AUTOMATIC DETECTION OF SOFTWARE SECURITY

VULNERABILITIES IN EXECUTABLE

PROGRAM FILES

Except where reference is made to the work of others, the work described in this dissertation is my own or was done in collaboration with my advisory committee. This dissertation does not include proprietary or classified information.

_____

Jay-Evan J. Tevis

Certificate of Approval:

_____	_____
Dean Hendrix Associate Professor Computer Science and Software Engineering	John A. Hamilton, Jr., Chair Associate Professor Computer Science and Software Engineering
_____	_____
David A. Umphress Associate Professor Computer Science and Software Engineering	Stephen L. McFarland Acting Dean Graduate School

AUTOMATIC DETECTION OF SOFTWARE SECURITY VULNERABILITIES IN EXECUTABLE PROGRAM FILES

Jay-Evan J. Tevis

A Dissertation

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Doctor of Philosophy

Auburn, Alabama
August 8, 2005

AUTOMATIC DETECTION OF SOFTWARE SECURITY

VULNERABILITIES IN EXECUTABLE

PROGRAM FILES

Jay-Evan J. Tevis

_____

Signature of Author

_____

Date of Graduation

DISSERTATION ABSTRACT

AUTOMATIC DETECTION OF SOFTWARE SECURITY

VULNERABILITIES IN EXECUTABLE

PROGRAM FILES

Jay-Evan J. Tevis

Doctor of Philosophy, August 8, 2005
(M.S., Air Force Institute of Technology, 1990)
(B.S., Iowa State University, 1985)
423 total pages

Directed by Dr. John A. Hamilton, Jr.

Secure programming describes those techniques that software developers use to provide security features in their applications. In addition to these techniques, software practitioners use static code security checkers to parse through and scan the source code, looking for potential security problems. Related to static code checking, runtime checkers have been developed that monitor the software while it is in use.

In an effort to counter the hacker threat, software security professionals need better methods and tools than these to analyze executable programs the way hackers do: from the binary data level. This level is where the hackers find the secret doorways and

security loopholes that are not evident in high-level source code. A few commercial companies have recently started marketing software products that will scan executable files for software security vulnerabilities; however, these products have unpublished methodologies and unverified test results. Consequently, software practitioners have only a loose collection of homegrown, commercial, and operating system software tools to perform their secure programming work and to do so in primarily a manual approach.

To help security analysts, programmers, and users detect security vulnerabilities in executable program files, we have created a methodology that uses information located in the headers, sections, and tables of a Windows NT/XP executable file, along with information derived from the overall contents of the file, as a means to detect specific software security vulnerabilities without having to disassemble the code. In addition, we have instantiated this methodology in a software utility program called findssv that automatically dissects an executable file and detects certain anomalies and software security vulnerabilities before installing and running the software.

We tested findssv on seven categories of files: software installation files, software development files, Windows XP operating system files, Microsoft application files, security-centric application files, and miscellaneous application files. We show through the test results on these 2700 files that findssv is able to detect table size anomalies, large zero-filled regions of bytes, unknown regions of bytes, compressed files, sections that are both writable and executable, and the use of functions susceptible to buffer overflow attacks. We also list sixteen key security vulnerability findings about software in the seven categories.

ACKNOWLEDGMENTS

The author would like to thank his advisor, Dr. Drew Hamilton, for his expert computer security guidance, constructive ideas, and "can-do" attitude throughout this time of research. Thanks also to the two graduate committee members, Dr. David Umphress and Dr. Dean Hendrix, for their review and comments on this research work. Special thanks to Dr. W. Homer Carlisle for his references on assembly language and his services as a sounding board for many of the novel approaches taken by this research. Finally, the author would like to thank his wife and children who endured many moves, travels, and separations for the author to complete this course of study and dissertation.

Style manual or journal used Journal of the ACM

Computer software used Microsoft Word 2000

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1 Making Software More Secure

The news of another virus threatening our computers has become a regularly expected occurrence. Moreover, we have come to accept the installation of a software patch as the preferred means to stop such malicious code. Companies have even devoted large amounts of resources to anti-virus teams and defense strategies to combat these problems. Such strategies include firewalls, intrusion detection mechanisms, honey pots, port monitors, system security scanners, and internet/e-mail content scanners [Grimes 2001]. Instead, we should look at ways to build more secure software and identify ways to detect vulnerable software code before installing it on our computers.

Secure programming describes the practices that software developers can use to provide security features in their applications. To study its relationship to software development, secure programming can be divided into the following categories: safe program initialization, access control, input validation, safe cryptographic usage, safe networking, safe random number generation, and anti-tampering.

In addition to secure programming practices, security researchers and practitioners have developed defense solutions targeted at decreasing the security vulnerabilities of computer systems. These solutions involve auditing all source code for vulnerabilities,

authenticating all software, giving security concerns a higher priority than increased functionality during software development, and preventing any unauthorized changes in the code baseline on a system [Grimes 2001].

1.2 Static Analysis of Source Code Files

For the vulnerability auditing solution, security experts have developed several software tools that provide a security check of C/C++ source code. The research concentrates on information about functions and data structures that pose a risk to the security of a computer system. These insecure items are a doorway through which malicious code enters to attack a system [Schaeffer 2002]. The security checkers search for these "doorways" in a source code file and alert the programmer to their presence.

Even though these security checkers provide some assistance in the prevention of security vulnerabilities, they have many weaknesses and only concentrate on the source code. The hackers, on the other hand, target the vulnerabilities in the executable code. In general, these are the same vulnerabilities that secure programming practices strive to prevent. By disassembling an executable program and looking for certain key indicators, hackers apply their tools of exploitation.

1.3 Static Analysis of Executable Files

In the past year, two commercial vendors began marketing security checking tools that scan executable files [HBGary 2004a, @stake 2004a]. According to their documentation, these tools perform an in-depth security analysis of the files; however,

they have published no detailed methodologies or test results demonstrating that their approaches actually work.

1.4 Objectives of this Research

This research effort also involves the scanning of executable files for software security vulnerabilities. Specifically, it involves the scanning of files that conform to the portable executable (PE) format designed for software running on Windows NT/XP computers. This effort sets out to prove the following hypothesis: A methodology can be devised that uses information located in the headers, sections, and tables of an executable file, along with information derived from the overall contents of the file, as a means to detect specific software security vulnerabilities without having to disassemble the code. Such a methodology can be instantiated in a software utility program that automatically detects certain software security vulnerabilities before installing and running the executable file.

To prove this hypothesis, this research effort first identifies specific information in the PE format that is useful in a security vulnerability analysis. It then formulates a methodology for identifying certain security vulnerabilities using this information. It incorporates this methodology into a software application called findssv that dissects a PE file and analyzes its parts. To test the hypothesis, the findssv program is run against seven categories of executable files (over 2700 files in all). Based on the test results, conclusions are drawn on the correctness of the hypothesis and on the usefulness of this approach.

2. LITERATURE REVIEW

This section summarizes the work done so far by others that impacts this research effort. It discusses secure programming techniques, general security defense strategies, specific software security vulnerabilities, static source code security checkers, common hacking techniques and tools, and examination of executable files.

2.1 Secure Programming as a Separate Discipline

The Internet and the World Wide Web continue to grow as extensions of the data storage and processing power of the personal computer. As a consequence of the resulting security risks, the need for secure programming practices also is growing. Moreover, secure programming is quickly becoming a separate discipline in computer programming and software engineering as evidenced by the books and articles published on the subject.

[Acar and Michener 2002] address the need for security to be a part of the initial software architecture rather than an add-on feature that users can opt to purchase. They also point out the need for security and software engineering to be integrated into computer science curriculums. [Anderson 2001] presents security engineering fundamentals, from protocols, passwords, cryptography, and access controls to the basics of security in distributed systems. [Evans 2004] discusses how security

properties can be placed as annotations into source code comments. These annotations provide a way for security scanning software to use preconditions to see if a function's implementation ensures the postconditions. [Goth 2002; McGraw 1998] talk about the need for software developers to move away from the "find and patch" method of software testing toward proactive software security and the establishment of security standards to reduce the impact of current vulnerabilities. [Howard 2004] lists a number of best practices to follow when developing software with a security mindset. [Soo Hoo, Sudbury and Jaquith 2001] discuss the return on investment of incorporating secure software engineering practices early in the software development life cycle. [Yoder and Barcalow 1997] provide a description of seven design patterns that provide a secure framework for building software applications.

[Graff and van Wyk 2003] cover the software side of security architecture, design, implementation, operations, and testing. In doing so, they use the approach of first identifying good and bad security practices and then presenting case studies. [Grimes 2001] looks at malicious code. It systematically dissects viruses, Trojans and worms, ActiveX and Java exploits, DOS viruses, macro viruses, browser-based exploits, e-mail attacks, and instant messaging attacks. [Hall and Chapman 2002] go through each of the stages of a software development process and propose security measures that developers can take at each stage. In these stages, formal methods are applied to greatly reduce security defects in the software. [Howard and LeBlanc 2002] discuss various secure coding techniques such as good access control, running with least

privilege, and using cryptography. They also look at socket security, RPC, ActiveX controls, DCOM, denial of service attacks, and web-based services.

[Kahn and Han 2002] look at how software components can make known their required and ensured security properties. They address how to characterize the security properties of components, how to analyze at runtime the internal security properties of a system comprising several atomic components, how to characterize the entire system's security properties, and how to make those characterized properties available at runtime. [Landwehr 1994] supplies a taxonomy showing how both intentional and unintentional security defects are introduced into software and where in the software development process that this can occur. For example, inadvertent security errors can be birthed through validation errors, object reuse, authentication errors, and boundary condition violations. [Du 1998] compares and contrasts several schemes for categorizing software security vulnerabilities and proposes a new approach based on cause, direct impact, and proposed fix. [Neumann 2003] discusses the problems of security vulnerabilities in end-user systems, routers, servers, and communication devices. It points out that software is regularly released with security flaws, and that patches introduce more flaws. The article closes by proposing that it is time to stop accepting bad software that is seriously unsecurable and require software security practices to be dramatically improved.

[Splaine 2002] covers how to plan a software security testing effort, how to test network and system configuration security, and how to check for security vulnerabilities in web-

based applications. [Viega and McGraw 2002] discuss guiding principles for building secure software. They strive to end the "penetrate and patch" approach to software security by emphasizing proper analysis and design techniques. [Viega and Messier 2003] supply a whole array of secure programming practices that are covered in more detail later in this section. [Wall 1999] provides information on secure programming in Linux. It addresses specific code issues for setuid programs, network servers, network clients, mail user agents, CGI programs, and various utilities and applications. [Wheeler 2004] lists Windows and Unix functions that are vulnerable to exploitation and discusses ways to reduce or eliminate these security problems.

[Short 2002] describes many kinds of vulnerabilities that are detectable by allowing unrestricted access to the source code. Although executable files can be disassembled or decompiled by tools that are easily obtained, these tools do not produce the original source code. Such source code is invaluable in understanding the logical flow of a program, the cause of the various branches, and the reason for certain function calls.

2.2 Secure Programming Techniques

Secure programming describes those techniques that software developers use to provide security features in their applications. Each of the following subsections summarizes a recommended secure programming technique. [Viega and Messier 2003] expand on each of these and provide examples using C and C++ code samples.

2.2.1 Safe Program Initialization

Safe program initialization refers to the values taken on by program constants and variables at the time a program begins execution. If also refers to the condition of any external resources utilized by the software. The goal is to validate as much of a program's environment as possible before any critical part of the application runs. A software application should make few, if any, assumptions about its environment in order to minimize the risks of many malicious attempts [Viega and Messier 2003].

2.2.2 Access Control

Access control refers to the need of a software application to protect the access to resources under its control. As soon as an application opens a file or port, access to that resource must be protected. In addition, an application should consistently use standard application program interfaces to system resources. Moreover, an application should use the minimum time needed in a privileged state in order to avoid race conditions whereby malicious attempts can try to gain access to specific resources during that time [Viega and Messier 2003].

2.2.3 Input Validation

Input validation refers to the need to confirm the content of any data read in by a program. No data should be assumed valid at any time during the life of a program. A policy of "default deny" should be enforced. In addition, no untrusted data should be used to control an application [Viega and Messier 2003; Whittaker 2003].

2.2.4 Safe Cryptographic Usage

Cryptography is the science of mathematical techniques for protecting data from malicious or unauthorized actions by transforming the data itself [Hankerson 2000]. Cryptographic protection involves the use of encryption and decryption techniques to safeguard the secrecy of data. It also involves message authentication techniques to detect data that has been tampered with [Viega and Messier 2003].

2.2.5 Safe Networking

Safe networking refers to the secure communication between two nodes over a network medium. This security ensures that both nodes are who they claim to be and the data exchanged by these nodes is protected from malicious attacks [Viega and Messier 2003].

2.2.6 Safe Random Number Generation

Safe random number generation refers to the creation of a continuing sequence of random numbers that is as close to nondeterministic as possible. This is a difficult task given that computers are inherently deterministic, and must be, to do the level and precision of calculations demanded of them. Methods for producing random numbers include insecure random number generators, cryptographic pseudo-random number generators, and entropy harvesters [Viega and Messier 2003].

2.2.7 Anti-tampering

Anti-tampering refers to techniques to protect the reverse engineering of binary code. Such action may be needed to protect software from malicious attempts to access proprietary data or algorithms, or find vulnerabilities in order to change the proper execution of the software [Viega and Messier 2003].

2.3 General Security Defense Rules When Developing Software

Along with security programming techniques are standard data validation rules to practice in software to prevent many software security vulnerabilities [Viega and Messier 2003]. These rules are based on the principle that all data should be filtered and then either accepted or rejected.

- Assume all input is invalid or incorrectly formatted until proven otherwise
- Prefer rejecting data to filtering data
- Perform data validation both at input points and at the component level
- Do not accept commands from the user unless you parse them yourself
- Beware of special commands, characters, and quoting
- Make policy decisions based on a "default deny" rule
- The better you understand the data, the better you can filter it

Along with good rules to follow in the source code, there are certain insecure coding practices that a software developer should avoid. See Appendix A for a summary of secure programming "Do nots" taken from [Graff and van Wyk 2003].

2.4 Specific Software Vulnerabilities to Avoid in Source Code

Attacks on software vulnerabilities vary from year to year as old bugs are fixed and new ones are found. Nevertheless, the one bug that invariably holds the top position in the list is buffer overflow. In a June 2000 study of the ten top vulnerabilities, cases of buffer/stack overflow were in the #1, #3, and #6 positions [Anderson 2001]. An interesting history of the birth of buffer overflow know-how is described in [Scambray, McClure, and Kurtz 2001]. It also describes how easy it is to use a web browser to modify a login page in an effort to abort a web server. This is done by changing the size of the userid or password field on the page, refreshing the page, entering long strings of characters into the text boxes, sending the client response, and watching the effect on the web server. [Schiffman 2001] contains a detailed scenario called Jack and Jill that recounts how an actual attack occurred on the Internet Information Server software using buffer overflows.

Buffer overflow attacks occur when a string of characters of unchecked length is entered into a program. This allows user-supplied input to overwrite other variables, thereby changing their values. Such attacks can change the value of a return address from a function call and cause control to jump to malicious code that was also entered via the buffer overflow. Some solutions are declaring all local variables in C as static to keep them off of the stack. Patches can be added to an operating system to make code in the stack non-executable. Modifications can be made to compilers to detect a possible buffer overflow situation [Lhee and Chapin 2002]. [Prasad and Chiueh 2003]

recommend a static translation of the contents of a binary file to incorporate a return address defense mechanism. This change protects the integrity of the return address on the stack by making a redundant copy of it. Canary values can be declared right next to string variables. The value of a canary value can then be checked after each string write using the assert function to see if its value has changed [Howard and LeBlanc 2002]. A version of the gcc compiler has been modified to automatically add canary values to functions [Wall, Watson, and Whitis 1999].

A variation of the buffer overflow attack is the exploitation of the mismatch between the sizes of Unicode characters and ANSI characters. The vulnerable function is MultiByteToWideChar() which has a buffer length parameter that can be changed by exploiting the stack [Howard and LeBlanc 2002]. Heap overflow attacks are also possible. A common way is to manipulate the bits maintained for each memory block in the free list. By doing so, a user can get calls to the free() function to overwrite memory locations with specific malicious data [Howard and LeBlanc 2002, Pincus and Baker 2004].

Array indexing attacks can allow a malicious user to write data to an arbitrary location in the data segment of a software application. Such data could change the constant value used in a conditional expression for example, thereby allowing the expression to return a true value to a larger range of user inputs [Howard and LeBlanc 2002]. This vulnerability exists because of the semantics of the array operator in C and C++. Such a vulnerability does not exist in Ada or Java because of the implicit bounds checking that

occurs in the runtime environment [Cohen 1986, Jaworksi and Perrone 2000; Lewis and Loftus 2005; Louden 2003].

Format string attacks using the "%n" specifier can make the printf() function write an integer value to an arbitrary location in memory. Because the printf() function allows a variable number of arguments, it doesn't know what number of arguments have been passed. A simple solution is to always pass a constant string as the format string, but then the values in this constant string could be changed [Andress 2002; Howard and LeBlanc 2002].

Standard C functions that do no range checking of character string inputs are vulnerable to function algorithm attacks. These functions include scanf(), gets(), sprintf(), vsprintf(), strcpy(), and strcat() [Schildt 2000]. Alternatives are available on some operating systems for each of these functions. These alternatives, such as fgets(), strncat, and strncpy require an additional string length parameter to counter any buffer overflow attempt [Miller and DeRaadt 1999]. Such measures are effective; however, a long string can still be entered into one function and the remaining unread part of the string will be input by the next function that reads input. [Viega and McGraw 2002] contain a two-page table of 31 standard C functions that should be used with caution or avoided altogether. A common characteristic across these functions is the passing of one or more character array parameters as character strings. The character array data structure in C and C++ has no built-in bounds checking for information written outside the range of the array indices.

Several types of system software applications can be exploited. Setuid programs have the setuid or setgid bits set, thus giving the program all the privileges of the file owner, which may be root. Network servers (daemons) can be continually attacked with data until one breaks and the attack is successful. Network clients are normally built with a lower concern for security than network servers. Clients such as browsers many times allow a server to execute code on the client machine. This code can easily be malicious software. Mail user agents are targets for buffer overflow attacks and malicious attachments.

CGI programs, which are run on a server, have the same if not more vulnerabilities as the server software because they are usually written in insecure scripting languages [Castro 2001; McComb 1997]. Utilities, such as those commonly available on a UNIX system, can be exploited through the use of special patterns of characters that may take advantage of buffer overflow or be interpreted in a special way by a shell program. Specific user applications such as office productivity software are vulnerable to malicious macro code embedded in documents [Wall, Watson, and Whitis 1999].

Inside each of these applications, various code features can be exploited. The use of certain commands in shell scripts such as eval() or function calls such as the system() function call in programs allows a malicious user to possibly execute any arbitrary command. An alternative to the system() call is the use of one of the functions in the

exec() family [Nutt 2002]. Although the exec() calls have fewer vulnerabilities, they should be chosen and used wisely [Viega and Messier 2003].

Changes in system environment variables can cause unexpected changes in the behavior of a program. An example is the change of LD_LIBRARY_PATH that can cause a program to link to code in a malicious library. Symbolic links can be changed or added to introduce vulnerabilities into a computer system. After such changes are made, many standard C functions (e.g., chmod(), chown(), link(), stat()) are vulnerable to allowing unauthorized access to certain files or directories [Wall, Watson, and Whitis 1999].

Host name attacks can occur. Information returned by the gethostbyname() function should not be trusted because a server can spoof the DNS response. A possible solution is to cross-check all responses using the gethostbyaddr() call [Wall, Watson, and Whitis 1999].

Signals that occur when a program is in a privileged state can cause vulnerabilities. [Wall, Watson, and Whitis 1999] list over 50 functions whose operation can be interrupted midstream by a signal. Through the use of a signal, a malicious user can induce a race condition involving a system command that executes in two or more user modes. An interruption while the command is in kernel mode can allow unbridled access to system-level files [Anderson 2001; Arce 2004].

One other area of exploitation to consider is core dumps. Malicious users can analyze a core dump to glean information on the value of program constants, variables, and registers. UNIX systems offer the setrlimit() function to disable memory dumps if an application crashes [Viega and Messier 2003].

2.5 Static Code Security Checkers

Static code security checkers parse through and scan the source code, looking for potential security problems. The process is similar to virus scanners. The static code checker looks through the source code for any of the known and previously defined problem conditions. Both false positives and false negatives may occur, and should therefore be used in conjunction with other security auditing and testing methods [Graff and van Wyk 2003].

The goal of static code security checkers is to focus the security analysis. Instead of the programmer searching the source code with a utility program such as grep(), the checker software is aware of known potential problems and searches for them based on encoded rules and entries in a database. These checkers not only find problems, they many also describe the problem and suggest possible remedies. In addition, they provide an assessment of the potential severity of each problem for an auditor to use in his overall assessment [Viega and McGraw 2002].

The security checkers differ according to the following criteria: the method of detecting security problems, the kinds of security problems detected, the way the problems are

reported, the suggestions offered for improvement, the host platforms, the availability of the auditing tool source code, and the proprietary or non-proprietary nature of the software. These criteria indicate a vast range in the features offered by each security auditing tool.

2.5.1 Inventory of Security Checkers

A list of currently-available static code security checkers is shown in Table 1. These checkers detect problematic code using proprietary heuristics to look for suspicious code segments, calls to specific utilities known to have vulnerability issues, or a combination of both [Chen and Wagner 2002; Dekok 2003; Evans and Larochelle 2002; Evans 2003; Gimpel 2003; Holzmann 2003; LDRA 2003; Parasoft 2003; Reasoning 2003; Secure Software 2004a; SPI Dynamics 2003; Viega et al. 2000; Wagner 2003; and Wheeler 2003]. The following paragraphs give a brief description of each of these code checkers.

Name	Focus	Author	Available From	Target Lang	License	Written in	URL
BOON	buffer overflow	David Wagner	University of California at Berkeley	C	Public	C	www.cs.berkeley.edu/~daw/boon/
CodeWizard	Dangerous code constructs	ParaSoft	ParaSoft	C, C++	Private		www.parasoft.com
Flawfinder	Function calls, gettext libraries	David Wheeler	David Wheeler	C, C++	GNU GPL	Python	www.dwheeler.com/flawfinder/
Illuma	Uninitialized variables, memory use and pointers	Reasoning	Reasoning	C, C++	Private		www.reasoning.com
ITS4	function calls, potential buffer overflows	John Viega	Cigital	C, C++	Public	C	www.cigital.com/its4
LDRA Testbed	general-purpose static code analysis	LDRA	LDRA Software Technology	Ada, C, C++, Java, plus many more	Private		www.ldra.co.uk
MOPS	violation of rules stated as temporal safety properties	Hao Chen	University of California at Berkeley	C	Public	Java	www.cs.berkeley.edu/~daw/mops/
PC-lint	general-purpose static code analysis	Gimpel	Gimpel Software	C, C++	Private		www.gimpel.com
PSCAN	Format strings	Alan DeKok	Alan Dekok	C	GNU GPL	C	www.striker.ottawa.on.ca/~aland/pscan/
RATS	Common security flaws	Secure Software	Secure Software	C, C++, Perl, Python, PHP	GNU GPL	C	www.securesoftware.com
Splint	buffer overflow, format errors	David Evans	University of Virginia	C	GNU GPL	C	www.splint.org
UNO	Uninitialized variables, nil pointers, index checking	Gerard Holzmann	Bell Labs	C	Public	C	http://spinroot.com/gerard/
WebInspect	Common security flaws	SPI Dynamics	SPI Dynamics	C, C++, Java	Proprietary		www.spidynamics.com

Table 1 - Static Code Security Checkers

BOON stands for Buffer Overrun Detection. As the full name implies, the software searches for buffer overruns in C source code. The concept behind BOON is that buffer overflow detection is an integer range analysis problem. The algorithm first takes the allocated size and the actual length of each character string and builds a corresponding value pair. This approach is also taken with the parameters of the standard C library functions that handle character strings. A comparison is then made to see if the inferred allocated size of the string is at least as large as its maximum length [Wagner et al. 2000].

CodeWizard is a proprietary general-purpose source code analyzer. It is not targeted specifically at security issues, and does not even advertise to do so, although the capabilities are there. Instead, it examines source code to locate violations of industry-accepted language-specific guidelines. This is done to reduce the opportunities for coding errors that could result in bugs. The analysis works by using a patented technology to search for patterns and then compare what is found to a set of rules [Parasoft 2003].

FlawFinder searches through C/C++ code looking for potential security flaws. After the code analysis is complete, it produces a list of potential flaws sorted by risk. FlawFinder's database contains both general rules that affect any program and specific Windows and Unix functions that are very vulnerable to exploitation [Wheeler 2003].

Illuma is proprietary software that searches C/C++ source code for problems such as memory leaks, null pointer dereferences, bad memory deallocation, out-of-bounds array access, and unitialized variables. It is used in conjunction with contracted services to assess the quality of a client's source code [Reasoning 2003].

ITS4 stand for It's The Software Stupid (Security Scanner) [Chess and McGraw 2004; Viega et al. 2000]. It statically scans C and C++ code for vulnerabilities, but it does not do so by parsing the actual source code used in a single build configuration. Instead, ITS4 looks at several files to check for vulnerabilities in multiple builds of the software. This is done for many reasons. First, it reduces the false negatives to almost zero.

Second, it avoids the complexities of real parsing that add no value to the security scanning requirement. Third, it allows the ITS4 software to be used real-time in integrated development environments to highlight potential errors from within an editor [Viega et al. 2000].

LDRA Testbed is proprietary software that performs a general-purpose static code analysis. It checks for such things as code complexity, unreachable code segments, variable interdependence, loop analysis, and correctness of procedure interfaces. It can also be used to verify a set of programming standards established by an organization; however, it does not specifically address security scanning as a possible use for the product [LDRA 2003].

MOPS stands for Model Checking Program for Security Properties. It checks for security vulnerabilities from a sequence of operations viewpoint. It uses model checking together with specific rules to detect the violation of temporal safety properties. A user describes the rules in the form of a finite state machine. If the software finds any problems related to a property, it prints out the offending path found in the source code. Such techniques can find potential issues with buffer overflow, user privileges, and array indexing [Chen and Wagner 2002; Wagner 2003].

PC-Lint is proprietary software that checks C/C++ source code to find such things as bugs, glitches, inconsistencies, non-portable constructs, and redundant code. The software can produce over 1900 distinct error messages. It does not specifically address

security vulnerabilities in its findings; however, a security analyst could spot many of these vulnerabilities in PC-Lint's error report. FlexeLint is a version of the PC-Lint software extended to non-PC platforms [Gimpel 2003].

PSCAN searches a C source code file for problematic uses of functions in the printf and scanf family, the syslog function, and a variety of functions used to display warning and error messages. It does not scan for normal buffer overflows or general misuse of function parameters [DeKok 2003].

RATS stands for Rough Auditing Tools for Security. It checks a variety of different language source code files for security-related problems such as buffer overflows and time-of-check vs. time-of-use race conditions. The software uses greedy pattern matching to find potential errors; consequently, false positives are prone to occur more often [Viega and McGraw 2002].

Splint stands for Secure Programming Lint. (It was previously known as LCLint.) The software checks that the source code is consistent with security properties stated in annotations. The annotations appear as comments and are associated with function parameters and return values, global variables, and structure fields. The annotations provide a way for the Splint software to use the preconditions to see if the function implementation ensures the postconditions. It resolves preconditions using postconditions from previous statements and annotated preconditions for the function [Evans 2003; Larochelle and Evans 2001].

UNO is named after the three focus areas of the software: use of unitialized variables, nil pointer references, and out of bounds index checking. It emphasizes these three areas to reduce the amount of false alarms produced by other static code checkers that try to look for everything. It also concentrates specifically on ANSI C source code. UNO has the ability to accept user-defined properties of application specific requirements, and then check the source code for strict compliance with these requirements [Holzmann 2003].

WebInspect is proprietary software that automates the discovery of security vulnerabilities in both traditional and web-based applications. It can be used in an integrated development environment to do static code analysis at the click of a button. WebInspect also makes recommendations on how to fix any potential security flaws that are found. SPI Dynamics, the maker of WebInspect, is part of a technical committee working on the definition of an Application Vulnerability Description Language (AVDL). The goal of the committee is to form an XML standard to define, categorize, and classify application vulnerabilities that can be understood and used by a variety of security products [SPI Dynamics 2003].

2.5.2 Critique of Static Code Security Checkers

Although source code checkers are very effective in detecting certain security vulnerabilities, they do have many shortcomings. The liberal syntax of C makes the language poorly suited to static analysis. The added object-oriented complexities of

C++ make it difficult to analyze. Static analysis in a multi-threaded environment is difficult because of the potential interaction of data. Performing a better static analysis using more advance algorithms is difficult and can cause an order of magnitude increase in scan time [Viega et al. 2000]. The static code checkers still require a significant level of expert knowledge. In other words, they work well for novice programmers; however, an expert can do a better job at manually evaluating the potential security vulnerabilities in the source code. Even for experts, analysis is still time consuming. The static code checker only cuts down about ¼ to 1/3 of the static code analysis that needs to be performed. The rest must still be done manually [Viega and McGraw 2002].

[Nazario 2002] points out more limitations in the current checkers. First, an automated scan has not been developed yet that catches many of the problems detected during manual analysis. Second, the scanners don't know the particulars of functions contained in libraries supplied by various domain-specific applications. Developers need to understand this so they don't think the checker looks at such things. Third, most checkers scan at most two languages. An exception is RATS, which can scan five. Fourth, the checkers perform no preprocessing that would expand macros or constant definitions.

[Wilander and Kamkar 2002] report similar limitations when comparing the performance of ITS4, FlawFinder, RATS, Splint, and BOON. They concluded the following:

"We have shown that the current state of static intrusion prevention tools is not satisfying. Tools built on lexical analysis produce too many false positives leading to manual work, and tools building on deeper analysis on the syntactical and semantical level produce too many false negatives leading to security risks. Thus the main usage for these tools would be as support during development and code auditing, not as a substitute for manual debugging and testing."

In a study on the reliability of static code security checkers, [Dor, Rodeh, and Sagiv 2003] discovered that the checkers miss certain character string errors, yield many false alarms, and cannot handle multilevel pointers and structures in C. In response to these problems, they formulated a way to detect all string manipulation errors by incorporating pre- and post-condition contracts into the source code of programs. These annotated programs are then subjected to a multi-stage analysis that performs static string verification in order to detect problem areas. Part of their methodology reduces the problem of checking for character string manipulations into a simpler problem of checking for integer manipulations.

Nevertheless, the checkers are still useful in a small way. These tools help to prevent the rush to check the security vulnerabilities of every piece of source code. Because of the prioritization and assessment features, they focus the analyst's attention on the more severe problems that may have manually been overlooked. They can help find real bugs. These tools actually work to find problems in just a few minutes that may have taken much longer to detect [Viega and McGraw 2002].

2.6 Runtime Code Security Checkers

Along with static code checking, runtime checkers have been developed also. Running in a layer between the application and operating system, these checkers work by intercepting system calls and screen each call for correctness before passing it to the operating system to be executed. Example products are Libsafe, PurifyPlus, and Immunix tools [Graff and van Wyk 2003].

Another related method is use profiling. The concept works as follows. The behavior of a program's system calls and file activity is studied for a number of software executions and then defined. The profile is then used as a basis to monitor the software activity for any behavior anomalies. Such anomalies could indicate malicious actions by an application or the presence of virus software. Example user profiling tools are Papillon, Janus, and gprof [Graff and van Wyk 2003].

Potential buffer overflow is undoubtedly the most searched for problem in static code checking. Run-time checking of this problem can also be done by executing destructive tests intentionally designed to detect the existence of a buffer overflow vulnerability. Example tools for testing buffer overflow are NTOMax and SendIP. Test tools can also be built to record and play back data submitted to a software application [Splaine 2002]. [Hunt and Brubacher 1999] have created a software library that instruments Win32 function calls without affecting the original binary files. This allows easier inspection and debugging of executable files. [Yong and Horwitz 2003] describe how they

automatically instrument the source code to check at runtime for invalid pointer dereferences. [Ghosh, O'Connor and McGraw 1998] have developed a process that uses fault injection analysis to automatically test the vulnerability of security-critical software.

[Cowan et al. 1998] describe the use of a patch to the Gnu C compiler to combat buffer overflow attacks. This patch, called StackGuard, virtually eliminates buffer overflow vulnerabilities through the use of canary values to detect changes in the data stored in locations outside the bounds of an array.

[Jiwnani and Zelkowitz 2004] recommend that security testing be focused in the areas of greatest vulnerabilities. They have created a susceptibility matrix of vulnerabilities by identifying error-prone system software components. For example, they found that implementation-level high-risk areas in software are common among Windows and Linux. This is in spite of different security policies and development histories.

2.7 Other Approaches Involving Executable Files

[DuVarney, Bhatkar, and Venkatakrishnan 2003] have proposed changes to the executable file format produced by linkers in order to enhance the security of programs. They recommend an extra section that contains the address, size, and alignment requirements for each code and static data item in the program. [Haugh and Bishop 2003] describe how to automatically instrument a source code file with additional code

so that the resultant executable file detects when and where buffer overflow attacks can occur.

In a related area, [Christodorescu and Jha 2003] investigated the usefulness of virus scanners to detect malicious patterns that had been obfuscated in executable code. They found that scanners could easily be defeated by simple code transformations. In response to this problem, they developed a methodology to detect virus code that involves the creation of a generalized automaton reflecting the virus code's dependency on certain data variables. This automaton is then converted into a control flow graph and compared to other control flow graphs formed for each procedure in the executable code.

2.8 Hacker Attacks

2.8.1 The Hacker Strategy

The hacker community considers itself to be a group of ordinary people providing a much-needed service to the computer software users of this world. Hackers proactively find holes and weaknesses in software to create their own exploits [Utimaco 2004, Wong 2001]. They cover their tracks by breaking into insecure systems and using them to launch attacks against other systems [Wall, Watson, and Whitis 1999]. Hackers see themselves as a pseudo-extension of the security teams paid by software companies to test software for vulnerabilities [Khalilzad, White, and Marshall 1999]. They feel that hacking is really just the act of finding a clever and counterintuitive solution to a problem. The hacks found in program exploits usually deal with using the rules of the

computer in ways never intended in order to achieve results that are usually focused on bypassing security [Erickson 2003, Kaspersky 2003]. One example uses a technique that relies on http traffic only to attack and penetrate web and applications servers [Shah 2004].

The Network Systems Survivability Program at Carnegie-Mellon University has identified a vulnerability exploit cycle that occurs with hackers [CERT 2002]. This cycle involves the following stages:

- Advanced intruders discover a new vulnerability through software testing and code examination
- Crude exploit tools are distributed in the form of scripts or a collection of command-line inputs
- Novice intruders pick up and use the crude exploit tools
- Automated scanning/exploit tools are developed and distributed via FTP servers, web sites, bulletin boards, or some physical means
- Widespread use of automated scanning/exploit tools occurs, thereby causing the use, attempt and success to be at its peak
- Patches are installed, but the number of exploits never become negligible because of poorly-maintained systems or systems redeployed with default configuations
- Intruders begin using new types of exploits on newly-discovered vulnerabilities

[Arbaugh, Fithen and McHugh 2000] used historical data obtained from the Network Systems Survivability Program to identify a similar life cyle model for software vulnerabilities. Their model consists of seven states. The first three states (birth, discovery, and disclosure) always occur in order. The next three states (correction, publicity, and scripting) occur in any order, with scripting occurring only in some circumstances. Finally, the vulnerability enters the death state. Sometimes this occurs after a few days, but often it takes many years because system administrators do not get all the holes patched immediately.

Many programmers write in high-level languages such as C or C++. Such languages assume that the programmer is responsible for data integrity. If this responsibility were shifted over to the compiler, the resulting executable programs would run significantly slower due to integrity checks on every variable. Also, this would remove a significant level of control from the programmer and complicate the language. When working at that high level, the programmer doesn't consider physical variable memory, stack calls, execution pointers, and other low-level machine commands. Hacking at the lower level involves knowing more of the rules and using them in ways never anticipated [Erickson 2003].

2.8.2 Hacker Tools

Hackers work with assembly code, or more accurately, an executable program that has been disassembled into assembly code. One of the major goals of a hacker is to get a

return instruction in a computer program to branch to an unplanned location in memory where a malicious payload awaits. Unplanned branching occasionally occurs in programs without hacker intervention due to logical errors when working with memory addresses. For example, such branching to a read-only code segment or to an invalid address causes a general protection fault in Protected Mode on an Intel32-based computer [Irvine 2003].

Low memory addresses	text segment
	data segment
	bss segment
	heap segment
	(unallocated)
High memory addresses	stack segment

Figure 1 – Program Organization in Memory

Hackers understand the organization of an executable program in computer memory and use it to their advantage. Program memory is normally divided into five segments: text, data, bss, heap, and stack as shown in Figure 1 [Erickson 2003].

The text or code segment is where the machine language instructions reside. This segment is read only. The data and bss segments store global and static program

variables. They are writeable and have a fixed size. The heap segment is used for dynamically allocated memory. It grows and shrinks over the course of program execution. The stack segment is used in the implementation of function calls and parameter passing. It also grows and shrinks over time. When a function is called, a stack frame (or activation record) is the area on the stack set aside for a procedure's return address, passed parameters, any saved registers and local variables [Erickson 2003, Irvine 2003].

Along with the memory organization, hackers understand registers. Registers are very fast memory located near or as a part of the central processor. The Intel32 family of processors has a standard set of registers categorized into general purpose, segment, control, and other registers [Intel 2004]. Examples of the general-purpose registers are EAX, EBX, and ECX, where the 'E' stands for extended. These are 32-bit registers that are used for computation and comparisons. The segment registers are 16-bit registers with names such as CS, DS, and SS. They are used to track the location of segments in memory. The "other" category is simply a collection of miscellaneous registers. A well-known one is the EFLAGS register, which contains various run-time status flags such as results from comparisons [Koziol 2004].

To exploit vulnerable executable programs, hackers spend weeks and months performing a line-by-line analysis of disassembled code. Over time they have collected or written an unorganized bag of software tools to semi-automatic their endeavors.

Some of these tools were originally designed for software development and maintenance, such as debuggers and execution tracers. Others have been built specifically for hackers to exploit software. In addition, others have been used in a process for performing vulnerability assessments on simulation software [Hamilton, Greaney, and Evans 2003]. Some researchers claim that hackers have a distinguishable programming pattern in the software tools they develop [Spafford and Weeber 1992]. Appendix B provides a list of commonly used hacker tools. [Schwarz, Debray, and Andrews 2002] have found that some disassemblers do not produce accurate assembly code and just fail silently. This occurs because of confusion from indirect jumps and the presence of non-executable data such as jump tables and alignment bytes.

2.8.3 Hacker Techniques

Hacking involves thinking about things that weren't anticipated. The two most common types of hacker techniques are buffer overflow exploits and format string exploits. The goal of both is to get an injection vector to strategically place a memory address so that it causes program control to transfer to a malicious payload [Erickson 2003].

Injection vectors must take into account several factors: the size of a buffer, the alignment of bytes, and restrictions on characters sets. Consequently, injection vectors are usually coded into a properly formatted protocol of some kind. The memory address of the payload must be known to the attacker and must be placed directly into the injection vector [Hoglund and McGraw 2004].

In the case of buffer flow, a program that runs fine for years might suddenly crash when a hacker decides to try to input a thousand characters into a field that normally only uses several dozen, such as a username field. Buffer overflow can occur either in the stack segment or in the heap segment. The ability to overwrite any arbitrary address in a program's memory space opens up many possibilities for exploitation. Basically, any section of memory that is writeable and contains a memory address that directs the flow of program execution can be targeted [Erickson 2003].

Buffer overflows are a relatively simple concept and explanations of them appear in most computer security books. Sometimes data can extend past the perceived boundaries of a record or array, and sometimes there are ways to take advantage of that. This is obviously true in languages such as C and C++ that do no bounds checking. With stack-based overflows, it's usually just a matter of finding the right location to place the return address. However, with heap-based overflows, creativity and innovation are needed because the exploit must involve a function calling another function [Erickson 2003]. In studies done by [Hoglund and McGraw 2004], they have found that corrupting memory remains the single most powerful technique for the attacker. They state that perhaps stack overflows will go away when programmers stop using the seriously broken Standard C library calls.

The injection vector and payload are usually delivered into a vulnerable program by means of character string input parameters. Consequently, the crafting of an injection vector and payload involves the proper choice of characters. Because the ASCII null character acts as the string termination symbol in C and C++ programs, hackers have had to find ways to avoid operands or operators that assemble into null bytes [Erickson 2003]. One way to overcome the negative effects of NULL bytes in injection vectors is to XOR the string [Hoglund and McGraw 2004]. Another way is to use the subset of machine instructions that correspond byte-wise to printable ASCII characters. This restriction makes writing shellcode, that is, code that spawns a command shell, significantly more difficult. [Erickson 2003] contains a number of examples of printable ASCII strings that correspond to executable code.

2.8.4 Common Hacker Targets

Based on experience with well-known techniques and tools, hackers tend to focus on certain targets. They include the following [Erickson 2003, Ahmad 2003]:

- Off-by-one error on arrays
- Adapting software from 8-bit ASCII to 16-bit Unicode
- Multiple backslashes in file paths
- Placing shellcode in an environment variable
- Printf format strings
- Other string-based Standard C library functions

- Constructors and destructors in GNU C
- Shellcode using printable characters
- Clearing out all stack memory
- Returning into the Standard C library (libc)
- Integer errors

The secure programming practices and input validation rules covered in earlier sections of this document are designed specifically to help eliminate these targets.

2.8.5 Hacking Over a Network

Hacking is not just limited to stand-alone software. Network software that involves communication protocols is also vulnerable. According to one hacker author, uninspiring and repetitious following of protocols may not be desirable for humans, but it's ideal work for a computer. The creativity and intelligence of a human mind is better suited to the design of protocols, the creation of programs that implement them, and the invention of hacks that use them to achieve interesting and unintended results [Erickson 2003].

Server software has come to be a major target of hacker exploits. The root cause of the server software problem is one of trusted input. Server software that exposes its functionality to whatever may come over the network must be built defensively. A common assumption of server software builders is that only the corresponding client

software will be used to access the servers. Instead, there is no need for an attacker to user particular client code to generate input to a server. The attacker can just send well-formed network traffic from a custom-built client program [Whittaker and Thompson 2004].

Along with server software problems, client software programs are almost never tested, let alone tested explicitly for security vulnerabilities. For example, the exploit code ends up executing with the same permissions that the user has [Whittaker and Thompson 2004].

2.9 Software Security on the Offensive

2.9.1 Informing the Software Developers

Countering the exploits of hackers are software security professionals [Arce and McGraw 2004]. These professionals believe that teaching software developers how hackers operate helps them build better defenses in their software and also apply secure programming practices. This is based on the philosophy that completely removing software security vulnerabilities from source code in advance is much better than trying to catch them when they are exploited at runtime. For example, when it comes to defending against malicious input, white listing is superior to a black listing approach. White listing is the exhaustive listing or defining of all acceptable program inputs [Hoglund and McGraw 2004].

Popular hacking books today focus mainly on existing exploits of network security issues. They do not strive to train the practitioner to find and eliminate new software exploits. Consequently, professionals writing software for secure systems are unaware of what they are up against [Bishop 2005, Hoglund and McGraw 2004].

Software risk from hacker exploits can only be measured and assessed relative to a particular environment. A threat may be a minimal risk in one setting, but be catastrophic in another. In the risk assessment approach proposed by [Hoglund and McGraw 2004], they measure only the damage to software assuming that a capable attacker exists. Consequently, if there are no capable attackers, there is no risk.

Software security is sometimes compared to software safety [Leveson 1995]. The difference between the two from an analysis standpoint is the addition of an intelligent adversary with the goal of making the system break. This so called "break" could have the side effect of compromising the operating system; consequently, exploiting a system through software becomes much easier. Advanced technology for scanning code is good at finding implementation-level mistakes, but there is no substitute for experience. Advanced technology for securing applications is excellent for making sure that only approved software is executed, but it is not good at finding vulnerabilities in executable programs. One collection of software security vulnerabilities is www.bugtraq.com. Another collection described by [Christey et al. 1999] is the Common Vulnerabilities

and Exposures (CVE) database administered by MITRE and located at www.cve.mitre.org.

[Arora and Telang 2005] studied the advantages and disadvantages of publicizing software security vulnerabilities. They first looked at the response by software vendors whose products were affected by vulnerabilities. For those vendors whose vulnerabilities were published by CERT (Computer Emergency Response Team), 77% of vendors responded with a software patch in an average of 242 days. For those vendors whose vulnerabilities were published by BugTraq, 60% of vendors responded with a software patch in an average of 390 days. They then studied the response by software attackers. They found that publishing vulnerabilities and patches both attracted attacks. This seems intuitive for publishing a vulnerability; however, it turns out that publishing a patch alerts the attackers on how to compromise the vulnerability even further.

From a software engineering perspective, [Rescorla 2005] questions whether the quality of software is improving even when security vulnerabilities are discovered and fixed. His results are inconclusive, but he does point out that current software development practices continue to produce security problems in software even with the knowledge of how to prevent them.

2.9.2 Attack Patterns

Software security vulnerabilities can be grouped together by common characteristics and fall prey to certain attack patterns. This is based on the premise that related programming errors give rise to similar exploit techniques. An attack by a hacker starts with breaking rules and undermining assumptions. One of these assumptions is "implicit trust". Attackers will break the rule on what the program expects the user to enter. [Hoglund and McGraw 2004] have outlined a collection of attack patterns. An attack pattern is a blueprint for exploiting a software vulnerability. As such, an attack pattern describes several critical features of the vulnerability and arms an attacker with the knowledge required to exploit the target system. Each attack consists of an injection vector and a payload.

A successful hacker attack takes several logical steps. First qualify the target, mainly to learn what input points exist in the software. Then figure out the kinds of transactions that are accepted at the input points. This will involve an in-depth machine instruction analysis. Once a vulnerability is discovered, try to exploit it and thereby gain access to the system.

Most vulnerabilities can be found by examining the following key areas [Bishop 2005, Hoglund and McGraw 2004]:

- Functions that do improper (or no) bounds checking
- Functions that pass through or consume user-supplied data in a format string

- Functions meant to enforce bounds checking in a format string
- Routines that get user input using a loop
- Lower-level byte copy operations
- Routines that use pointer arithmetic on user-supplied buffers
- So called "trusted" system calls that take dynamic input

In a typical case, white box analysis is used to find potential problem areas in the software, and black box testing is then used to develop working attacks against these areas [Ghosh and McGraw 1998; Potter and McGraw 2004]. This results in a kind of gray box approach that may include backtracing [Hoglund and McGraw 2004]. The first step in backtracing is to identify potentially vulnerable calls. Once the hostile input is determined, the tester tries to backtrace through the target program to determine whether an attacker can apply the hostile input from outside the program.

2.9.3 Counter Attacks

[Jim et al. 2002] describes the use of a security-focused version of C called Cyclone. This version keeps all the low-level features of C but prevents buffer overflows, format string attacks, and memory management errors.

[Barrantes et al. 2003] describe how to randomize an instruction set in order to disrupt binary code injection attacks. They have devised a way to randomize the binary code at

load time and then pass the code through an emulator to convert it back to the instruction set recognized by the computer processor.

Randomization is also used by [Bhatkar, DuVarney, and Sekar 2003] to thwart hacker attacks. However, they use randomization in a process called address obfuscation. This process repositions the locations of the data and code in a program at link and load time.

A means of preventing the execution of software on a specific computer is proposed by [Kirovski, Drinic, and Potkonjak 2002]. They describe a method of ensuring trusted software integrity by causing an attacker to solve a computationally intense problem concerning the format of an executable file before he can create a program that can be executed on the computer.

2.10 Examining Executable Files

Instead of scanning source code files for security vulnerabilities, it is possible to directly examine executable files. Two commercial companies are already marketing tools to perform such an analysis: BugScan from HBGary [HBGary 2004a] and SmartRisk Analyzer from @stake [@stake 2004a]. At first it may appear that the scanning of executable files would only be necessary if the source code is not available. However, we must remember the techniques of the hacker who looks directly at disassembled executable code to find vulnerabilities. In addition, an executable file contains much more useful information than just the program code and data.

2.10.1 The Format of an Executable File

The executable files of software applications that run under a version of Microsoft Windows conform to a specification developed by Microsoft [Microsoft Corporation 1999, Minasi 2001]. This specification describes the contents of object code files, executable files, and dynamic link library files [Dabak, Borate, and Phadke 1999]. The term "image file" is used to refer to both ".exe" and ".dll" files. Files of these two types differ only in their use, not in their content [Pietrek 2002a].

A compiler or assembler places information in an object code file according to the Microsoft common object file format (COFF). A typical 32-bit COFF object code file contains a file header, an optional header, a section table, a symbol table, a string table, an import table, possible other tables, and sections for code and data. The file header identifies the computer type, the number of sections, a time/date stamp when the file was created, a pointer to the symbol table, a count of the number of symbols in the symbol table, the size of the optional header, and flags indicating certain characteristics of the file. The optional header normally only appears in image files. The section table lists information on the various sections located in the image pages of the file.

A linker extracts information from one or more object code files and libraries in order to build a single executable file according to the Microsoft portable executable file format (PE). This PE format is designed to work on all versions of Microsoft Windows and supported CPUs. A typical 32-bit PE file contains an MS-DOS stub and a PE signature followed by the same areas that appear in an object code file. This is shown in Figure 2.

Type or Structure Name	Name in Specification
IMAGE_DOS_HEADER	DOS Header
None	MS-DOS Stub
DWORD	PE Signature (32 bits)
IMAGE_FILE_HEADER	File Header
IMAGE_OPTIONAL_HEADER	Optional Header
IMAGE_SECTION_HEADER	Section Table

Figure 2 – Typical Layout of the Portable Executable File Format

The MS-DOS Stub is a very small program that runs under MS-DOS and simply displays the message "This program cannot be run in DOS mode" when the executable file is run in MS-DOS.

2.10.2 Sections in an Executable File

An executable file may contain many sections, although only a code section and a data section are mandatory. Each section has a header in the section table. This header lists the location, length, and characteristics of the section. One interesting characteristic to hackers is if the section's attributes are set to read only or read/write [Woodmann 2004]. Although the text section is read only, both the normal data section and the uninitialized data sections are read/write. The combination of both the writeable attribute and the executable attribute for a section creates a vulnerability for hackers to exploit.

Some other sections are the exports section and the imports section [Pietrek 2002b]. The exports section contains the functions or variables made available by the file to other executable files. The imports section lists the functions or variables located external to the file including the files where they are contained. Such information can indicate the DLLs that are needed by a certain program. Hackers take advantage of the information stored in the section table and in the various sections by using available software tools, such as those listed in Appendix B, to examine executable files.

An executable file may also contain symbol table information that was placed earlier in the object code files and was used by the linker. Section names, file names, code symbols, and data symbols are listed in the symbol table. A linker option allows information from the symbol table to be stripped from the executable file or be included in the executable file for use by a debugger [Microsoft Corporation 1999].

[Huang 2003] found that the PE format is subject to a number of security vulnerabilities because of the amount of non-executable information stored in the file. Such vulnerabilities permit the file to be easily modified either manually or by other software.

2.10.3 BugScan from HBGary

BugScan is a commercial tool from HBGary that analyzes executable files to find security vulnerabilities. BugScan was unveiled to the public at the Black Hat USA conference on July 28-29, 2003 in Las Vegas, Nevada. It sells for $19,500 with a

$3,900 yearly maintenance fee after the first year. BugScan is a hardware and software combination appliance that attaches to a user's computer network. The hardware consists of a Dell PowerEdge 650 with a 2.4 GHz Pentium 4 processor, 256MB RAM, and 80GB of disk storage [InfoWorld 2004, Zacker 2001]. The software consists of an application server containing the actual binary code scanning utilities. According to the press release, "BugScan works simply by submitting binary files to the application server via a web interface. A report is generated for each file detailing specific coding errors found, coding error locations, problem severity and remediation advice. Binary locations of errors can be cross-referenced to source code lines or functions where programming errors can be fixed [HBGary 2004a]."

BugScan can be used to statically scan C and C++ programs. It looks for insecure C library calls (approximately 35 signatures), buffer overflow problems, format string vulnerabilities, poor use of the pseudorandom rand() function in C, signed/unsigned variable conversion errors, and poor exception handling [HBGary 2004a].

According to the technical white paper, BugScan works in the following manner [HBGary 2004b]. First, BugScan slices the code in the executable file into sections and creates a control flow map among the sections. This is done to determine what code is being called and from where. Next, BugScan applies pattern detection to find potentially vulnerable locations in the code such as certain API calls. For example, BugScan checks the size and type of the arguments passed to a function. It also uses backtracing from the vulnerable point in an attempt to reach a location in the code

where user-supplied input of data occurs. However, this process is normally too involved because the backtracing is too deep. In addition, branching may be encountered, thereby making the control flow only determinable at run time.

The technical white paper goes on to describe the report features of BugScan [HBGary 2004b]. The report contains red, yellow, and green alerts. Red alerts are high severity problems caused by a misuse of APIs. Yellow alerts are medium severity problems related to insecure coding practices when bad APIs are used. This includes functions related to buffer overflow, poor use of random number generation, and race conditions that may allow data disclosure or logic errors. Green alerts point out good coding practices such as the use of a more secure library function. Information provided with a red or yellow alert describes how to correct the problem and also gives a memory address to point to the location in the executable file where the problem was detected.

The user's manual for BugScan states that the product performs a static white box analysis providing 100% code coverage with no input from the user, providing an advantage over black box techniques [HBGary 2004c, Pressman 2005, and Sommerville 2001]. It also lists the following purposes for using BugScan:

- Ensure that only systems beyond a certain security threshold are being deployed
- Automate some aspects of security code reviews
- Verify and enforce that developers are writing secure code
- Provide online training for developers in writing secure code
- Evaluate commercial-off-the-shelf (COTS) software before a purchase

The manual points out that the supported file types for BugScan are Windows x86 portable executable (PE) binary files using the Microsoft Visual C library, and Linux x86 executable and linking format (ELF) binary files using GNU C library. In the troubleshooting area of the user's manual, it cautions that if a program has implemented its own version of C library calls, BugScan cannot currently detect their usage. In other words, BugScan can only detect the use of C library calls in the supported platforms, compilers, and libraries.

The Frequently-Asked Questions (FAQ) document for BugScan brings out some interesting information on the product from the point of view of the developers (HBGary 2004d]. They state that BugScan does not make it easy for hackers to develop new attacks. They also add that the security vulnerability information provided by BugScan only optimizes a small part of the exploit development process. It still requires a very skilled person to do the additional work to produce a working exploit.

The FAQ also states that BugScan makes no claim that a detected security coding error creates an exploitable vulnerability. They explain that it is difficult to determine with any amount of certainty if a problem detected is truly exploitable.

2.10.4 SmartRisk Analyzer from @stake

SmartRisk Analyzer is also a commercial tool that analyzes executable files to find security vulnerabilities. It is sold by a company called @stake. SmartRisk Analyzer

was announced to the public on May 24, 2004. Its selling price starts at $30,000 [PC Magazine 2004]. SmartRisk Analyzer supports C, C++, and Java (J2EE) languages running on Windows or Solaris platforms. The system requirements are Windows 2000 or higher, 2 GHz CPU, 2GB RAM, and 100MB disk space [@stake 2004b].

According to the press release [@stake 2004a], SmartRisk Analyzer uses deep static analysis of the application binary code that maps application control and data flow paths into a comprehensive security model. It looks at the variables introduced in the software's runtime environment and allows developers to identify security vulnerabilities introduced by third-party libraries. SmartRisk Analyzer builds a multidimensional model of the application and runs hundreds of risk analysis scans against the model to identify and prioritize security vulnerabilities. The scans find flaws related to insecure or improper use of programming languages and standard libraries and flaws that may result from the deployment platform on which the applications runs. It also detects vulnerabilities from input validation, command and script injection, backdoors, and malware (i.e., viruses). Flaws are classified and grouped by level of priority from severe to informational.

The press release and the product datasheet for SmartRisk Analyzer list the following features [@stake 2004a, @stack 2004c]:

- Automated secure code assurance (runs the code analysis automatically)
- Deep binary analysis (maps application control and data flow paths into a comprehensive security model)

- Linked library analysis (searches external libraries for security flaws)
- Comprehensive security scans (scans for hundreds of security flaws)
- Risk analysis module (reports flaws by type and severity)
- Remediation module (annotates errors on source code, if availability)
- Extension security rules module (allows new security scans to be added)
- Advanced vulnerability reporting (provides reports for QA and management)
- Program and library analysis (looks for improper use of programming languages and standard libraries)
- Application platform analysis (looks for platform-related vulnerabilities)
- Additional security modules (looks for vulnerabilities not covered by other modules)

The vulnerabilities detected by SmartRisk Analyzer include stack/heap buffer overruns, format string risks, integer overflows/underflows, threading/race conditions, return code checking, network access, privilege escalation, weak cryptography, input validation, backdoors, and command/script injection.

The white paper for SmartRisk Analyzer outlines the four-step process used in analyzing a binary file [@stake 2004d]:

1. Load the binary file, third party libraries, and platform environment data for comprehensive analysis
2. Develop the security model of the application including data flow, control flow, and range propagation

3. Perform security analysis on the application model and risk analyses on the output to rank results
4. Generate both detailed and summary reports for security flaws that were detected

In contrast to the approach taken by HBGary's BugScan to inspect the executable file itself, SmartRisk Analyzer loads the executable file into memory just like the operating system would do. This allows the analyzer to determine all of the interfaces that the program makes to its environment through dynamic link libraries. The analyzer then does a reverse link to break the program up into the individual functions in order to create a call graph. This call graph provides the data flow modeler with the information needed to graph the data flow among functions in the program. The control flow graph is then generated by analyzing the branch instructions in the program. This detailed control and data flow model of the program is subjected to an in-depth security analysis of over 400 scans to identify potential flaws that may become actual flaws. Each flaw is ranked as a severe error, a normal error, a possible error, a warning error, or an informational alert.

2.10.5 Code Security Evaluation by Secure Software

A third company, called Secure Software, is also involved with the security scanning of executable files; however, they are offering it as a service rather than as a product [Secure Software 2004b]. The service, entitled "Code Security Evaluation", provides a comprehensive and detailed evaluation of the source code or binary code of an

application to detect, identify, and validate security flaws. According to [Immix Technology 2004], the GSA schedule pricing for this service ranges from $31,200 to $46,800.

This scanning service has already been contracted by the Navy for use in scanning the software to be used on the prototype of the Navy-Marine Corps Intranet (NMCI). It is part of a two-pronged test aimed at determining whether specific applications create vulnerabilities and whether they work well under Microsoft Windows 2000 and XP. The source article states that Secure Software is using its Code Security Evaluation and Information Assurance Review software to run the tests. It also states that Code Security provides an automated security review process for software in development [Onley 2004].

3. STATEMENT OF RESEARCH OBJECTIVES

This research effort sets out to prove the following hypothesis: A methodology can be devised that uses information located in the headers, sections, and tables of an executable file, along with information derived from the overall contents of the file, as a means to detect specific software security vulnerabilities without having to disassemble the code. Such a methodology can be instantiated in a software utility program that automatically detects certain software security vulnerabilities before installing and running the executable file.

To prove this hypothesis, this research effort has the following objectives:

1) Do an in-depth study of the PE format of an executable file to discover any information that could be useful in detecting software security vulnerabilities. This involves the examination of the Microsoft PE format specification through the eyes of a security analyst armed with the list of "specific software security vulnerabilities" as described earlier in Section 2.

2) Create a software utility to dissect a PE file byte-by-byte from beginning to end (without disassembling) in order to identify, map, and categorize its contents. This means building a software utility that understands the various header,

section, and table information in a PE file and correctly translates the relative virtual addresses and pointer offsets found in those items. The dissection is necessary so that every part of a PE file will be accessible to whatever security vulnerability analysis will come next.

3) Formulate a methodology for combining and correlating the information found in a PE file in an effort to detect indicators of certain security vulnerabilities. This involves the need to look at the information from a viewpoint that does not consider what the loader does with the information or what happens at runtime, but rather what the information reveals about the security nature of the file itself.

4) Incorporate the methodology into the PE dissecting utility in order to automatically detect certain software security vulnerabilities in seconds. The automation factor is a must to make the methodology of any practical use.

5) Test the automated methodology on installation files, software development files, Windows XP operating system files, Microsoft application files, security-centric application files, and other application files. Such a wide breadth of testing is necessary to demonstrate that the software utility can handle PE files designed for a variety of purposes. Also, such testing will provide a large amount of results to evaluate the correctness of the hypothesis and the usefulness of the methodology.

6) Analyze the test results and make conclusions. This involves a thorough review of the results to see not only how many and what kind of security vulnerabilities were found but also what do these findings mean. The analysis will need to answer several questions. Do the vulnerability indicators (as stated in the methodology above) occur in the executable files of actual software? And when they do, what do they reveal about the overall security of the software? Are there trends in the test results that point to categories of software, or even specific software applications, which are more or less secure than others? Can we conclude that the methodology actually works, that is, is the hypothesis correct?

4. DESCRIPTION OF RESEARCH RESULTS

4.1 Explanation of Terms

Throughout this section, the terms listed below are used when describing the contents of a portable executable (PE) File.

Common Object File Format (COFF). This refers to a standardized file format created by Microsoft and used by the Microsoft compilers when creating object code files [Microsoft 1999]. Object code files created by the Cygwin GNU compilers also use this same format; however, the Borland compilers use a vendor-specific object code format.

Dynamic Link Library (DLL). This refers to a file containing executable code that can be called from another executable file. The only physical difference between a DLL and a typical ".exe" executable file is a bit flag in the file header stored in the file. Note that DLLs do not need to have a ".dll" extension to be a valid DLL file; other extensions such as ".ocx" and ".cpl" are used on certain Windows DLL files [Pietrek 2002].

Executable file. This refers to a file containing executable code (i.e., a program) that can be run on a computer by invoking the file's name. The program is invoked either

from the command line or from a double click of the mouse, depending on the Windows subsystem that the program expects. Note that executable files do not need to have a ".exe" extension to be a valid executable file; other extensions such as ".sys" and ".drv" are used on certain Windows executable files.

File offset/pointer. This refers to an actual byte address in a file, where the first byte in the file is numbered starting with zero.

Image file. This refers to a file that is either a typical executable file or a dynamic link library file that runs on a Windows NT system. Microsoft uses the name "image file" to refer to both kinds of files.

Loader. This refers to the operating system program that runs behind the scenes when an executable program is invoked by a user or another process. The loader reads through the contents of the PE file, loads specific information from that file into memory, and then begins execution of the program.

Portable Executable (PE) format. This refers to a standardized file format created by Microsoft and used by any image file that runs on a computer system running some version of the Windows NT operating system [Perry 2004]. The COFF is a subset of the PE format and governs the format of any region in an image file that also appears in an object code file such as the file header or the section table [Microsoft 1999]. The PE

format is different from the executable file format used for programs that run on LINUX or UNIX systems running on non-Intel platforms.

Relative virtual address (RVA). This refers to the byte address of image file information after the file is loaded into memory. This address is different than the location of the same information in the file. Consequently, any program reading through an image file or object code file must calculate the RVA delta or difference in order to locate the start of specific tables in a file. This RVA difference is calculated using an algorithm that involves the address data stored in section headers in the section table.

Region. This refers to a range of byte addresses in a file containing information of interest. A file contains many regions depending on the actual header, section, and table contents of an object code file or image file, and on the unmapped or zero-filled data found in a file.

winnt.h header file. This refers to the Microsoft header file that contains the constants and type definitions used by programs written in C or C++ that read files in the portable executable or common object file format.

4.2 The PE Format from a Security Point of View

The typical contents of a PE file are the DOS header, the MS-DOS stub, the PE signature, the file header, the optional header, the section table, the symbol table, the

string table, various sections (e.g., ”.text" and ".data"), the import table, and the export table. The winnt.h header file describes the byte format for each of these items by means of type definitions.

The purpose of the information below is not to give a detailed explanation of the PE format. Such an explanation is given later in this section when describing the PE dissecting software utility. Instead this subsection summarizes each typical item found in a PE file and points out any indicators of anomalies or security vulnerabilities that we find in these items.

4.2.1 The DOS Header

The DOS header is a single record of data that begins at the start of a file. It contains information mainly used by the loader program. It also contains a DOS signature of "MZ" that marks the file as an MS-DOS file. In addition, it contains the offset for the start of the PE signature that appears later in the file. We found no information in the DOS header that was useful for detecting security vulnerabilities.

4.2.2 The MS-DOS Stub

The MS-DOS stub directly follows the DOS header and is a holdover from the days before Microsoft Windows when PC software ran in the MS-DOS operating system. If a PE file is invoked on an MS-DOS system, a short default message will be displayed stating that the program cannot be run in MS-DOS mode. The program will then terminate. We found no information in the MS-DOS stub that was useful for detecting

security vulnerabilities. However, between the end of the MS-DOS stub and the location in the file where the PE signature is located, there should be a contiguous region of zero-filled bytes. The loader ignores any information in this area; consequently, software developers might use this area to store application-specific messages or data. Although we see no direct security risk in this information, we do consider any non-zero data in this region to be useful for detecting an anomaly in the file.

4.2.3 The File Header

The start of the file header immediately follows the PE signature. (The PE signature simply contains the letters "PE".) The file header is a single record that reveals much about the operating environment that the file expects, the contents of the file, and the characteristics of the file. It tells the number of sections in the file, the start location of the symbol table, the number of symbols in the symbol table, and the size of the variable-length optional header. It also implicitly tells the start location of the string table because that table directly follows the symbol table. The file header also contains status information that indicates if the file is an application file or a dynamic link library. This status information is the only way for a loader to tell if a file is a DLL or not; no other data in a PE file denotes this difference.

We found no information in the file header that was useful for detecting security vulnerabilities. However, the file header does reveal the presence or absence of the

symbol table and string table. As we will see later, these two tables are very helpful in detecting buffer overflow security vulnerabilities.

4.2.4 The Optional Header

The start of the optional header immediately follows the end of the file header. The name of the optional header is misleading. It is only optional in the COFF format used in an object code file; it is mandatory in the PE file format. Except for the last field of the optional header, the rest of the header is a fixed record size. The data fields in this record reveal even more detail than the file header does about the operating environment expected by the file and the runtime requirements required by the file. However, we found no information that was useful for detecting security vulnerabilities.

4.2.4.1 The Data Directory in the Optional Header

The last field in the optional header is referred to as the data directory. It contains a variable-length list of the start location and size of each optional table located in the file. For example, the location and size of the import table and export table will be found in this list. The start location is not given as a simple file offset, as was the case earlier for the symbol table and string table. Instead, the start location is a relative virtual address (RVA) that the loader uses when the file contents are placed into memory. However, its use in locating the start of a table statically in a PE file is complicated. This is explained in more detail later when the section table is reviewed.

Although we found no information in the optional header that is useful for detecting security vulnerabilities, we do compare the stated size of each table in the data directory to the actual size that we detect later when reading the contents of the table in the PE file. If there is a difference in these two sizes, we consider this a file anomaly. We also consider it to be an anomaly when the data directory lists an optional table that does not actually appear in the file.

4.2.5 The Section Table

The start of the section table immediately follows the end of the optional header. It contains one or more fixed-sized records containing information on the start location, size, and attributes of each section in the PE file. The number of entries in the section table is denoted by the value in the Number of Sections field in the file header. The section entries are listed in the order that the sections appear in the PE file.

4.2.5.1 Calculation of a File Offset for an Optional Table

Two specific fields in each section entry of the section table provide helpful information when calculating the static start location of an optional table in a PE file. These are the virtual address and the pointer to raw data fields. The data directory in the optional header contains the relative virtual address (RVA) for the start location of each optional table. This value is designed for use when the program is loaded in memory. To find the start location of a table statically in the PE file using the RVA, a file offset for the optional table needs to be calculated from it. This is done using the following steps:

1. Use the optional table's RVA to search through the section table entries to find the virtual addresses of two entries where the RVA falls between. Mark the section entry having the lower of the two virtual addresses
2. Calculate an address difference by subtracting the section entry's virtual address from its pointer to raw data value
3. Calculate the optional table's file offset by subtracting the address difference from the RVA of the optional table

4.2.5.2 Detecting Security Vulnerabilities in the Section Table

A security vulnerability can be detected in the section table by looking at the characteristics for a section. These characteristics tell the loader program if the contents of the section contain executable code or initialized/uninitialized data. They also tell if the section's contents can be executed, read from, or written to. Each of these characteristics has it own bit position in a 64-bit word. To be secure, any section that has the characteristic "contains executable code" or "can be executed" set should not have the characteristic "can be written to" also set.

A security vulnerability can be also be detected in a PE file by looking for a section entry that is located in the section table but does not have a corresponding section in the file, and a section entry in the section table that exceeds the number of sections denoted in the file header. Although both of these occurrences could be considered anomalies, they could also indicate that a virus has tampered with the contents of a file by deleting or adding a section.

4.2.6 The Symbol Table

The location and size of the symbol table are based on two fields in the file header that indicate this information. If the number of symbols is zero, then the file contains no symbol table. The linker uses the symbol table in an object code file but not in a PE file. The table is present in a PE file only for the use by a debugger. The symbol table contains the names of every data item and function declared by the program and any library or runtime environment software linked in with the program. Depending on which linker is used, the symbol table also contains the names of the application source code files for the program.

A search of the function names in the symbol table can be used to detect a buffer overflow security vulnerability in a program. Each function name listed in the symbol table can be compared to the list of standard C library functions susceptible to such a vulnerability. The resulting list of vulnerable functions may reveal function names that do not occur in the application source code at all. If this happens, then the names could be part of the implementation of the language by the compiler designer.

4.2.7 The String Table

The string table is located immediately following the symbol table. If a symbol table is not present in the file, then the string table is located where the symbol table would have started. If the start location of the symbol table field in the file header is zero, then

the file does not contain a string table. The linker needs a string table in an object code file but not in a PE file. The table is present in a PE file only for the use by a debugger. The string table contains the name of every variable name, section name, and constant character string that appears in a program or in any library or runtime environment code linked in with the program. The section table and symbol table only contain entry names up to eight bytes in length. For a name of any greater length, an offset is stored in the data field instead. This offset points to the start of a character string in the string table. Although we found no information in the string table that was directly useful for detecting security vulnerabilities, the contents are helpful indirectly in supplying the names used in the section table and symbol table.

4.2.8 Various Sections

The sections referred to in the section table fill out the remainder of a PE file. Some linkers place each optional table in its own section. For example, the export table may be in the ".edata" section. Other linkers place an optional table in the midst of other data in a section. For example, the import table may be in the ".text" section. The actual code and constant data that we commonly associate with an executable file are stored in one or more sections. These sections are usually referred to as the ".text" and ".data" sections, although they may have other names. Because we do not disassemble any executable code in this research effort, we found no information in the sections themselves that was useful for detecting security vulnerabilities.

4.2.9 The Import Table

The location of the import table in a PE file is found by calculating the file offset of the starting location. This is done using the RVA in the data directory of the optional header and the virtual address and pointer to raw data fields in the section table. This process is explained in more detail in the paragraphs above describing the section table. The import table contains the names of the DLL files and the functions in those files that the program requires in order to run. If a program does not use any DLL files, then the import table is not present.

As was described already in the case of the symbol table, the import table also contains the names of functions used in the file, albeit only functions imported from DLLs. This information can be used to determine a buffer overflow security vulnerability in a file similar to the method used with the symbol table. Note that if a file uses no DLLs and also contains no symbol table, then a search for the names of vulnerable functions will come up with an empty list. This should not be considered an indication that the program uses no vulnerable functions. Instead, it should point out the lack of enough information to make such a determination.

Although the use of a specific DLL by a program does not in itself constitute a security vulnerability, a security analyst may be interested in the DLLs used by a file. By looking at the purpose or function served by a DLL, this may reveal operations performed by the program that aren't readily apparent from the outside. This could include DLLs to assist in network connections, encryption, or installation of files.

4.2.10 The Export Table

The location of the export table in a PE file is found by calculating the file offset of the starting location as was described above for the import table. The export table contains the name of each function in the file that is available for use by another program. This is the normal behavior for a dynamic link library (DLL) file. We found no information in the export table that was useful for detecting security vulnerabilities.

4.2.11 Summary of Anomaly and Security Vulnerability Indicators

Our study of the PE format revealed many ways of detecting anomalies in a PE file. We can detect a file anomaly by looking at the region following the MS-DOS stub to see if data is stored there. We can also detect an anomaly by comparing the sizes given for the optional tables in the optional header to their actual sizes and note any differences.

Our study also found ways to detect certain security vulnerabilities in a PE file. We can detect executable code that can be written to in a section by looking at the characteristics in the section entries of the section table. We can detect hidden sections or sections that are listed but don't actually exist by comparing the number given for the entries in the section table to the actual entries found. We can detect buffer overflow vulnerabilities by comparing the list of function names found in the symbol table and

import table to those in the list of standard C library functions that are vulnerable to such attacks.

4.3 A Software Utility to Dissect a PE File

4.3.1 The Search for a PE Dissecting Program

After finding that we could detect anomalies and software security vulnerabilities in a PE file by statically analyzing it, we needed a means to dissect a PE file into all of its components. We looked at the features and output produced by four dump utilities: objdump from Cygwin [Cygwin 2004], tdump from Borland [Borland 2004], dumpbin from Microsoft [Visual Studio 2004], and pedump [Pietrek 2002b]. All provide PE format information that has first been filtered, summarized and text formatted. All rely on the values given in the header and look-up tables to locate and display section and table contents instead of looking directly at what is actually in the tables. Comparing results provided by each dump utility revealed inconsistencies in the section table and import table information. None provide a byte-for-byte account of the file contents in order to find hidden section contents, hidden table contents, or evidence of compressed files. Many aborted or printed out extraneous data when run against PE files that contain non-typical format information. Consequently, we found the need to first build our own PE dissecting utility and then add anomaly and vulnerability analysis capabilities to it later.

4.3.2 Program Modules and Classes

The PE dissecting utility consisted of 2700 source lines of code written in C++. The code was divided among one driver module, one utility module, and 16 class definitions. It later became version one of the findssv software described later in this section. The purpose of each of the modules and classes in the PE dissecting utility is described in the following paragraphs.

4.3.2.1 Driver Module

The driver module contains the main() function for the program. It calls methods to parse the command line, check the file type, and read the contents of an image file or object code file into a collection of data structures. It also calls methods to change, display, or search through the data in a file.

4.3.2.2 Utilities Module

The Utilities module contains three functions used for converting data formats to C++ strings so the program can store certain information in a string format. The ltostr() function converts a signed long number to a string. The ultostr() function converts an unsigned long number to a string. The toLowerCase function converts a string containing mixed upper and lowercase letters to a string with all lowercase letters.

4.3.2.3 FileTypeChecker Class

The FileTypeChecker class contains methods to quickly verify two types of files based on certain key signature information located in a file. The program uses the methods in this class to verify that it is reading an image file in Microsoft's portable executable

(PE) format or an object code file in Microsoft's common object file format (COFF). It considers all other file formats to be unknown. Valid files in the PE format are not just those with a ".exe" extension. Other valid PE files may have extensions such as ".dll", ".sys", or "drv".

4.3.2.4 File Handler Class

The File Handler class is implemented as a singleton. The class provides methods to open, close, and get the stream descriptor for an executable or object code file. This approach allows methods throughout the classes in the program to quickly and accurately access the file contents without having to repeatedly open and close the file.

4.3.2.5 PECOFF_Partitioner Class

The PECOFF_Partitioner class contains methods that call other class methods to read, store, and later analyze the information obtained from a COFF object code file or PE image file. It also calls class methods to print the data read by those classes. The class aggregates the DosHeader, FileHeader, OptionalHeader, SectionTable, SymbolTable, StringTable, ExportTable, ImportTable, DebugTable, and CoffRelocations classes.

The readData() method in the PECOFF_Partitioner class contains the main algorithm for reading the data from a COFF or PE formatted file. This algorithm is summarized below. (Note that the program reads the string table before reading the symbol table or section table. This allows the program to immediately retrieve any string information pointed to by entries in the symbol table or section table. Also note the action of

"mapping" the file contents. This is explained in more detail when the findssv classes are described later.)

- Count the number of bytes in a file and map the byte at the end of the file
- If the file is a PE file, read and map the DOS header and the PE signature
- If the file is a PE file, read and map the MS-DOS stub
- Read and map the file header
- If the file header indicates the presence of an optional header, read and map the optional header
- If the file header indicates the presence of a symbol table, first read and map the string table and then read and map the symbol table
- Read and map the section table
- Read and map the COFF relocations table if any relocations exist
- Map any of the tables that have entries in the data directory found in the optional header
- Read and map the sections
- Find and map the unknown regions in a file
- Find and map any zero-filled regions in a file that contain a certain minimum number of zeros

4.3.2.6 DosHeader Class

The DosHeader class contains the methods to read the DOS header data from an image file and display it in a report format. (Object code files do not contain a DOS header.) The winnt.h header file describes the format of the DOS header, which consists of a

single 64-byte record. Most of the fields in the record are not significant except for the first field named e_magic and the last field name e_lfanew. To be a valid executable file, the first field should contain the two letters "MZ", which stand for Mark Zbikowski who was one of the original designers of MS-DOS [Pietrek 2002]. The last field contains the absolute offset in the file to the portable executable (PE) signature field. A valid PE signature consists of four bytes containing the values "PE\0\0".

Between the DOS header and the PE signature field is the MS-DOS stub. It starts immediately following the 64 bytes of the DOS header. The MS-DOS stub is a small DOS program that by default displays the words "This program cannot run in DOS mode" when an image file that is targeted for a graphical user interface environment is invoked from the command line [Microsoft 1999].

4.3.2.7 FileHeader Class

The FileHeader class contains the methods to read the file header data from an object code file or image file and display it in a report format. The winnt.h header file describes the format of the file header, which consists of a single 20-byte record. The file header appears at the very start of an object code file or just after the PE signature in an image file. It has a number of significant fields. The NumberOfSections field indicates the number of sections in the section table. The PointerToSymbolTable field contains the absolute offset in the file to the start of the symbol table. This field still contains an offset if the symbol table does not exist but the string table does. This is because the string table follows directly after the symbol table in a file. The

NumberOfSymbols field tells how many symbols are in the symbol table. If this field is zero, then no symbol table exists.

The SizeOfOptionalHeader field tells the size in bytes of the optional header. This value is used to calculate the start of the section table, which immediately follows the optional header. The Characteristics field is a two-byte field, where each bit serves as a flag describing a characteristic of the file. For example, the flags tell if the symbol table has been stripped or not from the file, the type of architecture that the software expects, the big endian or little endian order of the bytes in the file, and if the file is a DLL or not. The DLL flag is the only valid indicator in a file for differentiating a DLL from a typical executable file.

4.3.2.8 OptionalHeader Class

The OptionalHeader class contains the methods to read the optional header data from an image file and display it in a report format. (Object code files normally do not contain an optional header.) The winnt.h header file describes the format of the optional header, which consists of a single 224-byte record. The word "optional" in the name of this header is misleading because an optional header is required in an image file. The optional header contains a number of significant fields. Some of the fields describe address details used when the file is loaded into memory. Other fields contain the operating system and subsystem versions that the software needs to run properly.

One significant field in the optional header is the NumberOfRvaAndSizes field. This field indicates the number of data descriptor records in the data directory that appears at the end of the optional header. The winnt.h header file describes the format of a data descriptor record in this directory. The purpose of this directory is to indicate the presence and location of certain standard tables in a file. These tables include, but are not limited to, the export table, import table, resource table, exception table, certificate table, base relocation table, and debug table.

The data directory entry for each table occurs at a specific index in the directory, ranging from 1 to 16. If an entry is present (i.e., if the data descriptor does not contain all zeros), then the data descriptor record contains the virtual address where the table starts in memory and the size of the table. The entries for the tables that appear most often occur in the lower indices in the directory.

4.3.2.9 SectionTable Class

The SectionTable class contains the methods to read the section table data from an object code file or image file and display it in a report format. The section table appears directly after the optional header in a file. The NumberOfSections field in the file header contains the number of section header entries in the section table. The winnt.h header file describes the format of a section header, which consists of a single 40-byte record.

A section header contains many significant fields. The Name field contains the section name. The VirtualAddress field contains the address where the section starts when the file is loaded into memory. The PointerToRawData field contains the file offset for the start of the section in the file. The program uses the difference between the VirtualAddress field and the PointerToRawData field, in conjunction with virtual addresses from the data directory in the optional header, to calculate the starting byte position of various tables in a file. Through the use of the "-M" map option, a security analyst can see the results from how the program has used this RVA difference calculation to map the location of tables inside of sections. The program stores the section table in a vector.

4.3.2.10 CoffRelocations Class

The CoffRelocations class contains the methods to read the COFF relocation data from an object code file or image file and display it in a report format. The NumberOfRelocations field in a section table entry indicates the number of relocations that exist for that section. The PointerToRelocations field in a section table entry points to the location of this data in an image file. COFF relocations specify how the section data should be modified when placed in the image file and later into memory [Microsoft 1999].

The program stores the COFF relocations data in a vector. Each member of the vector contains three fields: the section name connected with a set of relocations, the symbol name connected with a set of relocations, the actual relocation data. The winnt.h header

file describes the format of a relocations data record. Each record contains the address of the item being relocated, the index of the symbol in the symbol table, and the type of relocation. A type refers to the kind of relocation to perform, such as absolute addressing, virtual addressing, or relative virtual addressing.

4.3.2.11 SymbolTable Class

The SymbolTable class contains the methods to read the symbol table data from an object code file or image file and display it in a report format. The PointerToSymbolTable field in the file header points to the start of the symbol table in a file. The NumberOfSymbols field in the filed header tells how many symbols are in the symbol table. The symbol table contains section names, file names, code symbols, and data symbols. The table consists of an array of symbol records. The winnt.h header file describes the format of the 18-byte symbol record. Each symbol has at least one standard symbol record. There may be one or more auxiliary symbol records following a standard symbol to store additional data about the symbol. A standard symbol record that is zero filled marks the end of the symbol table.

A symbol table appears in object code files, but can be left out of an executable file. This is done by the use of a command line option to the linker that requests the symbol table be stripped out. The program stores the symbol table records in a vector.

4.3.2.12 StringTable Class

The StringTable class contains the methods to read the string table data from an object code file or image file and display it in a report format. The string table appears directly after the symbol table in a file. The first four bytes of the string table contain the total size in bytes of the table. This includes the first four bytes. The rest of the table contains null-terminated character strings that are pointed to by symbols in the symbol table [Microsoft 1999].

A string table appears in object code files, but can be left out of an executable file. This is done by the use of a command line option to the linker that requests the string table be stripped out. The program uses a vector to store the strings read from the string table.

4.3.2.13 ImportTable Class

The ImportTable class contains the methods to read the import table data from an object code file or image file and display it in a report format. The import table is referred to as the ".idata" section in some image files. It lists the functions (called by a program) whose definitions are located in a dynamic link library (DLL) file. The import table actually consists of one or more sets of tables. Each set corresponds to a certain dynamic link library and the functions used from that library.

The program stores the import table information in two vectors. The first vector contains the names of each of the DLLs and its corresponding descriptor record. The

winnt.h header file describes the format of the descriptor record. One significant field in the record is the name of the DLL. Another significant field is the FirstThunk field, which points to the record that heads up the link list of functions used from this DLL. The second vector contains all of the functions located in other DLLs. Each function name is stored with the name of the DLL where it is located.

4.3.2.14 ExportTable Class

The ExportTable class contains the methods to read the export table data from an object code file or image file and display it in a report format. The export table is referred to as the ".edata" section in some image files. This table is normally only found in dynamic link library files. It lists the functions callable by programs that use this DLL.

The export table actually consists of many tables: a directory table, an address table, an ordinal table, a name pointer table, an export name table, and a forwarder name table. All of these tables contribute to the mechanics necessary for software to call a function located in another file. Except for the directory table, which is just a single record, the program stores each of the remaining tables in vectors. The winnt.h header file describes the format of the directory table record. The members of the vectors are 16-bit words, 32-bit words, or strings.

4.3.2.15 DebugTable Class

The DebugTable class contains the methods to read the debug table data from an object code file or image file and display it in a report format. The debug table is referred to as

the ".debug" section in some image files. The format of the debug information is dependent on the specific vendor debug tool that is designed to read it; however, this data commonly consists of line numbers, indices in the symbol table, and indices in the string table.

The program stores the debug table data in a vector. The winnt.h header file describes the format of a debug record, which is the member of the vector. The Type field in this record serves as an indicator for various debug tools.

4.3.2.16 FileChanger Class

The FileChanger class contains methods to change the value of any designated byte in a file. The class provides services for the "-C" change option. The class works on any kind of file and writes data in byte, ASCII string, Unicode string, 16-bit word, and 32-bit word formats. The purpose of these methods is to give a security analyst the ability to easily change a value in a file in order to test the behavior of a file or to correct an anomaly detected in a file.

4.3.2.17 ValueDisplayer Class

The ValueDisplayer class contains methods to display the values in a range of byte addresses in a file. The class provides services for the "-D" display option. The ValueDisplayer works on any kind of file and displays the values in byte, ASCII string, Unicode string, 16-bit word, or 32-bit word formats. The purpose of these methods is to give a security analyst the ability to quickly display the contents of any byte addresses

in a file in a variety of formats. If the file is a PE or COFF file, the analyst can use the "-M" map option to first see the layout of a file, and then use the "-D" display option to see what values are stored in certain interesting file locations such as those marked as "Contents not known."

4.3.2.18 PatternFinder Class

The PatternFinder class contains methods to search for a value in a file. The class provides services for the "-S" search option. It works on any kind of file and looks for data in byte, ASCII string, Unicode string, 16-bit word, and 32-bit word formats. The purpose of these methods is to give a security analyst the ability to quickly search for the existence and location of a value in a file. If the file is a PE or COFF file, the analyst can note the address where a value was found and then use the "-M" option to find the name of the header, section, or table in which the value appears. The analyst can also use the "-D" option to display the range of values surrounding a value. When an ASCII or Unicode value pattern is searched for, the program displays the location of the matching pattern along with the context (i.e., other surrounding characters) in which it was found.

4.4 A Methodology for Finding Software Security Vulnerabilities in a PE File

This methodology describes how to statically analyze a PE file to provide useful information to a security analyst. When analyzing the file, the methodology categorizes its findings as facts, anomalies, or vulnerabilities. The methodology described in the following paragraphs is organized around these three categories.

4.4.1.1 Creating a File Fact Summary

Based on the administrative information stored in the DOS header, file header, and optional header, a file fact summary is generated. These facts include the actual file size in bytes, the target CPU and operating system, an indication that the file is a DLL file or not, an indication that the file has a symbol table or string table, the names of the text files containing the program source code, and a list of optional tables found in the file.

By examining the import table, a fact list is built of the DLLs required by a program. [Whittaker and Thompson 2004] describe how a hacker can exploit the user security level of an application by finding out the DLLs that a program uses, crafting a look-alike DLL with the same name and similar interface, and then placing that DLL in the application's current working directory. Now the program has the ability to use the same privilege level granted to the original DLL.

4.4.1.2 Detecting Anomalies When Reading the File

As stated earlier in this section, anomalies can be detected in the process of reading a PE file. One interesting anomaly is the mismatch between the size of a table as stated in the optional header and the actual size of the table in the file. Such anomalies may indicate an error in the linker or possibly manual tampering with the file contents.

4.4.1.3 Detecting Anomalies When Mapping the File Contents

Another interesting anomaly is the detection of regions in the file that contain either unknown information or possibly compressed files used in a software installation. Mapping the contents of the bytes of a file from the first byte to the last makes this possible. As each header, table, or section is read from the file, its start and stop location are recorded (i.e., mapped). After mapping the locations of every known entity in a PE file, a complete pass is made through the map to find byte regions between entities that are unaccounted for. Such a region is marked as "contents not known" if it falls outside the bounds of any known entity, or it is marked as "no additional details" if it is inside a section containing an optional table but not also inside the bounds of the table itself.

The detection of compressed files is found by comparing an unknown region's size to a maximum acceptable size constant. Based on a survey we made of the average maximum size of known regions in a variety of image files, we found that 200,000 bytes is a good constant. During the survey we also found that such large unknown areas occur in executable installation files where the files to be installed are stored in a compressed format in the executable file. These areas occur most often before or after the last known section in a file. Some also occur as part of the ".rsrc" resource section or a ".winzip" section. The loader ignores any bytes in a file that are not part of one of the listed headers, sections, or tables; consequently, an executable installation file can act as its own repository of hidden data.

4.4.1.4 Detecting Software Security Vulnerabilities

Our research has revealed three kinds of security vulnerabilities that can be detected when statically analyzing an image file (without doing any disassembling): sections in a file whose contents can be both written to and also executed, large unused zero-filled regions in a file, and the use of functions susceptible to buffer overflow attacks. The ways for detecting these vulnerabilities are described in the following paragraphs.

4.4.1.4.1 Detecting Sections That are Both Writable and Executable

Earlier in this document when we described the section table, we pointed out that each entry in the table contains bits in a 64-bit word that indicate the read, write, and execute characteristics of the contents of the corresponding section. A security vulnerability can be detected by finding the simultaneous occurrence of both the write and execute characteristics for a section. Such an occurrence may indicate an error in a linker. It may also indicate tampering by malicious software in an effort to later modify the program's executable code when it is loaded into memory and executed.

4.4.1.4.2 Detecting Large Unused File Regions

When mapping the regions of a file, the zero-filled regions are also tracked and a total is kept of the number of total zero bytes that are encountered. Based on a survey we made of the average size of a contiguous region of zero-filled bytes in a variety of image files, we found that a value less than or equal to 50 is acceptable. Any size above this constant indicates the vulnerability of malicious software employing these unused areas to store hidden code or data.

4.4.1.4.3 Detecting Vulnerable C Library Functions

Earlier in this section we described how examination of the symbol table and the import table could reveal the names of many of the functions used by a program. [Viega and McGraw 2002] provide a list of 31 commonly-recognized C library functions that are vulnerable to buffer overflow attacks on their character string parameters. We create a list of vulnerable functions used by a software program by comparing the function names found in the symbol table and import table to those found in the list of known vulnerable functions. We then store the matching names.

4.4.1.4.4 Understanding the Consequences of no Symbol Table or Import Table

The ability to detect the use of functions vulnerable to buffer overflow attacks obviously depends on the presence of the symbol table and/or the import table in the PE file. If the software developer had the linker strip the symbol table when the executable file was built, then that source of information does not exist in a PE file. Also, if the linker placed the function definitions in the executable file rather than arrange to have them be accessed through a dynamic link library, then the import table will not contain the names of the vulnerable functions. This finding is very important to understand. Without it, a person can get a false sense of security if no vulnerable function calls are found when statically analyzing a PE file using this approach. In other words, an empty list of vulnerable functions can only safely indicate the lack of enough information to detect any function names at all.

4.5 Automation of the Methodology: the findssv Software Utility

From a practical point of view it would be very difficult to manually apply the methodology described above to even the smallest of PE files. This is because of the tens of thousands of bytes in PE administrative information and runtime environment code included in the file. Instead, we incorporated the methodology into the PE dissecting software tool developed earlier to create a greatly enhanced version called "findssv", where "ssv" means software security vulnerabilities. Its purpose is to assist a person in performing a static analysis of executable and dynamic link library files.

Findssv operates similar to an MS-DOS command line utility and is designed to detect anomalies and certain software security vulnerabilities in files that run on Windows NT. It also works with object code files that use the Microsoft Common Object File Format (COFF). Findssv accepts an executable or object code file name (or a wildcard form of the file name) on the command line, followed by zero or more options. It displays information about each of the PE and COFF files in a report that is sent to the standard output device (i.e., the screen). It can be used to automatically check for security vulnerabilities in a single file or in a whole directory of files in just a few seconds.

4.5.1 Program Features

The findssv program has a variety of features to help a security analyst detect what is in a PE file. The following paragraphs describe each of the options available in findssv. Figure 3 contains the help screen that the program displays when only “findssv” is entered on the command line. Note that a file name entered on the command line can be

either a specific file name or a wildcard file name such as *.exe. If such a wildcard file name is entered, it must be enclosed in double quotes so that the MS-DOS command interpreter won't attempt to expand the asterisk.

```
========================================================================
Usage: findssv <filename> [-L] [-T] [-M] [-F] [-AV] [-P[<parts>] ]
       findssv <filename> -C <format> <start> <new>
       findssv <filename> -D <format> <start> <#values>
       findssv <filename> -S <format> <pattern>
       findssv <filename> -S {ascii | unicode} <#chars> <start> <stop>

Explanation:

   filename    : binary file name or wildcard name (Example: "*.exe")
   -AV         : display the anomalies and security vulnerabilities
                 (default)
   -C          : permanently change specific values in a file (Use with
                 care!)
      format   : format of changed value: byte | ascii | unicode |
                 word | dword
      new      : new value to replace current value in the file
      start    : starting byte location of the value to change
   -D          : display a range of file values in various data formats
      start    : starting byte location of the first value to display
      #values  : number of total values to display
      format   : format of values: byte | ascii | unicode |
                 word | dword
   -F          : display a summary of facts about the file (default)
   -L          : filename is a file containing a list of binary files
                 to read
   -M          : display an address map of the file
   -MZ         : map all zero regions, then display address map
   -P          : Display one or more standard parts found in a file
      parts    : one or more parts to display; if none then display all
                 parts
                    d - DOS header           y - symbol table
                    f - file header          t - string table
                    o - optional header      e - export table
                    s - section table        i - import table
                    c - COFF relocations     b - debug table
   -S          : search for a specific pattern of bytes in the file
      pattern  : the pattern to find
      format   : format of pattern:  byte | ascii | unicode |
                 word | dword
      #chars   : minimum number of characters in a string
      start    : starting address of string search
      stop     : stopping address of string search
   -T          : Turn program trace on
```

Figure 3 – Findssv Help Screen

Anomalies and Vulnerabilities. The “-AV” option is used to display a list of anomalies and security vulnerabilities that findssv discovered while reading through and analyzing

the file contents. This list is also displayed by default if only findssv and a file name are entered on the command line. There may be other anomalies or vulnerabilities in a file, but these are the ones that findssv is currently designed to detect.

Change. The "-C" option is used to permanently change a specific value starting at a certain byte position in a file. When making the change, findssv will first display the current value and then the success or failure of replacing it with the new value. The format of the value can be a single byte, an ASCII string, an ASCII string converted internally into Unicode, a 16-bit word, or a 32-bit double word. This option can be used on any type of file, not just executable or object code files.

Display. The "-D" option is used to display the values located at a range of byte addresses in a file. The output of the values can be displayed as bytes, ASCII values, Unicode values, 16-bit words, or 32-bit double words. The number of values is format-dependent in that it refers to the actual number of values using that format rather than the number of bytes in the range of displayed values. This option can be used on any type of file, not just executable or object code files.

Facts. The "-F" option is used to display a list of facts that findssv discovered while reading through and analyzing the file contents. This list is also displayed if only findssv and a file name are entered on the command line.

List. The "-L" option is used to submit a specific list of executable or object code files to findssv. The list is contained in a text file that is submitted to findssv on the command line. This approach to supplying file names to findssv can be used when a wildcard file name will not select the desired files to analyze.

Map. The “-M” option is used to display a diagram that acts similar to an x-ray of a file. The diagram shows the byte location and size of every header, table, and section in an executable or object code file. Each byte location from the start to the end of a file is examined. If a file does not contain enough information for findssv to use when describing a part of a file, the program instead displays “Contents not known” in that location in the file map.

Map Zero. The “-MZ” option performs the same function as the “-M” option. In addition, it checks for and notes zero-filled regions in a file before displaying the map diagram. For a zero-filled region to qualify, it must contain a certain minimum number of bytes. Currently that minimum number is 50.

Parts. The “-P” option is used to display the contents of one or more standard parts located in an executable file. These parts are the DOS header, the file header, the optional header, the section table, the COFF relocations, the symbol table, the string table, the export table, the import table, and the debug table. If only “-P” is entered with no part letter, then the contents of all of the parts are displayed; otherwise, a user may designate which part or parts to display. The order of the part letters makes no

difference in the order in which the data is displayed. Instead, the parts are displayed in the order that they normally appear in a file.

Search. The "-S" option has two separate forms. The first form of this option is used to search for the starting byte location of a specific pattern of information in a file. The format of the pattern may be a byte, an ASCII string, a Unicode string, a 16-bit word, or a 32-bit double word. When the ASCII or Unicode format is specified, findssv will display the starting locations of the pattern and the context of characters in which the pattern was found. The second form of this option is used to search for any ASCII or Unicode character strings in a file. The user specifies the minimum length of a string and the range of byte addresses to search in the file. The "-S" option in either of its forms can be used on any type of file, not just executable or object code files.

Trace. The "-T" option directs findssv to display a number of messages indicating the sequence of activities it performs when reading through and analyzing a file. It is mainly used to display details on problems that may have occurred when reading a specific file.

4.5.2 Program Classes

We built the findssv software features onto the existing version of the PE dissecting utility. The capability to automatically detect anomalies and software security vulnerabilities added 1100 source lines of code and six class definitions to the program. The subsections below describe the six new classes.

4.5.2.1 UserOptions Class

The UserOptions class contains methods to set and get the user options requested by a user on the command line. The user options fall into two categories. The first category contains the display choices of the various parts of a file using the "-P" parts option. The second category contains options to select the "-T" trace output, the "-M" file mapping functionality, the "-MZ" zero-region mapping functionality, the "-F" facts output, the "-AV" anomalies and vulnerabilities output, the "-C" change value functionality, the "-D" display value functionality, the "-S" search functionality, the "-H" hide data functionality, and the "-E" extract data functionality.

4.5.2.2 Analysis Manager Class

The Analysis Manager class contains methods to collect the names of all the files to analyze, submit those names one by one to the PECOFF_Partitioner, and then summarize the results of the analysis after all the files have been read and analyzed. This allows findssv to analyze multiple files one after another in a single run of the program.

4.5.2.3 DiscoveryTracker Class

The DiscoveryTracker class is implemented as a singleton. It serves as a collector and displayer of fact, anomaly, and security vulnerability information. Findssv discovers this information either while reading through a file or when analyzing and correlating the data gathered from a file. As soon as findssv comes upon a discovery, it enters the

category of the discovery and a description of the discovery in the DiscoveryTracker. After the file is completely read, findssv displays all of the fact, anomaly and security vulnerability findings it contains depending on the display desires requested in the user options.

The fact category contains information that describes the size, creation date, target cpu, and operating system for the file. It also contains a list of the dynamic link libraries (DLLs) needed by the file. If the symbol table and string table are present in the file, the fact category will also contain the names of the source code files that were used when building this file.

The anomaly category contains mismatches of totals and pointers that findssv detects while reading through a file. These findings don't point out vulnerabilities, but rather indicate something that is inconsistent in the file such as a section entry but no corresponding section, or a stated size of a table when it is actually a different size. These anomalies may be caused by logical errors in linkers or possibly manual tampering with the file that occurred after the link stage.

The vulnerability category contains specific software security vulnerabilities that findssv looks for after reading all the data from a file. Findssv does no disassembling of object code to find these vulnerabilities. Instead, it analyzes and correlates the data collected from the various headers, sections, and tables in a file.

Findssv stores the discovery tracker data in a vector. Each member of the vector contains four fields: the category of the discovery, the location of the discovery, an occurrence count, and a description of the discovery. Findssv only displays the location of a discovery when the "-T" trace option is used in conjunction with the "-AV" option on the command line.

4.5.2.4 FileMapper Class

The FileMapper class is implemented as a singleton. The class provides services for the "-M" map option and the "-MZ" map zero-filled regions option. It contains methods to collect, analyze, and display the byte-for-byte layout information of an object code file or image file. The map output shows the location and size of every header, section, and table in the file. It also points out the areas of the file whose contents are unknown and the areas of the file containing only a series of zeros. Appendix C contains examples of file maps created by findssv.

Findssv stores the file map information in a map data structure. The key for each entry in the map is the start address of a region (e.g., a header, section, or table) in the file. The corresponding data part of each entry contains the size of the region and a short description of its contents. Findssv detects any attempt to place an entry in the file map that matches a key (i.e., a starting address) already in the map. When such an event occurs, findssv makes adjustments in the map contents depending on the size of the original region and the size of the new region. It also keeps track of each of these occurrences and displays them in a summary after the end of a map display.

The FileMapper class contains complicated algorithms to locate and map unknown data regions in a file, to locate and map the zero-filled regions in a file, and to display the layout of all the regions in a file in a hierarchical format. This hierarchical format is necessary because some regions of a file may be contained within other regions. For example, the ".text" or ".data" section of a file may contain the import table or the export table along with other information.

The algorithm to locate and map unknown regions looks for byte ranges in the file that are unaccounted for in the file map. Findssv performs this action after all of the known regions of a file have been read and mapped. This process involves the tracking of the start and stop addresses of large regions in a file that may contain smaller regions. After findssv detects an unknown region, it enters the region in the file map. If the region is outside of any other region, it describes the region as "Contents not known"; otherwise it describes the region as "No additional details". This is because it occurs inside an already-mapped region.

The algorithm to locate and map zero-filled regions looks for byte ranges in the "Contents not known" regions of a file that contain a minimum number of consecutive zero bytes. Consequently, this algorithm is run after the algorithm to locate and map unknown regions. The minimum number of zeros allowed is defined as a constant in the FileMapper class. When findssv detects a "Contents not known" region exceeding this minimum, it inserts this information into the file map.

The algorithm to display the map layout of a file takes its information directly from the contents of the file map structure. In doing so, it keeps track of any smaller regions that are mapped inside of any larger regions. The resulting map display gives a security analyst an eye-opening view of how the contents of the file are structured. It also reveals areas, such as the regions marked as "Contents not known", that may require manual investigation using the "-D" display option.

4.5.2.5 FunctionCollector Class

The FunctionCollector class contains methods to collect the names of functions that are used by a program. Findssv gathers this information from a combination of data obtained from the symbol table, string table, and import table. As the FunctionCollector gathers names, it looks for functions that are known to be vulnerable to buffer overflow attacks and other hacker actions [McGraw and Viega]. When it finds such functions, it stores them along with one of the following risk levels: low, medium, high, very high, and ultra high.

Findssv stores both the list of all functions found and the list of vulnerable functions found in map data structures. The key for each map is the function name.

4.5.2.6 DLL Librarian Class

The DLL Librarian Class contains a table of DLL file names and descriptions along with methods to look up a description. The table currently contains the descriptions of

over 200 DLLs. Findssv uses this information in the file fact summary when it displays the names of DLLs used by a program.

4.6 Results from Testing the Automated Methodology

4.6.1 The Test Platform and Test Files

The test platform consisted of a Hewlett-Packard 531w personal computer with a 1.3GHz Intel Celeron processor, 512MB RAM, 40GB hard drive, running the Windows XP Home Edition operating system. The test files consisted of 2725 image files (i.e., PE files) already installed on the computer and used frequently for home office use, computer science research, personal entertainment, and software development.

4.6.2 Three Test Objectives

We had three test objectives. Our first was to test that findssv could detect the anomalies and security vulnerabilities that we had identified in the methodology. Our second was to test if findssv could correctly read a varied assortment of PE formatted files. Our third was to test if the automated methodology would produce meaningful and useful test data on the anomalies and security vulnerabilities detected in the test files.

4.6.3 Test Approach

To achieve our first objective, we tested findssv on a set of specific example files. This also allowed us to fine-tune the findssv software and the automated methodology. In addition, it allowed us to see what facts, anomalies, and security vulnerabilities we

could discover about software for which we had a special interest. To achieve our second and third objectives, we identified six categories of PE files: executable installation files, software development files, Windows XP operating system files, Microsoft application files, security-centric application files, and miscellaneous application files. The results of all of these tests are described in the rest of this subsection.

4.6.4 General Contents of the Test Results

The test results from the specific example files contain a fact summary, display of file parts, and file mapping output along with information on the detection of any anomalies and security vulnerabilities. The test results from the six test categories only contain information on the detection of anomalies and security vulnerabilities. Some of the test results for the six categories became extremely lengthy when hundreds of files were analyzed in the category. In such cases, the test results in Appendices C through I only contain the names of the files analyzed and the total number of anomalies and/or vulnerabilities detected in the files.

Paragraphs marked by the words “Key Finding” appear within many of the test result descriptions. These paragraphs highlight noteworthy findings about executable files that we discovered as a result of performing the tests and analyzing the results.

Each subsection of test results contains a results table with seven columns. The columns have the following meanings for the entries in each row:

- A short description of the kinds of files tested
- The total number of files tested
- The total number of files with one or more anomalies
- The total number of files with one to three security vulnerabilities
- The total number of files with four or more security vulnerabilities
- The total number of anomalies in all of the files
- The total number of security vulnerabilities in all of the files

A "*" found in the cell in a vulnerability column signifies that not enough information was available to test for security vulnerabilities in those file(s). This is mainly due to the absence of a symbol table or import table in the files.

4.6.5 Testing Specific Example Files

This subsection covers the results we obtained from testing findssv on specific example files. We selected these files for several reasons. First, the files demonstrate the overall features of findssv. This includes the fact summaries and the display of the contents of each header and table. Second, the files show how the C++ compiling and linking tools from three different vendors (Borland, Cygwin, and Microsoft) generate three different executable files, file maps, and test results for the same source code files. Third, the files are important to the actual software development for this research effort. This is described in more detail below. Table 2 lists a summary of the test results. The detailed test results are in Appendix C.

Description of File(s)	Total no. of Files	Total with Anom. (1+)	Total with Vul. (1-3)	Total with Vul. (4+)	Total no. of Anom.	Total no. of Vul.
Vulnerable – Borland C++	1	1	*	*	3	*
Vulnerable – Cygwin Gnu C++	1	1	0	1	1	24
Vulnerable – Microsoft VS C++	1	1	*	*	3	*
Helloworld – Borland C++	1	1	*	*	3	*
Helloworld – Cygwin Gnu C++	1	1	0	1	1	7
Helloworld – Microsoft VS C++	1	1	*	*	3	*
findssv – Borland C++	1	1	*	*	3	*
findssv – Cygwin Gnu C++	1	1	0	1	1	7
findssv – Microsoft C++	1	1	*	*	3	*
Cygwin Gnu Cygwin1.dll	1	1	0	1	3	12
Microsoft Windows XP kernel32.dll	1	1	1	0	5	2
JGRASP 1.7.5 IDE (.exe files)	3	3	*	*	12	*

Table 2 – Test Summary of Specific Example Files

4.6.5.1 Results from Testing for Functions Susceptible to Buffer Overflow Attacks

We tested findssv on a C++ program named vulnerable.cpp (that we created) to see if findssv would detect the use of functions susceptible to buffer overflow. The program had only a main function that contained calls to each of the vulnerable functions listed in [McGraw Viega and McGraw 2002]. We compiled and linked the file into different executable versions using three separate vendor products.

When findssv analyzed the executable file built by the Cygwin Gnu compiler and linker [Cygwin 2004], it was able to detect the use of the vulnerable functions by examining both the symbol table and the import table. The import table was available because the Gnu linker arranges for the program to use the standard cygwin1.dll dynamic link library rather than include the function code in the executable file. When findssv analyzed the executable file built by the Borland compiler and linker [Borland 2004], it

did not detect the use of any vulnerable functions. The same negative results occurred with the Visual Studio compiler and linker [Visual Studio 2004]. This is because both linkers automatically strip the symbol table from the executable file and include any standard C library code as a part of the executable file. This occurrence is made more evident by comparing the sizes of the three executable files:

- vulnerable.exe (built using the Gnu linker): 7,168 bytes
- vulnerable.exe (built using the Borland linker): 66,560 bytes
- vulnerable.exe (built using the Visual Studio linker): 45,056 bytes

Key Finding: It is possible for an executable file to reveal less information about the functions it uses to hackers by having its symbol table stripped and by having the linker include the language's standard function definitions in the executable file rather than reference functions in a dynamic link library.

4.6.5.2 Results from Testing a "Hello World" Program

We tested findssv on a hello world source code file that consisted simply of a main() function definition containing one C++ "cout" function call to print "Hello World" followed by one statement to return 0 from the function. We compiled and linked this file into different executable versions using three separate vendor products. The results shown in Appendix C were rather surprising. Findssv detected no vulnerabilities in the Borland and Visual Studio versions; however, it detected seven vulnerable functions in the Cygwin Gnu version. This means that the Gnu compiler used those seven

vulnerable functions either in its implementation of the C++ source code for "Hello World" or in its runtime environment included with the hello world program.

Key Finding: A program compiled and linked using the Cygwin Gnu tools will have standard C functions in it that are susceptible to buffer overflow attacks even when these functions are not explicitly used by the software developer.

4.6.5.3 Results from Testing the findssv Program

An obvious candidate to run the findssv software against is itself. When we did this, the test results showed that findssv contained no software security vulnerabilities, at least when compiled and linked using either the Borland or Microsoft Visual Studio tools. This was expected based on the evidence gained after compiling and linking the "vulnerable" program described above. However, the version of findssv generated using the Cygwin Gnu compiler and linker contained seven vulnerable functions: getc(), memcpy(), sprintf(), sscanf(), strcat(), strcpy(), and strncpy(). A text search of the findssv source code files for these function names resulted in no matches. These results were consistent with those for the hello world program.

Key Finding: The Cygwin Gnu C++ compiler and linker injected seven vulnerable function calls into the executable program of the findssv program.

4.6.5.4 Detection of Vendor-Specific Patterns in File Maps

We noticed something interesting in the file maps created by findssv for the vulnerable.cpp, helloworld.cpp, and findssv programs. Each set of compiler and linker tools created a different arrangement of sections and optional tables in the executable file. In other words, the executable code created by a certain vendor's linker had a pattern to it that was not dependent on the purpose of the application.

Key Finding: It may be possible to analyze the general layout of the sections and tables in a file map in order to detect a pattern that indicates the compiling and linking tools used to generate an executable file.

4.6.5.5 Results from Testing the Cygwin1.dll File

The apparent security weakness of the Cygwin C++ compiler to inject vulnerable functions into a software application extends beyond the compiler. Each executable program generated using the Cygwin C++ compiler and linker uses the Cygwin1.dll dynamic link library. When we tested findssv against Cygwin1.dll, it detected 12 sections in the file with attributes set that allow the contents of the 12 sections to be both written to and executed. This is a hacker's dream to find such an error because the sections in an executable program that contain executable code should have the read only attribute set instead. This stops hackers from placing executable code in these sections when the program is loaded in memory. The security concerns of Cygwin even get worse than this. Many of the software products that come bundled with Cygwin

contain software security vulnerabilities. For more details, see the upcoming subsection on the test results from the software development files.

Key Finding: The Cygwin1.dll file contains security vulnerabilities that allow executable code to be modified after the program is loaded into memory and executed.

4.6.5.6 Results from Testing the Kernel32.dll File

Almost every executable program that we tested findssv against uses the Kernel32.dll dynamic link library. It serves as a common DLL to provide a path to other DLLs on a computer system that may not have standard names. Findssv detected that kernel32.dll contains calls to two vulnerable functions: sprintf() and strncpy().

Key Finding: The Kernel32.dll contains functions that are susceptible to buffer overflow attacks.

4.6.5.7 Results from Testing the jGRASP Files

We used jGRASP as the main integrated development environment for the findssv software. Most of jGRASP is written in Java; consequently, we only tested findssv on the executable files that jGRASP uses as wedges to run compiler and linkers. Findssv detected no security vulnerabilities in the jGRASP executable files. However, if the jGRASP executable files were linked using the Borland or the Visual Studio linker, then these results could be misleading because findssv found no symbol table or import table in any of the files.

4.6.6 Testing Executable Installation Files

This subsection covers the results we obtained from testing findssv on executable installation files. The files in this test category all serve the same purpose: to act as a repository and a means for the installation of one or more files on a computer. Table 3 lists a summary of the test results. The detailed test results are in Appendix D.

Description of File(s)	**Total no. of Files**	**Total with Anom. (1+)**	**Total with Vul. (1-3)**	**Total with Vul. (4+)**	**Total no. of Anom.**	**Total no. of Vul.**
Adobe Acrobat Reader 5.0 installation file	1	1	*	*	3	*
Earthlink TotalAccess 5.0 installation file	1	1	*	*	4	*
Java SDK 1.4.2 installation file	1	1	1	0	4	1
Java SDK 1.5 installation file	1	1	1	0	4	1
JGRASP 1.7.5 installation file	1	1	*	*	4	*
JGRASP 1.7.5 (with JRE) installation file	1	1	*	*	4	*
Windows Media Player 9.0 installation file	1	1	*	*	4	*
Real One Player (Windows XP) installation file	1	1	*	*	4	*

Table 3 – Test Summary of Executable Installation Files

4.6.6.1 The Mechanics of Executable Installation Files

Executable installation files call either the Windows InstallShield program or their own internal functions to uncompress files that are stored in the file and then copy these files into specific subdirectories on a hard drive. They also contain function calls to the advapi32.dll dynamic link library to create and update information in the Windows registry.

4.6.6.2 Results from Testing Executable Installation Files

When we tested findssv on the installation files, only the Java SDK files revealed any security vulnerabilities. The Java SDK 1.4.2 installation file contained the use of the sprintf() function that is vulnerable to buffer overflow attack. The Java SDK 1.5 installation file contained 4492 bytes of unused zero-filled bytes that could be used to store malicious code or data.

4.6.6.3 Results from Testing for Compressed Files

Near the middle or end of each executable installation file, findssv detected a large region containing millions of bytes of compressed data (i.e., compressed files). Some of these regions were in a separate section of their own named ".winzip" or in the ".rsrc" section. Other regions were not in a specific section at all, but instead were located at the end of a file after all the sections and tables. This is possible because the Windows loader only loads specific section and table information into memory when a program is run. Any other information in an image file that does not conflict with the PE format is ignored.

Key Finding: An image file can take advantage of the flexibility of the PE format and serve as its own storehouse for millions of bytes of data.

4.6.7 Testing Software Development Files

This subsection covers the results we obtained from testing findssv on software development files. The files in this test category contain the executable programs and dynamic link libraries used by a programmer to create software using various vendor software tools. They consist of integrated development environments, compilers, linkers, debuggers, and other utilities. Table 4 lists a summary of the test results. The detailed test results are in Appendix E.

Description of File(s)	**Total no. of Files**	**Total with Anom. (1+)**	**Total with Vul. (1-3)**	**Total with Vul. (4+)**	**Total no. of Anom.**	**Total no. of Vul.**
Borland C++ Builder 5 executable files	22	22	*	*	79	*
Borland C++ Builder 5 DLL files	20	20	*	*	46	*
Cygwin executable files	325	325	107	177	331	1448
Cygwin DLL files	56	56	19	32	122	234
Sun Java 4.2 executable files	15	15	12	2	46	32
Sun Java 4.2 DLL files	41	41	27	1	181	44
Microsoft VS SDK executable files	29	29	5	2	117	25
Microsoft VS SDK DLL files	3	3	1	0	14	1
Microsoft VS VC7 executable files	15	15	5	3	64	27
Microsoft VS VC7 DLL files	6	6	2	3	34	21

Table 4 – Test Summary of Software Development Files

4.6.7.1 Results from Testing the Borland Files

When we tested findssv on the Borland software development files, it detected many anomalies but no security vulnerabilities. The anomalies dealt with inconsistencies in the actual sizes for the optional tables compared to the size values given in the data directory of the optional header. The lack of any security vulnerabilities is consistent

with our earlier findings when we tested the specific example files. It appears that Borland uses its own compiler and linker to build its software development tools.

4.6.7.2 Results from Testing the Cygwin Files

We tested findssv on the complete suite of software development files that come with the Cygwin download. This includes many directory and file utility programs in addition to software development files. Our tests detected a wealth of anomalies and vulnerabilities. Of the 325 executable files that we tested, findssv found vulnerabilities in 284 of them. Of those, 21 had 10 or more vulnerabilities. Some of these programs were: as, captoinfo, cvs, expect, ftp, gdb, gprof, info, infotocap, insight, ld, less, makinfo, mutt, ncftp, squid, tic, and wget. The gcc and g++ compilers both had six vulnerabilities. Of the 56 DLL files that we tested, findssv found vulnerabilities in 51 of them. Of those, four of the files had nine or more vulnerabilities. This included cygwin1.dll with 12 vulnerabilities.

Findssv detected not only calls to functions susceptible to buffer overflow attacks, but also found the following vulnerabilities:

- A writeable and executable ".idata" (import table) section in cyghistory4.dll, cygreadline4.dll, chgrp.exe, chmod.exe, chown.exe, cp.exe, data.exe, dd.exe, df.exe, dir.exe, dircolors.exe, du.exe, install.exe, ln.exe, ls .exe, mkdir.exe, mkfifo.exe, mknod.exe, mktemp.exe, mv.exe, rm.exe, rmdir.exe, and shred.exe

- A writeable and executable ".text" section in cygform6.dll, cyggdbm, cyggettextsrc-0-12-1.dll, cyggettextsrc-0-12-1.dll, cygmenu6.dll, cygncurses++6.dll, cygpanel6.dll, cygpcreposix-0.dll, Cygwin1.dll, fileman.exe, less.exe, mutt.exe, ncftp.exe, ncftpbookmarks.exe, rcp.exe, rl.exe, rlogin.exe, rlvsersion.exe, rsh.exe, scp.exe, sftp.exe, ssh-add.exe, and ssh-agent.exe
- A writeable and executable ".text" section and source code file names in cygform7.dll, cygmenu7.dll, cygpanel7.dll, clearn.exe, infocmp.exe, infotocap.exe, reset.exe, tack.exe, tic.exe, toe.exe, tput.exe, and tset.exe

In addition, findssv detected six executable files that listed the names of their source code files. These executable files were cygncurses7.dll, mingwm10.dll, awk.exe, captoinfo.exe, gawk.exe, and pgawk.exe

Key Finding: The Cygwin software development files and utility programs contain scores of security vulnerabilities. Therefore, we do not recommend them for secure programming activities.

4.6.7.3 Results from Testing the Sun Microsystems Java Files

We tested findssv on 15 executable files and 41 DLLs that are part of the Sun Microsystems Java development suite. Findssv detected security vulnerabilities in 14 of the executable files. This includes java.exe, the Java interpreter. The vulnerabilities that occurred most often were the use of the fgets() and sprintf() functions, which are susceptible to buffer overflow attacks. Many of the executable files also contained calls

to the fgetc() and sscanf() functions. Findssv detected security vulnerabilities in 28 of the DLL files. These mostly involved the use of one of more of the following functions: sprintf() sscanf(), fgets(), and fgetc().

Key Finding: 42 of the 56 Sun Microsystems Java software development files that we tested (including the Java interpreter) contained one or more functions that are susceptible to buffer overflow attacks.

4.6.7.4 Results from Testing the Microsoft Visual Studio SDK Files

We tested findssv on 29 executable files and three DLLs that are part of the Microsoft Visual Studio SDK. Findssv detected security vulnerabilities in seven of the executable files. The most severe was the ildasm.exe program (the MSIL disassembler), where findssv found the use of seven functions susceptible to buffer overflow attacks: fgets(), memcpy(), sprintf(), strcat(), strcpy(), strncpy(), and vsnprintf(). Findssv detected a security vulnerability in only one of the DLL files. This vulnerability was the use of the sprintf() function, which is also susceptible to a buffer overflow attack.

Key Finding: 8 of the 32 Microsoft Visual Studio SDK files that we tested (including the MSIL disassembler) contained one or more functions that are susceptible to buffer overflow attacks.

4.6.7.5 Results from Testing the Microsoft Visual C/C++ 7.0 Files

We tested findssv on 15 executable files and six DLLs that are part of the Microsoft Visual Studio SDK. Findssv detected security vulnerabilities in eight of the executable files. The two most severe were the cl.exe program and the link.exe program. The cl.exe program is the Visual C/C++ compiler. In this file findssv found the use of eight functions susceptible to buffer overflow attacks: fgets(), getchar() memcpy(), sprintf(), sscanf(), strcat(), strcpy(), and strncpy(). The link.exe program is the Visual C/C++ linker. In this file findssv found the use of seven vulnerable functions: fgets(), getc(), getchar(), read(), sprintf(), sscanf(), and strncpy(). Findssv detected a security vulnerability in five of the DLL files. The most severe was the c1xx.dll file. In this file findssv found the use of seven vulnerable functions: fgets(), read(), snprintf(), sprintf(), sscanf(), strncpy(), and vsnprintf().

Key Finding: 13 of the 21 Microsoft Visual Studio C/C++ 7.0 files that we tested (including the compiler and linker) contained one or more functions that are susceptible to buffer overflow attacks.

4.6.8 Testing Windows XP Home Edition Operating System Files

This subsection covers the results we obtained from testing findssv on Windows XP Home Edition operating system files. We tested all of the images files located in the c:\windows directory, the c:\windows\system directory, and the c:\windows\system32 directory. This included executable files, dynamic link libraries, and driver files.

Table 5 lists a summary of the findings. Appendix F contains the actual test results.

Description of File(s)	**Total no. of Files**	**Total with Anom. (1+)**	**Total with Vul. (1-3)**	**Total with Vul. (4+)**	**Total no. of Anom.**	**Total no. of Vul.**
Windows executable files (in c:\windows)	20	20	4	0	108	5
Windows DLL files (in c:\windows)	2	2	1	0	10	2
Windows System32 executable files (in c:\windows\system32)	279	279	57	9	1135	123
Windows System32 DLL files (in c:\windows\system32)	1304	1304	317	49	6245	743
Windows System32 DRV files (in c:\windows\system32)	5	5	*	*	26	*
Windows System DLL files (in c:\windows\system)	2	2	*	*	7	*
Windows System DRV files (in c:\windows\system)	1	1	*	*	6	*

Table 5 – Test Summary of Windows XP Operating System Files

4.6.8.1 Results from Testing Files in the C:\windows and C:\windows\system Directories

When we tested findssv on the executable and DLL files in the c:\windows directory, it detected very few security vulnerabilities. Of the 20 executable files, only four contained any vulnerabilities and these were all three or less occurrences. When we tested findssv on the files in the c:\windows\system directory, it was not able to identify any vulnerability information because of the missing symbol tables and import tables.

4.6.8.2 Results from Testing Files in the C:\windows\system32 Directory

We tested findssv on 279 executable files in the c:\windows\system32 directory. Findssv detected four or more security vulnerabilities in nine of the files. The most severe of these were the Ntsd.exe file and the MsPMSPSv.exe file. The MsPMSPSv.exe file uses seven functions susceptible to buffer overflow attacks: memcpy(), sprintf(), sscanf(), strcat(), strcpy(), strncpy(), and vcprintf(). The file Ntsd.exe also uses seven functions susceptible to buffer overflow attacks: fgetc(), fgets(), snprintf(), sprintf(), sscanf(), strncpy(), and vsnprintf().

We tested findssv on 1304 DLL files in the c:\windows\system32 directory. Findssv detected four or more security vulnerabilities in 49 of the files. The most severe of these were dbgeng.dll, drmv2clt.dll, LibZkr.dll, python15.dll, vsinit.dll, vsutil.dll, with eight vulnerabilities each, and ipebase12.dll with nine security vulnerabilities. These nine vulnerabilities consisted of nine functions susceptible to buffer overflow attacks: fgets(), fscanf(), getc(), read(), sprintf(), sscanf(), strncpy(), vsnprintf(), and vsprintf().

Findssv found one file named exsec32.dll in the C:\windows\system32 directory that revealed the names of the six source code files that were used to build it. This information was stored in the symbol table of the file.

Key Finding: In the Windows XP Home Edition c:\windows\system32 directory, approximately 25% of the executable files and dynamic link libraries use one or more standard C functions that are susceptible to buffer overflow attacks.

4.6.9 Testing Microsoft Application Files

This subsection covers the results we obtained from testing findssv on a variety of heavily-used Microsoft application files. This includes Microsoft Office software, multimedia software, and network-enabled applications. Table 6 lists a summary of the findings. Appendix G contains the actual test results.

Description of File(s)	**Total no. of Files**	**Total with Anom. (1+)**	**Total with Vul. (1-3)**	**Total with Vul. (4+)**	**Total no. of Anom.**	**Total no. of Vul.**
Microsoft Office 2000 executable files	14	14	5	1	59	10
Microsoft Office 2000 DLL files	48	48	17	3	253	42
Microsoft Outlook Express executable files	5	5	*	*	20	*
Microsoft Outlook Express DLL files	6	6	*	*	29	*
Windows Internet Explorer executable files	1	1	*	*	4	*
Windows Internet Explorer DLL plugin files	7	7	1	1	35	8
Windows Media Player 9 executable files	5	5	2	1	21	9
Windows Media Player 9 DLL files	8	8	2	0	45	4
Windows Messenger executable files	2	2	*	*	9	*
Windows Messenger DLL files	3	3	*	*	14	*
Windows MovieMaker executable files	1	1	0	1	5	5
Windows MovieMaker DLL files	3	3	2	0	15	3
Windows NetMeeting executable files	3	3	*	*	12	*
Windows NetMeeting DLL files	15	15	1	0	77	1

Table 6 – Test Summary of Microsoft Application Files

4.6.9.1 Results from Testing Microsoft Office Files

We tested findssv on 14 executable files and 48 dynamic link library files that are part of Microsoft Office. Findssv found no standard C functions susceptible to buffer

overflow attacks in excel.exe (Excel), outlook.exe (Outlook), powerpnt.exe (PowerPoint), winproj.exe (Project), or winword.exe (Word). It did detect four such vulnerabilities in the wavtoasf.exe file. These consisted of function calls to fgets(), sprintf(), sscanf(), and strncpy().

4.6.9.2 Results from Testing Network-Enabled Files

When we tested findssv on the executable files for Outlook Express, Internet Explorer, Messenger, and NetMeeting, it was unable to detect any security vulnerabilities. This was partially because the symbol tables and import tables are not present. However, findssv also searched for code sections with writable characteristics and large unused file regions in these files. Neither of these vulnerabilities was found. However, findssv did find seven security vulnerabilities in the NPDocBox.dll file, which is a DLL for Internet Explorer. This file uses calls to seven functions susceptible to buffer overflow attacks: memcpy(), snprintf(), sprintf(), sscanf(), strcat(), strcpy(), and strncpy().

4.6.9.3 Results from Testing Windows Media Player and MovieMaker Files

We tested findssv on five executable files and eight dynamic link library files that are part of Windows Media Player. Findssv found no security vulnerabilities in the wmplayer.exe file, but it did find four vulnerabilities in the migrate.exe file. That file contains calls to four vulnerable functions: memcpy(), strcat(), strcpy(), and strncpy(). Findssv found three security vulnerabilities in the npdrmv2.dll file. That file contains calls to three vulnerable functions: memcpy(), strcat(), and strcpy().

When we tested findssv on the moviemk.exe file, which is the major file for MovieMaker, it detected five security vulnerabilities. These consisted of calls to the following functions vulnerable to buffer overflow attacks: memcpy(), sprintf(), sscanf(), strcpy(), and strncpy().

4.6.10 Testing Security-Centric Application Files

This subsection covers the results we obtained from testing findssv on security-centric application files. The files in this test category consist of programs whose main purpose is to improve the security of the computing environment of a personal computer.

Description of File(s)	**Total no. of Files**	**Total with Anom. (1+)**	**Total with Vul. (1-3)**	**Total with Vul. (4+)**	**Total no. of Anom.**	**Total no. of Vul.**
Network Associates Common Framework executable files	7	7	4	1	28	19
Network Associates Common Framework DLL files	35	35	24	5	175	70
Network Associates VirusScan 7.0 executable files	11	11	1	0	45	1
Network Associates VirusScan 7.0 DLL files	16	16	*	*	81	*
Secure CRT 4.0 executable files	5	5	1	2	23	12
Secure CRT 4.0 DLL files	5	5	3	1	30	10
SpyBot 1.2 executable files	4	4	*	*	16	*
SpyBot 1.2 DLL files	7	7	*	*	12	*
WinSCP executable file	1	1	1	0	5	2
Zero Knowledge Freedom 3.0 executable files	5	5	1	0	19	1
Zero Knowledge Freedom 3.0 DLL files	34	34	23	1	173	42
Zone Alarm Pro 4 executable files	6	6	1	1	23	6
Zone Alarm Pro 4 DLL files	3	3	1	2	15	11

Table 7 – Test Summary of Security-Centric Application Files

Before testing this category, we had thought that these files would be our shining stars for secure programming and contain no security vulnerabilities; however, the results showed otherwise. Table 7 lists a summary of the findings. Appendix H contains the actual test results.

4.6.10.1 Results from Testing Network Associates' Virus Scanning Files

We tested findssv on both the Common Framework files and the VirusScan files that are part of the Network Associates software installation. The VirusScan files faired well, but one of the Common Framework executable files and five of the Common Framework dynamic link library files contained four or more security vulnerabilities. The executable file is McScript.exe. It contains the use of ten functions that are susceptible to buffer overflow: getc(), memcpy(), read(), snprintf(), sprintf(), sscanf(), strcat(), strcpy(), strncpy(), and vsnprintf().The DLL files are InternetManager,dll, ListenServer.dll, naCmnLib.dll, naisign.dll, and Scheduler.dll. All contain calls to vulnerable functions.

Key Finding: In the Network Associates Common Framework software installation that accompanies the VirusScan software installation, approximately 75% of the executable files and dynamic link libraries use one or more standard C functions that are susceptible to buffer overflow attacks.

4.6.10.2 Results from Testing SecureCRT Files

We tested findssv on five executable files and five dynamic link library files that are a part of SecureCRT. It found four or more security vulnerabilities in two of the executable files and one of the dynamic link library files. The SecureCRT.exe file contains calls to three vulnerable functions. The Vcp.exe file contains calls to four vulnerable functions. The Vsh.exe file contains calls to five vulnerable functions: gets(), sprintf(), sscanf(), strncpy(), and vsprintf(). One of these, the gets() function, is considerable an ultra high risk function for buffer overflow attacks, and its use is never recommended. The Mfc42.dll file contains 3409 unused zero-filled bytes that could be used to store malicious code or data. It also contains the use of four functions susceptible to buffer overflow attacks: fgets(), memcpy(), sprintf(), and vsprintf().

Key Finding: The SecureCRT 4.0 software contains executable files and DLL files that are highly vulnerable to buffer overflow attacks.

4.6.10.3 Results from Testing Zero Knowledge Freedom Files

Zero Knowledge Freedom is designed to be a firewall software product. When we tested findssv on the DLL files for Zero Knowledge Freedom, it found security vulnerabilities in 24 of the 34 files. The most severe was the NetworkR.dll file that contained the use of four vulnerable functions: memcpy(), snprintf(), strcat(), and strcpy().

Key Finding: In the Zero Knowledge Freedom software, approximately 70% of the dynamic link library files contain standard C functions that are susceptible to buffer overflow attacks.

4.6.10.4 Results from Testing Zone Alarm Pro Files

Zone Alarm Pro is also designed to be a firewall software product. When we tested findssv on Zone Alarm Pro files it faired better than Zero Knowledge Freedom; one executable file and two dynamic link libraries contained four or more security vulnerabilities. The executable file is zapro.exe, which is the main program for Zone Alarm Pro. It contains calls to five vulnerable functions: memcpy(), sprintf(), strcat(), strcpy(), and strncpy(). The two dynamic link library files are expert.dll and framewrk.dll. Both contain five calls to vulnerable functions.

Key Finding: In the Zone Alarm Pro software, all three of the dynamic link library files contain standard C functions that are susceptible to buffer overflow attacks.

4.6.11 Testing Miscellaneous Application Files

This subsection covers the results we obtained from testing findssv on a variety of miscellaneous application files that did not fit in any of the previous test categories. Table 8 lists a summary of the findings. Appendix I contains the actual test results.

Description of File(s)	Total no. of Files	Total with Anom. (1+)	Total with Vul. (1-3)	Total with Vul. (4+)	Total no. of Anom.	Total no. of Vul.
Adobe Acrobat Reader 5.0 executable files	1	1	*	*	5	*
Adobe Acrobat Reader 5.0 DLL files	7	7	1	1	31	11
Earthlink TotalAccess 5.0 executable files	13	13	8	1	56	27
Earthlink TotalAccess 5.0 DLL files	54	54	31	11	273	114
HP PC CoreTech executable files	1	1	1	0	4	2
HP PC CoreTech DLL files	7	7	2	0	34	4
Iomega ZIP Disk executable files	4	4	*	*	15	*
MusicMatch Jukebox executable files	11	11	2	3	43	17
MusicMatch Jukebox DLL files	44	44	18	11	219	78
OpenOffice 1.1 executable files	8	8	4	0	28	8
OpenOffice 1.1 DLL files	193	193	38	12	1002	123
Real One Player executable files	4	4	1	0	14	2
Real One Player DLL files	15	15	12	2	73	32
Veritas Update Manager image files	9	9	5	0	43	9
WinZIP 8.0 executable files	2	2	*	*	8	*
WinZIP 8.0 DLL files	7	7	*	*	32	*

Table 8 – Test Summary of Miscellaneous Application Files

4.6.11.1 Observing the State of Secure Programming Practices in Commercial Software

These test results gives us an indication of the state of secure programming practices in commercial software. As has been observed already in the previous tables, what really stands out in the test results are those programs in which findssv detected a large number of vulnerabilities and those in which it detects only a few. As for the programs in which findssv found no vulnerabilities, we must be cautious about give them a high rating. Recall that findssv can only detect the presence of security vulnerabilities, not confirm they don't exist. Nevertheless, when findssv does detect vulnerabilities, it is worth noticing as we point out concerning the DLL files below.

In Earthlink TotalAccess, findssv detected security vulnerabilities in 42 of the 54 DLL files. In MusicMatch Jukebox, findssv detected security vulnerabilities in 29 of the 44 DLL files. Of the 29 files, 11 have four or more vulnerabilities. In OpenOffice, findssv detected security vulnerabilities in only 50 of the 193 DLL files. But of those 50 files, 12 have four or more vulnerabilities. In Real One Player, findssv detected security vulnerabilities in 14 of the 15 DLL files.

This indicates a major lack of secure programming practices by the programmers who developed the dynamic link libraries for these application programs. This is in sharp contrast to the very low number of security vulnerabilities detected by findssv in the DLL files of the Windows application files. However, this high number of vulnerabilities corresponds closely to the large number of vulnerabilities found in the executable files and the DLL files in the c:\windows\system32 directory.

5. CONCLUSION

Our main goal throughout this research effort has been to devise and test techniques to automatically detect software security vulnerabilities in executable program files through static code analysis. Published open source auditing techniques only describe automated static code analysis of source code files. Because of the results of this research effort, a second approach directed at executable files now exists for the security analyst, the software developer and the computer user. This new approach takes a time-consuming manual process that required weeks to complete and replaces it with an automated methodology that not only finishes in just seconds, but until this research effort occurred, was not available in open source.

5.1 Proof of the Dissertation Hypothesis

5.1.1 The Hypothesis

This research effort involved the automatic scanning of executable files. Specifically, it involved the scanning of files that conform to the portable executable (PE) format designed for software running on Windows NT/XP computers. This effort set out to prove the following hypothesis:

A methodology can be devised that uses information located in the headers, sections, and tables of an executable file, along with information derived from the overall

contents of the file, as a means to detect specific software security vulnerabilities without having to disassemble the code. Such a methodology can be instantiated in a software utility program that automatically detects certain software security vulnerabilities before installing and running the executable file.

5.1.2 Achievement of Research Objectives

We proved our hypothesis by achieving four research objectives: identification of PE file information useful in a security vulnerability analysis, formulation of a methodology for conducting the analysis, automation of the methodology, and testing of the automated methodology.

5.1.2.1 Identification of PE File Information

We identified specific information in the PE file format that was useful in a security vulnerability analysis. The characteristics for each section entry in the section table can reveal executable code that can be written to during program execution. The mismatches in the number of expected entries and the number of actual entries in the section table can reveal hidden sections or sections that are listed but don't actually exist. Occurrences of C standard function names in the symbol table and import table can reveal the use of functions that are susceptible to buffer overflow attacks. A byte-for-byte mapping of the complete file contents can reveal areas of unused zero-filled space that could be used to store malicious code or data.

5.1.2.2 Formulation of a Methodology

Using the information discussed in the paragraph above, we formulated a methodology for conducting the analysis and identifying certain security vulnerabilities. These are the steps of the methodology:

- Create a file fact summary to understand the general layout of the file
- Detect anomalies when reading in the parts of a PE file
- Detect anomalies when mapping the complete file contents
- Detect sections that are both writable and executable
- Detect non-existent or spurious sections
- Detect large unused regions in the file
- Detect the use of C library functions that are susceptible to buffer overflow attack
- Report the anomalies and vulnerabilities that were found

5.1.2.3 Automation of the Methodology

We incorporated this methodology into a software application called findssv that dissects a PE file and analyzes its contents for anomalies and security vulnerabilities. The findssv software consists of 3800 source lines of C++ code. It has a driver module, a utility module, and 22 classes. The program runs in MS-DOS text mode and accepts a variety of options on the command line to assist the security analyst in checking either one executable file or a complete directory of files. The program produces results after only a few seconds of operation for each file.

5.1.2.4 Testing of the Automated Methodology

We ran the findssv software on seven categories of executable files totaling 2700 files in all. Findssv was able to automatically detect the kinds of anomalies and vulnerabilities that we had identified when examining the PE file format. Findssv also correctly read a varied assortment of PE formatted files. In addition, findssv produced meaningful and useful test data on the anomalies and security vulnerabilities detected in the test files.

5.1.3 Key Findings Extracted From the Test Results

We extracted the following sixteen key findings from analyzing the test results produced by findssv:

- It is possible for an executable file to reveal less information about the functions it uses to hackers by having its symbol table stripped and by having the linker include the language's standard function definitions in the executable file rather than reference the functions in a dynamic link library
- A program compiled and linked using the Cygwin Gnu tools will have standard C functions in it that are susceptible to buffer overflow attacks even when these functions are not explicitly used by the software developer
- The Cygwin Gnu C++ compiler and linker injected seven vulnerable function calls into the executable program of the findssv program
- It may be possible to analyze the general layout of the sections and tables in a file map of an executable file in order to detect a pattern that indicates the compiling and linking tools used to generate the file

- The Cygwin1.dll file contains security vulnerabilities that allow executable code to be modified after the program is loaded into memory and executed
- The Kernel32.dll file contains functions that are susceptible to buffer overflow attacks
- An executable or dynamic link library file can take advantage of the flexibility of the PE format and serve as its own storehouse for millions of bytes of data
- The Cygwin software development files and utility programs contain scores of security vulnerabilities. Therefore, we do not recommend them for secure programming activities
- 42 of the 56 Sun Microsystems Java software development files that we tested (including the Java interpreter) contained one or more functions that are susceptible to buffer overflow attacks
- 8 of the 32 Microsoft Visual Studio SDK files that we tested (including the MSIL disassembler) contained one or more functions that are susceptible to buffer overflow attacks
- 13 of the 21 Microsoft Visual Studio C/C++ 7.0 files that we tested (including the compiler and linker) contained one or more functions that are susceptible to buffer overflow attacks
- In the "c:\windows\system32" directory of the Windows XP Home Edition, approximately 25% of the executable files and dynamic link libraries use one or more standard C functions that are susceptible to buffer overflow attacks

- In the Network Associates Common Framework software installation that accompanies the VirusScan software installation, approximately 75% of the executable files and dynamic link libraries use one or more standard C functions that are susceptible to buffer overflow attacks
- The SecureCRT 4.0 software contains executable files and DLL files that are highly vulnerable to buffer overflow attacks
- In the Zero Knowledge Freedom software, approximately 70% of the dynamic link library files contain standard C functions that are susceptible to buffer overflow attacks
- In the Zone Alarm Pro 4.0 software, all three of the dynamic link library files contain standard C functions that are susceptible to buffer overflow attacks

These key findings show that it is possible and advisable to analyze executable files in an effort to detect security vulnerabilities. They also confirm that findssv can detect a certain subset of software security vulnerabilities by directly interrogating executable files in a static manner.

5.1.4 The Immediate Practical Uses of Findssv

Given a set of executable files, findssv provides a security analyst with the ability to quickly pare down those files to the ones in which secure programming was not an objective of the developers. This was made most evident in our test results of the Microsoft Office files compared to those of the other standard Microsoft application files and dynamic link libraries. It was clear that the Microsoft security experts had

gone through the Microsoft Office software with a fine-tooth comb looking for and removing security vulnerabilities. On the other hand, the other Microsoft applications were virtually ignored by their security specialists. Findssv can produce these same useful and focused results upon analyzing applications from any software development project. In addition, it can eliminate the occurrence of false positives because its approach to an analysis is purely a fact-finding approach. Consequently, findssv will answer the question, "Was secure programming a primary goal of the software development team?"

With findssv, a security analyst can do in seconds what could take days or weeks of semi-automated analysis using hex editors and file dump utilities. This is possible because it knows where to look in a PE file and what to look for. In addition, it knows when to stop looking for certain vulnerabilities when the indicators of those vulnerabilities do not exist in the file. An example is the absence of a symbol table, a string table, and an import table. When these three tables are missing from a file, findssv skips its search for buffer overflow vulnerabilities. This does not mean that such vulnerabilities do not exist in the file. Instead, it means that disassembling the executable sections of the code or searching those sections for unique function call signatures is the only other means of finding these vulnerabilities if they exist.

5.2 Performance of Findssv in a Real-World Security Vulnerability Analysis

5.2.1 Results Obtained by the Information Assurance Laboratory at Auburn University

Last year, the security research group in the Information Assurance Laboratory at Auburn University performed a security vulnerability assessment on simulation software used by the Department of Defense. Their goal was to determine if the executable files contained any classified information. In addition, they attempted to find buffer overflow vulnerabilities in the software. Using the automated file utilities currently available in Cygwin, Windows, and the Internet, the research group looked for indications of classified information in eighteen executable files (141 million total bytes) by running the software, disassembling and decompiling the software, and statically analyzing the executable files.

When running the software they discovered one unhandled exception from the use of a tab key. When using freeware decompilers, the utilities mostly produced unreadable or invalid code. When disassembling the software, the utilities produced 9.3 million lines of raw assembly code, which they attempted to analyze manually. When statically analyzing the executable files, they searched for the occurrence of character strings containing classified information.

5.2.2 Results Obtained by Using Findssv

At the close of this research effort we were given the opportunity to run findssv on these same simulation software files to look for security vulnerabilities. It took findssv less

than a minute to analyze all eighteen files and produce results. Table 9 contains a summary of the results.

File Nbr	File Size (bytes)	Total Vul.	Large Unknown Region (bytes)	Unused Zero-filled Bytes	Import Table Anomaly	Symbol and String Tables	Debug Table
1	6,622,124	12	4,381,612	100,726	80/1620	no	yes
2	4,961,816	12	3,356,184	56,560	80/1620	no	yes
3	34,304	0	0	0	40/927	no	no
4	4,841,964	13	3,269,100	76,212	80/1694	no	yes
5	34,816	0	0	0	40/927	no	no
6	23,255,612	14	16,046,652	314,600	180/4663	no	yes
7	23,043,168	12	15,219,658	364,746	100/2495	yes	yes
8	26,864,140	14	19,544,588	413,280	160/8341	no	yes
9	27,627,392	14	19,791,744	443,924	160/5339	no	yes
10	6,041,004	12	4,124,076	72,142	80/1556	no	yes
11	942,138	0	0	0	60/2533	no	yes
12	31,232	0	0	0	40/898	no	no
13	4,207,940	13	2,856,260	68,720	80/1570	no	yes
14	7,696,928	12	5,083,680	98,792	80/1623	no	yes
15	33,280	0	0	0	40/908	no	no
16	16,384	1	0	0	80/432	no	no
17	4,385,052	13	2,951,452	64,918	80/1594	no	yes
18	374,436	10	280,228	5022	40/1026	no	yes

Table 9 – Test Summary of Simulation Software Files

The first column in the table is a unique number identifier for each of the files. Because of security reasons, the actual file names were removed in this document. The second column is the size of the file in bytes. The sizes range from 16KB to 27MB.

The third column contains the total number of security vulnerabilities that findssv detected in each file. These vulnerabilities consisted of regions of unused zero-filled bytes and the use of functions susceptible to buffer overflow attacks. The fgetc(), fgets(), fscanf(), scanf(), sprintf(), sscanf(), strcat(), strcpy(), and strncpy() functions

were used in twelve of the files. The getchar() and memcpy() functions were used in eleven of the files. The read() function was used in four of the files. The getc() function was used in three of the files. The function with the highest risk of buffer overflow attack, gets(), was used in three of the files.

The fourth column contains the size in bytes of large unknown regions found in many of the files. These regions all appeared at the end of the files. The regions were not a part of any known section or table. This is very peculiar to have such large unknown regions in files that are not intended to install software. The fifth column contains the total number of bytes in unused zero-filled regions found throughout a file. We consider this a security vulnerability because these regions could be used to store malicious code or data.

The sixth column contains values related to an anomaly found with the import table in each of the files. The first value refers to the size in bytes for the import table as stated in the data directory in the optional header. The second value refers to the actual size of the table as found by findssv upon reading the import table contents. This anomaly could be an error in the linker used to build each one of the files. If an object code dump utility or program loader trusts the size value in the data directory, the information read from the import table will be incomplete.

The seventh column tells if the file contained a symbol table and a string table. Only File #7 contained the two tables. In all of the other files, the linker used by the

simulation development team must have removed the string tables and symbol tables. Either the programmers requested this using a command line option or it is a default action by the linker. The eighth column tells if the file contained a debug table. Thirteen of the files contained debug tables.

Findssv also detected eleven files that used functions from the dynamic link library intended for Windows socket programming. Such functions permit the transmission and receipt of information between programs over a network connection.

In summary, these results provide one example of how findssv detected anomalies and security vulnerabilities in real-world executable files consisting of tens of millions of bytes in less than a minute of operation. One member of the Auburn security group who worked on the analysis of the simulation software gave the following comment after experimenting later with findssv:

"The findssv software would have been very helpful during the analysis of the simulation software…The findssv software would have been useful in order to help determine critical portions of the simulation software and what values were contained in the program. This is especially important when the source code of the simulation model is not available and the executable itself must be examined for specific values [Chatam 2005]."

5.3 Future Work

The creation of the findssv software and the automation of our security vulnerability detection methodology is our first success at performing static analysis of executable files. It showed that a useful analysis could be done; however, we believe there is more security-related information that can be gleaned from executable files. Further research in the analysis of executable files is needed to identify more key indicators of software security vulnerabilities that can be detected by automated means through static file analysis. In the paragraphs below, we summarize some of our future research initiatives in this area.

5.3.1 Determining the Compiler and Linker used to build an Executable File

In Section 4 of this document we noted that it may be possible to analyze the general layout of the sections and tables in a file map of an executable file in order to detect a pattern that indicates the compiling and linking tools used to generate the file. This observation needs to be turned into a feature of findssv. First, we could study the layout patterns in the file maps of executable files generated by known compiler and linker utilities. Findssv is currently the only software we know of that generates a map of an executable file. Consequently, we would use findssv to discover information in order to expand the features of findssv. (This may seem unusual, but we used the early version of findssv when it was just a PE dissecting tool to figure out where sections and tables were stored in a PE file.) Second , we could draw relationships between the compiler and linker utility vendors and the layout patterns. Third, we could formulate a way to

incorporate this logic into findssv so it could report that an executable file was built using a certain compiler and linker.

5.3.2 Relationship of DLL Function Use to Program Purpose

Findssv only uses the import table as a place to check for the names of functions that are susceptible to buffer overflow attacks. These function names and their corresponding DLL files could also be used to determine the general activities performed by a program. The DLLs and functions used by known programs could be analyzed and relationships could be drawn among the purpose of the program and the DLLs and functions it uses. These relationships could then be incorporated into findssv and searched for in unknown executable files to ascertain their purpose.

5.3.3 More Details on Unknown Regions

We know how to use the start location, size, and relative virtual addresses in the PE file to build a file map. We also know how to scan that map to detect large unknown regions. The next step is to study the bytes in those regions to find out what they contain. Is it executable code or is it data? If it is code, does the program use it? If it is data, is it in some special format and does it belong to a table that is not a standard component of the PE format? In other words, do some linkers create their own unique tables of administrative information and place them at the end of executable files? All of these questions need to be answered.

5.3.4 Individual Names of Files Stored in Compressed File Regions

Findssv is able to detect regions in an executable file that most likely contain compressed files used by an installation program. We need to learn about the compression formats used in those regions so that findssv can uncompress enough of the data in order to report the list of individual file names and the proposed storage locations for those files.

5.3.5 Detecting the Use of Standard Functions By Way of Function Call Signatures

The only way that findssv can detect the use of standard C functions that are susceptible to buffer overflow attacks is by checking the function names in the symbol table and the import table. If neither of these tables is present in an executable file, findssv cannot perform the check for buffer overflow vulnerabilities.

An approach is needed that matches a series of particular byte values to a standard function call in binary code. This series of bytes could then be used as a signature to detect the use of certain vulnerable functions. Findssv could search for these signatures in the executable code sections of a PE file and report the use of the corresponding functions. This feature would allow us to remove the asterisks in the test result tables in Section 4 of this document and replace them with actual numbers. We could then analyze the executable files built with Borland and Microsoft linkers to see if those files contain any buffer overflow vulnerabilities.

Although this approach sounds promising, it has a major dependency. It can only detect the use of standard C functions for which we have identified a unique call signature. If a software application uses its own I/O function calls rather than those from the standard C library, then this approach will reap little benefit.

6. REFERENCES

[@stake 2004a]. @stake. SmartRisk Analyzer Press Release. *@stake Inc*. www.atstake.com. Accessed on August 28, 2004.

[@stake 2004b]. @stake. Technical Specifications for SmartRisk Analyzer. *@stake Inc*. www.atstake.com. Accessed on August 28, 2004.

[@stake 2004c]. @stake. SmartRisk Analyzer Product Datasheet. *@stake Inc*. www.atstake.com. Accessed on August 28, 2004.

[@stake 2004d]. @stake. SmartRisk Analyzer Product Whitepaper. *@stake Inc*. www.atstake.com. Accessed on August 28, 2004.

[Acar and Michener 2002] Acar, T. and Michener, J. Risk in Features vs. Assurance. *Communications of the ACM*, (Aug 2002), 112.

[Ahmad 2003] Ahmad, D. The Rising Threat of Vulnerabilities Due to Integer Errors. *IEEE Security and Privacy*, (Jul./Aug. 2003), 77-82.

[Anderson 2001] Anderson, R. *Security Engineering*. Wiley Computer Publishing, New York, NY, 2001.

[Andress 2002] Andress, M. *Surviving Security*. Sams Publishing, Indianapolis, IN, 2002.

[Arbaugh, Fithen and McHugh 2000] Arbaugh, B.; Fithen, B., and McHugh J. Windows of Vulnerability: A Case Study Analysis. *IEEE Computer*, 33 (10), Dec 2000.

[Arce 2004] Arce, I. The Kernel Kraze. *IEEE Security and Privacy*, (May/Jun. 2004).

[Arce and McGraw 2004] Arce, I. and McGraw, G. Why Attacking Systems Is a Good Idea. *IEEE Security and Privacy*, (Jul./Aug. 2004).

[Arora and Telang 2005] Arora, A. and Telang, R.. Economics of software vulnerability disclosure. *IEEE Security and Privacy*, (Jan./Feb. 2005).

[Barrantes et al. 2003] Barrantes, E.; Palmer, T.; Ackley, D.; Stefanovic, D.; Forrest, S. and Dai Zovi, D. Randomized instruction set emulation to disrupt binary code injection attacks. In the *Proceedings of the 10th ACM Conference on Computer and Communication Security* (Washington, DC, 2003). ACM, New York, NY, 2003.

[Bhatkar, DuVarney, and Sekar 2003] Bhatkar, S.; DuVarney, D. and Sekar, R. Address obfuscation: an efficient approach to combat a broad range of memory error exploits. In the *Proceedings of the 12th USENIX Security Symposium* (Washington, DC, August 2003). USENIX, Berkeley, CA, 2003.

[Bishop 2005] Bishop, M. *Introduction to Computer Security*. Addison Wesley, Boston, MA, 2005.

[Borland 2004] Borland. *C++ Builder*. Borland Website. www.borland.com. Accessed on Dec. 10, 2004.

[Castro 2001] Castro, E. *PERL and CGI for the World Wide Web*. Peachpit Press, Berkeley, CA, 2001.

[CERT 2002] CERT Coordination Center. Module 8: Threats, Vulnerabilities, and Attacks. *Information Security for Technical Staff*. Carnegie Mellon University. http://www.andrew.cmu.edu/. Accessed on August 30, 2004.

[Chatam 2005] Chatam W. E-mail message received at my Auburn University account on January 14, 2005.

[Chen and Wagner 2002] Chen, H. and Wagner, D. MOPS: An Infrastructure for Examining Security Properties of Software. In the *Proceedings of the 2002 Conference on Computer Communications and Security* (Washington, DC, Nov. 17-21). ACM, New York, NY, 2002.

[Chess and McGraw 2004] Chess, B. and McGraw G. Static analysis for security. *IEEE Security and Privacy*, (Nov./Dec. 2004).

[Christey et al. 1999] Christey, S., Baker, D., Hill, W. and Mann, D. The Development of a Common Vulnerabilities and Exposures List. In the *Proceedings of the 2nd International Workshop on Recent Advances in Intrusion Detection* (West Lafayette, IN, September 1999).

[Christodorescu and Jha 2003] Christodorescu, M. and Jha, S. Static Analysis of Executables to Detect Malicious Patterns. In the *Proceedings of the 13th USENIX Security Symposium* (Washington, DC, August 2003). USENIX, Berkeley, CA, 2003.

[Cohen 1986] Cohen, N. *Ada as a second language*. McGraw-Hill, New York City, NY, 1986.

[Cowan et al. 1998] Cowan, C.; Pu, C.; Maier, D.; Hinton, H.; Walpole, J.; Bakke, P.; Beattie, S.; Grier, A.; Wagle, P. and Zhang Q. StackGuard: Automatic Detection and Prevention of Buffer-Overflow Attacks. In the *Proceedings of the 7th USENIX Security Symposium* (San Antonio, TX, January 1998). USENIX, Berkeley, CA, 1998.

[Cygwin 2004] Cygwin. Cygwin Website. www.Cygwin.com. Accessed on Dec. 10, 2004.

[Dabak, Borate, and Phadke 1999] Dabak, P.; Borate M. and Phadke, S. *Undocumented Windows NT*. Wiley and Sons, New York, NY, 1999.

[Dekok 2003] Dekok, A. PScan. *Striker-On-Line*. www.striker.ottawa.on.ca/~aland/pscan/. Accessed Oct. 28, 2003.

[Dor, Rodeh, and Sagiv 2003] Dor, N., Rodeh, M., Sagiv, M. CSSV: Towards a realistic tool for statically detecting all buffer overflows in C. In the *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation* (San Diego, CA, 2003). ACM, New York, NY, 2003.

[Du 1998] Du, W. Categorization of Software Errors That Led to Security Breaches. In the *Proceedings of the 21st National Information Systems Security Conference* (Crystal City, VA, 1998).

[DuVarney, Bhatkar, and Venkatakrishnan 2003] DuVarney, D.; Bhatkar, S. and Venkatakrishnan, V. SELF: A Transparent Security Extension for ELF Binaries. In the *Proceedings of the 2002 New Security Paradigms Workshop* (Ascona, Switzerland, Aug. 18-21, 2003).

[Erickson 2003] Erickson, J. *Hacking: The Art of Exploitation*. No Starch Press, San Francisco, CA, 2003.

[Evans 2004] Evans, D. *Secure Programming Lint (SPLINT)*. Department of Computer Science, University of Virginia. www.splint.org, accessed on Feb. 22, 2004.

[Evans and Larochelle 2002] Evans, D. and D. Larochelle. Improving Security Using Extensible Lightweight Static Analysis. *IEEE Software*, no. 2 (Jan./Feb. 2002), 42-51.

[Fusco 2004] Fusco, J. Ten Commands Every Linux Developer Should Know. *LINUX Journal*, September 2004.

[Ghosh and McGraw 1998] Ghosh, A. and McGraw G. An approach for certifying security in software components. In the *Proceedings of the 21st National Information Systems Security Conference* (Crystal City, VA, October 1998).

[Ghosh, O'Connor and McGraw 1998] Ghosh, A.; O'Connor, T. and McGraw, G. An automated approach for identifying potential vulnerabilities in software. In the *Proceedings of the IEEE Symposium on Security and Privacy* (Oakland, CA, May 1998).

[Gimpel 2003] Gimpel. PC-Lint Software. *Gimpel Software*. www.gimpel.com. Accessed Nov. 1, 2003.

[Goth 2002] Goth, G. Federal Government Calls for More Secure Software Design. *IEEE Software*, (Jan./Feb. 2002), 90-94.

[Graff and van Wyk 2003] Graff, M. and van Wyk, K. *Secure Coding: Principles and Practices*. O'Reilly and Associates, Sebastopol, CA, 2003.

[Grimes 2001] Grimes, R. *Malicious Mobile Code*. O'Reilly and Associates, Sebastopol, CA, 2001.

[Hall and Chapman 2002] Hall, A. and Chapman, R. Correctness by Construction: Developing a Commercial Secure System. *IEEE Software*, (Jan./Feb. 2002), 18-25.

[Hamilton, Greaney, and Evans 2003] Hamilton J.; Greaney K.; and Evans G. Defining a Process for Simulation Software Vulnerability Assessments. *CrossTalk*, (November 2003).

[Hankerson et al. 2000] Hankerson, D. et al. *Coding Theory and Cryptography: The Essentials, 2nd Edition*. Marcel-Dekker, New York, NY, 2000.

[Haugh and Bishop 2003] Haugh, E. and Bishop, M. Testing C programs for buffer overflow vulnerabilities. In the *Proceedings of the 2003 Symposium on Network and Distributed System Security* (San Diego, CA, February 2003).

[HBGary 2004a] HBGary. BugScan 2003 Press Release on July 31, 2003. *HBGary Inc*. www.hbgary.com. Accessed on 22 July 2004.

[HBGary 2004b] HBGary. BugScan Technical White Paper. *HBGary Inc*. www.hbgary.com. Accessed on 22 July 2004.

[HBGary 2004c] HBGary. BugScan 2003 User's Manual. *HBGary Inc*. www.hbgary.com. Accessed on 22 July 2004.

[HBGary 2004d] HBGary. BugScan FAQ. *HBGary Inc*. www.hbgary.com. Accessed on 22 July 2004.

[Hoglund and McGraw 2004] Hoglund, G. and McGraw, G. *Exploiting Software: How to Break Code*. Addison Wesley, Boston, MA, 2004.

[Holzmann 2003] Holzmann, G. UNO Software. *Bell Labs*. www.spinroot.com/gerard/. Accessed Nov. 1, 2003.

[Howard 2004] Howard, M. Building more secure software with improved development processes. *IEEE Security and Privacy*, (Nov./Dec. 2004).

[Howard and LeBlanc 2002] Howard, M. and LeBlanc, D. *Writing Secure Code*. Microsoft Press, Redmond, WA, 2002.

[Huang 2003] Huang, Y. Vulnerabilities in Portable Executable (PE) File Format for Win32 Architecture. *OS Security*. www.ossecurity.ca. Accessed on August 30, 2004.

[Hunt and Brubacher 1999] Hunt, G. and Brubacher, D. Detours: Binary Interception of Win32 Functions. In the *Proceedings of the third USENIX NT Symposium (*Seattle, WA, July 12-15, 1999).

[Immix Technology 2004] Immix Technology. GSA Schedule Pricing. *Immix Technology*. www.immixtechnology.com. Accessed August 31, 2004.

[InfoWorld 2004]. InfoWorld. Code Catcher in a Box. *InfoWorld*. www.infoworld.com. Accessed on August 28, 2004.

[Intel 2004] Intel Corporation. *IA-32 Intel Architecture Software Developer's Manual, Volumes 1-3*. http://developer.intel.com/design. Accessed on August 30, 2004.

[Irvine 2003] Irvine, K. *Assembly Language for Intel-Based Computers*. Prentice Hall, Upper Saddle River, NJ, 2003.

[Jaworski and Perrone 2000] Jaworski, J. and Perrone, P. *Java Security Handbook*. Sams Publishing. Indianapolis, IN, 2000.

[Jim et al. 2002] Jim, T.; Morrisett, G.; Grossman, D.; Hicks, M.; Cheney, J. and Wang, Y. Cyclone: A safe dialect of C. In the *Proceedings of the USENIX Annual Technical Conference* (Monterey, CA, June 2002). USENIX, Berkeley, CA, 2002.

[Jiwnani and Zelkowitz 2004] Jiwnani, K. and Zelkowitz, M. Susceptibility Matrix: A New Aid to Software Auditing. *IEEE Security and Privacy*, (Mar./Apr. 2004).

[Kahn and Han 2002] Kahn, K. and Han, J. Composing Security-Aware Software. *IEEE Software*, (Jan./Feb. 2002), 34-40.

[Kaspersky 2003] Kaspersky, K. *Hacker Disassembling Uncovered*. A-List, Wayne, PA, 2003.

[Khalilzad, White, and Marshall 1999] Khalilzad, Z.; White, J.; and Marshall, A. *Strategic Appraisal: The Changing Role of Information in Warfare*. Rand Corporation. Santa Monica, CA, 1999.

[Kirovski, Drinic, and Potkonjak 2002] Kirovski, D., Drinic, M., and Potkonjak, M. Enabling trusted software integrity. In the *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems* (San Jose, CA, October 2002).

[Koziol et al. 2004] Koziol, J. et al. *The Shellcoder's Handbook*. Wiley Publishing, Indianapolis, IN, 2004.

[Landwehr 1994] Landwehr, C. et al. A Taxonomy of Computer Program Security Flaws with Examples. *ACM Computing Surveys*, (Sep 1994).

[Larochelle and Evans 2001] Larochelle, D. and Evans, D. Statistically detecting likely buffer overflow vulnerabilities. In the *Proceedings of the 10th USENIX Security Symposium* (Washington, DC, August 2001).

[LDRA 2003] LDRA. LDRA Testbed Software. *LDRA Software Technology*. www.ldra.co.uk/. Accessed Nov. 1, 2003.

[Leveson 1995] Leveson, N. *Safeware: System Safety and Computers*. Addison Wesley, Reading, MA, 1995.

[Lewis and Loftus 2005] Lewis, J., and Loftus, W. *Java Software Solutions*. Addison Wesley, Boston, MA, 2005.

[Lhee and Chapin 2002] Lhee, K. and Chapin, S. Type-Assisted Dynamic Buffer Overflow Detection. In the *Proceedings of the 2002 USENIX Conference* (San Francisco, CA, Aug. 5-9, 2002).

[Louden 2003] Louden, K. *Programming Languages: Principles and Practice, 2nd Edition*. Thomson Brooks/Cole, Pacific Grove, CA, 2003.

[McComb 1997] McComb, G. *Web Programming Languages*. Wiley and Sons, New York, NY, 1997.

[McGraw 1998] McGraw, G. Testing for Security During Development: Why we should scrap penetrate-and-patch. *IEEE Aerospace and Electronic Systems*, April 1998.

[Microsoft Corporation 1999] Microsoft Corporation. Microsoft Portable Executable and Common Object File Format Specification Revision 6.0 February 1999. *Microsoft Corporation*. www.microsoft.com/whdc/system/platform/firmware. Accessed on Aug. 28, 2004.

[Miller and DeRaadt 1999] Miller, T. and DeRaadt, T. strlcpy and strlcat—consistent, safe, string copy and concatenation. In the *Proceedings of the 1999 USENIX Conference* (Monterey, CA, June 1999). USENIX, Berkeley, CA, 1999.

[Minasi 2001] Minasi, M. *Mastering Windows XP Professional*. Sybex, San Francisco, CA, 2001.

[Nazario 2002] Nazario, J. Source Code Scanners for Better Code. *LinuxJournal.Com*. www.linuxjournal.com//article.php?sid=5673. Accessed January 26, 2002.

[Neumann 2003] Neumann, P. Information System Security Redux. *Communications of the ACM*, (Oct 2003), 126.

[Nutt 2002] Nutt, G. *Operating Systems: A Modern Perspective*. Addison Wesley, Boston, MA, 2002.

[Onley 2004] Onley, D. NMCI launches prototype for apps testing. *Government Computer News*, (Aug 2004).

[Parasoft 2003] Parasoft. CodeWizard. *Parasoft Inc*. www.parasoft.com. Accessed November 1, 2003.

[PC Magazine 2004] PC Magazine. Review of SmartRisk Analyzer 1.0. *PC Magazine*. www.pcmag.com. Accessed on August 28, 2004.

[Perry 2004] Perry, M. *Introduction to Reverse Engineering Software*. Mike Perry Website. www.acm.uiuc.edu/sigmil/RevEng. Accessed on Dec. 22, 2004.

[Petron 2000] Petron, E. *Linux Essential Reference*. New Riders Publishing, Indianapolis, IN, 2000.

[Pietrek 2002a] Pietrek, M. An In-Depth Look into the Win32 Portable Executable File Format, Part 1. *MSDN Magazine*, (Feb 2002).

[Pietrek 2002b] Pietrek, M. An In-Depth Look into the Win32 Portable Executable File Format, Part 2. *MSDN Magazine*, (Mar 2002).

[Pincus and Baker 2004] Pincus, J. and Baker, B. Beyond Stack Smashing: Recent Advances in Exploiting Buffer Overruns. *IEEE Security and Privacy*, (Jul./Aug. 2004).

[Potter and McGraw 2004] Potter, B. and McGraw G. Software security testing. *IEEE Security and Privacy*, (Nov./Dec. 2004).

[Prasad and Chiueh 2003] Prasad, M. and Chieueh, T. A Binary Rewriting Defense Against Stack-based Buffer Overflow Attacks. In the *Proceedings of the 2003 USENIX Conference* (San Antonio, TX, Jun. 9-14, 2003).

[Pressman 2005] Pressman, R. *Software Engineering: A Practitioner's Approach, 6th Edition*. McGraw-Hill, Boston, MA., 2005.

[Reasoning 2003] Reasoning. Illuma Software. *Reasoning Inc*. www.reasoning.com. Accessed November 1, 2003.

[Rescorla 2005] Rescorla, E. Is finding security holes a good idea. *IEEE Security and Privacy*, (Jan./Feb. 2005).

[Scambray, McClure, and Kurtz 2001] Scambray, J.; McClure, S.; and Kurtz, G. *Hacking Exposed, 3rd Edition*. Osborne/McGraw-Hill, Berkeley, CA., 2001.

[Schaeffer 2002] Schaeffer, F. *Surfing Anonymously*. Data Becker, Newton, MA., 2002.

[Schiffman 2001] Schiffman, M. *Hacker's Challenge*. Osborne/McGraw-Hill, Berkeley, CA., 2001.

[Schildt 2000] Schildt, H. *C: The Complete Reference, 4th Edition*. McGraw-Hill, Berkeley, CA., 2000.

[Schwarz, Debray, and Andrews 2002] Schwarz, B.; Debray, S.; and Andrews, G. Disassembly of executable code revisited. In the *Proceedings of the 2002 Working Conference on Reverse Engineering* (Richmond, VA, Oct. 29-Nov.1, 2002).

[Secure Software 2004a] Secure Software. Rough Auditing Tool for Security (RATS). *Secure Software Inc*. www.securesoftware.com. Accessed July 22, 2004.

[Secure Software 2004b] Secure Software. Code Security Evaluation. *Secure Software Inc*. www.securesoftware.com. Accessed August 31, 2004.

[Shah 2004] Shah, S. One-Way Web Hacking: Attacking Web and Application Servers. *Net Square*. http://www.net-square.com/papers/one_way/one_way.html. net-square, 2004.

[Short 2002] Short, C. Source Code Revelation Vulnerabilities. *SANS Institute*. www.sans.org. SANS Institute, 2002.

[Sommerville 2001] Sommerville, I. *Software Engineering, 6th Edition*. Addison-Wesley, New York, NY., 2001.

[Soo Hoo, Sudbury and Jaquith 2001] Soo Hoo, K.; Sudbury, A. and Jaquith, A. Tangible ROI through Secure Software Engineering. *Secure Business Quarterly*, (Vol 1:2).

[Spafford and Weeber 1992] Spafford, E. and Weeber, S. Software forensics: can we track code to its authors. In the *Proceedings of the 15th National Information Systems Security Conference* (Washington, DC, October 1992).

[SPI Dynamics 2003] SPI Dynamics. WebInspect Software. *SPI Dynamics Inc*. www.spidynamics.com. Accessed November 1, 2003.

[Splaine 2002] Splaine, S. *Testing Web Security*. Wiley Publishing, Indianapolis, IN, 2002.

[Utimaco 2004] Utimaco Safeware. White Paper on Vulnerabilities in Pure Software Security Systems. *Utimaco Safeware*. www.utimaco.com. Accessed on August 30, 2004.

[Viega et al. 2000] Viega, J.; Block, J.; Kohno, T.; and McGraw, G. ITS4: A Static Vulnerability Scanner for C and C++ Code. *Cigital Inc*. www.cigital.com/its4/. 2000.

[Viega and McGraw 2002] Viega, J. and McGraw, G. *Building Secure Software*. Addison-Wesley, Boston, MA, 2002.

[Viega and Messier 2003] Viega, J. and Messier, M. *Secure Programming Cookbook for C and C++*. O'Reilly, Sebastopol, CA, 2003.

[Visual Studio 2004] Visual Studio. Microsoft Visual Studio Website. msdn.microsoft.com/visualc. Accessed on Dec. 10, 2004.

[Wagner 2003] Wagner, D. Modelchecking Program for Security Properties (MOPS). *Computer Science Division, University of California Berkeley*. www.cs.berkeley.edu/~daw/mops/. Accessed October 28, 2003.

[Wagner et al. 2000] Wagner, D.; J. Foster; E. Brewer; and A. Aiken. A First Step Towards Automated Detection of Buffer Overrun Vulnerabilities. In the *Proceedings of the 2000 Network and Distributed Security Symposium* (San Diego, CA, Feb. 3-4, 2000). ISOC.

[Wall, Watson, and Whitis 1999] Wall, K.; Watson, M.; and Whitis, M. *Linux Programming Unleashed*. Sams Publishing, Indianapolis, IN, 1999.

[Wheeler 2003] Wheeler, D. Flawfinder. *David A. Wheeler*. www.dwheeler.com/flawfinder/. Accessed October 28, 2003.

[Wheeler 2004] Wheeler, D. *Secure Programming for Linux and Unix*. David Wheeler Website. www.dwheeler.com. Accessed on Feb. 22, 2004.

[Whittaker 2004] Whittaker, K. Why Secure Applications are Difficult to Write. *IEEE Security and Privacy*, (Jan./Feb. 2003).

[Whittaker and Thompson 2004] Whittaker, J. and Thompson, H. *How to Break Software Security*. Addison Wesley, Boston, MA, 2004

[Wilander and Kamkar 2002] Wilander, J. and Kamkar, M. A comparison of publicly available tools for static intrusion prevention. In the *Proceedings of the 7th Nordic Workshop on Secure IT Systems* (Karlstad, Sweden, November 2002).

[Wong 2001] Wong, K. Introduction to Hacking Methods and Ways of Counter-Measure. *Internet Industry Association.* www.security.iia.net.au. October 21, 2001. Accessed on August 30, 2004.

[Woodmann 2004] Woodmann, C. *Reverser's Archive Pages of Reverse Engineering*. Woodman Website. www.woodmann.com/fravia. Accessed on Dec. 12, 2004.

[Yoder and Barcalow 1997] Yoder, J. and Barcalow, J. Architectural Patterns for Enabling Application Security. In the *Proceedings of the Fourth Conference on Pattern Languages of Programming Conference* (Monticello, IL, September 1997).

[Yong and Horwitz 2003] Yong, S. and Horwitz, S. Protecting C programs from attacks via invalid pointer dereferences. In the *Proceedings of the 9th European Software Engineering Conference* (Helsinki, Finland, 2003).

[Zacker 2001] Zacker, C. *PC Hardware: The Complete Reference*. McGraw-Hill, Berkelely, CA, 2001.

APPENDICES

7. APPENDIX A – INSECURE CODING PRACTICES TO AVOID [Graff and van Wyk 2003]

1) Don't write code that uses relative filenames. Such names can be redirected to another location.

2) Don't refer to a file by its name twice in the same program. Such code can cause race conditions on which physical file is being referenced.

3) Don't invoke untrusted programs from within trusted ones.

4) Avoid using setuid or similar mechanisms whenever possible. In particular, do not setuid to an existing identity/profile that has interactive login capabilities.

5) Don't assume that your users are not malicious. This means to double-check every piece of external information read by your software.

6) Don't dump core. Instead, design your program to degrade gracefully.

7) Don't assume success. In other words, check the return status of all functions calls.

8) Don't confuse random with pseudo-random. Any cryptological algorithm requires a sound random number generator.

9) Don't invoke a shell or a command line from within your program.

10) Don't authenticate on untrusted criteria. This means to not blindly accept the identity of a user or process based on an IP address, a MAC address, or an e-mail address.

11) Don't use world-writable storage, even temporarily. This refers to a common storage area offered by an operating system.

12) Don't trust the integrity of user-writable storage. Hackers can tamper with it.

13) Don't keep sensitive data in a database without password protection.

14) Don't echo passwords or display them on the user's screen for any reason.

15) Don't issue passwords via e-mail.

16) Don't programmatically distribute sensitive information via e-mail.

17) Don't code user names or passwords into an application.

18) Don't store unencrypted passwords (or other highly sensitive information) on disk in an easy-to-read format, such as straight unencrypted text.

19) Don't transmit unencrypted passwords (or other highly sensitive information) between systems in an easy-to-read format, such as straight (unencrypted) text.

20) Don't rely on host-level file protection mechanisms as the sole means of preventing unauthorized file access. Such controls can be easily compromised.

21) Don't make access decisions based on environment variables or command-line parameters passed in at run-time.

22) Avoid, if reasonable, storing the application or key data on an NFS-mounted structure.

23) Avoid, as much as you can, relying on third-party software or services for critical operations. Such decisions result in dependencies and additional risks

8. APPENDIX B – LIST OF COMMONLY USED HACKER TOOLS

[Erickson 2003, Fusco 2004, Hoglund and McGraw 2004, Petron 2000]

NAME	PLATFORM	OBTAIN	HACKER PURPOSE
ADMutate	Linux	Freeware	XOR encrypt shellcode
APISpy	Windows	Freeware	Log function calls made by running programs
DebugView	Windows	Freeware	Monitor kernel mode and Win32 debug output
depends	Windows	Windows	Show a dll dependency tree
disasm	Windows	Microsoft	Disassemble an instruction in your own program
dissembler	Linux	Freeware	Generate printable ASCII bytecode from an existing piece of bytecode
dsniff	Linux	Freeware	Sniff packets and looks for user names and passwords
dumpbin	Windows	Microsoft	Identify functions imported by a program
DyninstAPI	Multiple	Freeware	Insert code patches in running programs
exehdr	Linux	Linux	Display header of an executable file
fenris	Linux	Linux	Show runtime trace of a process
filemon	Windows	Freeware	Monitor and display file system activity
ffp	Linux	Freeware	Use fuzzy footprint technology to alter crypto keys
file	Linux	Linux	Identify the general type of information in a file
fuser	Linux	Linux	Tell what processes have opened a given file
gdb	Linux	Linux	Disassemble and debug binary files
IDA 4.1	Windows	Freeware	Disassemble Intel32 binary code
John the Ripper	Linux	Freeware	Detect weak passwords
ltrace	Linux	Linux	Show dll calls from a process
nasm	Both	Freeware	Assemble 80x86 instructions into many Windows and Linux formats
netcat	Linux	Linux	Read and write data across network

			connections using TCP/IP
netstat	Windows	Windows	Audit a system for local sockets
nemesis	Linux	Freeware	Inject packets from the command line
nm	Linux	Linux	List symbols in an object code or executable file
objdump	Linux	Linux	Display contents and disassemble object code
od	Linux	Linux	Convert a binary file's contents to octal, decimal or hex format (See xxd)
OllyObg	Windows	Freeware	Analyze binary code through dumps
ps -o	Linux	Linux	Access many details of a running process
regmon	Windows	Freeware	Monitor which applications access the registry
SoftICE	Windows	Compuware	Provide kernel mode debugging
Speedbreak	Linux	Freeware	Set breakpoints in process and data areas
strace	Linux	Linux	Show system calls from a process
strings	Linux	Linux	Look for ASCII strings embedded in executable files
tcpdump	Linux	Freeware	Print headers of packets sensed on a network interface (packet sniffer)
tdump	Windows	Borland	Identify functions imported by a program
time	Linux	Linux	Use to understand the runtime performance of a process
truss	Solaris	Solaris	Track library API calls of a process
windump	Windows	Freeware	Print headers of packets sensed on a network interface (packet sniffer)
xxd	Linux	Linux	Convert a binary file's contents to octal, decimal or hexadecimal format, without disturbing the byte ordering (See od tool)

9. APPENDIX C – TEST RESULTS FROM ANALYZING SPECIFIC EXAMPLE FILES

9.1 vulnerable - Object code compiled using Cygwin Gnu g++

```
FILE NAME: vulnerable-gpp.o


==== File Fact Summary ====

  - Object code file in Windows NT common object file format (COFF)
  - Actual file size: 1657 bytes
  - Created on Wed Dec 31 18:00:00 1969
  - Target CPU: Intel 386 or later compatibles
  - Not an executable image file
  - Targeted for a 32-bit-word architecture
  - Contains a string table with 6 entries
  - Contains a symbol table with 25 entries

==== End of File Fact Summary ====

No anomalies were found


!!!! Security Vulnerabilities and Risks!!!!

  - Reveals that it was built from one source code file:
vulnerable.cpp
  - Uses 13 standard C functions susceptible to buffer overflow
attacks: fscanf (Very high risk), getopt (Very high risk), gets (Ultra
high risk), realpath (Very high risk), scanf (Very high risk), sprintf
(Very high risk), sscanf (Very high risk), strcat (Very high risk),
strcpy (Very high risk), vfscanf (Very high risk), vscanf (Very high
risk), vsprintf (Very high risk), vsscanf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 1
```

```
List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     0    14   vulnerable-gpp.o
```

9.2 vulnerable - Compiled and linked using Borland C++ Builder 5

```
FILE NAME: vulnerable-borland.exe


==== File Fact Summary ====

  - Image file in Windows NT portable executable (PE) format
  - Actual file size: 66560 bytes
  - Created on Tue Feb  1 16:25:56 2005
  - Target CPU: Intel 386 or later compatibles
  - Targeted for a 32-bit-word architecture
  - Debugging information has been removed
  - Designed for Windows Operating System version 4.0
  - Runs in the Windows character subsystem
  - Lists these table names in the data directory: Export, Import,
Resource, Relocation, TLS
  - Contains no string table
  - Contains no symbol table
  - Exports functions using the file name vulnerable.exe
  - Imports functions from
    -- KERNEL32.DLL (WinNT base API client)
    -- USER32.DLL (Windows NT user API client)

==== End of File Fact Summary ====


**** Anomalies ****

  - The file indicates a thread local storage table exists consisting
of 24 bytes; this table usually does not appear in an image file so it
was not read and only its start address was mapped
  - The data directory table in the optional header states that the
TLS Table (.tls section) (?) is 4 bytes in size when actually it is
512 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 2460 bytes in size when actually
it is 2560 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 1

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     3     0   vulnerable-borland.exe
```

9.3 vulnerable - Compiled and linked using Cygwin Gnu g++

```
FILE NAME: vulnerable-gpp.exe


==== File Fact Summary ====

  - Image file in Windows NT portable executable (PE) format
  - Actual file size: 7168 bytes
  - Created on Tue Feb  1 16:01:34 2005
  - Target CPU: Intel 386 or later compatibles
  - Targeted for a 32-bit-word architecture
  - Debugging information has been removed
  - Designed for Windows Operating System version 4.0
  - Runs in the Windows character subsystem
  - Lists these table names in the data directory: Import
  - Contains no string table
  - Contains no symbol table
  - Imports functions from
    -- cygwin1.dll (CYGWIN GNU base dynamic link library)
    -- KERNEL32.dll (WinNT base API client)

==== End of File Fact Summary ====


**** Anomalies ****

  - A section entry named .bss appears in the section table, but the
table doesn't contain the location of the 128 bytes for that section

**** End of Anomalies ****


!!!! Security Vulnerabilities and Risks!!!!

  - Uses 24 standard C functions susceptible to buffer overflow
attacks: bcopy (Low risk), fgetc (Medium risk), fgets (Low risk),
```

```
fscanf (Very high risk), getc (Medium risk), getopt (Very high risk),
getopt_long (Very high risk), getpass (Very high risk), gets (Ultra
high risk), memcpy (Low risk), read (Medium risk), realpath (Very high
risk), scanf (Very high risk), snprintf (Low risk), sprintf (Very high
risk), sscanf (Very high risk), strcat (Very high risk), strcpy (Very
high risk), strncpy (Low risk), vfscanf (Very high risk), vscanf (Very
high risk), vsnprintf (Low risk), vsprintf (Very high risk), vsscanf
(Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 1

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     1    24   vulnerable-gpp.exe
```

9.4 vulnerable - Compiled and linked using Microsoft Visual Studio

---

```
FILE NAME: vulnerable-vs.exe


==== File Fact Summary ====

  - Image file in Windows NT portable executable (PE) format
  - Actual file size: 45056 bytes
  - Created on Tue Feb  1 16:17:17 2005
  - Target CPU: Intel 386 or later compatibles
  - Targeted for a 32-bit-word architecture
  - Designed for Windows Operating System version 4.0
  - Runs in the Windows character subsystem
  - Lists these table names in the data directory: Import, Import
Address
  - Contains no string table
  - Contains no symbol table
  - Imports functions from
    -- KERNEL32.dll (WinNT base API client)

==== End of File Fact Summary ====


**** Anomalies ****
```

```
  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 212 bytes
exists starting at address 32768; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 40 bytes in size when actually it is
1165 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 1

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     3     0   vulnerable-vs.exe
```

9.5 findssv - Compiled and linked using Borland C++ Builder 5

```
FILE NAME: findssv.exe


---- File Map ----

 ADDRESS  DESCRIPTION

       0  +-------------------------------------------------------+
            DOS Header  [64 bytes]
      63  +-------------------------------------------------------+
      64  +-------------------------------------------------------+
            MS-DOS Stub  [56 bytes]
     119  +-------------------------------------------------------+
     120  +-------------------------------------------------------+
            (** Zero-filled region **)  [392 bytes]
     511  +-------------------------------------------------------+
     512  +-------------------------------------------------------+
            PE Signature  [4 bytes]
     515  +-------------------------------------------------------+
     516  +-------------------------------------------------------+
            File Header  [20 bytes]
     535  +-------------------------------------------------------+
     536  +-------------------------------------------------------+
```

```
            Optional Header  [224 bytes]

     759  +------------------------------------------------------------+
     760  +------------------------------------------------------------+
            Section Table  [320 bytes]
    1079  +------------------------------------------------------------+
    1080  +------------------------------------------------------------+
            (** Zero-filled region **)  [456 bytes]
    1535  +------------------------------------------------------------+
    1536  +------------------------------------------------------------+
            .text section  [973312 bytes]
  974847  +------------------------------------------------------------+
  974848  +------------------------------------------------------------+
            .data section  [167424 bytes]
 1142271  +------------------------------------------------------------+
 1142272  +------------------------------------------------------------+
            .tls section  [512 bytes]
 1142783  +------------------------------------------------------------+
 1142784  +------------------------------------------------------------+
            .rdata section  [512 bytes]
 1143295  +------------------------------------------------------------+
 1143296  +------------------------------------------------------------+
            .idata section  [1660 bytes]
 1144955  +------------------------------------------------------------+
 1144956  +------------------------------------------------------------+
            (** Zero-filled region **)  [388 bytes]
 1145343  +------------------------------------------------------------+
 1145344  +------------------------------------------------------------+
            .edata section  [107 bytes]
 1145450  +------------------------------------------------------------+
 1145451  +------------------------------------------------------------+
            (** Zero-filled region **)  [405 bytes]
 1145855  +------------------------------------------------------------+
 1145856  +------------------------------------------------------------+
            .rsrc section  [512 bytes]
 1146367  +------------------------------------------------------------+
 1146368  +------------------------------------------------------------+
            .reloc section  [31744 bytes]
 1178111  +------------------------------------------------------------+

---- End of File Map ----

NOTES ON FILE MAP CHANGES:
 - Changed "Export Table (.edata section)  [107 bytes]" at address
1145344
   by inserting ".edata section  [107 bytes]"
 - Changed "Import Table (.idata section)  [1660 bytes]" at address
1143296
   by inserting ".idata section  [1660 bytes]"
 - Changed "TLS Table (.tls section) (?)  [4 bytes]" at address
1142784
   by inserting ".rdata section  [512 bytes]"
 - Changed "Resource Table (.rsrc section)  [512 bytes]" at address
1145856
   by inserting ".rsrc section  [512 bytes]"
```

```
 - Changed "Relocation Table (.reloc section)  [31724 bytes]" at
address 1146368
   by inserting ".reloc section  [31744 bytes]"
 - Changed "(Contents not known)  [392 bytes]" at address 120
   by inserting "(** Zero-filled region **)  [392 bytes]"
 - Changed "(Contents not known)  [456 bytes]" at address 1080
   by inserting "(** Zero-filled region **)  [456 bytes]"
 - Changed "(Contents not known)  [388 bytes]" at address 1144956
   by inserting "(** Zero-filled region **)  [388 bytes]"
 - Changed "(Contents not known)  [405 bytes]" at address 1145451
   by inserting "(** Zero-filled region **)  [405 bytes]"



==== File Fact Summary ====

  - Image file in Windows NT portable executable (PE) format
  - Actual file size: 1178112 bytes
  - Created on Wed Jan 26 21:12:31 2005
  - Target CPU: Intel 386 or later compatibles
  - Targeted for a 32-bit-word architecture
  - Debugging information has been removed
  - Designed for Windows Operating System version 4.0
  - Runs in the Windows character subsystem
  - Lists these table names in the data directory: Export, Import,
Resource, Relocation, TLS
  - Contains no string table
  - Contains no symbol table
  - Exports functions using the file name findssv.exe
  - Imports functions from
    -- KERNEL32.DLL (WinNT base API client)
    -- USER32.DLL (Windows NT user API client)

==== End of File Fact Summary ====


**** Anomalies ****

  - The file indicates a thread local storage table exists consisting
of 24 bytes; this table usually does not appear in an image file so it
was not read and only its start address was mapped
  - The data directory table in the optional header states that the
TLS Table (.tls section) (?) is 4 bytes in size when actually it is
512 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 31724 bytes in size when actually
it is 31744 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 1

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     3     0   findssv.exe
```

9.6 findssv - Compiled and linked using Cygwin Gnu g++

```
FILE NAME: findssv.exe


---- File Map ----

 ADDRESS  DESCRIPTION

       0  +-----------------------------------------------------------+
            DOS Header  [64 bytes]
      63  +-----------------------------------------------------------+
      64  +-----------------------------------------------------------+
            MS-DOS Stub  [57 bytes]
     120  +-----------------------------------------------------------+
     121  +-----------------------------------------------------------+
            (Contents not known)  [7 bytes]
     127  +-----------------------------------------------------------+
     128  +-----------------------------------------------------------+
            PE Signature  [4 bytes]
     131  +-----------------------------------------------------------+
     132  +-----------------------------------------------------------+
            File Header  [20 bytes]
     151  +-----------------------------------------------------------+
     152  +-----------------------------------------------------------+
            Optional Header  [224 bytes]
     375  +-----------------------------------------------------------+
     376  +-----------------------------------------------------------+
            Section Table  [200 bytes]
     575  +-----------------------------------------------------------+
     576  +-----------------------------------------------------------+
            (** Zero-filled region **)  [448 bytes]
    1023  +-----------------------------------------------------------+
    1024  +-----------------------------------------------------------+
            .text section  [594944 bytes]
  595967  +-----------------------------------------------------------+
  595968  +-----------------------------------------------------------+
            .data section  [4608 bytes]
  600575  +-----------------------------------------------------------+
  600576  +-----------------------------------------------------------+
```

```
            .rdata section  [78848 bytes]
  679423  +------------------------------------------------------------+
  679424  +------------------------------------------------------------+
            .idata section  [1387 bytes]
  680810  +------------------------------------------------------------+
  680811  +------------------------------------------------------------+
            More of Import Table (.idata section)  [281 bytes]
  681091  +------------------------------------------------------------+
  681092  +------------------------------------------------------------+
            (** Zero-filled region **)  [380 bytes]
  681471  +------------------------------------------------------------+

---- End of File Map ----

NOTES ON FILE MAP CHANGES:
 - Changed "Import Table (.idata section)  [1668 bytes]" at address
679424
   by inserting ".idata section  [1387 bytes]"
 - Changed "(Contents not known)  [448 bytes]" at address 576
   by inserting "(** Zero-filled region **)  [448 bytes]"
 - Changed "(Contents not known)  [380 bytes]" at address 681092
   by inserting "(** Zero-filled region **)  [380 bytes]"



==== File Fact Summary ====

  - Image file in Windows NT portable executable (PE) format
  - Actual file size: 681472 bytes
  - Created on Tue Jan 25 19:32:32 2005
  - Target CPU: Intel 386 or later compatibles
  - Debugging information has been removed
  - Designed for Windows Operating System version 4.0
  - Runs in the Windows character subsystem
  - Lists these table names in the data directory: Import
  - Contains no string table
  - Contains no symbol table
  - Imports functions from
    -- cygwin1.dll (CYGWIN GNU base dynamic link library)
    -- KERNEL32.dll (WinNT base API client)

==== End of File Fact Summary ====


**** Anomalies ****

  - A section entry named .bss appears in the section table, but the
table doesn't contain the location of the 5392 bytes for that section

**** End of Anomalies ****



!!!! Security Vulnerabilities and Risks!!!!
```

```
  - Uses 7 standard C functions susceptible to buffer overflow
attacks: getc (Medium risk), memcpy (Low risk), sprintf (Very high
risk), sscanf (Very high risk), strcat (Very high risk), strcpy (Very
high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 1

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     1     7   findssv.exe
```

9.7 findssv - Compiled and linked using Microsoft Visual Studio

---

```
FILE NAME: findssv.exe


---- File Map ----

 ADDRESS  DESCRIPTION

       0  +-------------------------------------------------------+
            DOS Header  [64 bytes]
      63  +-------------------------------------------------------+
      64  +-------------------------------------------------------+
            MS-DOS Stub  [57 bytes]
     120  +-------------------------------------------------------+
     121  +-------------------------------------------------------+
            (Contents not known)  [95 bytes]
     215  +-------------------------------------------------------+
     216  +-------------------------------------------------------+
            PE Signature  [4 bytes]
     219  +-------------------------------------------------------+
     220  +-------------------------------------------------------+
            File Header  [20 bytes]
     239  +-------------------------------------------------------+
     240  +-------------------------------------------------------+
            Optional Header  [224 bytes]
     463  +-------------------------------------------------------+
     464  +-------------------------------------------------------+
            Section Table  [120 bytes]
     583  +-------------------------------------------------------+
```

```
     584  +-----------------------------------------------------+
            (Contents not known)  [3512 bytes]
    4095  +-----------------------------------------------------+
    4096  +-----------------------------------------------------+
            .text section  [331776 bytes]
  335871  +-----------------------------------------------------+
  335872  +-----------------------------------------------------+
            .rdata section  [32768 bytes]
  368639  +-----------------------------------------------------+
          335872  +-----------------------------------------+
                    (No additional details)  [28968 bytes]
          364839  +-----------------------------------------+
          364840  +-----------------------------------------+
                    Import Table (.idata section)  [1626 bytes]
          366465  +-----------------------------------------+
          366466  +-----------------------------------------+
                    (No additional details)  [2174 bytes]
          368639  +-----------------------------------------+
  368640  +-----------------------------------------------------+
            .data section  [53248 bytes]
  421887  +-----------------------------------------------------+

---- End of File Map ----

NOTES ON FILE MAP CHANGES:
 - Changed "Import Table (.idata section)  [40 bytes]" at address
364840
   by inserting "Import Table (.idata section)  [1626 bytes]"



==== File Fact Summary ====

  - Image file in Windows NT portable executable (PE) format
  - Actual file size: 421888 bytes
  - Created on Fri Jan 28 13:34:56 2005
  - Target CPU: Intel 386 or later compatibles
  - Targeted for a 32-bit-word architecture
  - Designed for Windows Operating System version 4.0
  - Runs in the Windows character subsystem
  - Lists these table names in the data directory: Import, Import
Address
  - Contains no string table
  - Contains no symbol table
  - Imports functions from
    -- KERNEL32.dll (WinNT base API client)

==== End of File Fact Summary ====


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
```

```
  - The file indicates an import address table consisting of 292 bytes
exists starting at address 335872; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 40 bytes in size when actually it is
1626 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 1

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     3     0   findssv.exe
```

9.8 Cygwin Gnu cygwin1.dll Dynamic Link Library

```
FILE NAME: cygwin1.dll


**** Anomalies ****

  - A section entry named .bss appears in the section table, but the
table doesn't contain the location of the 226272 bytes for that
section
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 1104 bytes in size when actually it
is 1536 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 43792 bytes in size when actually
it is 44032 bytes in size

**** End of Anomalies ****



!!!! Security Vulnerabilities and Risks!!!!

  - Has a section named .advapi32_text whose contents can be both
written to and executed
```

```
  - Has a section named .netapi32_text whose contents can be both
written to and executed
  - Has a section named .ntdll_text whose contents can be both written
to and executed
  - Has a section named .psapi_text whose contents can be both written
to and executed
  - Has a section named .secur32_text whose contents can be both
written to and executed
  - Has a section named .user32_text whose contents can be both
written to and executed
  - Has a section named .wsock32_text whose contents can be both
written to and executed
  - Has a section named .ws2_32_text whose contents can be both
written to and executed
  - Has a section named .iphlpapi_text whose contents can be both
written to and executed
  - Has a section named .ole32_text whose contents can be both written
to and executed
  - Has a section named .kernel32_text whose contents can be both
written to and executed
  - Has a section named .winmm_text whose contents can be both written
to and executed

!!!! End of Security Vulnerabilities and Risks!!!!
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 1

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     3    12   cygwin1.dll
```

9.9 helloworld - Compiled and linked using Borland C++ Builder 5

```
FILE NAME: helloworld-borland.exe


---- File Map ----

 ADDRESS  DESCRIPTION

       0  +-------------------------------------------------------+
            DOS Header  [64 bytes]
```

```
      63  +-----------------------------------------------------------+
      64  +-----------------------------------------------------------+
            MS-DOS Stub  [56 bytes]
     119  +-----------------------------------------------------------+
     120  +-----------------------------------------------------------+
            (** Zero-filled region **)  [392 bytes]
     511  +-----------------------------------------------------------+
     512  +-----------------------------------------------------------+
            PE Signature  [4 bytes]
     515  +-----------------------------------------------------------+
     516  +-----------------------------------------------------------+
            File Header  [20 bytes]
     535  +-----------------------------------------------------------+
     536  +-----------------------------------------------------------+
            Optional Header  [224 bytes]
     759  +-----------------------------------------------------------+
     760  +-----------------------------------------------------------+
            Section Table  [320 bytes]
    1079  +-----------------------------------------------------------+
    1080  +-----------------------------------------------------------+
            (** Zero-filled region **)  [456 bytes]
    1535  +-----------------------------------------------------------+
    1536  +-----------------------------------------------------------+
            .text section  [79872 bytes]
   81407  +-----------------------------------------------------------+
   81408  +-----------------------------------------------------------+
            .data section  [23040 bytes]
  104447  +-----------------------------------------------------------+
  104448  +-----------------------------------------------------------+
            .tls section  [512 bytes]
  104959  +-----------------------------------------------------------+
  104960  +-----------------------------------------------------------+
            .rdata section  [512 bytes]
  105471  +-----------------------------------------------------------+
  105472  +-----------------------------------------------------------+
            .idata section  [1450 bytes]
  106921  +-----------------------------------------------------------+
  106922  +-----------------------------------------------------------+
            (** Zero-filled region **)  [86 bytes]
  107007  +-----------------------------------------------------------+
  107008  +-----------------------------------------------------------+
            .edata section  [110 bytes]
  107117  +-----------------------------------------------------------+
  107118  +-----------------------------------------------------------+
            (** Zero-filled region **)  [402 bytes]
  107519  +-----------------------------------------------------------+
  107520  +-----------------------------------------------------------+
            .rsrc section  [512 bytes]
  108031  +-----------------------------------------------------------+
  108032  +-----------------------------------------------------------+
            .reloc section  [4608 bytes]
  112639  +-----------------------------------------------------------+

---- End of File Map ----
```

```
NOTES ON FILE MAP CHANGES:
 - Changed "Export Table (.edata section)  [110 bytes]" at address
107008
   by inserting ".edata section  [110 bytes]"
 - Changed "Import Table (.idata section)  [1450 bytes]" at address
105472
   by inserting ".idata section  [1450 bytes]"
 - Changed "TLS Table (.tls section) (?)  [4 bytes]" at address 104960
   by inserting ".rdata section  [512 bytes]"
 - Changed "Resource Table (.rsrc section)  [512 bytes]" at address
107520
   by inserting ".rsrc section  [512 bytes]"
 - Changed "Relocation Table (.reloc section)  [4412 bytes]" at
address 108032
   by inserting ".reloc section  [4608 bytes]"
 - Changed "(Contents not known)  [392 bytes]" at address 120
   by inserting "(** Zero-filled region **)  [392 bytes]"
 - Changed "(Contents not known)  [456 bytes]" at address 1080
   by inserting "(** Zero-filled region **)  [456 bytes]"
 - Changed "(Contents not known)  [86 bytes]" at address 106922
   by inserting "(** Zero-filled region **)  [86 bytes]"
 - Changed "(Contents not known)  [402 bytes]" at address 107118
   by inserting "(** Zero-filled region **)  [402 bytes]"



==== File Fact Summary ====

  - Image file in Windows NT portable executable (PE) format
  - Actual file size: 112640 bytes
  - Created on Wed Jan 26 17:27:01 2005
  - Target CPU: Intel 386 or later compatibles
  - Targeted for a 32-bit-word architecture
  - Debugging information has been removed
  - Designed for Windows Operating System version 4.0
  - Runs in the Windows character subsystem
  - Lists these table names in the data directory: Export, Import,
Resource, Relocation, TLS
  - Contains no string table
  - Contains no symbol table
  - Exports functions using the file name helloworld.exe
  - Imports functions from
    -- KERNEL32.DLL (WinNT base API client)
    -- USER32.DLL (Windows NT user API client)

==== End of File Fact Summary ====


**** Anomalies ****

  - The file indicates a thread local storage table exists consisting
of 24 bytes; this table usually does not appear in an image file so it
was not read and only its start address was mapped
```

```
  - The data directory table in the optional header states that the
TLS Table (.tls section) (?) is 4 bytes in size when actually it is
512 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 4412 bytes in size when actually
it is 4608 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found



---------- DOS Header ----------
DOS signature:                    MZ
Bytes on last page of file:       80
Pages in file:                    2
Relocations:                      0
Size of header in paragraphs:     4
Minimum extra paragraphs:         15
Maximum extra paragraphs:         65535
Initial (relative) SS value:      0
Initial SP value:                 184
Checksum:                         0
Initial IP value:                 0
Initial (relative) CS value:      0
File address of relocation table: 64
Overlay number:                   26
Reserved words:                   0  0  0  0
OEM identifier:                   0
OEM information:                  0
Reserved words:                   0  0
Offset to PE signature and header: 512

-------- End of DOS Header --------

---------- File Header ----------
Target CPU:              Intel 386 or later compatibles
Number of Sections:      8
Time Date Stamp:         Wed Jan 26 17:27:01 2005
Ptr to Symbol Table:     0
Number of Symbols:       0
Size of Optional Header: 224
Characteristics:
   - This is an executable image
   - COFF line numbers have been removed
   - COFF symbol table entries for local symbols have been removed
   - Software is targeted for a 32-bit-word architecture
   - Debugging information has been removed from the image file

--------- End of File Header -------

---------- Optional Header ----------
Magic:                       267  (PE32 executable)
Linker Version:              5.0
```

```
Size of Code:                81920
Size of Initialized Data:    28672
Size of Uninitialized Data:  0
Address of Entry Point:      4096
Base of Code:                4096
Base of Data:                86016
Image Base:                  4194304
Section Alignment:           4096
File Alignment:              512
OS Version:                  4.0
Major Image Version:         0
Minor Image Version:         0
Major Subsystem Version:     4
Minor Subsystem Version:     0
Size of Image:               143360
Size of Headers:             1536
Check Sum:                   0
Required Windows Subsystem:  Runs in the Windows character subsystem
DLL Characteristics:         0
Size of Stack Reserve:       1048576
Size of Stack Commit:        8192
Size of Heap Reserve:        1048576
Size of Heap Commit:         4096
Loader Flags:                0
Nbr of RVS and Sizes:        16
Data Directory:
   Export Table                  RVA: 0126976  Size: 110
   Import Table                  RVA: 0122880  Size: 1450
   Resource Table                RVA: 0131072  Size: 512
   Exception Table               RVA: 0000000  Size: 0
   Certificate Table             RVA: 0000000  Size: 0
   Base Relocation Table         RVA: 0135168  Size: 4412
   Debug                         RVA: 0000000  Size: 0
   Architecture                  RVA: 0000000  Size: 0
   Global Ptr                    RVA: 0000000  Size: 0
   TLS Table                     RVA: 0118784  Size: 24
   Load Config Table             RVA: 0000000  Size: 0
   Bound Import                  RVA: 0000000  Size: 0
   Import Address Table          RVA: 0000000  Size: 0
   Delay Import Descriptor       RVA: 0000000  Size: 0
   COM+ Runtime Header           RVA: 0000000  Size: 0
   Reserved                      RVA: 0000000  Size: 0

-------- End of Optional Header --------

---- Section Table (8 entries) ----

Entry#0:
   Full Name:             .text
   Stored name:           .text
   Virtual Size:          81920
   Virtual Address:       4096
   Size of Raw Data:      79872
   Ptr to Raw Data:       1536
   Ptr to Relocations:    0
```

```
   Ptr to Line Numbers:   No COFF line numbers
   Number of Relocations: 0
   Number of Linenumbers: 0
   Characteristics:
      - Contains executable code
      - Can be executed as code
      - Can be read from
      - Cannot be written to

Entry#1:
   Full Name:             .data
   Stored name:           .data
   Virtual Size:          28672
   Virtual Address:       86016
   Size of Raw Data:      23040
   Ptr to Raw Data:       81408
   Ptr to Relocations:    0
   Ptr to Line Numbers:   No COFF line numbers
   Number of Relocations: 0
   Number of Linenumbers: 0
   Characteristics:
      - Contains initialized data
      - Can be read from
      - Can be written to

Entry#2:
   Full Name:             .tls
   Stored name:           .tls
   Virtual Size:          4096
   Virtual Address:       114688
   Size of Raw Data:      512
   Ptr to Raw Data:       104448
   Ptr to Relocations:    0
   Ptr to Line Numbers:   No COFF line numbers
   Number of Relocations: 0
   Number of Linenumbers: 0
   Characteristics:
      - Contains initialized data
      - Can be read from
      - Can be written to

Entry#3:
   Full Name:             .rdata
   Stored name:           .rdata
   Virtual Size:          4096
   Virtual Address:       118784
   Size of Raw Data:      512
   Ptr to Raw Data:       104960
   Ptr to Relocations:    0
   Ptr to Line Numbers:   No COFF line numbers
   Number of Relocations: 0
   Number of Linenumbers: 0
   Characteristics:
      - Contains initialized data
      - Can be shared in memory
```

```
      - Can be read from
      - Cannot be written to

Entry#4:
   Full Name:             .idata
   Stored name:           .idata
   Virtual Size:          4096
   Virtual Address:       122880
   Size of Raw Data:      1536
   Ptr to Raw Data:       105472
   Ptr to Relocations:    0
   Ptr to Line Numbers:   No COFF line numbers
   Number of Relocations: 0
   Number of Linenumbers: 0
   Characteristics:
      - Contains initialized data
      - Can be read from
      - Cannot be written to

Entry#5:
   Full Name:             .edata
   Stored name:           .edata
   Virtual Size:          4096
   Virtual Address:       126976
   Size of Raw Data:      512
   Ptr to Raw Data:       107008
   Ptr to Relocations:    0
   Ptr to Line Numbers:   No COFF line numbers
   Number of Relocations: 0
   Number of Linenumbers: 0
   Characteristics:
      - Contains initialized data
      - Can be read from
      - Cannot be written to

Entry#6:
   Full Name:             .rsrc
   Stored name:           .rsrc
   Virtual Size:          4096
   Virtual Address:       131072
   Size of Raw Data:      512
   Ptr to Raw Data:       107520
   Ptr to Relocations:    0
   Ptr to Line Numbers:   No COFF line numbers
   Number of Relocations: 0
   Number of Linenumbers: 0
   Characteristics:
      - Contains initialized data
      - Can be read from
      - Cannot be written to

Entry#7:
   Full Name:             .reloc
   Stored name:           .reloc
   Virtual Size:          8192
```

```
   Virtual Address:       135168
   Size of Raw Data:      4608
   Ptr to Raw Data:       108032
   Ptr to Relocations:    0
   Ptr to Line Numbers:   No COFF line numbers
   Number of Relocations: 0
   Number of Linenumbers: 0
   Characteristics:
      - Contains initialized data
      - Can be shared in memory
      - Can be read from
      - Cannot be written to

-------- End of Section Table --------

No Symbol Table to display

No String Table to display

---- Export Table ----

Directory Table:
   Characteristics:              0
   Time Date Stamp:              Wed Dec 31 18:00:00 1969
   Major Version:                0
   Major Version:                0
   Name of DLL:                  helloworld.exe
   Starting Ordinal Number:      1
   Number of Function Addresses: 2
   Number of Names:              2
   Location of Address Table:    0x1f028
   Location of Name Ptr Table:   0x1f030
   Location of Ordinal Table:    0x1f038

Export Name Table (2 names):
   Entry#1: __GetExceptDLLinfo
   Entry#2: ___CPPdebugHook

Forwarder Name Table (0 names):

---- End of Export Table ----

---- Import Table ----
Import Directory Table:

Entry#0:
   RVA of Lookup Table:            122940
   Time Date Stamp:                Tue Apr 14 17:09:36 1970
   Start Index of Forwarder Chain: 25886720
   DLL Name:                       KERNEL32.DLL
   RVA of Import Lookup Table:     0x1e10c
   Import Lookup Table Contents:
      CloseHandle
      CreateFileA
      EnterCriticalSection
```

```
      ExitProcess
      GetACP
      GetCPInfo
      GetCommandLineA
      GetCurrentThreadId
      GetEnvironmentStrings
      GetFileType
      GetLastError
      GetLocalTime
      GetLocaleInfoA
      GetModuleFileNameA
      GetModuleHandleA
      GetOEMCP
      GetProcAddress
      GetProcessHeap
      GetStartupInfoA
      GetStdHandle
      GetStringTypeA
      GetStringTypeW
      GetSystemDefaultLangID
      GetUserDefaultLCID
      GetVersion
      GetVersionExA
      GlobalMemoryStatus
      HeapAlloc
      HeapFree
      IsValidLocale
      LCMapStringA
      LeaveCriticalSection
      LoadLibraryA
      MultiByteToWideChar
      RaiseException
      ReadFile
      RtlUnwind
      SetConsoleCtrlHandler
      SetFilePointer
      SetHandleCount
      SetLastError
      SetThreadLocale
      TlsAlloc
      TlsFree
      TlsGetValue
      TlsSetValue
      UnhandledExceptionFilter
      VirtualAlloc
      VirtualFree
      WideCharToMultiByte
      WriteFile

Entry#1:
   RVA of Lookup Table:            123356
   Time Date Stamp:                Wed Dec 31 20:16:32 1969
   Start Index of Forwarder Chain: 4305980
   DLL Name:                       USER32.DLL
   RVA of Import Lookup Table:     0x1e1ec
```

```
   Import Lookup Table Contents:
      EnumThreadWindows
      MessageBoxA
      wsprintfA

---- End of Import Table ----

No Debug Table data to display
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 1

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     3     0   helloworld-borland.exe
```

9.10 helloworld - Compiled and linked using Cygwin Gnu g++

```
FILE NAME: helloworld-gnu.exe


---- File Map ----

 ADDRESS  DESCRIPTION

       0  +-------------------------------------------------------+
            DOS Header  [64 bytes]
      63  +-------------------------------------------------------+
      64  +-------------------------------------------------------+
            MS-DOS Stub  [57 bytes]
     120  +-------------------------------------------------------+
     121  +-------------------------------------------------------+
            (Contents not known)  [7 bytes]
     127  +-------------------------------------------------------+
     128  +-------------------------------------------------------+
            PE Signature  [4 bytes]
     131  +-------------------------------------------------------+
     132  +-------------------------------------------------------+
            File Header  [20 bytes]
     151  +-------------------------------------------------------+
     152  +-------------------------------------------------------+
            Optional Header  [224 bytes]
     375  +-------------------------------------------------------+
     376  +-------------------------------------------------------+
            Section Table  [200 bytes]
```

```
     575  +------------------------------------------------------------+
     576  +------------------------------------------------------------+
            (** Zero-filled region **)  [448 bytes]
    1023  +------------------------------------------------------------+
    1024  +------------------------------------------------------------+
            .text section  [176128 bytes]
  177151  +------------------------------------------------------------+
  177152  +------------------------------------------------------------+
            .data section  [4608 bytes]
  181759  +------------------------------------------------------------+
  181760  +------------------------------------------------------------+
            .rdata section  [35840 bytes]
  217599  +------------------------------------------------------------+
  217600  +------------------------------------------------------------+
            .idata section  [1183 bytes]
  218782  +------------------------------------------------------------+
  218783  +------------------------------------------------------------+
            More of Import Table (.idata section)  [241 bytes]
  219023  +------------------------------------------------------------+
  219024  +------------------------------------------------------------+
            (** Zero-filled region **)  [112 bytes]
  219135  +------------------------------------------------------------+

---- End of File Map ----

NOTES ON FILE MAP CHANGES:
 - Changed "Import Table (.idata section)  [1424 bytes]" at address
217600
   by inserting ".idata section  [1183 bytes]"
 - Changed "(Contents not known)  [448 bytes]" at address 576
   by inserting "(** Zero-filled region **)  [448 bytes]"
 - Changed "(Contents not known)  [112 bytes]" at address 219024
   by inserting "(** Zero-filled region **)  [112 bytes]"



==== File Fact Summary ====

  - Image file in Windows NT portable executable (PE) format
  - Actual file size: 219136 bytes
  - Created on Wed Jan 26 17:24:34 2005
  - Target CPU: Intel 386 or later compatibles
  - Debugging information has been removed
  - Designed for Windows Operating System version 4.0
  - Runs in the Windows character subsystem
  - Lists these table names in the data directory: Import
  - Contains no string table
  - Contains no symbol table
  - Imports functions from
    -- cygwin1.dll (CYGWIN GNU base dynamic link library)
    -- KERNEL32.dll (WinNT base API client)

==== End of File Fact Summary ====
```

```
**** Anomalies ****

  - A section entry named .bss appears in the section table, but the
table doesn't contain the location of the 4464 bytes for that section

**** End of Anomalies ****



!!!! Security Vulnerabilities and Risks!!!!

  - Uses 7 standard C functions susceptible to buffer overflow
attacks: getc (Medium risk), memcpy (Low risk), sprintf (Very high
risk), sscanf (Very high risk), strcat (Very high risk), strcpy (Very
high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!



---------- DOS Header ----------
DOS signature:                    MZ
Bytes on last page of file:       144
Pages in file:                    3
Relocations:                      0
Size of header in paragraphs:     4
Minimum extra paragraphs:         0
Maximum extra paragraphs:         65535
Initial (relative) SS value:      0
Initial SP value:                 184
Checksum:                         0
Initial IP value:                 0
Initial (relative) CS value:      0
File address of relocation table: 64
Overlay number:                   0
Reserved words:                   0  0  0  0
OEM identifier:                   0
OEM information:                  0
Reserved words:                   0  0
Offset to PE signature and header: 128

-------- End of DOS Header --------

---------- File Header ----------
Target CPU:              Intel 386 or later compatibles
Number of Sections:      5
Time Date Stamp:         Wed Jan 26 17:24:34 2005
Ptr to Symbol Table:     0
Number of Symbols:       0
Size of Optional Header: 224
Characteristics:
   - Base relocations have been stripped (default linker action),
     so file must be loaded at preferred base address
   - This is an executable image
   - COFF line numbers have been removed
```

```
   - COFF symbol table entries for local symbols have been removed
   - Debugging information has been removed from the image file

--------- End of File Header -------

---------- Optional Header ----------
Magic:                       267  (PE32 executable)
Linker Version:              2.56
Size of Code:                176128
Size of Initialized Data:    218112
Size of Uninitialized Data: 4608
Address of Entry Point:      4096
Base of Code:                4096
Base of Data:                180224
Image Base:                  4194304
Section Alignment:           4096
File Alignment:              512
OS Version:                  4.0
Major Image Version:         1
Minor Image Version:         0
Major Subsystem Version:     4
Minor Subsystem Version:     0
Size of Image:               237568
Size of Headers:             1024
Check Sum:                   235824
Required Windows Subsystem: Runs in the Windows character subsystem
DLL Characteristics:         0
Size of Stack Reserve:       2097152
Size of Stack Commit:        4096
Size of Heap Reserve:        1048576
Size of Heap Commit:         4096
Loader Flags:                0
Nbr of RVS and Sizes:        16
Data Directory:
   Export Table                 RVA: 0000000  Size: 0
   Import Table                 RVA: 0233472  Size: 1424
   Resource Table               RVA: 0000000  Size: 0
   Exception Table              RVA: 0000000  Size: 0
   Certificate Table            RVA: 0000000  Size: 0
   Base Relocation Table        RVA: 0000000  Size: 0
   Debug                        RVA: 0000000  Size: 0
   Architecture                 RVA: 0000000  Size: 0
   Global Ptr                   RVA: 0000000  Size: 0
   TLS Table                    RVA: 0000000  Size: 0
   Load Config Table            RVA: 0000000  Size: 0
   Bound Import                 RVA: 0000000  Size: 0
   Import Address Table         RVA: 0000000  Size: 0
   Delay Import Descriptor      RVA: 0000000  Size: 0
   COM+ Runtime Header          RVA: 0000000  Size: 0
   Reserved                     RVA: 0000000  Size: 0

-------- End of Optional Header --------

---- Section Table (5 entries) ----
```

```
Entry#0:
   Full Name:             .text
   Stored name:           .text
   Virtual Size:          176112
   Virtual Address:       4096
   Size of Raw Data:      176128
   Ptr to Raw Data:       1024
   Ptr to Relocations:    0
   Ptr to Line Numbers:   No COFF line numbers
   Number of Relocations: 0
   Number of Linenumbers: 0
   Characteristics:
      - Contains executable code
      - Contains initialized data
      - Can be executed as code
      - Can be read from
      - Cannot be written to

Entry#1:
   Full Name:             .data
   Stored name:           .data
   Virtual Size:          4508
   Virtual Address:       180224
   Size of Raw Data:      4608
   Ptr to Raw Data:       177152
   Ptr to Relocations:    0
   Ptr to Line Numbers:   No COFF line numbers
   Number of Relocations: 0
   Number of Linenumbers: 0
   Characteristics:
      - Contains initialized data
      - Can be read from
      - Can be written to

Entry#2:
   Full Name:             .rdata
   Stored name:           .rdata
   Virtual Size:          35484
   Virtual Address:       188416
   Size of Raw Data:      35840
   Ptr to Raw Data:       181760
   Ptr to Relocations:    0
   Ptr to Line Numbers:   No COFF line numbers
   Number of Relocations: 0
   Number of Linenumbers: 0
   Characteristics:
      - Contains initialized data
      - Can be read from
      - Can be written to

Entry#3:
   Full Name:             .bss
   Stored name:           .bss
   Virtual Size:          4464
   Virtual Address:       225280
```

```
   Size of Raw Data:      0
   Ptr to Raw Data:       0
   Ptr to Relocations:    0
   Ptr to Line Numbers:   No COFF line numbers
   Number of Relocations: 0
   Number of Linenumbers: 0
   Characteristics:
      - Contains uninitialized data
      - Can be read from
      - Can be written to

Entry#4:
   Full Name:             .idata
   Stored name:           .idata
   Virtual Size:          1424
   Virtual Address:       233472
   Size of Raw Data:      1536
   Ptr to Raw Data:       217600
   Ptr to Relocations:    0
   Ptr to Line Numbers:   No COFF line numbers
   Number of Relocations: 0
   Number of Linenumbers: 0
   Characteristics:
      - Contains initialized data
      - Can be read from
      - Can be written to

-------- End of Section Table --------

No Symbol Table to display

No String Table to display

No Export Table to display

---- Import Table ----
Import Directory Table:

Entry#0:
   RVA of Lookup Table:            233536
   Time Date Stamp:                Wed Dec 31 18:00:00 1969
   Start Index of Forwarder Chain: 0
   DLL Name:                       cygwin1.dll
   RVA of Import Lookup Table:     0x39124
   Import Lookup Table Contents:
      __errno
      __main
      _ctype_
      abort
      atoi
      calloc
      cygwin_internal
      dll_crt0__FP11per_process
      fclose
      fdopen
```

```
      fflush
      fileno
      fopen
      fprintf
      fread
      free
      fseek
      ftell
      fwrite
      getc
      malloc
      memchr
      memcpy
      memmove
      memset
      printf
      pthread_mutex_lock
      pthread_mutex_unlock
      pthread_once
      realloc
      setlocale
      setvbuf
      sprintf
      sscanf
      strcat
      strcmp
      strcoll
      strcpy
      strdup
      strftime
      strlen
      strncpy
      strtod
      strtol
      strtoll
      strtoul
      strtoull
      strxfrm
      ungetc

Entry#1:
   RVA of Lookup Table:            233740
   Time Date Stamp:                Wed Dec 31 18:00:00 1969
   Start Index of Forwarder Chain: 0
   DLL Name:                       KERNEL32.dll
   RVA of Import Lookup Table:     0x391f0
   Import Lookup Table Contents:
      AddAtomA
      FindAtomA
      GetAtomNameA
      GetModuleHandleA

---- End of Import Table ----

No Debug Table data to display
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 1

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     1     7   helloworld-gnu.exe
```

9.11 helloworld - Compiled and linked using Microsoft Visual Studio

```
FILE NAME: helloworld-vs.exe


---- File Map ----

 ADDRESS  DESCRIPTION

       0  +-------------------------------------------------------+
            DOS Header  [64 bytes]
      63  +-------------------------------------------------------+
      64  +-------------------------------------------------------+
            MS-DOS Stub  [57 bytes]
     120  +-------------------------------------------------------+
     121  +-------------------------------------------------------+
            (Contents not known)  [87 bytes]
     207  +-------------------------------------------------------+
     208  +-------------------------------------------------------+
            PE Signature  [4 bytes]
     211  +-------------------------------------------------------+
     212  +-------------------------------------------------------+
            File Header  [20 bytes]
     231  +-------------------------------------------------------+
     232  +-------------------------------------------------------+
            Optional Header  [224 bytes]
     455  +-------------------------------------------------------+
     456  +-------------------------------------------------------+
            Section Table  [120 bytes]
     575  +-------------------------------------------------------+
     576  +-------------------------------------------------------+
            (Contents not known)  [3520 bytes]
    4095  +-------------------------------------------------------+
    4096  +-------------------------------------------------------+
            .text section  [131072 bytes]
  135167  +-------------------------------------------------------+
  135168  +-------------------------------------------------------+
```

```
            .rdata section  [28672 bytes]
  163839  +-------------------------------------------------------+
          135168  +-----------------------------------------------+
                    (No additional details)  [24056 bytes]
          159223  +-----------------------------------------------+
          159224  +-----------------------------------------------+
                    Import Table (.idata section)  [1546 bytes]
          160769  +-----------------------------------------------+
          160770  +-----------------------------------------------+
                    (No additional details)  [3070 bytes]
          163839  +-----------------------------------------------+
  163840  +-------------------------------------------------------+
            .data section  [8192 bytes]
  172031  +-------------------------------------------------------+

---- End of File Map ----

NOTES ON FILE MAP CHANGES:
 - Changed "Import Table (.idata section)  [40 bytes]" at address
159224
   by inserting "Import Table (.idata section)  [1546 bytes]"



==== File Fact Summary ====

  - Image file in Windows NT portable executable (PE) format
  - Actual file size: 172032 bytes
  - Created on Fri Jan 28 13:40:30 2005
  - Target CPU: Intel 386 or later compatibles
  - Targeted for a 32-bit-word architecture
  - Designed for Windows Operating System version 4.0
  - Runs in the Windows character subsystem
  - Lists these table names in the data directory: Import, Import
Address
  - Contains no string table
  - Contains no symbol table
  - Imports functions from
    -- KERNEL32.dll (WinNT base API client)

==== End of File Fact Summary ====


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 292 bytes
exists starting at address 335872; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 40 bytes in size when actually it is
1626 bytes in size

**** End of Anomalies ****
```

```
No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 1

 List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     3     0   helloworld-vs.exe
```

9.12 jGRASP IDE 1.7.5 Executable Files

```
FILE NAME: jgrasp.exe



**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 1232
bytes exists starting at address 131072; this table often does not
appear in an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 160 bytes in size when actually it is
6545 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 7904 bytes in size when actually it
is 8192 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
FILE NAME: jGRASPjava.exe


**** Anomalies ****
```

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 268 bytes exists starting at address 45056; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 592 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 8600 bytes in size when actually it is 12288 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: winconfig.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 916 bytes exists starting at address 90112; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 160 bytes in size when actually it is 4819 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 8960 bytes in size when actually it is 12288 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

---------- Summary of File Security Analysis ----------

Total number of files submitted: 3

List of files containing anomalies (A), vulnerabilities (V) or risks (R)

A	V/R	Filename
4	0	jgrasp.exe
4	0	jGRASPjava.exe
4	0	winconfig.exe

9.13 Microsoft Windows XP kernel32.dll Dynamic Link Library

```
FILE NAME: \windows\system32\kernel32.dll


---- File Map ----

 ADDRESS  DESCRIPTION

       0  +-------------------------------------------------------+
            DOS Header  [64 bytes]
      63  +-------------------------------------------------------+
      64  +-------------------------------------------------------+
            MS-DOS Stub  [57 bytes]
     120  +-------------------------------------------------------+
     121  +-------------------------------------------------------+
            (Contents not known)  [127 bytes]
     247  +-------------------------------------------------------+
     248  +-------------------------------------------------------+
            PE Signature  [4 bytes]
     251  +-------------------------------------------------------+
     252  +-------------------------------------------------------+
            File Header  [20 bytes]
     271  +-------------------------------------------------------+
     272  +-------------------------------------------------------+
            Optional Header  [224 bytes]
     495  +-------------------------------------------------------+
     496  +-------------------------------------------------------+
            Section Table  [160 bytes]
     655  +-------------------------------------------------------+
     656  +-------------------------------------------------------+
            Bound Import Table  [28 bytes]
     683  +-------------------------------------------------------+
     684  +-------------------------------------------------------+
            (** Zero-filled region **)  [340 bytes]
    1023  +-------------------------------------------------------+
    1024  +-------------------------------------------------------+
            .text section  [477184 bytes]
  478207  +-------------------------------------------------------+
            1024  +-----------------------------------------------+
                    (No additional details)  [135488 bytes]
          136511  +-----------------------------------------------+
          136512  +-----------------------------------------------+
                    Export Table (.edata section)  [26300 bytes]
          162811  +-----------------------------------------------+
          162812  +-----------------------------------------------+
                    More of Export Table (.edata section)  [716 bytes]
          163527  +-----------------------------------------------+
          163528  +-----------------------------------------------+
                    (No additional details)  [303868 bytes]
          467395  +-----------------------------------------------+
          467396  +-----------------------------------------------+
```

```
                    Import Table (.idata section)  [10330 bytes]
          477725  +-------------------------------------------------+
          477726  +-------------------------------------------------+
                    (No additional details)  [2 bytes]
          477727  +-------------------------------------------------+
          477728  +-------------------------------------------------+
                    Debug Table (.debug section)  [56 bytes]
          477783  +-------------------------------------------------+
          477784  +-------------------------------------------------+
                    (No additional details)  [424 bytes]
          478207  +-------------------------------------------------+
  478208  +---------------------------------------------------------+
            .data section  [9216 bytes]
  487423  +---------------------------------------------------------+
          478208  +-------------------------------------------------+
                    (No additional details)  [1704 bytes]
          479911  +-------------------------------------------------+
          479912  +-------------------------------------------------+
                    Load Config Table  [64 bytes]
          479975  +-------------------------------------------------+
          479976  +-------------------------------------------------+
                    (No additional details)  [7448 bytes]
          487423  +-------------------------------------------------+
  487424  +---------------------------------------------------------+
            .rsrc section  [417792 bytes]
  905215  +---------------------------------------------------------+
  905216  +---------------------------------------------------------+
            .reloc section  [21504 bytes]
  926719  +---------------------------------------------------------+

---- End of File Map ----

NOTES ON FILE MAP CHANGES:
 - Changed "Export Table (.edata section)  [27016 bytes]" at address
136512
   by inserting "Export Table (.edata section)  [26300 bytes]"
 - Changed "Import Table (.idata section)  [40 bytes]" at address
467396
   by inserting "Import Table (.idata section)  [10330 bytes]"
 - Changed "Debug Table (.debug section)  [56 bytes]" at address
477728
   by inserting "Debug Table (.debug section)  [56 bytes]"
 - Changed "Resource Table (.rsrc section)  [417496 bytes]" at address
487424
   by inserting ".rsrc section  [417792 bytes]"
 - Changed "Relocation Table (.reloc section)  [21264 bytes]" at
address 905216
   by inserting ".reloc section  [21504 bytes]"
 - Changed "(Contents not known)  [340 bytes]" at address 684
   by inserting "(** Zero-filled region **)  [340 bytes]"



==== File Fact Summary ====
```

```
  - Image file in Windows NT portable executable (PE) format
  - Actual file size: 926720 bytes
  - Created on Sat Aug 18 00:33:02 2001
  - Target CPU: Intel 386 or later compatibles
  - Targeted for a 32-bit-word architecture
  - Image file is a dynamic link library (DLL)
  - Designed for Windows Operating System version 5.1
  - Runs in the Windows character subsystem
  - Lists these table names in the data directory: Export, Import,
Resource, Relocation, Debug, Load Config, Bound Import, Import Address
  - Contains no string table
  - Contains no symbol table
  - Exports functions using the file name KERNEL32.dll
  - Imports functions from
    -- ntdll.dll (NT layer)
  - Contains a large area of 303868 bytes starting at address 163528
which may indicate a group of compressed files

==== End of File Fact Summary ====


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 1544
bytes exists starting at address 1024; this table often does not
appear in an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 40 bytes in size when actually it is
10330 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 417496 bytes in size when actually
it is 417792 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 21264 bytes in size when actually
it is 21504 bytes in size

**** End of Anomalies ****



!!!! Security Vulnerabilities and Risks!!!!

  - Uses 2 standard C functions susceptible to buffer overflow
attacks: sprintf (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!



---------- DOS Header ----------
DOS signature:                    MZ
Bytes on last page of file:       144
Pages in file:                    3
```

```
Relocations:                        0
Size of header in paragraphs:       4
Minimum extra paragraphs:           0
Maximum extra paragraphs:           65535
Initial (relative) SS value:        0
Initial SP value:                   184
Checksum:                           0
Initial IP value:                   0
Initial (relative) CS value:        0
File address of relocation table:   64
Overlay number:                     0
Reserved words:                     0  0  0  0
OEM identifier:                     0
OEM information:                    0
Reserved words:                     0  0
Offset to PE signature and header: 248

-------- End of DOS Header --------

---------- File Header ----------
Target CPU:               Intel 386 or later compatibles
Number of Sections:       4
Time Date Stamp:          Sat Aug 18 00:33:02 2001
Ptr to Symbol Table:      0
Number of Symbols:        0
Size of Optional Header: 224
Characteristics:
   - This is an executable image
   - COFF line numbers have been removed
   - COFF symbol table entries for local symbols have been removed
   - Software is targeted for a 32-bit-word architecture
   - This image file is a dynamic link library (DLL)

--------- End of File Header -------

---------- Optional Header ----------
Magic:                       267  (PE32 executable)
Linker Version:              7.0
Size of Code:                477184
Size of Initialized Data:    450048
Size of Uninitialized Data: 0
Address of Entry Point:      107073
Base of Code:                4096
Base of Data:                462848
Image Base:                  2011561984
Section Alignment:           4096
File Alignment:              512
OS Version:                  5.1
Major Image Version:         5
Minor Image Version:         1
Major Subsystem Version:     4
Minor Subsystem Version:     0
Size of Image:               937984
Size of Headers:             1024
Check Sum:                   952210
```

```
Required Windows Subsystem: Runs in the Windows character subsystem
DLL Characteristics:        0
Size of Stack Reserve:      262144
Size of Stack Commit:       4096
Size of Heap Reserve:       1048576
Size of Heap Commit:        4096
Loader Flags:               0
Nbr of RVS and Sizes:       16
Data Directory:
   Export Table                 RVA: 0139584  Size: 27016
   Import Table                 RVA: 0470468  Size: 40
   Resource Table               RVA: 0495616  Size: 417496
   Exception Table              RVA: 0000000  Size: 0
   Certificate Table            RVA: 0000000  Size: 0
   Base Relocation Table        RVA: 0913408  Size: 21264
   Debug                        RVA: 0480800  Size: 56
   Architecture                 RVA: 0000000  Size: 0
   Global Ptr                   RVA: 0000000  Size: 0
   TLS Table                    RVA: 0000000  Size: 0
   Load Config Table            RVA: 0485032  Size: 64
   Bound Import                 RVA: 0656  Size: 28
   Import Address Table         RVA: 04096  Size: 1544
   Delay Import Descriptor      RVA: 0000000  Size: 0
   COM+ Runtime Header          RVA: 0000000  Size: 0
   Reserved                     RVA: 0000000  Size: 0

-------- End of Optional Header --------

---- Section Table (4 entries) ----

Entry#0:
   Full Name:             .text
   Stored name:           .text
   Virtual Size:          476760
   Virtual Address:       4096
   Size of Raw Data:      477184
   Ptr to Raw Data:       1024
   Ptr to Relocations:    0
   Ptr to Line Numbers:   No COFF line numbers
   Number of Relocations: 0
   Number of Linenumbers: 0
   Characteristics:
      - Contains executable code
      - Can be executed as code
      - Can be read from
      - Cannot be written to

Entry#1:
   Full Name:             .data
   Stored name:           .data
   Virtual Size:          10442
   Virtual Address:       483328
   Size of Raw Data:      9216
   Ptr to Raw Data:       478208
   Ptr to Relocations:    0
```

```
   Ptr to Line Numbers:   No COFF line numbers
   Number of Relocations: 0
   Number of Linenumbers: 0
   Characteristics:
      - Contains initialized data
      - Can be read from
      - Can be written to

Entry#2:
   Full Name:             .rsrc
   Stored name:           .rsrc
   Virtual Size:          417496
   Virtual Address:       495616
   Size of Raw Data:      417792
   Ptr to Raw Data:       487424
   Ptr to Relocations:    0
   Ptr to Line Numbers:   No COFF line numbers
   Number of Relocations: 0
   Number of Linenumbers: 0
   Characteristics:
      - Contains initialized data
      - Can be read from
      - Cannot be written to

Entry#3:
   Full Name:             .reloc
   Stored name:           .reloc
   Virtual Size:          21264
   Virtual Address:       913408
   Size of Raw Data:      21504
   Ptr to Raw Data:       905216
   Ptr to Relocations:    0
   Ptr to Line Numbers:   No COFF line numbers
   Number of Relocations: 0
   Number of Linenumbers: 0
   Characteristics:
      - Contains initialized data
      - Can be discarded as needed
      - Can be read from
      - Cannot be written to

-------- End of Section Table --------

No Symbol Table to display

No String Table to display

---- Export Table ----

Directory Table:
   Characteristics:             0
   Time Date Stamp:             Fri Aug 17 22:24:08 2001
   Major Version:               0
   Major Version:               0
   Name of DLL:                 KERNEL32.dll
```

```
   Starting Ordinal Number:      1
   Number of Function Addresses: 928
   Number of Names:              928
   Location of Address Table:    0x22168
   Location of Name Ptr Table:   0x22fe8
   Location of Ordinal Table:    0x23e68

Export Name Table (928 names):
   Entry#1: ActivateActCtx
   Entry#2: AddAtomA
   Entry#3: AddAtomW
   . . .
   Entry#926: lstrlen
   Entry#927: lstrlenA
   Entry#928: lstrlenW

Forwarder Name Table (0 names):

---- End of Export Table ----

---- Import Table ----
Import Directory Table:

Entry#0:
   RVA of Lookup Table:            470518
   Time Date Stamp:                Wed Dec 31 17:59:59 1969
   Start Index of Forwarder Chain: 4294967295
   DLL Name:                       ntdll.dll
   RVA of Import Lookup Table:     0x1000
   Import Lookup Table Contents:
      _wcsnicmp
      NtFsControlFile
      NtCreateFile
      . . .
      NtQueryDefaultLocale
      _strlwr
      RtlUnwind

---- End of Import Table ----


---- Debug Table ----
Debug Directory Table (2 entries):

Entry#0:
   Characteristics:            0
   Time Date Stamp:            Fri Aug 17 22:24:08 2001
   Format Major Version:       0
   Format Minor Version:       0
   Debug Type:                 CodeView debug information
   Size of Debug Data:         29
   Image Address of Raw Data:  470180
   File Pointer to Raw Data:   467108

Entry#1:
```

```
   Characteristics:            0
   Time Date Stamp:            Fri Aug 17 22:24:08 2001
   Format Major Version:       0
   Format Minor Version:       0
   Debug Type:                 Unknown information (10)
   Size of Debug Data:         0
   Image Address of Raw Data:  0
   File Pointer to Raw Data:   0

---- End of Debug Table ----
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 1

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     5     2   \windows\system32\kernel32.dll
```

10. APPENDIX D – TEST RESULTS FROM ANALYZING EXECUTABLE INSTALLATION FILES

10.1 Adobe Acrobat Reader 5.0 Installation File

```
FILE NAME: rp500enu.exe


==== File Fact Summary ====

  - Image file in Windows NT portable executable (PE) format
  - Actual file size: 10236296 bytes
  - Created on Thu Mar 26 08:31:20 1998
  - Target CPU: Intel 386 or later compatibles
  - Targeted for a 32-bit-word architecture
  - Designed for Windows Operating System version 4.0
  - Runs in the Windows GUI subsystem
  - Lists these table names in the data directory: Import, Resource,
Certificate, Import Address
  - Contains no string table
  - Contains no symbol table
  - Imports functions from
    -- KERNEL32.dll (WinNT base API client)
    -- USER32.dll (Windows NT user API client)
    -- GDI32.dll (Graphics device interface client)
    -- COMCTL32.dll (Custom controls library)
    -- ADVAPI32.dll (Routines to read and modify the Windows NT
registry)
    -- SHELL32.dll (Windows shell common)
    -- LZ32.dll (LZ expand/compress API)
  - Contains a large area of 10078496 bytes starting at address 130560
which may indicate a group of compressed files

==== End of File Fact Summary ====


**** Anomalies ****

  - The file indicates an import address table consisting of 588 bytes
exists starting at address 85228; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 160 bytes in size when actually it is
3174 bytes in size
```

```
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 41640 bytes in size when actually it
is 41984 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 1

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     3     0   rp500enu.exe
```

10.2 Earthlink TotalAccess 5.0 Installation File

```
FILE NAME: TA2005_1.exe



==== File Fact Summary ====

  - Image file in Windows NT portable executable (PE) format
  - Actual file size: 427096 bytes
  - Created on Sat Jun 19 00:10:13 2004
  - Target CPU: Intel 386 or later compatibles
  - Targeted for a 32-bit-word architecture
  - Designed for Windows Operating System version 4.0
  - Runs in the Windows GUI subsystem
  - Lists these table names in the data directory: Import, Resource,
Certificate, Import Address
  - Contains no string table
  - Contains no symbol table
  - Imports functions from
    -- KERNEL32.dll (WinNT base API client)
    -- USER32.dll (Windows NT user API client)
    -- GDI32.dll (Graphics device interface client)
    -- comdlg32.dll (Common dialogs)
    -- WINSPOOL.DRV (Purpose unknown)
    -- ADVAPI32.dll (Routines to read and modify the Windows NT
registry)
    -- SHELL32.dll (Windows shell common)
    -- COMCTL32.dll (Custom controls library)
    -- SHLWAPI.dll (Purpose unknown)
    -- WININET.dll (Purpose unknown)
```

```
    -- OLEAUT32.dll (OLE 2.20 for Windows NT and Windows 95)
    -- OLEACC.dll (Purpose unknown)
  - Contains an unusual area of 212992 bytes starting at address
192512 which may indicate a group of compressed files

==== End of File Fact Summary ====


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 1324
bytes exists starting at address 147456; this table often does not
appear in an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 260 bytes in size when actually it is
7206 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 227664 bytes in size when actually
it is 229376 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 1

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     4     0   TA2005_1.exe
```

10.3 Java 1.4.2 Installation Files

```
FILE NAME: j2re-1_4_2_01-windows-i586-iftw.exe


==== File Fact Summary ====

  - Image file in Windows NT portable executable (PE) format
  - Actual file size: 1418120 bytes
  - Created on Tue Aug 19 21:32:03 2003
  - Target CPU: Intel 386 or later compatibles
```

```
  - Targeted for a 32-bit-word architecture
  - Designed for Windows Operating System version 4.0
  - Runs in the Windows GUI subsystem
  - Lists these table names in the data directory: Import, Resource,
Certificate, Import Address
  - Contains no string table
  - Contains no symbol table
  - Imports functions from
    -- KERNEL32.dll (WinNT base API client)
    -- USER32.dll (Windows NT user API client)
    -- WININET.dll (Purpose unknown)
    -- MSVCRT.dll (C runtime library)

==== End of File Fact Summary ====


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 208 bytes
exists starting at address 8192; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 100 bytes in size when actually it is
972 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 1394544 bytes in size when actually
it is 1396736 bytes in size

**** End of Anomalies ****



!!!! Security Vulnerabilities and Risks!!!!

  - Uses one standard C function susceptible to buffer overflow
attack: sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!
```

```
FILE NAME: j2sdk-1_4_2_01-windows-i586-iftw.exe


==== File Fact Summary ====

  - Image file in Windows NT portable executable (PE) format
  - Actual file size: 364544 bytes
  - Created on Tue Aug 19 21:34:40 2003
  - Target CPU: Intel 386 or later compatibles
  - Targeted for a 32-bit-word architecture
  - Designed for Windows Operating System version 4.0
```

```
  - Runs in the Windows GUI subsystem
  - Lists these table names in the data directory: Import, Resource,
Import Address
  - Contains no string table
  - Contains no symbol table
  - Imports functions from
    -- KERNEL32.dll (WinNT base API client)
    -- USER32.dll (Windows NT user API client)
    -- WININET.dll (Purpose unknown)
    -- MSVCRT.dll (C runtime library)

==== End of File Fact Summary ====


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 208 bytes
exists starting at address 8192; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 100 bytes in size when actually it is
972 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 345680 bytes in size when actually
it is 348160 bytes in size

**** End of Anomalies ****



!!!! Security Vulnerabilities and Risks!!!!

  - Uses one standard C function susceptible to buffer overflow
attack: sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!
```

```
FILE NAME: jdk-1_5_0-beta2-windows-i586.exe


==== File Fact Summary ====

  - Image file in Windows NT portable executable (PE) format
  - Actual file size: 45836078 bytes
  - Created on Wed Feb  4 10:43:10 2004
  - Target CPU: Intel 386 or later compatibles
  - Targeted for a 32-bit-word architecture
  - Designed for Windows Operating System version 4.0
  - Runs in the Windows GUI subsystem
```

```
  - Lists these table names in the data directory: Import, Resource,
Import Address
  - Contains no string table
  - Contains no symbol table
  - Imports functions from
    -- VERSION.dll (Version checking and file installation)
    -- SHELL32.dll (Windows shell common)
    -- COMCTL32.dll (Custom controls library)
    -- KERNEL32.dll (WinNT base API client)
    -- USER32.dll (Windows NT user API client)
    -- GDI32.dll (Graphics device interface client)
    -- ADVAPI32.dll (Routines to read and modify the Windows NT
registry)
    -- ole32.dll (OLE 2.1 16/32 interoperability library)
    -- OLEAUT32.dll (OLE 2.20 for Windows NT and Windows 95)
  - Contains an unknown region of 45610798 bytes starting at address
225280 which may indicate a group of compressed files

==== End of File Fact Summary ====


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 1084
bytes exists starting at address 143360; this table often does not
appear in an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 200 bytes in size when actually it is
4682 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 41696 bytes in size when actually it
is 45056 bytes in size

**** End of Anomalies ****



!!!! Security Vulnerabilities and Risks!!!!

  - Contains 4492 bytes of unused zero-filled space that could be used
to store malicious code or data

!!!! End of Security Vulnerabilities and Risks!!!!

_____________________________________________________________________

---------- Summary of File Security Analysis ----------

Total number of files submitted: 3

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)
```

```
A   V/R   Filename
-   ---   --------
4     1   j2re-1_4_2_01-windows-i586-iftw.exe
4     1   j2sdk-1_4_2_01-windows-i586-iftw.exe
4     1   jdk-1_5_0-beta2-windows-i586.exe
```

10.4 jGRASP 1.7.5 Installation File

```
FILE NAME: jgrasp175.exe


---- File Map ----

 ADDRESS  DESCRIPTION

       0  +-------------------------------------------------------+
            DOS Header  [64 bytes]
      63  +-------------------------------------------------------+
      64  +-------------------------------------------------------+
            MS-DOS Stub  [57 bytes]
     120  +-------------------------------------------------------+
     121  +-------------------------------------------------------+
            (Contents not known)  [87 bytes]
     207  +-------------------------------------------------------+
     208  +-------------------------------------------------------+
            PE Signature  [4 bytes]
     211  +-------------------------------------------------------+
     212  +-------------------------------------------------------+
            File Header  [20 bytes]
     231  +-------------------------------------------------------+
     232  +-------------------------------------------------------+
            Optional Header  [224 bytes]
     455  +-------------------------------------------------------+
     456  +-------------------------------------------------------+
            Section Table  [200 bytes]
     655  +-------------------------------------------------------+
     656  +-------------------------------------------------------+
            (** Zero-filled region **)  [368 bytes]
    1023  +-------------------------------------------------------+
    1024  +-------------------------------------------------------+
            .text section  [24064 bytes]
   25087  +-------------------------------------------------------+
   25088  +-------------------------------------------------------+
            .rdata section  [4608 bytes]
   29695  +-------------------------------------------------------+
           25088  +-----------------------------------------------+
                    (No additional details)  [816 bytes]
           25903  +-----------------------------------------------+
           25904  +-----------------------------------------------+
                    Import Table (.idata section)  [3637 bytes]
```

```
          29540  +-------------------------------------------------+
          29541  +-------------------------------------------------+
                   (No additional details)  [155 bytes]
          29695  +-------------------------------------------------+
   29696  +-------------------------------------------------------+
            .data section  [1024 bytes]
   30719  +-------------------------------------------------------+
   30720  +-------------------------------------------------------+
            .rsrc section  [12800 bytes]
   43519  +-------------------------------------------------------+
   43520  +-------------------------------------------------------+
            More of Resource Table (.rsrc section)  [3584 bytes]
   47103  +-------------------------------------------------------+
   47104  +-------------------------------------------------------+
            (Contents not known)  [2053221 bytes]
 2100324  +-------------------------------------------------------+

---- End of File Map ----

NOTES ON FILE MAP CHANGES:
 - Changed "Import Table (.idata section)  [180 bytes]" at address
25904
   by inserting "Import Table (.idata section)  [3637 bytes]"
 - Changed "Resource Table (.rsrc section)  [16384 bytes]" at address
30720
   by inserting ".rsrc section  [12800 bytes]"
 - Changed "(Contents not known)  [368 bytes]" at address 656
   by inserting "(** Zero-filled region **)  [368 bytes]"



==== File Fact Summary ====

  - Image file in Windows NT portable executable (PE) format
  - Actual file size: 2100325 bytes
  - Created on Sat Feb  7 11:26:28 2004
  - Target CPU: Intel 386 or later compatibles
  - Targeted for a 32-bit-word architecture
  - Designed for Windows Operating System version 4.0
  - Runs in the Windows GUI subsystem
  - Lists these table names in the data directory: Import, Resource,
Import Address
  - Contains no string table
  - Contains no symbol table
  - Imports functions from
    -- COMCTL32.dll (Custom controls library)
    -- KERNEL32.dll (WinNT base API client)
    -- USER32.dll (Windows NT user API client)
    -- GDI32.dll (Graphics device interface client)
    -- ADVAPI32.dll (Routines to read and modify the Windows NT
registry)
    -- SHELL32.dll (Windows shell common)
    -- ole32.dll (OLE 2.1 16/32 interoperability library)
    -- VERSION.dll (Version checking and file installation)
```

```
  - Contains an unknown region of 2053221 bytes starting at address
47104 which may indicate a group of compressed files

==== End of File Fact Summary ====


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 672 bytes
exists starting at address 25088; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 180 bytes in size when actually it is
3637 bytes in size
  - A section entry named .ndata appears in the section table, but the
table doesn't contain the location of the 61440 bytes for that section

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 1

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     4     0   jgrasp175.exe
```

10.5 jGRASP 1.7.5 (with JRE) Installation File

```
FILE NAME: jgraspjre175.exe



---- File Map ----

 ADDRESS  DESCRIPTION

       0  +-------------------------------------------------------+
            DOS Header  [64 bytes]
      63  +-------------------------------------------------------+
      64  +-------------------------------------------------------+
```

```
          MS-DOS Stub  [57 bytes]
      120 +----------------------------------------------------------+
      121 +----------------------------------------------------------+
          (Contents not known)  [87 bytes]
      207 +----------------------------------------------------------+
      208 +----------------------------------------------------------+
          PE Signature  [4 bytes]
      211 +----------------------------------------------------------+
      212 +----------------------------------------------------------+
          File Header  [20 bytes]
      231 +----------------------------------------------------------+
      232 +----------------------------------------------------------+
          Optional Header  [224 bytes]
      455 +----------------------------------------------------------+
      456 +----------------------------------------------------------+
          Section Table  [200 bytes]
      655 +----------------------------------------------------------+
      656 +----------------------------------------------------------+
          (** Zero-filled region **)  [368 bytes]
     1023 +----------------------------------------------------------+
     1024 +----------------------------------------------------------+
          .text section  [24064 bytes]
    25087 +----------------------------------------------------------+
    25088 +----------------------------------------------------------+
          .rdata section  [4608 bytes]
    29695 +----------------------------------------------------------+
           25088  +-------------------------------------------------+
                    (No additional details)  [816 bytes]
           25903  +-------------------------------------------------+
           25904  +-------------------------------------------------+
                    Import Table (.idata section)  [3637 bytes]
           29540  +-------------------------------------------------+
           29541  +-------------------------------------------------+
                    (No additional details)  [155 bytes]
           29695  +-------------------------------------------------+
    29696 +----------------------------------------------------------+
          .data section  [1024 bytes]
    30719 +----------------------------------------------------------+
    30720 +----------------------------------------------------------+
          .rsrc section  [12800 bytes]
    43519 +----------------------------------------------------------+
    43520 +----------------------------------------------------------+
          More of Resource Table (.rsrc section)  [3584 bytes]
    47103 +----------------------------------------------------------+
    47104 +----------------------------------------------------------+
          (Contents not known)  [17384875 bytes]
17431978 +----------------------------------------------------------+

---- End of File Map ----

NOTES ON FILE MAP CHANGES:
 - Changed "Import Table (.idata section)  [180 bytes]" at address
25904
   by inserting "Import Table (.idata section)  [3637 bytes]"
```

```
 - Changed "Resource Table (.rsrc section)  [16384 bytes]" at address
30720
   by inserting ".rsrc section  [12800 bytes]"
 - Changed "(Contents not known)  [368 bytes]" at address 656
   by inserting "(** Zero-filled region **)  [368 bytes]"


==== File Fact Summary ====

  - Image file in Windows NT portable executable (PE) format
  - Actual file size: 17431979 bytes
  - Created on Sat Feb  7 11:26:28 2004
  - Target CPU: Intel 386 or later compatibles
  - Targeted for a 32-bit-word architecture
  - Designed for Windows Operating System version 4.0
  - Runs in the Windows GUI subsystem
  - Lists these table names in the data directory: Import, Resource,
Import Address
  - Contains no string table
  - Contains no symbol table
  - Imports functions from
    -- COMCTL32.dll (Custom controls library)
    -- KERNEL32.dll (WinNT base API client)
    -- USER32.dll (Windows NT user API client)
    -- GDI32.dll (Graphics device interface client)
    -- ADVAPI32.dll (Routines to read and modify the Windows NT
registry)
    -- SHELL32.dll (Windows shell common)
    -- ole32.dll (OLE 2.1 16/32 interoperability library)
    -- VERSION.dll (Version checking and file installation)
  - Contains an unknown region of 17384875 bytes starting at address
47104 which may indicate a group of compressed files

==== End of File Fact Summary ====


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 672 bytes
exists starting at address 25088; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 180 bytes in size when actually it is
3637 bytes in size
  - A section entry named .ndata appears in the section table, but the
table doesn't contain the location of the 61440 bytes for that section

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 1

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     4     0   jgraspjre175.exe
```

10.6 Windows Media Player 9.0 Installation File

```
FILE NAME: MPSetupXP-9.exe


---- File Map ----

 ADDRESS  DESCRIPTION

       0  +-------------------------------------------------------+
            DOS Header  [64 bytes]
      63  +-------------------------------------------------------+
      64  +-------------------------------------------------------+
            MS-DOS Stub  [57 bytes]
     120  +-------------------------------------------------------+
     121  +-------------------------------------------------------+
            (Contents not known)  [79 bytes]
     199  +-------------------------------------------------------+
     200  +-------------------------------------------------------+
            PE Signature  [4 bytes]
     203  +-------------------------------------------------------+
     204  +-------------------------------------------------------+
            File Header  [20 bytes]
     223  +-------------------------------------------------------+
     224  +-------------------------------------------------------+
            Optional Header  [224 bytes]
     447  +-------------------------------------------------------+
     448  +-------------------------------------------------------+
            Section Table  [120 bytes]
     567  +-------------------------------------------------------+
     568  +-------------------------------------------------------+
            (** Zero-filled region **)  [456 bytes]
    1023  +-------------------------------------------------------+
    1024  +-------------------------------------------------------+
            .text section  [34816 bytes]
   35839  +-------------------------------------------------------+
            1024  +-----------------------------------------------+
```

```
                    (No additional details)  [528 bytes]
             1551  +-----------------------------------------------+
             1552  +-----------------------------------------------+
                     Debug Table (.debug section)  [28 bytes]
             1579  +-----------------------------------------------+
             1580  +-----------------------------------------------+
                    (No additional details)  [30852 bytes]
            32431  +-----------------------------------------------+
            32432  +-----------------------------------------------+
                     Import Table (.idata section)  [2834 bytes]
            35265  +-----------------------------------------------+
            35266  +-----------------------------------------------+
                    (No additional details)  [574 bytes]
            35839  +-----------------------------------------------+
   35840  +-------------------------------------------------------+
            .data section  [1024 bytes]
   36863  +-------------------------------------------------------+
   36864  +-------------------------------------------------------+
            .rsrc section  [10092032 bytes]
10128895  +-------------------------------------------------------+
            36864  +-----------------------------------------------+
                    (No additional details)  [10079744 bytes]
         10116607  +-----------------------------------------------+
         10116608  +-----------------------------------------------+
                     Certificate Table  [6792 bytes]
         10123399  +-----------------------------------------------+
         10123400  +-----------------------------------------------+
                    (No additional details)  [12288 bytes]
         10135687  +-----------------------------------------------+

---- End of File Map ----

NOTES ON FILE MAP CHANGES:
 - Changed "Import Table (.idata section)  [140 bytes]" at address
32432
   by inserting "Import Table (.idata section)  [2834 bytes]"
 - Changed "Debug Table (.debug section)  [28 bytes]" at address 1552
   by inserting "Debug Table (.debug section)  [28 bytes]"
 - Changed "Resource Table (.rsrc section)  [10092012 bytes]" at
address 36864
   by inserting ".rsrc section  [10092032 bytes]"
 - Changed "(Contents not known)  [456 bytes]" at address 568
   by inserting "(** Zero-filled region **)  [456 bytes]"


==== File Fact Summary ====

  - Image file in Windows NT portable executable (PE) format
  - Actual file size: 10135688 bytes
  - Created on Fri Aug 17 20:42:57 2001
  - Target CPU: Intel 386 or later compatibles
  - Targeted for a 32-bit-word architecture
  - Designed for Windows Operating System version 5.1
  - Runs in the Windows GUI subsystem
```

```
  - Lists these table names in the data directory: Import, Resource,
Certificate, Debug, Import Address
  - Contains no string table
  - Contains no symbol table
  - Imports functions from
    -- ADVAPI32.dll (Routines to read and modify the Windows NT
registry)
    -- KERNEL32.dll (WinNT base API client)
    -- GDI32.dll (Graphics device interface client)
    -- USER32.dll (Windows NT user API client)
    -- COMCTL32.dll (Custom controls library)
    -- VERSION.dll (Version checking and file installation)
  - Contains an unusual area of 10079744 bytes starting at address
36864 which may indicate a group of compressed files

==== End of File Fact Summary ====


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 528 bytes
exists starting at address 1024; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 140 bytes in size when actually it is
2834 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 10092012 bytes in size when actually
it is 10092032 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 1

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     4     0   MPSetupXP-9.exe
```

10.7 Real One Player (ME/XP) Installation File

FILE NAME: RealPlayer10-5GOLD_bb.exe

---- File Map ----

```
 ADDRESS  DESCRIPTION

       0  +-------------------------------------------------------+
            DOS Header  [64 bytes]
      63  +-------------------------------------------------------+
      64  +-------------------------------------------------------+
            MS-DOS Stub  [57 bytes]
     120  +-------------------------------------------------------+
     121  +-------------------------------------------------------+
            (Contents not known)  [111 bytes]
     231  +-------------------------------------------------------+
     232  +-------------------------------------------------------+
            PE Signature  [4 bytes]
     235  +-------------------------------------------------------+
     236  +-------------------------------------------------------+
            File Header  [20 bytes]
     255  +-------------------------------------------------------+
     256  +-------------------------------------------------------+
            Optional Header  [224 bytes]
     479  +-------------------------------------------------------+
     480  +-------------------------------------------------------+
            Section Table  [160 bytes]
     639  +-------------------------------------------------------+
     640  +-------------------------------------------------------+
            (** Zero-filled region **)  [3456 bytes]
    4095  +-------------------------------------------------------+
    4096  +-------------------------------------------------------+
            .text section  [98304 bytes]
  102399  +-------------------------------------------------------+
  102400  +-------------------------------------------------------+
            .rdata section  [12288 bytes]
  114687  +-------------------------------------------------------+
          102400  +-----------------------------------------------+
                    (No additional details)  [688 bytes]
          103087  +-----------------------------------------------+
          103088  +-----------------------------------------------+
                    Debug Table (.debug section)  [28 bytes]
          103115  +-----------------------------------------------+
          103116  +-----------------------------------------------+
                    (No additional details)  [6556 bytes]
          109671  +-----------------------------------------------+
          109672  +-----------------------------------------------+
                    Import Table (.idata section)  [2665 bytes]
          112336  +-----------------------------------------------+
          112337  +-----------------------------------------------+
                    (No additional details)  [2351 bytes]
          114687  +-----------------------------------------------+
  114688  +-------------------------------------------------------+
            .data section  [20480 bytes]
```

```
  135167  +----------------------------------------------------------+
  135168  +----------------------------------------------------------+
            .rsrc section  [10354688 bytes]
10489855  +----------------------------------------------------------+
          135168  +-------------------------------------------------+
                    (No additional details)  [10346544 bytes]
        10481711  +-------------------------------------------------+
        10481712  +-------------------------------------------------+
                    Certificate Table  [5616 bytes]
        10487327  +-------------------------------------------------+
        10487328  +-------------------------------------------------+
                    (No additional details)  [8192 bytes]
        10495519  +-------------------------------------------------+

---- End of File Map ----

NOTES ON FILE MAP CHANGES:
 - Changed "Import Table (.idata section)  [140 bytes]" at address
109672
   by inserting "Import Table (.idata section)  [2665 bytes]"
 - Changed "Debug Table (.debug section)  [28 bytes]" at address
103088
   by inserting "Debug Table (.debug section)  [28 bytes]"
 - Changed "Resource Table (.rsrc section)  [10351928 bytes]" at
address 135168
   by inserting ".rsrc section  [10354688 bytes]"
 - Changed "(Contents not known)  [3456 bytes]" at address 640
   by inserting "(** Zero-filled region **)  [3456 bytes]"



==== File Fact Summary ====

  - Image file in Windows NT portable executable (PE) format
  - Actual file size: 10495520 bytes
  - Created on Tue Oct 19 18:51:27 2004
  - Target CPU: Intel 386 or later compatibles
  - Targeted for a 32-bit-word architecture
  - Designed for Windows Operating System version 4.0
  - Runs in the Windows GUI subsystem
  - Lists these table names in the data directory: Import, Resource,
Certificate, Debug, Import Address
  - Contains no string table
  - Contains no symbol table
  - Imports functions from
    -- KERNEL32.dll (WinNT base API client)
    -- USER32.dll (Windows NT user API client)
    -- ADVAPI32.dll (Routines to read and modify the Windows NT
registry)
    -- GDI32.dll (Graphics device interface client)
    -- COMCTL32.dll (Custom controls library)
    -- VERSION.dll (Version checking and file installation)
  - Contains an unusual area of 10346544 bytes starting at address
135168 which may indicate a group of compressed files
```

==== End of File Fact Summary ====

**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub contains some kind of information
  - The file indicates an import address table consisting of 676 bytes exists starting at address 102400; this table often does not appear in an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the Import Table (.idata section) is 140 bytes in size when actually it is 2665 bytes in size
  - The data directory table in the optional header states that the Resource Table (.rsrc section) is 10351928 bytes in size when actually it is 10354688 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

---------- Summary of File Security Analysis ----------

Total number of files submitted: 1

List of files containing anomalies (A), vulnerabilities (V) or risks (R)

A	V/R	Filename
4	0	RealPlayer10-5GOLD_bb.exe

11. APPENDIX E – TEST RESULTS FROM ANALYZING SOFTWARE DEVELOPMENT FILES

11.1 Dynamic Link Library (DLL) files for Borland CBuilder 5

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 20

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     3     0   bcbedit.dll
     2     0   bcbmm.dll
     2     0   borlndmm.dll
     1     0   brcide.dll
     6     0   cc3250.dll
     6     0   cc3250mt.dll
     1     0   comp32p.dll
     3     0   dcc50.dll
     2     0   delphimm.dll
     1     0   ilink32.dll
     4     0   imged32.dll
     1     0   ixxml50.dll
     2     0   lnkdfm50.dll
     1     0   rlink32.dll
     3     0   rw32core.dll
     1     0   typelibimport.dll
     2     0   vcltest3.dll
     1     0   xmlide.dll
     4     0   xprtfltr.dll
```

11.2 Executable (EXE) files for Borland CBuilder 5

```
---------- Summary of File Security Analysis ----------
```

```
Total number of files submitted: 22

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     2     0   bcb.exe
     3     0   bcc32.exe
     4     0   bpr2mak.exe
     5     0   brc32.exe
     5     0   brcc32.exe
     4     0   coff2omf.exe
     4     0   convert.exe
     3     0   cpp32.exe
     5     0   dcc32.exe
     3     0   grep.exe
     3     0   ilink32.exe
     4     0   imagedit.exe
     5     0   impdef.exe
     5     0   implib.exe
     3     0   instreg.exe
     5     0   make.exe
     5     0   tdump.exe
     3     0   tlib.exe
     3     0   touch.exe
     5     0   tregsvr.exe
```

11.3 Dynamic Link Library (DLL) files for Cygwin

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 56

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     2     2   cygbz2-1.dll
     6     9   cygcrypto-0.9.7.dll
     2     9   cygcrypto.dll
     2     8   cygdb-3.1.dll
     2     9   cygdb_cxx-3.1.dll
     2     4   cygform5.dll
     2     5   cygform6.dll
     2     6   cygform7.dll
     2     3   cyggdbm-3.dll
     2     3   cyggdbm-4.dll
     2     4   cyggdbm.dll
```

```
2      3   cyggdbm_compat-3.dll
2      3   cyggdbm_compat-4.dll
2      6   cyggettextlib-0-12-1.dll
2      0   cyggettextpo-0.dll
2      4   cyggettextsrc-0-12-1.dll
2      5   cyghistory4.dll
2      4   cyghistory5.dll
2      4   cygiconv-2.dll
2      6   cygintl-1.dll
2      6   cygintl-2.dll
2      0   cygjbig1.dll
2      3   cygjpeg-62.dll
2      3   cygjpeg6b.dll
2      2   cygmenu5.dll
2      3   cygmenu6.dll
2      4   cygmenu7.dll
2      4   cygminires.dll
3      5   cygncurses++5.dll
2      6   cygncurses++6.dll
2      9   cygncurses5.dll
2      9   cygncurses6.dll
2     11   cygncurses7.dll
2      0   cygpanel5.dll
2      1   cygpanel6.dll
2      2   cygpanel7.dll
2      1   cygpcre-0.dll
2      1   cygpcre.dll
2      3   cygpcreposix-0.dll
2      3   cygpcreposix.dll
2      7   cygperl5_8_0.dll
2      3   cygpng12.dll
2      6   cygpopt-0.dll
2      7   cygreadline4.dll
2      6   cygreadline5.dll
2      1   cygssl-0.9.7.dll
2      1   cygssl.dll
2      4   cygtiff3.dll
2      4   cygtiff4.dll
3     12   cygwin1.dll
2      5   cygz.dll
5      0   glut32.dll
2      1   mingwm10.dll
3      4   tcl84.dll
1      0   tclpip84.dll
3      6   tk84.dll
```

11.4 Executable (EXE) files for Cygwin

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 325
```

List of files containing anomalies (A), vulnerabilities (V) or risks (R)

A	V/R	Filename
1	5	a2p.exe
1	6	addftinfo.exe
1	8	addr2line.exe
1	8	ar.exe
1	10	as.exe
1	1	ascii.exe
1	8	awk.exe
1	0	banner.exe
1	0	basename.exe
1	9	bash.exe
1	4	bison.exe
1	4	bunzip2.exe
1	4	bzcat.exe
1	4	bzip2.exe
1	4	bzip2recover.exe
1	6	c++.exe
1	7	c++filt.exe
1	1	cal.exe
1	10	captoinfo.exe
1	1	cat.exe
1	5	chgrp.exe
1	5	chmod.exe
1	6	chown.exe
1	0	chroot.exe
1	3	cjpeg.exe
1	1	cksum.exe
1	2	clearn.exe
1	7	client.exe
1	2	cmp.exe
1	1	col.exe
1	2	colcrt.exe
1	1	colrm.exe
1	1	column.exe
1	2	comm.exe
1	3	conv.exe
1	9	cp.exe
1	8	cpio.exe
1	6	cpp.exe
1	0	crypt.exe
1	5	csplit.exe
1	2	cut.exe
1	11	cvs.exe
1	5	cygcheck.exe
1	4	cygpath.exe
1	7	cygrunsrv.exe
1	6	cygserver.exe
1	1	cygstart.exe
1	3	d2u.exe
1	2	date.exe

```
1      5   dd.exe
1      2   ddate.exe
1      8   df.exe
1      5   diff.exe
1      2   diff3.exe
1      7   dir.exe
1      5   dircolors.exe
1      1   dirname.exe
1      2   djpeg.exe
1      9   dlltool.exe
1      7   dllwrap.exe
1      3   dos2unix.exe
1      7   du.exe
1      1   dump.exe
1      9   dumper.exe
1      3   dumpgdbm.exe
1      0   echo.exe
1      0   env.exe
1      7   eqn.exe
1      1   expand.exe
1     10   expect.exe
1      3   expr.exe
1      2   factor.exe
1      0   false.exe
1      1   fax2ps.exe
1      0   fax2tiff.exe
1      7   file.exe
1      6   fileman-stat.exe
1      3   fileman.exe
1      5   find.exe
1      6   flex++.exe
1      6   flex.exe
1      1   fmt.exe
1      2   fold.exe
1     11   ftp.exe
1      4   funzip.exe
1      6   g++.exe
1      6   g77.exe
1      8   gawk.exe
1      6   gcc.exe
1      6   gcj.exe
1      5   gcjh.exe
1      6   gcov.exe
2     13   gdb.exe
1      0   getclip.exe
1      2   getfacl.exe
1      1   getopt.exe
1      1   gettext.exe
1      1   gif2tiff.exe
1      8   gij.exe
1     10   gprof.exe
1      4   grep.exe
1      5   grepjar.exe
1      9   grn.exe
1      8   grodvi.exe
```

```
1      8   groff.exe
1      8   grolbp.exe
1      8   grolj4.exe
1      8   grops.exe
1      8   grotty.exe
1      5   gunzip.exe
1      5   gzip.exe
1      1   head.exe
1      0   hostname.exe
1      6   hpftodit.exe
1      6   i686-pc-cygwin-c++.exe
1      6   i686-pc-cygwin-g++.exe
1      6   i686-pc-cygwin-gcc.exe
1      0   id.exe
1      6   indxbib.exe
1     10   info.exe
1      8   infocmp.exe
1      9   infokey.exe
1     10   infotocap.exe
2     13   insight.exe
1      6   install-info.exe
1      9   install.exe
1      6   jar.exe
1      0   jbgtopbm.exe
1      5   jcf-dump.exe
1      2   join.exe
1      3   jpegtran.exe
1      8   jv-convert.exe
1      4   jv-scan.exe
1      1   kill.exe
1     11   ld.exe
1     11   less.exe
1      0   lessecho.exe
1      3   lesskey.exe
1      6   lkbib.exe
1      8   ln.exe
1      3   loadgdbm.exe
1      2   locate.exe
1      3   logger.exe
1      7   login.exe
1      0   logname.exe
1      7   lookbib.exe
1      7   lpr.exe
1      7   ls.exe
1      8   m4.exe
1      7   make.exe
1     11   makeinfo.exe
1      9   man.exe
1      3   man2html.exe
1      2   mcookie.exe
1      2   md5sum.exe
1      6   mkdir.exe
1      4   mkfifo.exe
1      3   mkgroup.exe
1      4   mknod.exe
```

```
1      3   mkpasswd.exe
1      4   mkshortcut.exe
1      4   mktemp.exe
1      2   mount.exe
1     13   mutt.exe
1      9   mv.exe
1      0   namei.exe
1     13   ncftp.exe
1      8   ncftpbatch.exe
1     10   ncftpbookmarks.exe
1     10   ncftpget.exe
1     10   ncftpls.exe
1      9   ncftpput.exe
1      8   ncftpspooler.exe
1      1   ngettext.exe
1      0   nice.exe
1      5   nl.exe
1      8   nm.exe
1      8   objcopy.exe
1      8   objdump.exe
1      2   od.exe
1      8   openssl.exe
1      0   pal2rgb.exe
1      2   passwd.exe
1      1   paste.exe
1      5   patch.exe
1      1   pathchk.exe
1      3   pbmtojbg.exe
1      0   perl.exe
1      0   perl5.8.0.exe
1      1   pfbtops.exe
1      8   pgawk.exe
1      0   pgpewrap.exe
1      7   pgpring.exe
1      7   pic.exe
1      5   pinky.exe
1      8   post-grohtml.exe
1      2   ppm2tiff.exe
1      3   pr.exe
1      8   pre-grohtml.exe
1      0   printenv.exe
1      4   printf.exe
1      3   ps.exe
1      3   ptx.exe
1      2   putclip.exe
1      0   pwd.exe
1      8   ranlib.exe
1      0   ras2tiff.exe
1      1   raw2tiff.exe
1      4   rcp.exe
1      1   rdjpgcom.exe
1      8   readelf.exe
1      1   readlink.exe
1      1   realpath.exe
1      7   refer.exe
```

```
1     1   regtool.exe
1     9   reset.exe
1     1   rev.exe
1     1   rgb2ycbcr.exe
1     6   rl-stat.exe
1     1   rl.exe
1     3   rlogin.exe
1     6   rltest-stat.exe
1     0   rltest.exe
1     6   rlversion-stat.exe
1     1   rlversion.exe
1     6   rm.exe
1     4   rmdir.exe
1     8   rmic.exe
1     8   rmiregistry.exe
1     3   rsh.exe
1     6   scp.exe
1     3   sdiff.exe
1     5   sed.exe
1     1   seq.exe
1     1   setfacl.exe
1     8   sftp.exe
1     4   sh.exe
1     2   sha1sum.exe
1     6   shred.exe
1     7   size.exe
1     0   sleep.exe
1     5   soelim.exe
1     3   sort.exe
1     2   split.exe
1    12   squid.exe
1     8   ssh-add.exe
1     7   ssh-agent.exe
1     8   ssh-keygen.exe
1     9   ssh-keyscan.exe
1     9   ssh.exe
1     1   ssp.exe
1     6   strace.exe
1     7   strings.exe
1     8   strip.exe
1     2   stty.exe
1     2   su.exe
1     3   sum.exe
1     4   sync.exe
1     3   syslog.exe
1     4   tac.exe
1     8   tack.exe
1     4   tail.exe
1     8   talk.exe
1     9   tar.exe
1     6   tbl.exe
2     0   tclsh.exe
2     0   tclsh84.exe
1     3   tcsh.exe
1     1   tee.exe
```

```
1     6   telnet.exe
1     0   test.exe
1     3   testdbm.exe
1     4   testgdbm.exe
1     2   testndbm.exe
1     8   texindex.exe
1     6   tfmtodit.exe
1     6   tftp.exe
1     0   thumbnail.exe
1    10   tic.exe
1     1   tiff2bw.exe
1     1   tiff2ps.exe
1     0   tiff2rgba.exe
1     0   tiffcmp.exe
1     0   tiffcp.exe
1     1   tiffdither.exe
1     1   tiffdump.exe
1     0   tiffinfo.exe
1     0   tiffmedian.exe
1     0   tiffset.exe
1     2   tiffsplit.exe
1     8   toe.exe
1     4   touch.exe
1     6   tput.exe
1     3   tr.exe
1     8   troff.exe
1     0   true.exe
1     9   tset.exe
1     3   tsort.exe
1     0   tty.exe
1     3   u2d.exe
1     0   umount.exe
1     0   uname.exe
1     1   unexpand.exe
1     1   uniq.exe
1     3   unix2dos.exe
1     6   unzip.exe
1     5   unzipsfx.exe
1     1   users.exe
1     7   vdir.exe
1     2   wc.exe
1    10   wget.exe
1     2   which.exe
1     3   who.exe
1     0   whoami.exe
1     8   windres.exe
2     1   wish.exe
2     1   wish84.exe
1     3   wrjpgcom.exe
1     3   xargs.exe
1     0   yes.exe
1     5   zcat.exe
1     8   zip.exe
1     3   zipcloak.exe
1     4   zipnote.exe
```

```
1     5   zipsplit.exe
```

11.5 Dynamic Link Library (DLL) files for Java 1.4.2

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 41

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     5     2   awt.dll
     5     1   axbridge.dll
     4     1   cmm.dll
     4     0   dcpr.dll
     4     1   dt_shmem.dll
     4     1   dt_socket.dll
     5     0   eula.dll
     4     2   fontmanager.dll
     4     3   hpi.dll
     4     2   hprof.dll
     4     0   ioser12.dll
     4     0   jaas_nt.dll
     4     4   java.dll
     4     0   jawt.dll
     4     3   jcov.dll
     4     1   JdbcOdbc.dll
     4     2   jdwp.dll
     4     1   jpeg.dll
     5     0   jpicom32.dll
     5     1   jpiexp32.dll
     4     0   jpins4.dll
     4     0   jpins6.dll
     4     0   jpins7.dll
     5     2   jpinsp.dll
     5     2   jpishare.dll
     4     2   jsound.dll
     6     1   msvcrt.dll
     4     2   net.dll
     4     0   nio.dll
     5     1   NPJava11.dll
     5     1   NPJava12.dll
     5     1   NPJava13.dll
     5     1   NPJava14.dll
     5     1   NPJava32.dll
     5     1   NPJPI142_01.dll
     5     1   NPOJI610.dll
     5     2   RegUtils.dll
     4     0   rmi.dll
```

```
4     0   verify.dll
4     0   w2k_lsa_auth.dll
4     1   zip.dll
```

11.6 Executable (EXE) files for Java 1.4.2

```
FILE NAME: java.exe


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 180 bytes
exists starting at address 12288; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 80 bytes in size when actually it is
800 bytes in size

**** End of Anomalies ****



!!!! Security Vulnerabilities and Risks!!!!

  - Uses 2 standard C functions susceptible to buffer overflow
attacks: fgets (Low risk), sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!
```

```
FILE NAME: javaw.exe


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 216 bytes
exists starting at address 16384; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 100 bytes in size when actually it is
974 bytes in size

**** End of Anomalies ****
```

!!!! Security Vulnerabilities and Risks!!!!

- Uses 2 standard C functions susceptible to buffer overflow attacks: fgets (Low risk), sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: jpicpl32.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 92 bytes exists starting at address 8192; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 419 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: jucheck.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 932 bytes exists starting at address 53248; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 280 bytes in size when actually it is 4566 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 166192 bytes in size when actually it is 167936 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 4 standard C functions susceptible to buffer overflow attacks: fgetc (Medium risk), sprintf (Very high risk), sscanf (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: jusched.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 364 bytes exists starting at address 20480; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 140 bytes in size when actually it is 1780 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 4 standard C functions susceptible to buffer overflow attacks: fgetc (Medium risk), sprintf (Very high risk), sscanf (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: keytool.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 180 bytes exists starting at address 16384; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 800 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 2 standard C functions susceptible to buffer overflow attacks: fgets (Low risk), sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: kinit.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 180 bytes exists starting at address 16384; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 800 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 2 standard C functions susceptible to buffer overflow attacks: fgets (Low risk), sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: klist.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 180 bytes exists starting at address 16384; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 800 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 2 standard C functions susceptible to buffer overflow attacks: fgets (Low risk), sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: ktab.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 180 bytes exists starting at address 16384; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 800 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 2 standard C functions susceptible to buffer overflow attacks: fgets (Low risk), sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: orbd.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 180 bytes exists starting at address 16384; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 800 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

  - Uses 2 standard C functions susceptible to buffer overflow attacks: fgets (Low risk), sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: policytool.exe

**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub contains some kind of information
  - The file indicates an import address table consisting of 180 bytes exists starting at address 16384; this table often does not appear in an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 800 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

  - Uses 2 standard C functions susceptible to buffer overflow attacks: fgets (Low risk), sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: rmid.exe

**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub contains some kind of information
  - The file indicates an import address table consisting of 180 bytes exists starting at address 16384; this table often does not appear in an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 800 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 2 standard C functions susceptible to buffer overflow attacks: fgets (Low risk), sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: rmiregistry.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 180 bytes exists starting at address 16384; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 800 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 2 standard C functions susceptible to buffer overflow attacks: fgets (Low risk), sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: servertool.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 180 bytes exists starting at address 16384; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 800 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 2 standard C functions susceptible to buffer overflow attacks: fgets (Low risk), sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: tnameserv.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 180 bytes exists starting at address 16384; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 800 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 2 standard C functions susceptible to buffer overflow attacks: fgets (Low risk), sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

---------- Summary of File Security Analysis ----------

Total number of files submitted: 15

List of files containing anomalies (A), vulnerabilities (V) or risks (R)

A	V/R	Filename
3	2	java.exe
3	2	javaw.exe

```
3     0   jpicpl32.exe
4     4   jucheck.exe
3     4   jusched.exe
3     2   keytool.exe
3     2   kinit.exe
3     2   klist.exe
3     2   ktab.exe
3     2   orbd.exe
3     2   policytool.exe
3     2   rmid.exe
3     2   rmiregistry.exe
3     2   servertool.exe
3     2   tnameserv.exe
```

11.7 Dynamic Link Library (DLL) files for Microsoft Visual Studio SDK

```
FILE NAME: Microsoft.VisualStudio.Designer.Interfaces.dll


**** Anomalies ****

  - The file indicates an import address table consisting of 8 bytes
exists starting at address 1024; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The file indicates a COM runtime header consisting of 72 bytes
exists starting at address 1032; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Debug Table (.debug section) is 28 bytes in size when actually it is
512 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 1640 bytes in size when actually it
is 2048 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 12 bytes in size when actually it
is 512 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
FILE NAME: Microsoft.VisualStudio.dll


**** Anomalies ****
```

- The file indicates an import address table consisting of 8 bytes exists starting at address 4096; this table often does not appear in an image file so it was not read and it was also not mapped
- The file indicates a COM runtime header consisting of 72 bytes exists starting at address 4104; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1520 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 12 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: msdis130.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 188 bytes exists starting at address 102400; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 1954 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1000 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 21728 bytes in size when actually it is 24576 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses one standard C function susceptible to buffer overflow attack: sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 3

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     5     0   Microsoft.VisualStudio.Designer.Interfaces.dll
     4     0   Microsoft.VisualStudio.dll
     5     1   msdis130.dll
```

11.8 Executable (EXE) files for Microsoft Visual Studio SDK

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 29

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     4     0   AxImp.exe
     4     0   cert2spc.exe
     4     1   certmgr.exe
     4     0   ChkTrust.exe
     4     6   cordbg.exe
     4     0   disco.exe
     5     0   FUSLOGVW.exe
     4     0   gacutil.exe
     4     7   ildasm.exe
     4     0   lc.exe
     4     0   makecert.exe
     4     0   MgmtClassGen.exe
     4     0   mscordmp.exe
     4     3   nmake.exe
     4     2   PermView.exe
     4     3   PEVerify.exe
     4     0   ResGen.exe
     4     0   SecUtil.exe
     4     0   setreg.exe
     4     0   signcode.exe
     4     3   sn.exe
     4     0   SoapSuds.exe
     4     0   StoreAdm.exe
     4     0   TlbExp.exe
```

```
4      0   TlbImp.exe
4      0   WinCV.exe
4      0   WinRes.exe
4      0   wsdl.exe
4      0   xsd.exe
```

11.9 Dynamic Link Library (DLL) files for Microsoft Visual Studio VC7

```
FILE NAME: atlprov.dll


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 364 bytes
exists starting at address 159744; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The file indicates a delay import descriptor consisting of 160
bytes exists starting at address 192848; this item often does not
appear in an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 60 bytes in size when actually it is
1189 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 42272 bytes in size when actually it
is 45056 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 11300 bytes in size when actually
it is 16384 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
FILE NAME: c1.dll


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 608 bytes
exists starting at address 405504; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The file indicates a delay import descriptor consisting of 96
bytes exists starting at address 474284; this item often does not
appear in an image file so it was not read and it was also not mapped
```

- The data directory table in the optional header states that the Import Table (.idata section) is 100 bytes in size when actually it is 2764 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 265384 bytes in size when actually it is 266240 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 37796 bytes in size when actually it is 40960 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 5 standard C functions susceptible to buffer overflow attacks: fgets (Low risk), read (Medium risk), snprintf (Low risk), sprintf (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: c1xx.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 764 bytes exists starting at address 1314816; this table often does not appear in an image file so it was not read and it was also not mapped
- The file indicates a delay import descriptor consisting of 160 bytes exists starting at address 1515864; this item often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 3400 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 269632 bytes in size when actually it is 270336 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 102648 bytes in size when actually it is 106496 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 7 standard C functions susceptible to buffer overflow attacks: fgets (Low risk), read (Medium risk), snprintf (Low risk), sprintf (Very high risk), sscanf (Very high risk), strncpy (Low risk), vsnprintf (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: c2.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 416 bytes exists starting at address 1359872; this table often does not appear in an image file so it was not read and it was also not mapped
- The file indicates a delay import descriptor consisting of 128 bytes exists starting at address 1453448; this item often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 100 bytes in size when actually it is 1946 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 17440 bytes in size when actually it is 20480 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 64036 bytes in size when actually it is 65536 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 5 standard C functions susceptible to buffer overflow attacks: fgets (Low risk), memcpy (Low risk), sprintf (Very high risk), strncpy (Low risk), vsprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: rcdll.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information

- The file indicates an import address table consisting of 292 bytes exists starting at address 1024; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 1176 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 27496 bytes in size when actually it is 27648 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 6504 bytes in size when actually it is 7168 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 3 standard C functions susceptible to buffer overflow attacks: fgetc (Medium risk), sprintf (Very high risk), sscanf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: wmiscriptutils.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 304 bytes exists starting at address 1024; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 160 bytes in size when actually it is 1603 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 5576 bytes in size when actually it is 5632 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 1332 bytes in size when actually it is 2048 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

```
  - Uses one standard C function susceptible to buffer overflow
attack: memcpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 6

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     6     0   atlprov.dll
     6     5   c1.dll
     6     7   c1xx.dll
     6     5   c2.dll
     5     3   rcdll.dll
     5     1   wmiscriptutils.dll
```

11.10 Executable (EXE) files for Microsoft Visual Studio VC7

```
FILE NAME: bscmake.exe


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 328 bytes
exists starting at address 4096; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 80 bytes in size when actually it is
1441 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 5352 bytes in size when actually it
is 8192 bytes in size

**** End of Anomalies ****


!!!! Security Vulnerabilities and Risks!!!!
```

- Uses 4 standard C functions susceptible to buffer overflow attacks: getc (Medium risk), read (Medium risk), sprintf (Very high risk), vsnprintf (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: cl.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates a thread local storage table exists consisting of 24 bytes; this table usually does not appear in an image file so it was not read and only its start address was mapped
- The file indicates an import address table consisting of 548 bytes exists starting at address 36864; this table often does not appear in an image file so it was not read and it was also not mapped
- The file indicates a delay import descriptor consisting of 64 bytes exists starting at address 51700; this item often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 100 bytes in size when actually it is 2529 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 16280 bytes in size when actually it is 16384 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 8 standard C functions susceptible to buffer overflow attacks: fgets (Low risk), getchar (Medium risk), memcpy (Low risk), sprintf (Very high risk), sscanf (Very high risk), strcat (Very high risk), strcpy (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: clstencil.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information

- The file indicates an import address table consisting of 612 bytes exists starting at address 131072; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 160 bytes in size when actually it is 2362 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 3064 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: cvtres.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 288 bytes exists starting at address 4096; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 60 bytes in size when actually it is 1304 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 2616 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 2 standard C functions susceptible to buffer overflow attacks: sprintf (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: dumpbin.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information

- The file indicates an import address table consisting of 136 bytes exists starting at address 1024; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 60 bytes in size when actually it is 680 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 936 bytes in size when actually it is 1024 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: editbin.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 136 bytes exists starting at address 1024; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 60 bytes in size when actually it is 680 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 936 bytes in size when actually it is 1024 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: h2inc.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 388 bytes exists starting at address 278956; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 40 bytes in size when actually it is 2150 bytes in size

- The data directory table in the optional header states that the Debug Table (.debug section) is 28 bytes in size when actually it is 8192 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1670 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: lib.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 136 bytes exists starting at address 1024; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 60 bytes in size when actually it is 680 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 920 bytes in size when actually it is 1024 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: link.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 812 bytes exists starting at address 4096; this table often does not appear in an image file so it was not read and it was also not mapped
- The file indicates a delay import descriptor consisting of 224 bytes exists starting at address 592608; this item often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 2516 bytes in size

- The data directory table in the optional header states that the Resource Table (.rsrc section) is 37504 bytes in size when actually it is 40960 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 7 standard C functions susceptible to buffer overflow attacks: fgets (Low risk), getc (Medium risk), getchar (Medium risk), read (Medium risk), sprintf (Very high risk), sscanf (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: ml.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 276 bytes exists starting at address 278528; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 40 bytes in size when actually it is 1545 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 20680 bytes in size when actually it is 24576 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: nmake.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 448 bytes exists starting at address 4096; this table often does not appear in an image file so it was not read and it was also not mapped

- The data directory table in the optional header states that the Import Table (.idata section) is 60 bytes in size when actually it is 1900 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 10280 bytes in size when actually it is 12288 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 3 standard C functions susceptible to buffer overflow attacks: fgets (Low risk), getc (Medium risk), sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: rc.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 88 bytes exists starting at address 1536; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 445 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 952 bytes in size when actually it is 1024 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: sproxy.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information

- The file indicates an import address table consisting of 408 bytes exists starting at address 89600; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 120 bytes in size when actually it is 1223 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 27320 bytes in size when actually it is 27648 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses one standard C function susceptible to buffer overflow attack: sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: undname.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 152 bytes exists starting at address 3072; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 60 bytes in size when actually it is 699 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 984 bytes in size when actually it is 1024 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses one standard C function susceptible to buffer overflow attack: memcpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: vcdeploy.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 632 bytes exists starting at address 28672; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 160 bytes in size when actually it is 2553 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 9208 bytes in size when actually it is 9216 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses one standard C function susceptible to buffer overflow attack: vsprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

---------- Summary of File Security Analysis ----------

Total number of files submitted: 15

List of files containing anomalies (A), vulnerabilities (V) or risks (R)

A	V/R	Filename
4	4	bscmake.exe
6	8	cl.exe
4	0	clstencil.exe
4	2	cvtres.exe
4	0	dumpbin.exe
4	0	editbin.exe
5	0	h2inc.exe
4	0	lib.exe
5	7	link.exe
4	0	ml.exe
4	3	nmake.exe
4	0	rc.exe
4	1	sproxy.exe
4	1	undname.exe
4	1	vcdeploy.exe

12. APPENDIX F – TEST RESULTS FROM ANALYZING WINDOWS XP HOME EDITION OPERATING SYSTEM FILES

12.1 Dynamic Link Library (DLL) Files for Windows XP (C:\windows directory)

```
FILE NAME: twain_32.dll


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 328 bytes
exists starting at address 1024; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 80 bytes in size when actually it is
1259 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 4728 bytes in size when actually it
is 5120 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 2008 bytes in size when actually
it is 2560 bytes in size

**** End of Anomalies ****



!!!! Security Vulnerabilities and Risks!!!!

  - Uses 2 standard C functions susceptible to buffer overflow
attacks: read (Medium risk), sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!
```

```
FILE NAME: vmmreg32.dll
```

```
**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 100 bytes
exists starting at address 1024; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 80 bytes in size when actually it is
489 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 1008 bytes in size when actually it
is 1024 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 344 bytes in size when actually
it is 512 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 3

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     5     2   twain_32.dll
     5     0   vmmreg32.dll
```

12.2 Executable (EXE) Files for Windows XP (C:\windows directory)

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 26

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     5     2   delttsul.exe
     6     0   explorer.exe
     4     0   GPInstall.exe
     4     0   hh.exe
```

```
4     0   ieuninst.exe
4     0   IsUninst.exe
4     0   muninst.exe
4     0   NOTEPAD.EXE
4     0   oeuninst.exe
4     0   Q330994.exe
4     0   regedit.exe
4     1   rmud.exe
5     0   setdebug.exe
4     0   TASKMAN.EXE
4     1   twunk_32.exe
4     0   uinst001.exe
4     0   uninst.exe
4     0   unvise32.exe
4     0   unvise32qt.exe
4     1   winhlp32.exe
```

12.3 Dynamic Link Library (DLL) Files for Windows XP (C:\windows\system directory)

```
FILE NAME: CTL3D32.DLL


**** Anomalies ****

  - A section entry named .bss appears in the section table, but the
table doesn't contain the location of the 0 bytes for that section
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 2384 bytes in size when actually it
is 2560 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 568 bytes in size when actually
it is 5120 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
FILE NAME: CW3215.DLL


**** Anomalies ****

  - The length of 256 forwarder name(s) in the export table exceeded
the buffer size of 255 bytes
  - Tried to read a directory table entry in the import table but
didn't find the amount of data expected
```

```
  - The data directory table in the optional header states that the
Import Table (.idata section) is 2256 bytes in size when actually it
is 2560 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 5640 bytes in size when actually
it is 6144 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 18

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     3     0   CTL3D32.DLL
     4     0   CW3215.DLL
```

12.4 Driver (DRV) Files for Windows XP (C:\windows\system directory)

```
FILE NAME: WINSPOOL.DRV


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 708 bytes
exists starting at address 1024; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The file indicates a delay import descriptor consisting of 224
bytes exists starting at address 112252; this item often does not
appear in an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 160 bytes in size when actually it is
4041 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 2512 bytes in size when actually it
is 2560 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 5260 bytes in size when actually
it is 5632 bytes in size

**** End of Anomalies ****
```

No security vulnerabilities or security risks were found

---------- Summary of File Security Analysis ----------

Total number of files submitted: 11

List of files containing anomalies (A), vulnerabilities (V) or risks (R)

A	V/R	Filename
6	0	WINSPOOL.DRV

12.5 Dynamic Link Library (DLL) Files for Windows XP (C:\windows\system32 directory)

---------- Summary of File Security Analysis ----------

Total number of files submitted: 1348

List of files containing anomalies (A), vulnerabilities (V) or risks (R)

A	V/R	Filename
5	2	6to4svc.dll
5	0	aaaamon.dll
3	0	acctres.dll
5	0	acledit.dll
6	0	aclui.dll
5	1	activeds.dll
6	0	actxprxy.dll
5	0	admparse.dll
5	0	adptif.dll
5	1	adsldp.dll
5	0	adsldpc.dll
5	1	adsmsext.dll
5	0	adsnt.dll
6	2	advapi32.dll
5	0	advpack.dll
5	0	alrsvc.dll
5	0	amstream.dll
5	3	apcups.dll
6	3	apphelp.dll
3	0	asferror.dll
5	0	asfsipc.dll
5	2	asycfilt.dll
5	0	atkctrs.dll

```
6     0   atl.dll
6     0   atl70.dll
6     1   atmfd.dll
5     2   atmlib.dll
5     0   atmpvcno.dll
5     1   atrace.dll
5     0   audiosrv.dll
6     0   authz.dll
6     0   autodisc.dll
5     0   avicap32.dll
5     0   avifil32.dll
5     1   avmeter.dll
5     0   avtapi.dll
5     0   avwav.dll
5     0   basesrv.dll
5     0   batmeter.dll
6     0   batt.dll
2     0   bcbmm.dll
5     4   bfc42.dll
5     5   bfc42d.dll
6     0   bidispl.dll
5     0   bitsprx2.dll
5     0   bitsprx3.dll
5     3   blackbox.dll
4     1   Bocof.dll
6     1   bootvid.dll
2     0   borlndmm.dll
3     0   browselc.dll
4     0   browser.dll
6     0   BROWSEUI.DLL
5     0   browsewm.dll
5     0   cabinet.dll
5     0   cabview.dll
5     0   camocx.dll
5     0   capesnpn.dll
6     0   cards.dll
5     0   catsrv.dll
5     0   catsrvps.dll
5     0   catsrvut.dll
6     0   cc3250.dll
6     0   cc3250mt.dll
5     0   ccfgnt.dll
5     0   cddbcontrol.dll
6     0   cdfview.dll
6     1   cdm.dll
5     1   cdmodem.dll
5     4   cdosys.dll
5     2   cehelper.dll
5     3   certcli.dll
5     0   certmgr.dll
5     0   CEWMDM.dll
5     0   cfgbkend.dll
4     0   cfgmgr32.dll
5     0   ciadmin.dll
5     0   cic.dll
```

```
5     0   ciodm.dll
5     0   clb.dll
5     0   clbcatex.dll
5     1   clbcatq.dll
6     2   cliconfg.dll
5     5   ClientBR.dll
5     0   clusapi.dll
5     0   cmcfg32.dll
5     0   cmdial32.dll
6     0   cmpbk32.dll
5     0   cmprops.dll
5     0   cmutil.dll
5     1   cnbjmon.dll
5     0   cnetcfg.dll
4     0   cnvfat.dll
6     1   colbact.dll
5     0   comaddin.dll
4     0   comcat.dll
5     0   comctl32.dll
5     0   comdlg32.dll
5     0   compatUI.dll
5     0   compstui.dll
5     0   comrepl.dll
6     0   comres.dll
5     0   comsnap.dll
5     1   comsvcs.dll
5     0   comuid.dll
5     1   confmsp.dll
5     0   console.dll
5     0   corpol.dll
3     0   cpuinf32.dll
6     0   credui.dll
4     0   crtdll.dll
6     2   crypt32.dll
5     0   cryptdlg.dll
5     0   cryptdll.dll
6     1   cryptext.dll
5     1   cryptnet.dll
6     0   cryptsvc.dll
6     1   cryptui.dll
6     2   cscdll.dll
6     0   cscui.dll
5     2   csrsrv.dll
5     0   csseqchk.dll
4     0   ctl3d32.dll
5     4   d3d8.dll
6     0   d3d8thk.dll
5     3   d3d9.dll
5     2   d3dim.dll
6     0   d3dpmesh.dll
4     0   d3dramp.dll
5     1   d3drm.dll
5     2   d3dxof.dll
5     1   danim.dll
5     0   dataclen.dll
```

```
5     0   datime.dll
5     0   davclnt.dll
6     8   dbgeng.dll
5     4   dbghelp.dll
6     1   dbmsadsn.dll
5     2   dbmsrpcn.dll
5     1   dbmsvinn.dLL
5     2   DBnetlib.dll
5     1   dbnmpntw.dll
5     0   dciman32.dll
5     0   ddraw.dll
5     0   ddrawex.dll
2     0   delphimm.dll
5     0   deskadp.dll
5     0   deskmon.dll
5     0   deskperf.dll
5     0   devenum.dll
6     0   devmgr.dll
2     0   dfrgres.dll
5     0   dfrgsnap.dll
5     0   dfrgui.dll
5     0   dfsshlex.dll
4     0   dgnet.dll
5     0   dgrpsetu.dll
5     0   dgsetup.dll
5     1   dhcpcsvc.dll
5     2   dhcpmon.dll
5     0   dhcpsapi.dll
4     0   diactfrm.dll
6     0   digest.dll
5     1   dimap.dll
5     0   dinput.dll
5     0   dinput8.dll
5     0   diskcopy.dll
5     0   dispex.dll
5     0   dmband.dll
5     0   dmcompos.dll
5     3   dmconfig.dll
5     0   dmdlgs.dll
5     0   dmdskmgr.dll
4     0   dmdskres.dll
5     0   dmime.dll
5     0   dmintf.dll
5     0   dmloader.dll
5     0   dmocx.dll
5     0   dmscript.dll
5     1   dmserver.dll
5     0   dmstyle.dll
5     0   dmsynth.dll
5     0   dmusic.dll
5     1   dmutil.dll
6     6   dnsapi.dll
5     4   dnsrslvr.dll
5     0   docprop.dll
5     0   docprop2.dll
```

```
5     1   dpcdll.dll
5     0   dplay.dll
5     0   dplayx.dll
5     0   dpmodemx.dll
4     0   dpnaddr.dll
5     2   dpnet.dll
5     0   dpnhpast.dll
5     0   dpnhupnp.dll
3     0   dpnlobby.dll
5     0   dpnmodem.dll
5     1   dpnwsock.dll
5     0   dpserial.dll
5     0   dpvacm.dll
5     2   dpvoice.dll
5     0   dpvvox.dll
5     0   dpwsock.dll
5     0   dpwsockx.dll
5     7   drmclien.dll
5     0   drmstor.dll
5     8   drmv2clt.dll
5     0   drprov.dll
6     0   ds32gt.dll
5     0   dsauth.dll
5     1   dsdmo.dll
5     1   dsdmoprp.dll
5     0   dskquota.dll
5     0   dskquoui.dll
5     0   dsound.dll
6     0   dsound3d.dll
5     0   dsprop.dll
5     0   dsquery.dll
5     0   dssdata.dll
5     0   dssec.dll
6     0   dssenh.dll
5     0   dsuiext.dll
5     0   dswave.dll
6     1   duser.dll
4     0   dx3j.dll
5     0   dx7vb.dll
5     4   dx8vb.dll
5     0   dxdiagn.dll
6     4   dxmasf.dll
5     3   dxmrtp.dll
5     0   dxtmsft.dll
5     0   dxtrans.dll
5     0   els.dll
5     0   encapi.dll
5     0   EqnClass.Dll
5     0   ersvc.dll
5     1   es.dll
5     6   esent.dll
5     5   esent97.dll
5     1   esentprf.dll
5     1   eventcls.dll
6     0   eventlog.dll
```

```
5     0   expsrv.dll
5     2   EXSEC32.DLL
5     3   exts.dll
5     1   faultrep.dll
5     0   feclient.dll
5     0   filemgmt.dll
5     0   fldrclnr.dll
4     0   FLORA32.DLL
5     0   FM20.DLL
3     0   FM20ENU.DLL
5     0   fmifs.dll
5     2   fontext.dll
5     0   fontsub.dll
6     1   framebuf.dll
5     3   fsusd.dll
5     0   ftsrch.dll
5     0   fxsapi.dll
5     0   fxscfgwz.dll
5     0   fxsclntR.dll
5     0   fxscom.dll
5     0   fxscomex.dll
5     0   fxsdrv.dll
6     0   fxsevent.dll
6     0   fxsext32.dll
5     0   fxsmon.dll
5     0   fxsperf.dll
5     0   fxsres.dll
5     0   fxsroute.dll
5     0   fxsst.dll
5     3   fxst30.dll
6     0   fxstiff.dll
5     0   fxsui.dll
5     1   fxswzrd.dll
5     0   fxsxp32.dll
5     0   gcdef.dll
5     1   gdi32.dll
6     0   getuname.dll
5     2   glmf32.dll
5     0   glu32.dll
5     1   gpkcsp.dll
3     0   gpkrsrc.dll
5     1   h323msp.dll
4     3   hal.dll
5     0   hccutils.dll
5     4   HfxClasses45.dll
5     1   HfxGui45.dll
5     2   hhsetup.dll
5     0   hid.dll
5     0   hlink.dll
4     0   hlp25632.dll
4     0   hlp95en.dll
6     0   hnetcfg.dll
5     0   hnetmon.dll
5     0   hnetwiz.dll
6     0   hotplug.dll
```

5	0	hpaghlpr.dll
5	0	Hpgdtppg.dll
5	0	hpgdtt.dll
5	1	hpgdtuu.dll
5	2	hpgreg32.dll
4	4	hpgt34.dll
5	1	hpgt34tk.dll
5	0	hpgtmcro.dll
5	3	hpREG.DLL
5	0	hpsj32.dll
5	0	hpsjvset.dll
5	0	HPUNINST.DLL
5	0	hpvaut32.dll
7	3	hpvcp70.dll
5	0	hpvcr70.dll
5	0	hpzcoi09.dll
5	0	hpzcon09.dll
5	0	hpzlnt09.dll
5	0	hticons.dll
5	0	htui.dll
5	5	hypertrm.dll
4	0	i81xcoin.dll
6	1	i81xdnt5.dll
5	0	i81xgdev.dll
5	0	i81xgicd.dll
5	0	iasacct.dll
5	0	iasads.dll
5	0	iashlpr.dll
5	0	iasnap.dll
5	0	iaspolcy.dll
5	0	iasrad.dll
5	0	iasrecst.dll
5	1	iassam.dll
5	0	iassdo.dll
5	0	iassvcs.dll
4	2	icaapi.dll
2	0	iccvid.dll
6	0	icfgnt5.dll
5	0	icm32.dll
4	0	icmp.dll
5	0	icmui.dll
5	0	icwdial.dll
5	0	icwphbk.dll
5	0	idq.dll
1	0	IDUNINST.DLL
5	0	ieakeng.dll
5	0	ieaksie.dll
3	0	ieakui.dll
6	0	iedkcs32.dll
5	0	iepeers.dll
5	0	iernonce.dll
5	0	iesetup.dll
5	1	ifmon.dll
5	1	ifsutil.dll
5	0	igfxdev.dll

```
4     0   igfxdgps.dll
5     0   igfxdo.dll
5     0   igfxeud.dll
5     0   igfxhk.dll
5     0   igfxpph.dll
5     0   igfxres.dll
5     0   igfxsrvc.dll
5     0   igmpagnt.dll
5     0   ils.dll
6     3   imagehlp.dll
6     1   imeshare.dll
5     0   imgutil.dll
5     0   imm32.dll
5     0   inetcfg.dll
5     0   INETCOMM.DLL
3     0   inetcplc.dll
5     0   inetmib1.dll
6     0   inetpp.dll
5     0   inetppui.dll
3     0   inetres.dll
5     0   INETWH32.DLL
5     4   infosoft.dll
5     0   initpki.dll
3     0   INLOADER.DLL
5     0   input.dll
5     0   inseng.dll
5     0   InstAdm.dll
5     0   InstExp.dll
3     0   iologmsg.dll
5     0   ioRdyUI.dll
5     0   ioReady.dll
5     1   ipeapi12.dll
5     9   ipebase12.dll
5     1   ipeistor12.dll
6     2   iphlpapi.dll
5     0   ipl.dll
5     0   ipla6.dll
5     0   iplm5.dll
5     0   iplm6.dll
5     0   iplp6.dll
5     0   iplpx.dll
5     0   iplw7.dll
5     1   ipmontr.dll
5     2   ipnathlp.dll
5     1   ippromon.dll
4     0   iprop.dll
5     0   iprtprio.dll
5     2   iprtrmgr.dll
5     0   ipsecsnp.dll
5     0   ipsecsvc.dll
5     0   ipsmsnap.dll
5     0   ipv6mon.dll
5     1   ipxmontr.dll
5     0   ipxpromn.dll
5     0   ipxrip.dll
```

```
5     0   ipxrtmgr.dll
5     1   ipxsap.dll
5     0   ipxwan.dll
3     0   ir32_32.dll
5     0   ir41_qc.dll
4     0   ir41_qcx.dll
5     0   ir50_32.dll
5     0   ir50_qc.dll
5     0   ir50_qcx.dll
5     0   irclass.dll
5     0   isign32.dll
5     0   isrdbg32.dll
5     0   itircl.dll
5     0   itss.dll
5     1   iuctl.dll
5     1   iuengine.dll
5     0   ixsso.dll
5     0   iyuv_32.dll
5     0   javacypt.dll
5     0   javaee.dll
5     0   javaprxy.dll
5     0   javart.dll
5     2   jet500.dll
4     0   jgaw400.dll
4     0   jgdw400.dll
3     0   jgmd400.dll
3     0   jgpl400.dll
3     0   jgsd400.dll
3     0   jgsh400.dll
5     0   jit.dll
5     1   jobexec.dll
5     1   jscript.dll
5     0   jsproxy.dll
3     0   KBDAL.DLL
2     0   kbdaze.dll
3     0   kbdazel.dll
3     0   kbdbe.dll
3     0   kbdbene.dll
3     0   kbdblr.dll
3     0   kbdbr.dll
3     0   kbdbu.dll
2     0   kbdca.dll
3     0   kbdcan.dll
2     0   kbdcr.dll
3     0   kbdcz.dll
3     0   kbdcz1.dll
2     0   kbdcz2.dll
3     0   kbdda.dll
3     0   kbddv.dll
3     0   kbdes.dll
3     0   kbdest.dll
3     0   kbdfc.dll
3     0   kbdfi.dll
3     0   kbdfo.dll
3     0   kbdfr.dll
```

```
3      0    kbdgae.dll
3      0    kbdgkl.dll
3      0    kbdgr.dll
3      0    kbdgr1.dll
3      0    kbdhe.dll
2      0    kbdhe220.dll
2      0    kbdhe319.dll
3      0    kbdhela2.dll
3      0    kbdhela3.dll
2      0    kbdhept.dll
3      0    kbdhu.dll
2      0    kbdhu1.dll
3      0    kbdic.dll
3      0    kbdir.dll
3      0    kbdit.dll
3      0    kbdit142.dll
3      0    kbdkaz.dll
3      0    kbdkyr.dll
3      0    kbdla.dll
3      0    kbdlt.dll
3      0    kbdlt1.dll
3      0    kbdlv.dll
3      0    kbdlv1.dll
3      0    kbdmac.dll
3      0    kbdmon.dll
3      0    kbdne.dll
3      0    kbdnec.dll
3      0    kbdno.dll
3      0    kbdpl.dll
2      0    kbdpl1.dll
3      0    kbdpo.dll
3      0    kbdro.dll
3      0    kbdru.dll
2      0    kbdru1.dll
3      0    kbdsf.dll
3      0    kbdsg.dll
3      0    kbdsl.dll
3      0    kbdsl1.dll
3      0    kbdsp.dll
3      0    kbdsw.dll
3      0    kbdtat.dll
3      0    kbdtuf.dll
3      0    kbdtuq.dll
3      0    kbduk.dll
3      0    kbdur.dll
3      0    kbdus.dll
3      0    kbdusl.dll
3      0    kbdusr.dll
2      0    kbdusx.dll
3      0    kbduzb.dll
2      0    kbdycc.dll
3      0    kbdycl.dll
6      2    kd1394.dll
6      1    kdcom.dll
6      3    kerberos.dll
```

```
5     2   kernel32.dll
6     0   keymgr.dll
5     0   ksuser.dll
5     0   langwrbk.dll
5     0   laprxy.dll
3     0   lfavi80n.dll
3     0   lfawd80n.dll
4     0   lfbmp11n.dll
3     0   lfbmp80n.dll
3     0   lfcal80n.dll
5     0   LFCMP11n.DLL
3     0   LFCMP70n.DLL
3     0   Lfcmp80n.dll
3     0   Lfdic80n.dll
4     0   lfeps11n.dll
3     0   lfeps80n.dll
4     0   lffax11n.dll
3     0   lffax70n.dll
3     0   Lffax80n.dll
4     1   Lffpx7.dll
3     0   lffpx70n.dll
3     0   lffpx80n.dll
4     0   lfgif11n.dll
3     0   lfgif70n.dll
3     0   lfgif80n.dll
3     0   lfica80n.dll
3     0   lfimg80n.dll
3     0   Lfkodak.dll
3     0   lflma80n.dll
3     0   lflmb80n.dll
3     0   lfmac80n.dll
3     0   lfmsp80n.dll
4     0   lfpcd11n.dll
3     0   lfpcd80n.dll
3     0   lfpct80n.dll
4     0   lfpcx11n.dll
3     0   lfpcx70n.dll
3     0   lfpcx80n.dll
5     0   Lfpng11n.dll
3     0   lfpng70n.dll
3     0   lfpng80n.dll
4     0   lfpsd11n.dll
3     0   lfpsd80n.dll
3     0   lfras80n.dll
4     0   lftga11n.dll
3     0   lftga80n.dll
4     0   lftif11n.dll
3     0   lftif70n.dll
3     0   Lftif80n.dll
3     0   lfwfx80n.dll
4     0   lfwmf11n.dll
3     0   lfwmf80n.dll
3     0   lfwpg80n.dll
6     5   libeay32.dll
4     0   libmred204_000.dll
```

```
4     0   libmzgc204_000.dll
4     0   libmzsch204_000.dll
5     0   libthrdR.dll
5     8   LibZkR.dll
5     1   LibZkTestR.dll
5     0   licdll.dll
5     0   licmgr10.dll
5     0   licwmi.dll
6     0   linkinfo.dll
4     1   lmhsvc.dll
5     1   lmrt.dll
5     2   loadperf.dll
5     0   localsec.dll
6     0   localspl.dll
5     0   localui.dll
5     0   loghours.dll
5     0   lpk.dll
5     2   lprhelp.dll
5     0   lprmonui.dll
6     3   lsasrv.dll
3     0   ltann80n.dll
4     0   LTDIS11n.dll
3     0   Ltefx80n.dll
4     0   ltfil11n.DLL
3     0   ltfil70n.DLL
3     0   ltfil80n.DLL
4     0   ltimg11n.dll
3     0   Ltimg80n.dll
4     0   ltkrn11n.dll
3     0   ltkrn70n.dll
3     0   Ltkrn80n.dll
3     0   Lttwn80n.dll
3     0   LTWND80n.DLL
7     0   Ltwvc11n.dll
3     0   lz32.dll
4     0   mag_hook.dll
5     0   mapi32.dll
5     0   mapistub.dll
5     0   mcastmib.dll
5     0   mcd32.dll
6     1   mcdsrv32.dll
5     0   mchgrcoi.dll
5     0   mciavi32.dll
5     0   mcicda.dll
5     0   mciole32.dll
5     0   mciqtz32.dll
5     0   mciseq.dll
5     0   mciwave.dll
5     0   mdhcp.dll
5     0   mdminst.dll
5     2   mdwmdmsp.dll
5     0   mf3216.dll
8     5   mfc40.dll
8     2   mfc40u.dll
9     5   mfc42.dll
```

```
3     0   MFC42ENU.DLL
9     2   mfc42u.dll
8     6   mfc70.dll
8     2   mfc70u.dll
5     0   mfcsubs.dll
5     0   mgmtapi.dll
5     0   midimap.dll
5     0   miglibnt.dll
5     1   mimefilt.dll
5     4   mindex.dll
5     1   mlang.dll
5     0   mll_hp.dll
5     0   mll_mtf.dll
5     0   mll_qic.dll
5     0   mmcbase.dll
5     0   mmcndmgr.dll
5     0   mmcshext.dll
5     0   mmdrv.dll
5     0   mmfutil.dll
5     0   mmutilse.dll
6     1   mnmdd.dll
5     0   mobsync.dll
5     0   modemui.dll
6     1   modex.dll
3     0   moricons.dll
5     1   mp43dmod.dll
5     1   mp4sdmod.dll
5     1   mpg4dmod.dll
6     0   mpr.dll
5     2   mprapi.dll
5     1   mprddm.dll
6     3   mprdim.dll
5     0   mprmsg.dll
6     0   mprui.dll
5     1   msaatext.dll
5     0   msacm32.dll
4     0   msafd.dll
5     2   msapsspc.dll
5     0   msasn1.dll
3     0   msaudite.dll
5     0   msawt.dll
6     0   mscat32.dll
5     0   mscms.dll
5     0   msconf.dll
3     0   mscpx32r.dLL
5     0   mscpxl32.dLL
5     0   msctf.dll
5     0   MSCTFP.dll
5     1   msdart.dll
5     2   msdmo.dll
5     2   msdtclog.dll
5     3   msdtcprx.dll
5     4   msdtctm.dll
5     2   msdtcuiu.dll
5     0   msdvdopt.dll
```

```
4     0   msdxmlc.dll
5     0   msencode.dll
6     0   msexch40.dll
6     1   msexcl40.dll
6     1   msgina.dll
5     1   msgsvc.dll
5     0   MSHTML.DLL
5     0   mshtmled.dll
3     0   mshtmler.dll
5     0   msi.dll
5     0   msident.dll
5     0   msidle.dll
3     0   msidntld.dll
6     0   msieftp.dll
5     0   msihnd.dll
5     0   msimg32.dll
4     0   MSIMRT.DLL
4     0   MSIMRT32.DLL
3     0   msimsg.dll
5     0   MSIMTF.dll
4     0   MSIMUSIC.DLL
5     1   msisam11.dll
5     0   msisip.dll
5     0   msjava.dll
6     0   msjdbc10.dll
5     1   msjet35.dll
6     1   msjet40.dll
6     1   msjetoledb40.dll
4     0   MSJINT32.DLL
4     0   msjint35.dll
6     0   msjint40.dll
6     2   MSJT3032.DLL
4     1   MSJTER32.DLL
5     1   msjter35.dll
5     0   msjter40.dll
6     1   msjtes40.dll
5     0   mslbui.dll
2     0   msls2.dll
3     0   msls31.dll
5     0   msltus40.dll
5     3   msnetobj.dll
5     2   msnsspc.dll
3     0   msobjs.dll
5     0   msoeacct.dll
5     0   msoert2.dll
3     0   msorc32r.dll
5     2   msorcl32.dll
5     0   mspatcha.dll
6     1   mspbde40.dll
5     6   mspmsnsv.dll
5     6   mspmsp.dll
5     6   mspmspsv.dll
5     0   msports.dll
3     0   msprivs.dll
4     1   msr2c.dll
```

```
4     0   msr2cenu.dll
3     0   msratelc.dll
5     0   msrating.dll
5     0   msrclr40.dll
4     0   MSRD2X32.DLL
5     0   MSRD2X35.DLL
6     1   msrd2x40.dll
6     1   msrd3x40.dll
7     0   MSRDO20.DLL
4     0   MSRECR40.DLL
7     1   msrepl35.dll
6     1   msrepl40.dll
5     0   msrle32.dll
5     3   msscp.dll
5     1   mssign32.dll
6     0   mssip32.dll
6     0   msstdfmt.dll
6     0   MSSTKPRP.DLL
6     0   msswch.dll
5     0   mstask.dll
6     1   mstext40.dll
5     0   mstime.dll
5     1   mstlsapi.dll
5     0   mstscax.dll
5     0   mstvca.dll
5     0   mstvgs.dll
6     1   msuni11.dll
5     0   msutb.dll
6     3   msv1_0.dll
5     1   msvbvm50.dll
5     1   msvbvm60.dll
5     3   msvcirt.dll
6     2   msvcp50.dll
8     4   msvcp60.dll
7     3   msvcp70.dll
5     0   msvcr70.dll
5     0   msvcrt.dll
2     0   Msvcrt10.dll
4     0   msvcrt20.dll
6     0   msvcrt40.dll
5     0   msvfw32.dll
5     0   msvidc32.dll
6     0   msvidctl.dll
5     0   msw3prt.dll
5     0   mswdat10.dll
4     0   mswebdvd.dll
5     1   mswmdm.dll
6     4   mswsock.dll
6     1   mswstr10.dll
6     1   msxbde40.dll
6     0   msxml.dll
6     0   msxml2.dll
4     0   msxml2r.dll
6     0   msxml3.dll
3     0   msxml3r.dll
```

```
6     0   MSXML4.dll
3     0   MSXML4a.dll
3     0   MSXML4r.dll
3     0   msxmlr.dll
5     0   msyuv.dll
5     2   mtxclu.dll
4     0   mtxdm.dll
4     0   mtxex.dll
5     0   mtxlegih.dll
5     1   mtxoci.dll
5     0   mycomput.dll
5     0   mydocs.dll
5     0   narrhook.dll
5     0   nbicdnt.dll
5     0   ncobjapi.dll
5     0   ncxpnt.dll
5     1   nddeapi.dll
5     4   nddenb32.dll
6     3   netapi32.dll
6     2   netcfgx.dll
3     0   netevent.dll
3     0   neth.dll
5     0   netid.dll
5     2   netlogon.dll
6     1   netman.dll
3     0   netmsg.dll
6     0   netplwiz.dll
5     1   netrap.dll
6     0   netshell.dll
5     1   netui0.dll
5     0   netui1.dll
7     0   netui2.dll
5     0   newdev.dll
5     0   nlhtml.dll
3     0   nmevtmsg.dll
5     0   nmmkcert.dll
4     0   npplg80n.dll
5     3   npptools.dll
5     2   npwmsdrm.dll
4     0   ntdll.dll
6     1   ntdsapi.dll
5     0   ntlanman.dll
5     0   ntlanui.dll
4     0   ntlanui2.dll
5     0   ntlsapi.dll
6     0   ntmarta.dll
5     0   ntmsapi.dll
6     3   ntmsdba.dll
6     0   ntmsevt.dll
5     0   ntmsmgr.dll
5     1   ntmssvc.dll
5     0   ntprint.dll
5     2   ntsdexts.dll
5     0   ntshrui.dll
5     0   ntvdmd.dll
```

```
5     1   nv4.dll
5     0   nvcpl.dll
5     0   nvdesk32.dll
5     0   nvdmcpl.dll
5     0   nvinstnt.dll
5     0   nvoglnt.dll
5     0   nvqtwk.dll
3     0   nvrsda.dll
3     0   nvrsde.dll
3     0   nvrseng.dll
3     0   nvrses.dll
3     0   nvrsfi.dll
3     0   nvrsfr.dll
3     0   nvrsit.dll
3     0   nvrsja.dll
3     0   nvrsko.dll
3     0   nvrsnl.dll
3     0   nvrsno.dll
3     0   nvrsptb.dll
3     0   nvrssv.dll
3     0   nvrszhc.dll
3     0   nvrszht.dll
5     0   nwprovau.dll
5     1   oakley.dll
5     0   objsel.dll
8     6   OC30.DLL
6     0   occache.dll
4     0   ochlp30e.dll
6     1   ocmanage.dll
5     2   ODBC32.dll
6     0   odbc32gt.dll
5     0   odbcbcp.dll
6     0   odbcconf.dll
5     1   ODBCCP32.dll
5     0   odbccr32.dll
5     0   odbccu32.dll
3     0   odbcint.dll
5     0   odbcji32.dll
5     3   odbcjt32.dll
5     0   ODBCMON.DLL
3     0   odbcp32r.dll
5     3   odbctrac.dll
6     0   oddbse32.dll
6     0   odexl32.dll
6     0   odfox32.dll
6     0   odpdx32.dll
6     0   odtext32.dll
5     4   OemLibR.dll
5     1   offfilt.dll
4     1   ole32.dll
5     0   oleacc.dll
3     0   oleaccrc.dll
5     1   oleaut32.dll
5     0   olecli32.dll
5     0   olecnv32.dll
```

```
5     0   oledlg.dll
6     0   oleprn.dll
5     0   olepro32.dll
5     0   olesvr32.dll
5     0   olethk32.dll
5     0   opengl32.dll
5     2   osuninst.dll
6     4   OUTLWAB.DLL
5     0   panmap.dll
5     2   paqsp.dll
5     0   pautoenr.dll
3     0   PCDLIB32.DLL
5     2   pdh.dll
5     1   perfctrs.dll
5     0   perfdisk.dll
5     0   perfnet.dll
5     0   perfos.dll
5     0   perfproc.dll
5     0   perfts.dll
5     0   PGPhk.dll
5     0   photowiz.dll
5     0   pid.dll
5     0   pidgen.dll
3     0   pifmgr.dll
5     1   pjlmon.dll
5     2   PlugFile.dll
5     0   plustab.dll
5     0   pncrt.dll
4     0   pndx5032.dll
5     0   PNGFILT.DLL
5     0   polstore.dll
2     0   POLVGA.DLL
5     0   powrprof.dll
3     0   prflbmsg.dll
5     0   printui.dll
5     0   profmap.dll
6     1   psapi.dll
5     0   psbase.dll
5     0   pschdprf.dll
5     0   psisdecd.dll
5     0   psnppagn.dll
5     0   pstorec.dll
5     0   pstorsvc.dll
7     0   px.dll
5     0   pxdrv.dll
5     0   pxmas.dll
5     0   pxwave.dll
5     0   pxwma.dll
6     8   python15.dll
5     1   PythonCOM15.dll
5     0   PyWinTypes15.dll
5     0   qasf.dll
5     0   qcap.dll
5     0   qdv.dll
5     0   qdvd.dll
```

```
6     1   qedit.dll
3     0   qedwipes.dll
6     2   qmgr.dll
5     0   qmgrprxy.dll
5     0   qosname.dll
6     2   quartz.dll
5     3   query.dll
4     0   racpldlg.dll
5     0   rasadhlp.dll
6     1   rasapi32.dll
5     1   rasauto.dll
5     0   raschap.dll
5     0   rasctrs.dll
6     2   rasdlg.dll
5     0   rasman.dll
5     1   rasmans.dll
5     0   rasmontr.dll
5     2   rasmxs.dll
5     3   rasppp.dll
5     0   rasrad.dll
5     0   rassapi.dll
4     1   rasser.dll
4     2   rastapi.dll
5     0   rastls.dll
5     0   rcbdyctl.dll
1     0   RDBios32.DLL
5     0   rdchost.dll
7     0   RDOCURS.DLL
4     0   rdpcfgex.dll
6     1   rdpdd.dll
5     0   rdpsnd.dll
5     1   rdpwsx.dll
6     0   regapi.dll
5     0   regsvc.dll
5     2   regwizc.dll
5     0   remotepg.dll
5     0   rend.dll
4     0   resutils.dll
3     0   RHMMPLAY.DLL
6     0   riched20.dll
5     0   riched32.dll
5     4   rmoc3260.dll
4     0   rnr20.dll
5     0   Roboex32.dll
6     0   routetab.dll
5     0   rpcns4.dll
4     1   rpcrt4.dll
5     0   rpcss.dll
6     0   rsaenh.dll
6     0   rshx32.dll
5     0   rsmps.dll
3     0   rsvpmsg.dll
5     1   rsvpperf.dll
5     3   rsvpsp.dll
5     4   rtcdll.dll
```

```
4     1   rtipxmib.dll
5     1   rtm.dll
5     0   rtutils.dll
5     0   s3appdll.dll
5     0   S3Gamma.dll
6     1   s3gNB.dll
5     0   s3swtch2.dll
5     1   safrcdlg.dll
5     0   safrdm.dll
5     0   safrslv.dll
5     0   samlib.dll
6     4   samsrv.dll
5     0   scarddlg.dll
5     0   scardssp.dll
5     0   sccbase.dll
5     0   sccsccp.dll
5     0   scecli.dll
7     0   scesrv.dll
5     0   schannel.dll
6     0   schedsvc.dll
5     0   sclgntfy.dll
2     0   SCP32.DLL
5     0   scredir.dll
5     1   scripto.dll
5     1   scrobj.dll
5     2   scrrun.dll
5     1   sdpblb.dll
5     0   seclogon.dll
6     1   secur32.dll
5     0   security.dll
5     0   sendcmsg.dll
5     0   sendmail.dll
6     0   sens.dll
5     0   sensapi.dll
6     0   senscfg.dll
5     0   serialui.dll
5     0   servdeps.dll
5     0   serwvdrv.dll
6     0   setupapi.dll
5     3   setupdll.dll
5     0   sfc.dll
5     0   sfcfiles.dll
6     0   sfc_os.dll
5     0   sfmapi.dll
3     0   shdoclc.dll
6     0   SHDOCVW.DLL
6     0   shell32.dll
4     0   shellstyle.dll
5     0   shfolder.dll
6     0   shgina.dll
4     3   shimeng.dll
6     1   shimgvw.dll
6     0   SHLWAPI.DLL
6     0   shmedia.dll
6     0   shscrap.dll
```

```
6     0   shsvcs.dll
5     0   sigtab.dll
5     0   sisbkup.dll
5     0   skdll.dll
5     0   slayerxp.dll
5     1   slbcsp.dll
6     0   slbiop.dll
4     0   slbrccsp.dll
5     0   smlogcfg.dll
5     2   snmpapi.dll
5     0   snmpsnap.dll
6     0   softpub.dll
3     0   spmsg.dll
5     2   spnike.dll
5     0   spoolss.dll
5     0   SPORDER.DLL
5     2   sprio600.dll
5     2   sprio800.dll
5     1   spxcoins.dll
5     2   SQLSRV32.dll
5     3   sqlunirl.dll
5     0   sqlwid.dll
5     1   sqlwoa.dll
5     0   srclient.dll
5     0   srrstr.dll
5     0   srsvc.dll
5     0   srvsvc.dll
5     0   ssdpapi.dll
6     0   ssdpsrv.dll
4     0   ssleay32.dll
5     0   stclient.dll
2     0   stdvcl32.dll
2     0   stdvcl40.dll
5     3   sti.dll
5     3   sti_ci.dll
6     0   stobject.dll
5     0   storprop.dll
5     0   streamci.dll
5     4   strmdll.dll
6     0   svcpack.dll
5     1   swprv.dll
5     2   sxlrt232.dll
6     3   sxs.dll
5     0   synceng.dll
5     0   syncui.dll
5     0   syscontr.dll
5     0   sysinv.dll
6     0   syssetup.dll
5     0   t2embed.dll
5     1   tapi3.dll
5     1   tapi32.dll
5     0   tapiperf.dll
5     1   tapisrv.dll
3     0   tapiui.dll
5     2   tcpmib.dll
```

```
5     2   tcpmon.dll
6     0   tcpmonui.dll
5     1   termmgr.dll
6     2   termsrv.dll
6     0   themeui.dll
5     0   traffic.dll
5     0   trkwks.dll
6     1   tsappcmp.dll
5     0   tsbyuv.dll
5     0   tscfgwmi.dll
5     0   tsd32.dll
6     1   tsddd.dll
4     0   TV_ENG32.DLL
3     4   Twain_32.dll
5     1   txflog.dll
5     1   udhisapi.dll
5     1   ufat.dll
5     1   ulib.dll
5     0   umandlg.dll
6     0   umdmxfrm.dll
5     0   umloader.dll
5     0   umpnpmgr.dll
5     0   unimdmat.dll
5     0   uniplat.dll
5     1   untfs.dll
6     1   upnp.dll
6     1   upnphost.dll
6     0   upnpui.dll
6     0   ureg.dll
6     0   url.dll
6     0   URLMON.DLL
5     0   usbmon.dll
6     0   usbui.dll
6     1   user32.dll
6     0   userenv.dll
5     0   usp10.dll
5     0   usrcntra.dll
5     0   usrcoina.dll
5     0   usrdpa.dll
5     0   usrdtea.dll
5     0   usrfaxa.dll
5     0   usrlbva.dll
5     0   usrrtosa.dll
5     0   usrsdpia.dll
5     0   usrsvpia.dll
5     0   usrv42a.dll
5     0   usrv80a.dll
5     0   usrvoica.dll
5     0   usrvpa.dll
5     1   utildll.dll
5     0   uxtheme.dll
6     3   VB40032.DLL
5     0   VB5DB.DLL
4     0   VB5STKIT.DLL
5     0   vbajet32.dll
```

5	0	VBAME.DLL
4	1	VBAR2232.DLL
5	0	VBAR332.DLL
5	1	vbscript.dll
5	0	vcdex.dll
3	0	vcfidl32.dll
3	0	vcfiwz32.dll
5	0	vdmdbg.dll
5	1	vdmredir.dll
5	1	verifier.dll
5	1	version.dll
6	0	vfpodbc.dll
6	1	vga.dll
6	1	vga256.dll
6	1	vga64k.dll
5	0	vjoy.dll
5	0	vmhelper.dll
5	1	vsdata.dll
6	8	vsinit.dll
5	1	vsmonapi.dll
3	0	vspell32.dll
5	4	vspubapi.dll
6	1	vssapi.dll
5	0	vss_ps.dll
6	8	vsutil.dll
5	1	vsxml.dll
5	1	VXBLOCK.dll
5	0	VxDMDcDlg.dll
5	0	w32time.dll
4	0	w32topl.dll
5	1	wavemsp.dll
5	0	WBDBT32I.DLL
8	1	WBDBV32I.DLL
6	1	wdigest.dll
6	0	webcheck.dll
5	1	webclnt.dll
5	1	webhits.dll
5	0	webvw.dll
5	0	wh2robo.dll
5	0	wiadefui.dll
5	3	wiadss.dll
6	0	wiafbdrv.dll
5	1	wiascr.dll
5	3	wiaservc.dll
6	0	wiashext.dll
5	3	wiavideo.dll
5	3	wiavusd.dll
5	0	win32spl.dll
6	0	winfax.dll
5	1	winhttp.dll
6	2	WINHTTP5.DLL
6	1	WININET.DLL
5	0	winipsec.dll
5	1	winmm.dll
5	0	winntbbu.dll

```
5     0   winrnr.dll
5     0   winscard.dll
6     0   winsrv.dll
5     0   winsta.dll
6     0   winstrm.dll
6     0   wintrust.dll
6     3   wkssvc.dll
5     0   wldap32.dll
6     2   wlnotify.dll
5     1   wmadmod.dll
5     2   wmadmoe.dll
5     0   wmasf.dll
5     0   wmdmlog.dll
5     0   wmdmps.dll
3     0   wmerrenu.dll
3     0   wmerror.dll
4     0   wmi.dll
5     0   wmidx.dll
5     0   wmiprop.dll
6     5   wmnetmgr.dll
6     5   wmp.dll
5     0   wmpasf.dll
5     0   wmpcd.dll
5     0   wmpcore.dll
5     0   wmpdxm.dll
3     0   wmploc.dll
5     0   wmpshell.dll
5     0   wmpui.dll
5     2   wmsdmod.dll
5     1   wmsdmoe.dll
5     2   wmsdmoe2.dll
5     1   wmspdmod.dll
5     2   wmspdmoe.dll
6     4   wmstream.dll
5     1   wmv8dmod.dll
5     2   wmv8dmoe.dll
6     7   wmvcore.dll
5     2   wmvdmod.dll
5     1   wmvdmoe.dll
5     2   wmvdmoe2.dll
5     2   wow32.dll
5     1   wowfax.dll
5     0   wowfaxui.dll
6     0   ws2help.dll
6     2   ws2_32.dll
5     0   wshatm.dll
5     1   wshcon.dll
5     1   wshext.dll
5     0   wship6.dll
5     0   wshisn.dll
5     0   wshnetbs.dll
5     0   WshRm.dll
5     0   wshtcpip.dll
5     0   wsnmp32.dll
5     0   wsock32.dll
```

```
5     2   wstdecod.dll
5     0   wtsapi32.dll
5     1   wuaueng.dll
4     0   wuauserv.dll
5     0   wupdinfo.dll
5     2   wuv3is.dll
5     0   wzcdlg.dll
5     0   wzcsapi.dll
5     2   wzcsvc.dll
5     1   xactsrv.dll
6     1   xenroll.dll
5     1   xolehlp.dll
3     0   xpob2res.dll
5     0   zipfldr.dll
5     3   ZKLSPR.dll
5     2   zlib.dll
```

12.6 Driver (DRV) Files for Windows XP (C:\windows\system32 directory)

```
FILE NAME: msacm32.drv


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 308 bytes
exists starting at address 1024; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 100 bytes in size when actually it is
1560 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 7472 bytes in size when actually it
is 7680 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 492 bytes in size when actually
it is 1024 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
FILE NAME: msh261.drv


**** Anomalies ****
```

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 160 bytes exists starting at address 4096; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 120 bytes in size when actually it is 816 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 3816 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 2952 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: msh263.drv

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 132 bytes exists starting at address 4096; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 120 bytes in size when actually it is 682 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 3816 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 4508 bytes in size when actually it is 8192 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: wdmaud.drv

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information

- The file indicates an import address table consisting of 232 bytes exists starting at address 1024; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 100 bytes in size when actually it is 1342 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1000 bytes in size when actually it is 1024 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 1128 bytes in size when actually it is 1536 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: winspool.drv

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 708 bytes exists starting at address 1024; this table often does not appear in an image file so it was not read and it was also not mapped
- The file indicates a delay import descriptor consisting of 224 bytes exists starting at address 112252; this item often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 160 bytes in size when actually it is 4041 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 2512 bytes in size when actually it is 2560 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 5260 bytes in size when actually it is 5632 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

---------- Summary of File Security Analysis ----------

Total number of files submitted: 17

List of files containing anomalies (A), vulnerabilities (V) or risks (R)

A	V/R	Filename
5	0	msacm32.drv
5	0	msh261.drv
5	0	msh263.drv
5	0	wdmaud.drv
6	0	winspool.drv

12.7 Executable (EXE) Files for Windows XP (C:\windows\system32 directory)

---------- Summary of File Security Analysis ----------

Total number of files submitted: 301

List of files containing anomalies (A), vulnerabilities (V) or risks (R)

A	V/R	Filename
4	0	accwiz.exe
4	0	actmovie.exe
4	2	ahui.exe
3	0	alg.exe
4	2	arp.exe
4	2	at.exe
4	1	atmadm.exe
4	0	attrib.exe
5	1	autochk.exe
5	1	autoconv.exe
5	1	autofmt.exe
5	0	autolfn.exe
4	0	bootok.exe
4	0	bootvrfy.exe
4	1	cacls.exe
4	0	calc.exe
4	0	charmap.exe
4	0	chkdsk.exe
4	0	chkntfs.exe
3	0	cidaemon.exe
4	0	cisvc.exe
4	0	ckcnv.exe
4	0	cleanmgr.exe
4	0	cliconfg.exe
4	1	clipbrd.exe
4	1	clipsrv.exe
5	0	clspack.exe
5	1	cmd.exe
4	0	cmdl32.exe
5	0	cmmon32.exe
4	0	cmstp.exe

```
4      0   comp.exe
4      0   compact.exe
4      0   conime.exe
4      0   control.exe
4      0   convert.exe
4      0   cscript.exe
5      0   csrss.exe
4      0   ctfmon.exe
4      0   dcomcnfg.exe
4      1   ddeshare.exe
4      0   defrag.exe
4      0   dfrgfat.exe
4      0   dfrgntfs.exe
4      3   diantz.exe
4      0   diskpart.exe
4      0   diskperf.exe
4      0   dllhost.exe
4      0   dllhst3g.exe
4      1   dmadmin.exe
4      0   dmremote.exe
4      0   doskey.exe
3      0   dplaysvr.exe
4      0   dpnsvr.exe
4      1   dpvsetup.exe
5      1   drwtsn32.exe
4      0   dumprep.exe
4      0   dvdplay.exe
4      0   dvdupgrd.exe
4      0   dwwin.exe
4      0   dxdiag.exe
4      0   dxdllreg.exe
4      1   esentutl.exe
4      0   eudcedit.exe
4      0   eventvwr.exe
4      0   expand.exe
4      4   extrac32.exe
4      1   fc.exe
4      0   find.exe
4      3   findstr.exe
4      0   finger.exe
4      0   fixmapi.exe
4      0   fontview.exe
4      1   forcedos.exe
4      0   freecell.exe
4      0   fsutil.exe
4      5   ftp.exe
4      0   fxsclnt.exe
4      0   fxscover.exe
4      0   fxssend.exe
4      0   fxssvc.exe
4      0   GkSui18.EXE
4      0   grpconv.exe
4      0   help.exe
4      0   hkcmd.exe
4      0   hostname.exe
```

```
4     0   ie4uinit.exe
4     0   iexpress.exe
4     0   igfxcfg.exe
4     0   igfxdiag.exe
4     0   igfxtray.exe
4     0   imapi.exe
4     0   InstallDriver.exe
4     1   InstUtl.exe
4     1   ipconfig.exe
4     0   ipsec6.exe
4     0   ipv6.exe
4     1   ipxroute.exe
3     2   java.exe
3     2   javaw.exe
5     0   jdbgmgr.exe
5     0   jview.exe
4     0   label.exe
4     0   lights.exe
4     0   lnkstub.exe
4     1   locator.exe
4     0   lodctr.exe
4     5   logagent.exe
4     0   logoff.exe
5     0   logonui.exe
4     0   lpq.exe
4     0   lpr.exe
4     0   lsass.exe
3     0   ltremove.exe
4     0   magnify.exe
4     3   makecab.exe
4     0   MAPISRVR.EXE
4     0   migpwd.exe
4     0   mmc.exe
4     0   mnmsrvc.exe
4     1   mobsync.exe
4     0   mountvol.exe
4     0   mplay32.exe
4     1   mpnotify.exe
4     0   mrinfo.exe
4     0   msdtc.exe
4     1   msg.exe
4     0   mshearts.exe
3     0   mshta.exe
4     0   msiexec.exe
5     0   mspaint.exe
5     7   MsPMSPSv.exe
4     0   msswchx.exe
4     0   mstinit.exe
4     0   mstsc.exe
4     0   narrator.exe
4     3   nbtstat.exe
4     0   nddeapir.exe
5     1   net.exe
4     2   net1.exe
4     4   netdde.exe
```

```
4      0   netsetup.exe
4      1   netsh.exe
4      2   netstat.exe
4      0   notepad.exe
4      5   nslookup.exe
5      1   ntkrnlpa.exe
5      1   ntoskrnl.exe
5      7   ntsd.exe
4      0   ntvdm.exe
4      0   nvsvc32.exe
4      1   odbcad32.exe
4      0   odbcconf.exe
4      0   osk.exe
4      0   osuninst.exe
4      0   packager.exe
4      0   pathping.exe
4      1   pentnt.exe
4      0   perfmon.exe
4      0   ping.exe
4      0   ping6.exe
4      0   print.exe
4      0   progman.exe
4      0   proquota.exe
4      0   ps2.EXE
3      0   pxhpinst.exe
4      1   qappsrv.exe
4      0   qprocess.exe
3      0   qttask.exe
4      0   qwinsta.exe
4      1   rasautou.exe
4      2   rasdial.exe
4      0   rasphone.exe
4      0   rcimlby.exe
4      2   rcp.exe
4      1   rdpclip.exe
4      0   rdsaddin.exe
4      0   rdshost.exe
4      0   recover.exe
4      0   reg.exe
4      0   regedt32.exe
4      0   regini.exe
4      0   regsvr32.exe
4      0   regwiz.exe
4      0   replace.exe
4      0   reset.exe
4      0   rexec.exe
4      2   route.exe
4      0   routemon.exe
4      0   rsh.exe
4      0   rsm.exe
4      0   rsmsink.exe
4      0   rsmui.exe
4      4   rsvp.exe
4      0   rtcshare.exe
4      0   runas.exe
```

```
4      0   rundll32.exe
4      0   runonce.exe
4      0   rwinsta.exe
4      0   S3tray2.exe
4      0   S3Uninst.exe
4      1   savedump.exe
4      0   sc.exe
5      1   scardsvr.exe
4      0   sdbinst.exe
5      0   services.exe
4      0   sessmgr.exe
4      0   sethc.exe
4      0   setup.exe
4      0   sfc.exe
4      0   shadow.exe
4      0   shmgrate.exe
4      0   shrpubw.exe
4      0   shutdown.exe
4      0   sigverif.exe
4      0   skeys.exe
4      0   smlogsvc.exe
5      1   smss.exe
4      0   sndrec32.exe
4      0   sndvol32.exe
4      0   sol.exe
4      0   sort.exe
4      0   spider.exe
5      0   spoolsv.exe
4      0   SPORDER.EXE
6      0   sprestrt.exe
3      0   stimon.exe
4      0   subst.exe
5      0   svchost.exe
4      0   syncapp.exe
4      0   syskey.exe
4      0   sysocmgr.exe
5      0   systray.exe
4      0   taskman.exe
5      0   taskmgr.exe
4      0   tcmsetup.exe
4      0   tcpsvcs.exe
4      1   telnet.exe
4      3   tftp.exe
4      0   tourstart.exe
4      0   tracert.exe
4      0   tracert6.exe
4      0   tscon.exe
4      0   tscupgrd.exe
4      0   tsdiscon.exe
4      0   tskill.exe
4      1   tsshutdn.exe
4      0   Twunk_32.exe
4      0   unlodctr.exe
4      0   upnpcont.exe
4      0   ups.exe
```

```
5     0   userinit.exe
4     0   usrmlnka.exe
4     0   usrprbda.exe
4     0   usrshuta.exe
4     0   utilman.exe
4     0   verifier.exe
4     1   vssadmin.exe
4     1   vssvc.exe
4     0   w32tm.exe
3     0   wextract.exe
4     0   wiaacmgr.exe
4     0   winchat.exe
4     0   winhlp32.exe
5     2   winlogon.exe
4     0   winmine.exe
4     0   winmsd.exe
4     0   winver.exe
5     0   wjview.exe
4     0   wmpstub.exe
4     0   wpabaln.exe
4     0   wpnpinst.exe
4     0   write.exe
4     1   wscript.exe
4     1   wuauclt.exe
4     0   wupdmgr.exe
4     0   xcopy.exe
4     4   XMNT2001.EXE
4     1   xpsp1hfm.exe
```

13. APPENDIX G – TEST RESULTS FROM ANALYZING MICROSOFT APPLICATION FILES

13.1 Dynamic Link Library (DLL) Files for Microsoft Office 2000

---------- Summary of File Security Analysis ----------

Total number of files submitted: 50

List of files containing anomalies (A), vulnerabilities (V) or risks (R)

A	V/R	Filename
5	0	ADJDATE.DLL
5	0	ANLYZTS.DLL
6	1	ATLCONV.DLL
5	0	AW.DLL
5	0	BLNMGR.DLL
5	0	BLNMGRPS.DLL
5	0	DBCONV.DLL
5	4	DLGSETP.DLL
6	1	ENVELOPE.DLL
4	0	HLP95EN.DLL
5	8	IMPMAIL.DLL
5	0	MDHELPER.DLL
5	3	MIMEDIR.DLL
5	0	MLSHEXT.DLL
5	0	MSDETECT.DLL
8	1	MSO9.DLL
4	0	MSO97FX.DLL
5	0	MSODRAA9.DLL
5	0	MSOHEV.DLL
7	1	MSOWC.DLL
5	0	MSOWCF.DLL
5	0	MSOWCW.DLL
5	1	OLKFSTUB.DLL
5	1	OUTLACCT.DLL
5	0	OUTLAS9.DLL
5	2	OUTLCTL.DLL
8	6	OUTLLIB.DLL
6	1	OUTLMIME.DLL
6	2	OUTLRPC.DLL
5	0	OUTLVBS.DLL

```
6     0   OWS.DLL
5     0   OWSDSC.DLL
5     0   PDIGRAPH.DLL
5     0   PERTANL.DLL
6     1   PJ9OD9.DLL
6     1   PJ9TM9.DLL
5     0   PJBKND09.DLL
5     0   PJPROTS.DLL
6     1   PRJRES9.DLL
5     2   RECALL.DLL
4     0   REFEDIT.DLL
6     3   RTFHTML.DLL
4     0   SELFREG.DLL
6     1   SENDTO9.DLL
6     1   SERCONV.DLL
5     0   STARTWIZ.DLL
5     0   WEBPAGE.DLL
3     0   XLCALL32.DLL
```

13.2 Executable (EXE) Files for Microsoft Office 2000

```
FILE NAME: EXCEL.EXE


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 5712
bytes exists starting at address 8192; this table often does not
appear in an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 210 bytes in size when actually it is
15413 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 134076 bytes in size when actually
it is 135168 bytes in size
  - The data directory table in the optional header states that the
(** Zero-filled region **) is 3373 bytes in size when actually it is
7468 bytes in size

**** End of Anomalies ****


!!!! Security Vulnerabilities and Risks!!!!

  - Contains 3373 bytes of unused zero-filled space that could be used
to store malicious code or data

!!!! End of Security Vulnerabilities and Risks!!!!
```

FILE NAME: FINDER.EXE

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 140 bytes exists starting at address 4096; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 60 bytes in size when actually it is 742 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 2416 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: GRAPH9.EXE

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 3200 bytes exists starting at address 8192; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 210 bytes in size when actually it is 10939 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 24204 bytes in size when actually it is 24576 bytes in size
- The data directory table in the optional header states that the (** Zero-filled region **) is 3373 bytes in size when actually it is 7468 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Contains 3373 bytes of unused zero-filled space that could be used to store malicious code or data

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: MSO7FTP.EXE

**** Anomalies ****

- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1284 bytes in size when actually it is 1536 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 8 bytes in size when actually it is 512 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: MSO7FTPA.EXE

**** Anomalies ****

- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1284 bytes in size when actually it is 1536 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 8 bytes in size when actually it is 512 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: MSO7FTPS.EXE

**** Anomalies ****

- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1284 bytes in size when actually it is 1536 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 8 bytes in size when actually it is 512 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: MSOHTMED.EXE

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 408 bytes exists starting at address 33276; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 100 bytes in size when actually it is 1468 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 2054 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: OSA9.EXE

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 460 bytes exists starting at address 4096; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 120 bytes in size when actually it is 2320 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 31512 bytes in size when actually it is 32768 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: OUTLOOK.EXE

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 140 bytes exists starting at address 4096; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 60 bytes in size when actually it is 742 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 26004 bytes in size when actually it is 28672 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: PJSPOOL.EXE

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 676 bytes exists starting at address 81920; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 160 bytes in size when actually it is 3304 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 8840 bytes in size when actually it is 12288 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: POWERPNT.EXE

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates a thread local storage table exists consisting of 24 bytes; this table usually does not appear in an image file so it was not read and only its start address was mapped
- The file indicates an import address table consisting of 4940 bytes exists starting at address 8192; this table often does not appear in an image file so it was not read and it was also not mapped

- The data directory table in the optional header states that the Import Table (.idata section) is 244 bytes in size when actually it is 13650 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 84952 bytes in size when actually it is 86016 bytes in size
- The data directory table in the optional header states that the (** Zero-filled region **) is 3333 bytes in size when actually it is 7428 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Contains 3333 bytes of unused zero-filled space that could be used to store malicious code or data

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: WAVTOASF.EXE

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The data directory table in the optional header states that the DOS Header is 64 bytes in size when actually it is 52488 bytes in size
- The file indicates an import address table consisting of 332 bytes exists starting at address 1536; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 140 bytes in size when actually it is 1529 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 4056 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Contains 446 bytes of unused zero-filled space that could be used to store malicious code or data
- Uses 4 standard C functions susceptible to buffer overflow attacks: fgets (Low risk), sprintf (Very high risk), sscanf (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: WINPROJ.EXE

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 3840 bytes exists starting at address 5775360; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 304 bytes in size when actually it is 13945 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 14864 bytes in size when actually it is 16384 bytes in size
- The data directory table in the optional header states that the (** Zero-filled region **) is 3201 bytes in size when actually it is 7296 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Contains 3201 bytes of unused zero-filled space that could be used to store malicious code or data

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: WINWORD.EXE

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates a thread local storage table exists consisting of 24 bytes; this table usually does not appear in an image file so it was not read and only its start address was mapped
- The file indicates an import address table consisting of 6568 bytes exists starting at address 8192; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 276 bytes in size when actually it is 17654 bytes in size
- A section entry named .CRT appears in the section table, but the table doesn't contain the location of the 8 bytes for that section

```
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 73592 bytes in size when actually it
is 77824 bytes in size
  - The data directory table in the optional header states that the
(** Zero-filled region **) is 3249 bytes in size when actually it is
7344 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

  - Contains 3249 bytes of unused zero-filled space that could be used
to store malicious code or data

!!!! End of Security Vulnerabilities and Risks!!!!
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 14

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)
```

A	V/R	Filename
5	1	EXCEL.EXE
4	0	FINDER.EXE
5	1	GRAPH9.EXE
2	0	MSO7FTP.EXE
2	0	MSO7FTPA.EXE
2	0	MSO7FTPS.EXE
4	0	MSOHTMED.EXE
4	0	OSA9.EXE
4	0	OUTLOOK.EXE
4	0	PJSPOOL.EXE
6	1	POWERPNT.EXE
5	5	WAVTOASF.EXE
5	1	WINPROJ.EXE
7	1	WINWORD.EXE

13.3 Dynamic Link Library (DLL) Files for Microsoft Outlook Express

---

```
FILE NAME: MSOE.DLL

**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
```

- The file indicates an import address table consisting of 2744 bytes exists starting at address 1024; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 220 bytes in size when actually it is 13851 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 9368 bytes in size when actually it is 9728 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 49332 bytes in size when actually it is 55808 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: msoeres.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 2477960 bytes in size when actually it is 2478080 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 8 bytes in size when actually it is 512 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: oeimport.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 588 bytes exists starting at address 1024; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 180 bytes in size when actually it is 3152 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 4664 bytes in size when actually it is 5120 bytes in size

- The data directory table in the optional header states that the Relocation Table (.reloc section) is 5960 bytes in size when actually it is 8192 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: oemiglib.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 328 bytes exists starting at address 1024; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 120 bytes in size when actually it is 1792 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 3680 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 1288 bytes in size when actually it is 2048 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: wabfind.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 108 bytes exists starting at address 1024; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 120 bytes in size when actually it is 647 bytes in size
- A section entry named .data appears in the section table, but the table doesn't contain the location of the 20 bytes for that section
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 25112 bytes in size when actually it is 25600 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 216 bytes in size when actually it is 512 bytes in size

```
**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
FILE NAME: wabimp.dll


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 316 bytes
exists starting at address 1024; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 120 bytes in size when actually it is
1661 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 6400 bytes in size when actually it
is 6656 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 3028 bytes in size when actually
it is 3584 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 6

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     5     0   MSOE.DLL
     3     0   msoeres.dll
     5     0   oeimport.dll
     5     0   oemiglib.dll
     6     0   wabfind.dll
     5     0   wabimp.dll
```

13.4 Executable (EXE) Files for Microsoft Outlook Express

FILE NAME: msimn.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 152 bytes exists starting at address 1024; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 100 bytes in size when actually it is 705 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 48824 bytes in size when actually it is 49152 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: oemig50.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 460 bytes exists starting at address 1024; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 120 bytes in size when actually it is 2493 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 8304 bytes in size when actually it is 8704 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: setup50.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information

- The file indicates an import address table consisting of 464 bytes exists starting at address 1024; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 140 bytes in size when actually it is 2537 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 4768 bytes in size when actually it is 5120 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: wab.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 296 bytes exists starting at address 1024; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 120 bytes in size when actually it is 1573 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 26664 bytes in size when actually it is 27136 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: wabmig.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 308 bytes exists starting at address 1024; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 100 bytes in size when actually it is 1387 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 2352 bytes in size when actually it is 2560 bytes in size

```
**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 5

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     4     0   msimn.exe
     4     0   oemig50.exe
     4     0   setup50.exe
     4     0   wab.exe
     4     0   wabmig.exe
```

13.5 Dynamic Link Library (DLL) Files for Windows Internet Explorer Plugins

---

```
FILE NAME: NPDocBox.dll


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 848 bytes
exists starting at address 118784; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 180 bytes in size when actually it is
3417 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 60952 bytes in size when actually it
is 61440 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 10032 bytes in size when actually
it is 12288 bytes in size

**** End of Anomalies ****


!!!! Security Vulnerabilities and Risks!!!!
```

- Uses 7 standard C functions susceptible to buffer overflow attacks: memcpy (Low risk), snprintf (Low risk), sprintf (Very high risk), sscanf (Very high risk), strcat (Very high risk), strcpy (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: nppdf32.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 412 bytes exists starting at address 69632; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 100 bytes in size when actually it is 2188 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 2776 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 2788 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Contains 486 bytes of unused zero-filled space that could be used to store malicious code or data

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: npqtplugin.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 228 bytes exists starting at address 45056; this table often does not appear in an image file so it was not read and it was also not mapped

- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 1240 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 4640 bytes in size when actually it is 8192 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 3092 bytes in size when actually it is 12288 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: npqtplugin2.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 228 bytes exists starting at address 45056; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 1240 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 4640 bytes in size when actually it is 8192 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 3092 bytes in size when actually it is 12288 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: npqtplugin3.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 228 bytes exists starting at address 45056; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 1240 bytes in size

- The data directory table in the optional header states that the Resource Table (.rsrc section) is 4640 bytes in size when actually it is 8192 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 3092 bytes in size when actually it is 12288 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: npqtplugin4.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 228 bytes exists starting at address 45056; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 1240 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 4640 bytes in size when actually it is 8192 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 3092 bytes in size when actually it is 12288 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: npqtplugin5.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 228 bytes exists starting at address 45056; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 1240 bytes in size

```
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 4640 bytes in size when actually it
is 8192 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 3092 bytes in size when actually
it is 12288 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 7

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     5     7   NPDocBox.dll
     5     1   nppdf32.dll
     5     0   npqtplugin.dll
     5     0   npqtplugin2.dll
     5     0   npqtplugin3.dll
     5     0   npqtplugin4.dll
     5     0   npqtplugin5.dll
```

13.6 Executable (EXE) Files for Windows Internet Explorer

```
FILE NAME: IEXPLORE.EXE


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 280 bytes
exists starting at address 1024; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 120 bytes in size when actually it is
1337 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 83784 bytes in size when actually it
is 83968 bytes in size

**** End of Anomalies ****
```

```
No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 1

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     4     0   IEXPLORE.EXE
```

13.7 Dynamic Link Library (DLL) Files for Windows Media Player 9

```
FILE NAME: custsat.dll


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 668 bytes
exists starting at address 4096; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 180 bytes in size when actually it is
2650 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 21384 bytes in size when actually it
is 24576 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 4160 bytes in size when actually
it is 8192 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
FILE NAME: npdrmv2.dll


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
```

- The file indicates an import address table consisting of 348 bytes exists starting at address 1024; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 160 bytes in size when actually it is 1739 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1176 bytes in size when actually it is 1536 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 4124 bytes in size when actually it is 6144 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 3 standard C functions susceptible to buffer overflow attacks: memcpy (Low risk), strcat (Very high risk), strcpy (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: npdsplay.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1228 bytes exists starting at address 225280; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 240 bytes in size when actually it is 6372 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 11800 bytes in size when actually it is 12288 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 19200 bytes in size when actually it is 28672 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: npwmsdrm.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 140 bytes exists starting at address 1024; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 120 bytes in size when actually it is 633 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1168 bytes in size when actually it is 1536 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 336 bytes in size when actually it is 512 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses one standard C function susceptible to buffer overflow attack: sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: pidgen.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 124 bytes exists starting at address 1024; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 664 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1784 bytes in size when actually it is 2048 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 172 bytes in size when actually it is 512 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: wmpband.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 844 bytes exists starting at address 4096; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 240 bytes in size when actually it is 4398 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 3288 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 3860 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: wmpns.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 904 bytes exists starting at address 4096; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 220 bytes in size when actually it is 4665 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 984 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 7312 bytes in size when actually it is 16384 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: wmpvis.dll

```
**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 328 bytes
exists starting at address 90112; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 160 bytes in size when actually it is
1570 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 413032 bytes in size when actually
it is 413696 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 2292 bytes in size when actually
it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 9

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)
```

A	V/R	Filename
5	0	custsat.dll
5	3	npdrmv2.dll
5	0	npdsplay.dll
5	1	npwmsdrm.dll
5	0	pidgen.dll
5	0	wmpband.dll
5	0	wmpns.dll
5	0	wmpvis.dll

13.8 Executable (EXE) Files for Windows Media Player 9

```
FILE NAME: dlimport.exe


**** Anomalies ****
```

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 984 bytes exists starting at address 212992; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 240 bytes in size when actually it is 4472 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 2032 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 3 standard C functions susceptible to buffer overflow attacks: memcpy (Low risk), strcat (Very high risk), strcpy (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: migrate.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1444 bytes exists starting at address 4096; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 300 bytes in size when actually it is 7543 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 992 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 4 standard C functions susceptible to buffer overflow attacks: memcpy (Low risk), strcat (Very high risk), strcpy (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: mplayer2.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 16 bytes exists starting at address 1536; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 60 bytes in size when actually it is 114 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 2320 bytes in size when actually it is 2560 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: setup_wm.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1652 bytes exists starting at address 4096; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 360 bytes in size when actually it is 8539 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 471576 bytes in size when actually it is 475136 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 2 standard C functions susceptible to buffer overflow attacks: strncpy (Low risk), vsnprintf (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

```
FILE NAME: wmplayer.exe


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 192 bytes
exists starting at address 4096; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The file indicates a delay import descriptor consisting of 96
bytes exists starting at address 7844; this item often does not appear
in an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 80 bytes in size when actually it is
1055 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 54080 bytes in size when actually it
is 57344 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 5

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     4     3   dlimport.exe
     4     4   migrate.exe
     4     0   mplayer2.exe
     4     2   setup_wm.exe
     5     0   wmplayer.exe
```

13.9 Dynamic Link Library (DLL) Files for Windows Messenger

```
FILE NAME: msgsc.dll


**** Anomalies ****
```

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 316 bytes exists starting at address 8192; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 140 bytes in size when actually it is 1666 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1160 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 3184 bytes in size when actually it is 8192 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: msgslang.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The data directory table in the optional header states that the Debug Table (.debug section) is 28 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 206056 bytes in size when actually it is 208896 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 8 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: rtcimsp.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 740 bytes exists starting at address 1024; this table often does not appear in an image file so it was not read and it was also not mapped

```
  - The data directory table in the optional header states that the
Import Table (.idata section) is 140 bytes in size when actually it is
2361 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 31104 bytes in size when actually it
is 31232 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 7272 bytes in size when actually
it is 13312 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 3

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     5     0   msgsc.dll
     4     0   msgslang.dll
     5     0   rtcimsp.dll
```

13.10 Executable (EXE) Files for Windows Messenger

```
_

FILE NAME: msmsgs.exe


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 1876
bytes exists starting at address 4096; this table often does not
appear in an image file so it was not read and it was also not mapped
  - The file indicates a delay import descriptor consisting of 160
bytes exists starting at address 861548; this item often does not
appear in an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 280 bytes in size when actually it is
9819 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 571616 bytes in size when actually
it is 573440 bytes in size
```

```
**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
FILE NAME: msmsgsin.exe


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 220 bytes
exists starting at address 4096; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 100 bytes in size when actually it is
1113 bytes in size
  - A section entry named .data appears in the section table, but the
table doesn't contain the location of the 4 bytes for that section

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 2

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     5     0   msmsgs.exe
     4     0   msmsgsin.exe
```

13.11 Dynamic Link Library (DLL) Files for Windows MovieMaker

---

```
FILE NAME: wmmfilt.dll


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
```

- The file indicates an import address table consisting of 468 bytes exists starting at address 81920; this table often does not appear in an image file so it was not read and it was also not mapped
- The file indicates a delay import descriptor consisting of 64 bytes exists starting at address 102872; this item often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 120 bytes in size when actually it is 2485 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 2104 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 5052 bytes in size when actually it is 8192 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses one standard C function susceptible to buffer overflow attack: memcpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: wmmres.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The data directory table in the optional header states that the Debug Table (.debug section) is 28 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 306200 bytes in size when actually it is 307200 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 8 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: wmmutil.dll

```
**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 772 bytes
exists starting at address 45056; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 160 bytes in size when actually it is
3983 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 944 bytes in size when actually it
is 4096 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 2904 bytes in size when actually
it is 4096 bytes in size

**** End of Anomalies ****



!!!! Security Vulnerabilities and Risks!!!!

  - Uses 2 standard C functions susceptible to buffer overflow
attacks: memcpy (Low risk), strcpy (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 3

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     6     1   wmmfilt.dll
     4     0   wmmres.dll
     5     2   wmmutil.dll
```

13.12 Executable (EXE) Files for Windows MovieMaker

```
FILE NAME: moviemk.exe



**** Anomalies ****
```

```
  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 1188
bytes exists starting at address 688128; this table often does not
appear in an image file so it was not read and it was also not mapped
  - The file indicates a delay import descriptor consisting of 448
bytes exists starting at address 773624; this item often does not
appear in an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 180 bytes in size when actually it is
6643 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 27392 bytes in size when actually it
is 28672 bytes in size

**** End of Anomalies ****



!!!! Security Vulnerabilities and Risks!!!!

  - Uses 5 standard C functions susceptible to buffer overflow
attacks: memcpy (Low risk), sprintf (Very high risk), sscanf (Very
high risk), strcpy (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 1

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     5     5   moviemk.exe
```

13.13 Dynamic Link Library (DLL) Files for Windows NetMeeting

---

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 15

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)
```

```
A   V/R   Filename
-   ---   --------
5     0   callcont.dll
5     0   confmrsl.dll
5     0   dcap32.dll
5     0   h323cc.dll
5     1   mst120.dll
6     0   MST123.DLL
5     0   nac.dll
5     0   nmas.dll
6     0   nmasnt.dll
5     0   nmchat.dll
5     0   nmcom.dll
5     0   nmft.dll
5     0   nmoldwb.dll
5     0   nmwb.dll
5     0   rrcm.dll
```

13.14 Executable (EXE) Files for Windows NetMeeting

```
FILE NAME: cb32.exe


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 24 bytes
exists starting at address 4096; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 60 bytes in size when actually it is
129 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 1184 bytes in size when actually it
is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
FILE NAME: conf.exe


**** Anomalies ****
```

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1860 bytes exists starting at address 4096; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 340 bytes in size when actually it is 9818 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 672040 bytes in size when actually it is 675840 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: wb32.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 52 bytes exists starting at address 4096; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 100 bytes in size when actually it is 281 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1200 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

---------- Summary of File Security Analysis ----------

Total number of files submitted: 3

List of files containing anomalies (A), vulnerabilities (V) or risks (R)

A	V/R	Filename
4	0	cb32.exe
4	0	conf.exe
4	0	wb32.exe

14. APPENDIX H – TEST RESULTS FROM ANALYZING SECURITY-CENTRIC APPLICATION FILES

14.1 Dynamic Link Libraries (DLL) Files for Network Associates Common Framework

---

---------- Summary of File Security Analysis ----------

Total number of files submitted: 35

List of files containing anomalies (A), vulnerabilities (V) or risks (R)

A	V/R	Filename
5	2	Agent.dll
5	3	AgentPlugin.dll
4	1	ClientUI.dll
5	1	ComponentSubSystem.dll
5	0	ComponentUserInterface.dll
5	2	FrmPlugin.dll
6	3	GenEvtInf.dll
6	5	InternetManager.dll
5	0	LicWrap.dll
6	5	ListenServer.dll
5	2	Logging.dll
5	2	Management.dll
5	0	mcurial.dll
5	5	naCmnLib.dll
5	2	nagshr32.dll
5	2	naicrt32.dll
5	2	nailog.dll
5	2	naInet.dll
5	4	naisign.dll
5	2	naPolicyManager.dll
5	1	naSPIPE.dll
5	2	naXML.dll
5	2	nmcomn32.dll
2	0	patchw32.dll
5	0	PcrPlug.dll
5	3	PoEvtInf.dll
5	1	PSAPI.dll
5	4	Scheduler.dll

```
5     2   ScriptSubSys.dll
5     2   SecureFrameworkFactory.dll
5     0   unicows.dll
5     2   UpdateSubSys.dll
5     1   UpdPlug.dll
6     2   UserSpace.dll
5     3   XMLWrap.dll
```

14.2 Exectutable (EXE) Files for Network Associates Common Framework

---

```
FILE NAME: Cleanup.exe

**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 204 bytes
exists starting at address 8192; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 100 bytes in size when actually it is
1058 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 1032 bytes in size when actually it
is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
FILE NAME: CmdAgent.exe

**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 192 bytes
exists starting at address 24576; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 60 bytes in size when actually it is
963 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 1000 bytes in size when actually it
is 4096 bytes in size

**** End of Anomalies ****
```

No security vulnerabilities or security risks were found

FILE NAME: FrameworkService.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 756 bytes exists starting at address 65536; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 200 bytes in size when actually it is 4965 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 10120 bytes in size when actually it is 12288 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 2 standard C functions susceptible to buffer overflow attacks: memcpy (Low risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: FrmInst.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 808 bytes exists starting at address 86016; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 200 bytes in size when actually it is 6379 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 4736 bytes in size when actually it is 8192 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 2 standard C functions susceptible to buffer overflow attacks: memcpy (Low risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: McScript.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 988 bytes exists starting at address 126976; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 180 bytes in size when actually it is 4817 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1192 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 10 standard C functions susceptible to buffer overflow attacks: getc (Medium risk), memcpy (Low risk), read (Medium risk), snprintf (Low risk), sprintf (Very high risk), sscanf (Very high risk), strcat (Very high risk), strcpy (Very high risk), strncpy (Low risk), vsnprintf (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: naPrdMgr.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 576 bytes exists starting at address 77824; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 200 bytes in size when actually it is 3802 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 9208 bytes in size when actually it is 12288 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 3 standard C functions susceptible to buffer overflow attacks: memcpy (Low risk), strcpy (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: UpdaterUI.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 860 bytes exists starting at address 81920; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 220 bytes in size when actually it is 5451 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 15000 bytes in size when actually it is 16384 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 2 standard C functions susceptible to buffer overflow attacks: memcpy (Low risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

---------- Summary of File Security Analysis ----------

Total number of files submitted: 7

List of files containing anomalies (A), vulnerabilities (V) or risks (R)

A	V/R	Filename
4	0	Cleanup.exe
4	0	CmdAgent.exe
4	2	FrameworkService.exe
4	2	FrmInst.exe

```
4    10   McScript.exe
4     3   naPrdMgr.exe
4     2   UpdaterUI.exe
```

14.3 Dynamic Link Libraries (DLL) Files for Network Associates VirusScan 7.0

```
FILE NAME: adslokuu.dll


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 372 bytes
exists starting at address 53248; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 160 bytes in size when actually it is
881 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 952 bytes in size when actually it
is 4096 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 3460 bytes in size when actually
it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
FILE NAME: ftcfg.dll

**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 468 bytes
exists starting at address 90112; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 140 bytes in size when actually it is
2154 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 936 bytes in size when actually it
is 4096 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 6572 bytes in size when actually
it is 12288 bytes in size
```

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: ftl.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates a thread local storage table exists consisting of 24 bytes; this table usually does not appear in an image file so it was not read and only its start address was mapped
- The file indicates an import address table consisting of 348 bytes exists starting at address 106496; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 1951 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 912 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 5824 bytes in size when actually it is 8192 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: midutil.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates a thread local storage table exists consisting of 24 bytes; this table usually does not appear in an image file so it was not read and only its start address was mapped
- The file indicates an import address table consisting of 512 bytes exists starting at address 65536; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 120 bytes in size when actually it is 2843 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1064 bytes in size when actually it is 4096 bytes in size

- The data directory table in the optional header states that the Relocation Table (.reloc section) is 3524 bytes in size when actually it is 8192 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: naeventu.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 860 bytes exists starting at address 126976; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 220 bytes in size when actually it is 3632 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 944 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 7192 bytes in size when actually it is 8192 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: naiann.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 536 bytes exists starting at address 61440; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 140 bytes in size when actually it is 2940 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1048 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 5780 bytes in size when actually it is 8192 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: naievent.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 37640 bytes in size when actually it is 40960 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 8 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: nailite.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 652 bytes exists starting at address 196608; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 120 bytes in size when actually it is 2778 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 442880 bytes in size when actually it is 446464 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 12324 bytes in size when actually it is 16384 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: nakrnlu.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 432 bytes exists starting at address 69632; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 100 bytes in size when actually it is 1291 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 928 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 3904 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: nautilu.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 760 bytes exists starting at address 126976; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 200 bytes in size when actually it is 3101 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 928 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 8568 bytes in size when actually it is 12288 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: ntclient.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 416 bytes exists starting at address 49152; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 100 bytes in size when actually it is 1061 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 936 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 3468 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: scanemal.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 940 bytes exists starting at address 167936; this table often does not appear in an image file so it was not read and it was also not mapped
- The file indicates a delay import descriptor consisting of 64 bytes exists starting at address 175496; this item often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 160 bytes in size when actually it is 4167 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 928 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 9220 bytes in size when actually it is 12288 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: shext.dll

**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub contains some kind of information
  - The file indicates an import address table consisting of 356 bytes exists starting at address 32768; this table often does not appear in an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the Import Table (.idata section) is 120 bytes in size when actually it is 967 bytes in size
  - The data directory table in the optional header states that the Resource Table (.rsrc section) is 904 bytes in size when actually it is 4096 bytes in size
  - The data directory table in the optional header states that the Relocation Table (.reloc section) is 1752 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: shutil.dll

**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub contains some kind of information
  - The file indicates an import address table consisting of 876 bytes exists starting at address 143360; this table often does not appear in an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the Import Table (.idata section) is 240 bytes in size when actually it is 3560 bytes in size
  - The data directory table in the optional header states that the Resource Table (.rsrc section) is 904 bytes in size when actually it is 4096 bytes in size
  - The data directory table in the optional header states that the Relocation Table (.reloc section) is 12340 bytes in size when actually it is 16384 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: VS7Plugin.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 600 bytes exists starting at address 106496; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 160 bytes in size when actually it is 2205 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 3872 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 9276 bytes in size when actually it is 12288 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: vsplugin.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 616 bytes exists starting at address 106496; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 160 bytes in size when actually it is 2245 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 3872 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 9408 bytes in size when actually it is 12288 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

---------- Summary of File Security Analysis ----------

Total number of files submitted: 16

List of files containing anomalies (A), vulnerabilities (V) or risks (R)

A	V/R	Filename

```
-   ---   --------
5     0   adslokuu.dll
5     0   ftcfg.dll
6     0   ftl.dll
6     0   midutil.dll
5     0   naeventu.dll
5     0   naiann.dll
3     0   naievent.dll
5     0   nailite.dll
5     0   nakrnlu.dll
5     0   nautilu.dll
5     0   ntclient.dll
6     0   scanemal.dll
5     0   shext.dll
5     0   shutil.dll
5     0   VS7Plugin.dll
5     0   vsplugin.dll
```

14.4 Executable (EXE) Files for Network Associates VirusScan 7.0

```
FILE NAME: mcconsol.exe

**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 1016
bytes exists starting at address 90112; this table often does not
appear in an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 220 bytes in size when actually it is
5309 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 1776 bytes in size when actually it
is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
FILE NAME: mcshield.exe


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
```

- The file indicates a thread local storage table exists consisting of 24 bytes; this table usually does not appear in an image file so it was not read and only its start address was mapped
- The file indicates an import address table consisting of 876 bytes exists starting at address 163840; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 100 bytes in size when actually it is 4991 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 936 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: mcupdate.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 840 bytes exists starting at address 90112; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 220 bytes in size when actually it is 3311 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1776 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: naiavfin.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 248 bytes exists starting at address 12288; this table often does not appear in an image file so it was not read and it was also not mapped

- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 1169 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 960 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses one standard C function susceptible to buffer overflow attack: vsnprintf (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: scan32.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1208 bytes exists starting at address 196608; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 160 bytes in size when actually it is 5270 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1784 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: scncfg32.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 900 bytes exists starting at address 114688; this table often does not appear in an image file so it was not read and it was also not mapped

- The data directory table in the optional header states that the Import Table (.idata section) is 180 bytes in size when actually it is 3808 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1848 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: scnstat.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 540 bytes exists starting at address 36864; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 180 bytes in size when actually it is 2363 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1800 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: shcfg32.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 740 bytes exists starting at address 61440; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 240 bytes in size when actually it is 3877 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1808 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: shstat.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 860 bytes exists starting at address 65536; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 200 bytes in size when actually it is 4676 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1816 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: svcpwd.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 388 bytes exists starting at address 49152; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 729 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 5576 bytes in size when actually it is 8192 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: vstskmgr.exe

**** Anomalies ****

```
  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 792 bytes
exists starting at address 94208; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 140 bytes in size when actually it is
3461 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 976 bytes in size when actually it
is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 11

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     4     0   mcconsol.exe
     5     0   mcshield.exe
     4     0   mcupdate.exe
     4     1   naiavfin.exe
     4     0   scan32.exe
     4     0   scncfg32.exe
     4     0   scnstat.exe
     4     0   shcfg32.exe
     4     0   shstat.exe
     4     0   svcpwd.exe
     4     0   vstskmgr.exe
```

14.5 Dynamic Link Libraries (DLL) Files for Secure CRT 4.0

```
FILE NAME: ConnectDialog10.dll


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
```

- The file indicates an import address table consisting of 1068 bytes exists starting at address 65536; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 180 bytes in size when actually it is 2561 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 9504 bytes in size when actually it is 12288 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 4340 bytes in size when actually it is 8192 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: License33.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1040 bytes exists starting at address 61440; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 160 bytes in size when actually it is 2312 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 287136 bytes in size when actually it is 290816 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 7132 bytes in size when actually it is 12288 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 2 standard C functions susceptible to buffer overflow attacks: sprintf (Very high risk), sscanf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: Mfc42.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 2068 bytes exists starting at address 638976; this table often does not appear in an image file so it was not read and it was also not mapped
- The file indicates a delay import descriptor consisting of 896 bytes exists starting at address 648896; this item often does not appear in an image file so it was not read and it was also not mapped
- Invalid directory table data found in the export table; 6932 functions are being exported but the maximum allowed by this program is 2000
- The file indicates an export table exists but the export data could not be read
- The data directory table in the optional header states that the Import Table (.idata section) is 148 bytes in size when actually it is 7116 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 41600 bytes in size when actually it is 45056 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 61260 bytes in size when actually it is 61440 bytes in size
- The data directory table in the optional header states that the (** Zero-filled region **) is 3409 bytes in size when actually it is 7504 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Contains 3409 bytes of unused zero-filled space that could be used to store malicious code or data
- Uses 4 standard C functions susceptible to buffer overflow attacks: fgets (Low risk), memcpy (Low risk), sprintf (Very high risk), vsprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: Msvcrt.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 580 bytes exists starting at address 192512; this table often does not appear in an image file so it was not read and it was also not mapped

- The data directory table in the optional header states that the Import Table (.idata section) is 54 bytes in size when actually it is 3314 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 936 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 8908 bytes in size when actually it is 12288 bytes in size
- The data directory table in the optional header states that the (** Zero-filled region **) is 3433 bytes in size when actually it is 7528 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Contains 3433 bytes of unused zero-filled space that could be used to store malicious code or data

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: SSH2Client23.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1604 bytes exists starting at address 176128; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 180 bytes in size when actually it is 6801 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 21288 bytes in size when actually it is 24576 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 15960 bytes in size when actually it is 20480 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 2 standard C functions susceptible to buffer overflow attacks: strncpy (Low risk), vsprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

```
_

---------- Summary of File Security Analysis ----------

Total number of files submitted: 6

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     5     0   ConnectDialog10.dll
     5     2   License33.dll
     9     5   Mfc42.dll
     6     1   Msvcrt.dll
     5     2   SSH2Client23.dll
```

14.6 Executable (EXE) Files for Secure CRT 4.0

```
_

FILE NAME: ACTIVATOR.EXE


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 644 bytes
exists starting at address 57344; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 140 bytes in size when actually it is
2620 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 16192 bytes in size when actually it
is 16384 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

---------------------------------------------------------------------

FILE NAME: SecureCRT.EXE


**** Anomalies ****
```

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 3424 bytes exists starting at address 737280; this table often does not appear in an image file so it was not read and it was also not mapped
- The file indicates a delay import descriptor consisting of 64 bytes exists starting at address 881232; this item often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 260 bytes in size when actually it is 9035 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 154432 bytes in size when actually it is 155648 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 3 standard C functions susceptible to buffer overflow attacks: sprintf (Very high risk), sscanf (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: UNINSTAL.EXE

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 888 bytes exists starting at address 67072; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 160 bytes in size when actually it is 3899 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 75424 bytes in size when actually it is 75776 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: VCP.EXE

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 884 bytes exists starting at address 266240; this table often does not appear in an image file so it was not read and it was also not mapped
- The file indicates a delay import descriptor consisting of 64 bytes exists starting at address 315568; this item often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 200 bytes in size when actually it is 4309 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 8912 bytes in size when actually it is 12288 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 4 standard C functions susceptible to buffer overflow attacks: gets (Ultra high risk), sscanf (Very high risk), strncpy (Low risk), vsprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: VSH.EXE

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1000 bytes exists starting at address 262144; this table often does not appear in an image file so it was not read and it was also not mapped
- The file indicates a delay import descriptor consisting of 64 bytes exists starting at address 311288; this item often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 200 bytes in size when actually it is 4724 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 8848 bytes in size when actually it is 12288 bytes in size

**** End of Anomalies ****

```
!!!! Security Vulnerabilities and Risks!!!!

  - Uses 5 standard C functions susceptible to buffer overflow
attacks: gets (Ultra high risk), sprintf (Very high risk), sscanf
(Very high risk), strncpy (Low risk), vsprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 5

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     4     0   ACTIVATOR.EXE
     5     3   SecureCRT.EXE
     4     0   UNINSTAL.EXE
     5     4   VCP.EXE
     5     5   VSH.EXE
```

14.7 Dynamic Link Libraries (DLL) Files for SpyBot 1.2

```
FILE NAME: advcheck.dll


**** Anomalies ****

  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 7452 bytes in size when actually
it is 7680 bytes in size
  - The data directory table in the optional header states that the
BSS section is 0 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
FILE NAME: borlndmm.dll


**** Anomalies ****
```

- The data directory table in the optional header states that the Relocation Table (.reloc section) is 1216 bytes in size when actually it is 1536 bytes in size
- The data directory table in the optional header states that the BSS section is 0 bytes in size when actually it is 1175 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: delphimm.dll

**** Anomalies ****

- The data directory table in the optional header states that the Relocation Table (.reloc section) is 772 bytes in size when actually it is 1024 bytes in size
- The data directory table in the optional header states that the BSS section is 0 bytes in size when actually it is 1113 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: SDHelper.dll

**** Anomalies ****

- The data directory table in the optional header states that the Relocation Table (.reloc section) is 32940 bytes in size when actually it is 33280 bytes in size
- The data directory table in the optional header states that the BSS section is 0 bytes in size when actually it is 9725 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: Tools.dll

**** Anomalies ****

- The data directory table in the optional header states that the Relocation Table (.reloc section) is 24016 bytes in size when actually it is 24064 bytes in size

- The data directory table in the optional header states that the BSS section is 0 bytes in size when actually it is 9061 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: UnzDll.dll

**** Anomalies ****

- The data directory table in the optional header states that the Relocation Table (.reloc section) is 3384 bytes in size when actually it is 3584 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: ZipDll.dll

**** Anomalies ****

- The data directory table in the optional header states that the Relocation Table (.reloc section) is 3716 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

---------- Summary of File Security Analysis ----------

Total number of files submitted: 7

List of files containing anomalies (A), vulnerabilities (V) or risks (R)

A	V/R	Filename
2	0	advcheck.dll
2	0	borlndmm.dll
2	0	delphimm.dll
2	0	SDHelper.dll
2	0	Tools.dll
1	0	UnzDll.dll

```
     1     0   ZipDll.dll
```

14.8 Executable (EXE) Files for SpyBot 1.2

```
FILE NAME: blindman.exe


**** Anomalies ****

  - The file indicates a thread local storage table exists consisting
of 24 bytes; this table usually does not appear in an image file so it
was not read and only its start address was mapped
  - The data directory table in the optional header states that the
.tls section is 0 bytes in size when actually it is 512 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 728 bytes in size when actually
it is 1024 bytes in size
  - The data directory table in the optional header states that the
BSS section is 0 bytes in size when actually it is 811 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
FILE NAME: SpybotSD.exe


**** Anomalies ****

  - The file indicates a thread local storage table exists consisting
of 24 bytes; this table usually does not appear in an image file so it
was not read and only its start address was mapped
  - The data directory table in the optional header states that the
.tls section is 0 bytes in size when actually it is 512 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 114196 bytes in size when
actually it is 114688 bytes in size
  - The data directory table in the optional header states that the
BSS section is 0 bytes in size when actually it is 13312 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
FILE NAME: unins000.exe
```

**** Anomalies ****

- The file indicates a thread local storage table exists consisting of 24 bytes; this table usually does not appear in an image file so it was not read and only its start address was mapped
- The data directory table in the optional header states that the .tls section is 0 bytes in size when actually it is 512 bytes in size
- A section entry named .reloc appears in the section table, but the table doesn't contain the location of the 2588 bytes for that section
- The data directory table in the optional header states that the BSS section is 0 bytes in size when actually it is 4608 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: Update.exe

**** Anomalies ****

- The file indicates a thread local storage table exists consisting of 24 bytes; this table usually does not appear in an image file so it was not read and only its start address was mapped
- The data directory table in the optional header states that the .tls section is 0 bytes in size when actually it is 512 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 22652 bytes in size when actually it is 23040 bytes in size
- The data directory table in the optional header states that the BSS section is 0 bytes in size when actually it is 8704 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

---------- Summary of File Security Analysis ----------

Total number of files submitted: 4

List of files containing anomalies (A), vulnerabilities (V) or risks (R)

A	V/R	Filename
4	0	blindman.exe
4	0	SpybotSD.exe
4	0	unins000.exe
4	0	Update.exe

14.9 Executable (EXE) Files for WinSCP

```
FILE NAME: WinSCP.exe


**** Anomalies ****

  - The file indicates a thread local storage table exists consisting
of 24 bytes; this table usually does not appear in an image file so it
was not read and only its start address was mapped
  - Invalid directory table data found in the export table; 2817015669
functions are being exported but the maximum allowed by this program
is 2000
  - The file indicates an export table exists but the export data
could not be read
  - The data directory table in the optional header states that the
UPX0 section is 0 bytes in size when actually it is 391168 bytes in
size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 13872 bytes in size when actually it
is 14848 bytes in size

**** End of Anomalies ****



!!!! Security Vulnerabilities and Risks!!!!

  - Has a section named UPX0 whose contents can be both written to and
executed
  - Has a section named UPX1 whose contents can be both written to and
executed

!!!! End of Security Vulnerabilities and Risks!!!!
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 1

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     5     2   WinSCP.exe
```

14.10 Dynamic Link Libraries (DLL) Files for Zero Knowledge Freedom 3.0

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 34

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     5     0   AdblockR.dll
     5     0   BandObjs.dll
     5     0   ClntpR.dll
     4     1   ConvertR.dll
     6     2   CookieR.dll
     6     1   DialogsR.dll
     5     0   DInetR.dll
     5     2   FCryptR.dll
     5     2   FireR.dll
     5     2   FirewallUIR.dll
     5     2   FreeBHOR.dll
     5     1   frkyqryR.dll
     5     2   FrSecR.dll
     5     2   inethlpR.dll
     5     0   IpcSrvR.dll
     5     0   libbz2R.dll
     5     2   LibzkipR.dll
     5     2   libzkmR.dll
     5     4   NetWorkR.dll
     5     1   PacketR.dll
     5     1   PersistR.dll
     5     3   ProxiesR.dll
     5     3   ServiceR.dll
     5     1   SktShimR.dll
     5     1   TConfigR.dll
     5     2   TGenNetR.dll
     5     0   TTInfoR.dll
     6     0   WalletR.dll
     6     1   WordScnR.dll
     5     1   WzSetupR.dll
     5     2   YarrowR.dll
     5     0   zkrandR.dll
     5     1   ZKUIR.dll
     5     0   ZkYarrR.dll
```

14.11 Executable (EXE) Files for Zero Knowledge Freedom 3.0

```
FILE NAME: AutoStarterR.exe
```

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 452 bytes exists starting at address 8192; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 160 bytes in size when actually it is 3026 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: DiagR.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1412 bytes exists starting at address 45056; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 360 bytes in size when actually it is 7234 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 4048 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: Freedom.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 3136 bytes exists starting at address 53248; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 640 bytes in size when actually it is 36172 bytes in size

- The data directory table in the optional header states that the Resource Table (.rsrc section) is 4048 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses one standard C function susceptible to buffer overflow attack: strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: RestoreR.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 708 bytes exists starting at address 12288; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 160 bytes in size when actually it is 3514 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 4048 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: zkInstallDriver.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 160 bytes exists starting at address 1024; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 140 bytes in size when actually it is 883 bytes in size

```
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 1296 bytes in size when actually it
is 1536 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 5

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     3     0   AutoStarterR.exe
     4     0   DiagR.exe
     4     1   Freedom.exe
     4     0   RestoreR.exe
     4     0   zkInstallDriver.exe
```

14.12 Dynamic Link Libraries (DLL) Files for Zone Alarm Pro 4

```
FILE NAME: expert.dll


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 420 bytes
exists starting at address 90112; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 180 bytes in size when actually it is
2071 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 45792 bytes in size when actually it
is 49152 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 7004 bytes in size when actually
it is 8192 bytes in size

**** End of Anomalies ****
```

!!!! Security Vulnerabilities and Risks!!!!

- Uses 5 standard C functions susceptible to buffer overflow attacks: memcpy (Low risk), snprintf (Low risk), sprintf (Very high risk), strcpy (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: framewrk.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1320 bytes exists starting at address 241664; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 240 bytes in size when actually it is 6702 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 465456 bytes in size when actually it is 466944 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 19444 bytes in size when actually it is 28672 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 5 standard C functions susceptible to buffer overflow attacks: memcpy (Low risk), sprintf (Very high risk), strcat (Very high risk), strcpy (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: tutorwiz.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 172 bytes exists starting at address 20480; this table often does not appear in an image file so it was not read and it was also not mapped

```
  - The data directory table in the optional header states that the
Import Table (.idata section) is 140 bytes in size when actually it is
835 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 637520 bytes in size when actually
it is 638976 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 2116 bytes in size when actually
it is 4096 bytes in size

**** End of Anomalies ****


!!!! Security Vulnerabilities and Risks!!!!

  - Uses one standard C function susceptible to buffer overflow
attack: sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 3

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     5     5   expert.dll
     5     5   framewrk.dll
     5     1   tutorwiz.dll
```

14.13 Executable (EXE) Files for Zone Alarm Pro 4

```
FILE NAME: bbuninst.exe


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 668 bytes
exists starting at address 61440; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 140 bytes in size when actually it is
2649 bytes in size
```

- The data directory table in the optional header states that the Resource Table (.rsrc section) is 2632 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: runlink.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 368 bytes exists starting at address 28672; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 1269 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: zapro.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1020 bytes exists starting at address 135168; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 260 bytes in size when actually it is 5343 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 249944 bytes in size when actually it is 253952 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 5 standard C functions susceptible to buffer overflow attacks: memcpy (Low risk), sprintf (Very high risk), strcat (Very high risk), strcpy (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: zatutor.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 140 bytes exists starting at address 8192; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 120 bytes in size when actually it is 742 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 2464 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: zauninst.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 68 bytes exists starting at address 1536; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 60 bytes in size when actually it is 400 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 165228 bytes in size when actually it is 165376 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: zonealarm.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 104 bytes exists starting at address 8192; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 374 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 10656 bytes in size when actually it is 12288 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses one standard C function susceptible to buffer overflow attack: strcat (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

---------- Summary of File Security Analysis ----------

Total number of files submitted: 6

List of files containing anomalies (A), vulnerabilities (V) or risks (R)

A	V/R	Filename
4	0	bbuninst.exe
3	0	runlink.exe
4	5	zapro.exe
4	0	zatutor.exe
4	0	zauninst.exe
4	1	zonealarm.exe

15. APPENDIX I – TEST RESULTS FROM ANALYZING MISCELLANEOUS APPLICATION FILES

15.1 Dynamic Link Library (DLL) Files for Adobe Acrobat Reader 5.0

```
FILE NAME: AceLite.dll

**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 328 bytes
exists starting at address 290816; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 100 bytes in size when actually it is
1830 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 968 bytes in size when actually it
is 4096 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 16312 bytes in size when actually
it is 20480 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
FILE NAME: ACROFX32.DLL

**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 276 bytes
exists starting at address 30208; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 60 bytes in size when actually it is
1263 bytes in size
```

- The data directory table in the optional header states that the Relocation Table (.reloc section) is 2060 bytes in size when actually it is 3584 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: Agm.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 512 bytes exists starting at address 831488; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 100 bytes in size when actually it is 1898 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 976 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 60628 bytes in size when actually it is 69632 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: Bib.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 212 bytes exists starting at address 90112; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 40 bytes in size when actually it is 1151 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 968 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 7672 bytes in size when actually it is 12288 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: CoolType.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 560 bytes exists starting at address 1007616; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 160 bytes in size when actually it is 2374 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1016 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 66024 bytes in size when actually it is 77824 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: vdk150.dll

**** Anomalies ****

- A section entry named .bss appears in the section table, but the table doesn't contain the location of the 260 bytes for that section
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 17992 bytes in size when actually it is 18432 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 3 standard C functions susceptible to buffer overflow attacks: strcat (Very high risk), strcpy (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: WHA Library.dll

```
**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 572 bytes
exists starting at address 106496; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 200 bytes in size when actually it is
4635 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 11784 bytes in size when actually it
is 12288 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 8956 bytes in size when actually
it is 12288 bytes in size

**** End of Anomalies ****


!!!! Security Vulnerabilities and Risks!!!!

  - Uses 8 standard C functions susceptible to buffer overflow
attacks: memcpy (Low risk), read (Medium risk), sprintf (Very high
risk), sscanf (Very high risk), strcat (Very high risk), strcpy (Very
high risk), strncpy (Low risk), vsprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 7

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)
```

A	V/R	Filename
5	0	AceLite.dll
4	0	ACROFX32.DLL
5	0	Agm.dll
5	0	Bib.dll
5	0	CoolType.dll
2	3	vdk150.dll
5	8	WHA Library.dll

15.2 Executable (EXE) Files for Adobe Acrobat Reader 5.0

```
FILE NAME: AcroRd32.exe

**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 2688
bytes exists starting at address 2240512; this table often does not
appear in an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 360 bytes in size when actually it is
12831 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 889256 bytes in size when actually
it is 892928 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 140380 bytes in size when
actually it is 167936 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 1

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     5     0   AcroRd32.exe
```

15.3 Dynamic Link Library (DLL) Files for EarthLink TotalAccess 5.0

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 54

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     5     3   AddrBook.dll
     5     2   AuthMgr.dll
     5     1   CntctMgr.dll
     5     2   Dialer.dll
```

5	4	E60Cmmon.dll
5	0	E60MAPI.dll
5	1	EAuthMgr.dll
5	3	EConfig.dll
5	3	Ecrypt.dll
5	5	EFtp.dll
5	5	EImpExp.dll
5	0	ElnIE.dll
5	2	ElsAol.dll
5	1	ElsHotmail.dll
6	1	ElsMain.dll
5	2	ElsYahoo.dll
5	4	emsmtp.dll
6	4	Epic.dll
5	3	EventLog.dll
5	2	HSconfig.DLL
5	1	IdentityMgr.dll
5	1	imap4.dll
5	3	Location.dll
6	0	MagicCtl.dll
5	8	MailDoc.dll
5	7	MailEng.dll
5	0	mailstore.dll
5	4	MailStoreConverter.DLL
5	1	MailStoreDB.dll
5	4	MCE60Cmmon.dll
5	3	MCEcrypt.dll
5	3	MCLocation.dll
5	2	MCUtils.dll
5	2	MCWin.dll
5	0	MonIdle.dll
5	1	Notify.dll
5	1	Parse822.dll
5	0	PnEL.dll
5	0	PnEL_UI.dll
5	0	PnMsgBlk.dll
5	0	pop3.dll
5	3	Register.dll
4	0	RzTp.dll
5	5	SetupKrn.dll
5	1	smtp.dll
5	1	SpamBlocker.dll
5	0	Swi_Cdma1x.dll
5	1	SynchEng.dll
6	7	tmTools.dll
5	2	Utils.dll
5	2	Win.dll
5	0	WrSetupUtils.dll
5	1	XMLCol.dll
5	2	zlib.dll

15.4 Executable (EXE) Files for EarthLink TotalAccess 5.0

FILE NAME: elnbonus.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1892 bytes exists starting at address 110592; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 300 bytes in size when actually it is 5621 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 13072 bytes in size when actually it is 16384 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 3 standard C functions susceptible to buffer overflow attacks: sprintf (Very high risk), vsnprintf (Low risk), vsprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: ELNhelp.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1492 bytes exists starting at address 86016; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 260 bytes in size when actually it is 6245 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 28664 bytes in size when actually it is 28672 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 3 standard C functions susceptible to buffer overflow attacks: memcpy (Low risk), strcpy (Very high risk), vsprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: FixMail.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1648 bytes exists starting at address 196608; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 320 bytes in size when actually it is 8928 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 77280 bytes in size when actually it is 77824 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: IEAccnt.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 912 bytes exists starting at address 45056; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 220 bytes in size when actually it is 4133 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 3 standard C functions susceptible to buffer overflow attacks: memcpy (Low risk), sscanf (Very high risk), vsprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: MailClnt.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 3352 bytes exists starting at address 581632; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 460 bytes in size when actually it is 18402 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 93840 bytes in size when actually it is 94208 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 3 standard C functions susceptible to buffer overflow attacks: sprintf (Very high risk), sscanf (Very high risk), vsprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: MailSvr.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 660 bytes exists starting at address 24576; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 200 bytes in size when actually it is 2218 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 29552 bytes in size when actually it is 32768 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: MsiUtils.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1180 bytes exists starting at address 126976; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 260 bytes in size when actually it is 5895 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 28552 bytes in size when actually it is 28672 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses one standard C function susceptible to buffer overflow attack: vsprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: PrivacyHelper.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 484 bytes exists starting at address 28672; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 160 bytes in size when actually it is 1919 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 3944 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: PuB.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 300 bytes exists starting at address 28672; this table often does not appear in an image file so it was not read and it was also not mapped

- The data directory table in the optional header states that the Import Table (.idata section) is 140 bytes in size when actually it is 841 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 2128 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: TaskPanl.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 4644 bytes exists starting at address 692224; this table often does not appear in an image file so it was not read and it was also not mapped
- The file indicates a delay import descriptor consisting of 64 bytes exists starting at address 835044; this item often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 440 bytes in size when actually it is 24485 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 31496 bytes in size when actually it is 32768 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 3 standard C functions susceptible to buffer overflow attacks: sprintf (Very high risk), sscanf (Very high risk), vsprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: uninstll.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information

- The file indicates an import address table consisting of 2308 bytes exists starting at address 188416; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 220 bytes in size when actually it is 6790 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 49184 bytes in size when actually it is 53248 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 6 standard C functions susceptible to buffer overflow attacks: memcpy (Low risk), sprintf (Very high risk), strcat (Very high risk), strcpy (Very high risk), vsnprintf (Low risk), vsprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: UpdMgr.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1940 bytes exists starting at address 77824; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 320 bytes in size when actually it is 8200 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 93088 bytes in size when actually it is 94208 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 2 standard C functions susceptible to buffer overflow attacks: sscanf (Very high risk), vsprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: Webspace.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1580 bytes exists starting at address 40960; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 260 bytes in size when actually it is 6304 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 29928 bytes in size when actually it is 32768 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 3 standard C functions susceptible to buffer overflow attacks: sprintf (Very high risk), sscanf (Very high risk), vsprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

---------- Summary of File Security Analysis ----------

Total number of files submitted: 14

List of files containing anomalies (A), vulnerabilities (V) or risks (R)

A	V/R	Filename
4	3	elnbonus.exe
4	3	ELNhelp.exe
4	0	FixMail.exe
3	3	IEAccnt.exe
4	3	MailClnt.exe
4	0	MailSvr.exe
4	1	MsiUtils.exe
4	0	PrivacyHelper.exe
4	0	PuB.exe
5	3	TaskPanl.exe
4	6	uninstll.exe
4	2	UpdMgr.exe
4	3	Webspace.exe

15.5 Dynamic Link Library (DLL) Files for Hewlett-Packard PC CoreTech

---

```
FILE NAME: hpcmpmgr.dll


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 608 bytes
exists starting at address 61440; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 140 bytes in size when actually it is
2386 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 11872 bytes in size when actually it
is 12288 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 3840 bytes in size when actually
it is 8192 bytes in size

**** End of Anomalies ****


!!!! Security Vulnerabilities and Risks!!!!

  - Uses one standard C function susceptible to buffer overflow
attack: vsprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!
```

```
FILE NAME: hpvaut32.dll


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 1092
bytes exists starting at address 8192; this table often does not
appear in an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 120 bytes in size when actually it is
5570 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 676 bytes in size when actually it
is 4096 bytes in size
```

- The data directory table in the optional header states that the Relocation Table (.reloc section) is 28360 bytes in size when actually it is 32768 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: hpvcp70.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 440 bytes exists starting at address 188416; this table often does not appear in an image file so it was not read and it was also not mapped
- Invalid directory table data found in the export table; 2896 functions are being exported but the maximum allowed by this program is 2000
- The file indicates an export table exists but the export data could not be read
- The data directory table in the optional header states that the Import Table (.idata section) is 60 bytes in size when actually it is 2221 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 960 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 12448 bytes in size when actually it is 16384 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 3 standard C functions susceptible to buffer overflow attacks: fgetc (Medium risk), memcpy (Low risk), sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: hpvcr70.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information

- The file indicates an import address table consisting of 616 bytes exists starting at address 233472; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 40 bytes in size when actually it is 3467 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 952 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 10872 bytes in size when actually it is 12288 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: msxml4.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 552 bytes exists starting at address 1024; this table often does not appear in an image file so it was not read and it was also not mapped
- The file indicates a delay import descriptor consisting of 256 bytes exists starting at address 987372; this item often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 2786 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 110512 bytes in size when actually it is 110592 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 60540 bytes in size when actually it is 60928 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: msxml4a.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information

```
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 43368 bytes in size when actually it
is 43520 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 8 bytes in size when actually it
is 512 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
FILE NAME: msxml4r.dll


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 81288 bytes in size when actually it
is 81408 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 8 bytes in size when actually it
is 512 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 7

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     5     1   hpcmpmgr.dll
     5     0   hpvaut32.dll
     7     3   hpvcp70.dll
     5     0   hpvcr70.dll
     6     0   msxml4.dll
     3     0   msxml4a.dll
     3     0   msxml4r.dll
```

15.6 Executable (EXE) Files for Hewlett-Packard PC CoreTech

```
FILE NAME: hpcmpmgr.exe


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 984 bytes
exists starting at address 147456; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 160 bytes in size when actually it is
4015 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 14992 bytes in size when actually it
is 16384 bytes in size

**** End of Anomalies ****



!!!! Security Vulnerabilities and Risks!!!!

  - Uses 2 standard C functions susceptible to buffer overflow
attacks: sprintf (Very high risk), vsprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 1

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     4     2   hpcmpmgr.exe
```

15.7 Executable (EXE) Files for Iomega Zip Disk 100 Utilities

---

```
FILE NAME: ActivityDisk.exe


**** Anomalies ****
```

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 444 bytes exists starting at address 45056; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 1557 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1728 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: AppServices.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 452 bytes exists starting at address 49152; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 1565 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1608 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: RegW2KInst.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 196 bytes exists starting at address 16384; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 449 bytes in size

```
**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
FILE NAME: Win2kDrivers.exe


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 336 bytes
exists starting at address 53248; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 80 bytes in size when actually it is
987 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 1256 bytes in size when actually it
is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 4

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     4     0   ActivityDisk.exe
     4     0   AppServices.exe
     3     0   RegW2KInst.exe
     4     0   Win2kDrivers.exe
```

15.8 Dynamic Link Library (DLL) Files for MusicMatch Jukebox 7

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 47
```

List of files containing anomalies (A), vulnerabilities (V) or risks (R)

A	V/R	Filename
5	1	analog.dll
5	0	CDDBControl.dll
4	6	cdr.dll
5	4	cds.dll
4	0	DestinationWavDll.dll
5	1	digital.dll
5	1	EventMgr.dll
5	1	FileAssoc.dll
5	1	FileCacheMgr.dll
5	3	fileco.dll
5	1	FWRun.dll
5	4	JewelCasePrinter.dll
5	1	linein.dll
5	4	mixer.dll
5	5	mmdb.dll
5	0	mmfwloc.dll
5	2	MMInet.dll
5	0	mmInstall.dll
5	0	mmjbloc.dll
5	5	mmportal.dll
5	5	MMRadioEngine.dll
5	1	mmreg.dll
5	2	mmrio.dll
5	1	mmsal32.dll
5	6	MMSecurity.dll
5	4	mmsiteserv.dll
5	3	mmuiserv.dll
5	1	mmzip32.dll
4	7	mrbupd.dll
4	0	mscdex32.dll
8	4	msvcp60.dll
5	0	ObjectManager.dll
5	1	PortableDevice.dll
5	1	PortableDevice2.dll
5	0	preferences.dll
5	1	record.dll
5	1	StgCdr.dll
5	0	TrackListPrinter.dll
5	0	unmatch.dll
5	0	unzip32.dll
5	0	wnaspint.dll
5	0	xanalyze.dll
5	0	xaudio.dll
5	0	zip32.dll

15.9 Executable (EXE) Files for MusicMatch Jukebox 7

FILE NAME: mmdiag.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1392 bytes exists starting at address 32768; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 120 bytes in size when actually it is 2730 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 12680 bytes in size when actually it is 16384 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 4 standard C functions susceptible to buffer overflow attacks: sprintf (Very high risk), strcat (Very high risk), strcpy (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: mmjb.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 5564 bytes exists starting at address 1220608; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 440 bytes in size when actually it is 24246 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 306912 bytes in size when actually it is 307200 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 7 standard C functions susceptible to buffer overflow attacks: fgets (Low risk), memcpy (Low risk), sprintf (Very high risk), sscanf (Very high risk), strcat (Very high risk), strcpy (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: MMJBBurn.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 3024 bytes exists starting at address 425984; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 260 bytes in size when actually it is 8137 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 34184 bytes in size when actually it is 36864 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses one standard C function susceptible to buffer overflow attack: sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: MMJBLaunch.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 240 bytes exists starting at address 1024; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 100 bytes in size when actually it is 1079 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 32152 bytes in size when actually it is 32256 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: mmjbrun.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 184 bytes exists starting at address 16384; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 394 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: MmjbUpdt.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 572 bytes exists starting at address 86016; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 120 bytes in size when actually it is 2100 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 8824 bytes in size when actually it is 12288 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: MMPurchase.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 380 bytes exists starting at address 12288; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 100 bytes in size when actually it is 982 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 920 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses one standard C function susceptible to buffer overflow attack: strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: mm_tray.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 676 bytes exists starting at address 40960; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 200 bytes in size when actually it is 4755 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 32872 bytes in size when actually it is 36864 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 4 standard C functions susceptible to buffer overflow attacks: fgetc (Medium risk), sprintf (Very high risk), strcat (Very high risk), strcpy (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: RefreshIcon.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 240 bytes exists starting at address 24576; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 100 bytes in size when actually it is 633 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 912 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: ti.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 548 bytes exists starting at address 245760; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 120 bytes in size when actually it is 1775 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 127080 bytes in size when actually it is 131072 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: UpdtStub.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information

```
  - The file indicates an import address table consisting of 232 bytes
exists starting at address 8192; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 100 bytes in size when actually it is
1088 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 1072 bytes in size when actually it
is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 11

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     4     4   mmdiag.exe
     4     7   mmjb.exe
     4     1   MMJBBurn.exe
     4     0   MMJBLaunch.exe
     3     0   mmjbrun.exe
     4     0   MmjbUpdt.exe
     4     1   MMPurchase.exe
     4     4   mm_tray.exe
     4     0   RefreshIcon.exe
     4     0   ti.exe
     4     0   UpdtStub.exe
```

15.10 Dynamic Link Library (DLL) Files for OpenOffice 1.1

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 198

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     5     0   abp645mi.dll
     5     0   acceptor.uno.dll
     5     0   adabas2.dll
```

```
5     0   ado2.dll
5     1   analysis645mi.dll
5     0   basctl645mi.dll
5     0   bib645mi.dll
5     0   bridgefac.uno.dll
5     0   cached1.dll
5     0   calc645mi.dll
5     1   cfgmgr2.dll
5     0   cmdmail.dll
5     1   cnt645mi.dll
7     0   comphelp3MSC.dll
5     0   connector.uno.dll
5     0   corereflection.uno.dll
5     0   cppu3.dll
6     0   cppuhelper3MSC.dll
5     0   ctl645mi.dll
5     0   date645mi.dll
5     0   dba645mi.dll
5     2   dbase645mi.dll
5     6   dbghelp.dll
5     0   dbi645mi.dll
5     0   dbp645mi.dll
5     0   dbpool2.dll
7     1   dbtools2.dll
5     1   dbu645mi.dll
5     0   del645mi.dll
5     0   dict_ja.dll
5     0   dict_zh.dll
7     0   dl645mi.dll
5     0   dnd.dll
5     0   dtrans.dll
5     0   emser645mi.dll
5     0   evtatt.dll
7     0   file645mi.dll
5     0   flash645mi.dll
5     0   flat645mi.dll
5     0   fop.dll
5     0   fps.dll
5     1   frm645mi.dll
5     0   ftransl.dll
7     1   fwe645mi.dll
7     0   fwi645mi.dll
5     1   fwk645mi.dll
5     0   fwl645mi.dll
5     0   go645mi.dll
5     1   hyphen645mi.dll
5     0   i18n645mi.dll
5     1   i18npool645mi.dll
5     0   i18nregexpMSC.dll
5     0   i18nsearch.dll
5     0   i18nutilMSC.dll
2     0   icudt22l.dll
5     0   icuin22.dll
5     0   icule22.dll
5     4   icuuc22.dll
```

```
5     0   implreg.uno.dll
5     0   introspection.uno.dll
5     0   invocadapt.uno.dll
5     0   invocation.uno.dll
5     0   j645mi_g.dll
5     0   javaloader.uno.dll
5     0   javavm.uno.dll
5     0   java_uno.dll
5     0   java_uno_accessbridge.dll
5     0   jdbc2.dll
5     0   jpipe.dll
5     4   js3250.dll
6     0   juh.dll
5     0   juhx.dll
5     0   jvm645mi.dll
5     1   jvmaccess3MSC.dll
4     6   libcurl.dll
5     7   libdb32.dll
4     2   libdb_java32.dll
5     1   lng645mi.dll
5     0   localedata_en.dll
5     0   localedata_es.dll
5     0   localedata_euro.dll
5     0   localedata_others.dll
5     1   lth645mi.dll
5     0   mcnttype.dll
5     0   mozab2.dll
5     0   mozabdrv2.dll
5     2   mozreg.dll
5     1   msci_uno.dll
5     3   msgbsutl.dll
7     3   msvcp70.dll
5     0   msvcr70.dll
5     0   mysql2.dll
5     0   namingservice.uno.dll
5     0   nestedreg.uno.dll
4     3   nsldap32v50.dll
4     0   nsldappr32v50.dll
5     2   nspr4.dll
5     0   nsreg.dll
5     0   odbc2.dll
7     0   odbcbase2.dll
5     0   ofa645mi.dll
5     0   offacc645mi.dll
5     0   officebean.dll
5     0   oleautobridge.uno.dll
5     2   opc645mi.dll
5     0   package2.dll
5     0   pcr645mi.dll
5     0   pdffilter645mi.dll
5     0   pk645mi.dll
5     0   pkgchk645mi.dll
5     0   placewaremi.dll
5     0   plc4.dll
5     0   plds4.dll
```

```
5     0   preload645mi.dll
5     0   proxyfac.uno.dll
5     0   proxyset.dll
5     8   python22.dll
5     0   pythonloader.uno.dll
5     0   pyuno.dll
5     1   reg3.dll
5     0   reg4msdoc645mi.dll
5     0   regactivex645mi.dll
5     0   regtypeprov.uno.dll
5     0   remotebridge.uno.dll
5     0   res645mi.dll
5     0   rmcxt3.dll
5     4   sal3.dll
5     0   salhelper3MSC.dll
5     0   sax.uno.dll
7     2   sb645mi.dll
5     2   sc645mi.dll
5     0   sch645mi.dll
5     1   scn645mi.dll
5     0   sd645mi.dll
5     0   sdbc2.dll
5     0   security.uno.dll
5     0   servicemgr.uno.dll
7     6   set645mi.dll
7     3   sfx645mi.dll
5     0   shlibloader.uno.dll
5     0   shlxthdl.dll
5     0   simplereg.uno.dll
5     1   sm645mi.dll
5     0   smplmail.dll
7     1   so645mi.dll
5     0   sot645mi.dll
5     2   so_activex.dll
5     1   spell645mi.dll
5     0   spl645mi.dll
5     0   srtrs1.dll
7     4   stlport_vc745.dll
5     0   store3.dll
5     0   streams.uno.dll
5     0   sts645mi.dll
5     0   svg645mi.dll
7     1   svx645mi.dll
5     1   sw645mi.dll
5     0   sysdtrans.dll
5     0   syssh.dll
5     0   textinstream.uno.dll
5     0   textoutstream.uno.dll
7     0   tk645mi.dll
7     3   tl645mi.dll
5     0   tplx645mi.dll
5     0   tvhlp1.dll
5     0   typeconverter.uno.dll
5     0   typemgr.uno.dll
5     0   ucb1.dll
```

```
7     0   ucbhelper2MSC.dll
5     1   ucpchelp1.dll
5     6   ucpdav1.dll
5     0   ucpfile1.dll
5     0   ucpftp1.dll
5     0   ucphier1.dll
5     0   ucppkg1.dll
5     0   ulingu645mi.dll
5     0   unicows.dll
5     0   urp_uno.dll
5     1   usp645mi.dll
7     0   utl645mi.dll
5     0   uui645mi.dll
5     0   uuresolver.uno.dll
5     2   uwinapi.dll
7     2   vcl645mi.dll
5     0   vos3MSC.dll
5     0   xcr645mi.dll
5     0   xmlfa645mi.dll
5     0   xmlfd645mi.dll
5     0   xmx645mi.dll
7     0   xo645mi.dll
7     5   xpcom.dll
5     0   xsltdlg645mi.dll
5     5   zlib.dll
```

15.11 Executable (EXE) Files for OpenOffice 1.1

```
FILE NAME: crashrep.exe


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 632 bytes
exists starting at address 24576; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 220 bytes in size when actually it is
4096 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 197472 bytes in size when actually
it is 200704 bytes in size

**** End of Anomalies ****


!!!! Security Vulnerabilities and Risks!!!!
```

- Uses 3 standard C functions susceptible to buffer overflow attacks: fgets (Low risk), sprintf (Very high risk), sscanf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: jvmsetup.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1152 bytes exists starting at address 49152; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 260 bytes in size when actually it is 3333 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 3 standard C functions susceptible to buffer overflow attacks: fgetc (Medium risk), sprintf (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: pkgchk.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 272 bytes exists starting at address 8192; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 140 bytes in size when actually it is 1452 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: quickstart.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 220 bytes exists starting at address 8192; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 100 bytes in size when actually it is 942 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 42624 bytes in size when actually it is 45056 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: regsvrex.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 132 bytes exists starting at address 4096; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 60 bytes in size when actually it is 695 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: setup.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 2568 bytes exists starting at address 114688; this table often does not appear in an image file so it was not read and it was also not mapped

- The data directory table in the optional header states that the Import Table (.idata section) is 320 bytes in size when actually it is 4853 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 12200 bytes in size when actually it is 12288 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses one standard C function susceptible to buffer overflow attack: strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: soffice.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1916 bytes exists starting at address 139264; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 320 bytes in size when actually it is 7233 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 239128 bytes in size when actually it is 241664 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses one standard C function susceptible to buffer overflow attack: sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: testtool.exe

**** Anomalies ****

```
  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 3896
bytes exists starting at address 237568; this table often does not
appear in an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 300 bytes in size when actually it is
5655 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 8

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     4     3   crashrep.exe
     3     3   jvmsetup.exe
     3     0   pkgchk.exe
     4     0   quickstart.exe
     3     0   regsvrex.exe
     4     1   setup.exe
     4     1   soffice.exe
     3     0   testtool.exe
```

15.12 Dynamic Link Library (DLL) Files for Real One Player (ME/XP)

```
FILE NAME: dunzip32.dll


**** Anomalies ****

  - A section entry named .bss appears in the section table, but the
table doesn't contain the location of the 42888 bytes for that section
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 2264 bytes in size when actually it
is 2560 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 5528 bytes in size when actually
it is 6144 bytes in size

**** End of Anomalies ****
```

No security vulnerabilities or security risks were found

FILE NAME: ierjplug.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 268 bytes exists starting at address 13824; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 140 bytes in size when actually it is 1175 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 4120 bytes in size when actually it is 4608 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 788 bytes in size when actually it is 2560 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 2 standard C functions susceptible to buffer overflow attacks: snprintf (Low risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: mmcdda32.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 196 bytes exists starting at address 15872; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 120 bytes in size when actually it is 984 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1064 bytes in size when actually it is 1536 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 760 bytes in size when actually it is 1536 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 2 standard C functions susceptible to buffer overflow attacks: sprintf (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: rjbres.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 92 bytes exists starting at address 7168; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 478 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 783208 bytes in size when actually it is 783360 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 256 bytes in size when actually it is 2560 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses one standard C function susceptible to buffer overflow attack: sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: rjbxfade.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 260 bytes exists starting at address 13312; this table often does not appear in an image file so it was not read and it was also not mapped

- The data directory table in the optional header states that the Import Table (.idata section) is 100 bytes in size when actually it is 1005 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 42944 bytes in size when actually it is 43008 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 636 bytes in size when actually it is 1536 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses one standard C function susceptible to buffer overflow attack: sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: rjdlg.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1268 bytes exists starting at address 124416; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 180 bytes in size when actually it is 8052 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 284984 bytes in size when actually it is 285184 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 10068 bytes in size when actually it is 13312 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 6 standard C functions susceptible to buffer overflow attacks: fgets (Low risk), read (Medium risk), sprintf (Very high risk), sscanf (Very high risk), strncpy (Low risk), vsprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: rjprog.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 208 bytes exists starting at address 11264; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 1055 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 2392 bytes in size when actually it is 2560 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 628 bytes in size when actually it is 1536 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses one standard C function susceptible to buffer overflow attack: sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: rmbe3260.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 640 bytes exists starting at address 336384; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 140 bytes in size when actually it is 2829 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 36168 bytes in size when actually it is 36352 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 14356 bytes in size when actually it is 17920 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 4 standard C functions susceptible to buffer overflow attacks: read (Medium risk), sprintf (Very high risk), strncpy (Low risk), vsprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: rpau3260.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 264 bytes exists starting at address 13312; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 140 bytes in size when actually it is 1388 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 4864 bytes in size when actually it is 5120 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 884 bytes in size when actually it is 2048 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses one standard C function susceptible to buffer overflow attack: sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: rpwa3260.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 160 bytes exists starting at address 7168; this table often does not appear in an image file so it was not read and it was also not mapped

- The data directory table in the optional header states that the Import Table (.idata section) is 100 bytes in size when actually it is 786 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1064 bytes in size when actually it is 1536 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 332 bytes in size when actually it is 1024 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 2 standard C functions susceptible to buffer overflow attacks: fgets (Low risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: tmdedit.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1020 bytes exists starting at address 73216; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 220 bytes in size when actually it is 7353 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 162416 bytes in size when actually it is 162816 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 4736 bytes in size when actually it is 7168 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 2 standard C functions susceptible to buffer overflow attacks: sprintf (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: tnetdtct.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 252 bytes exists starting at address 8704; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 140 bytes in size when actually it is 1101 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1048 bytes in size when actually it is 1536 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 608 bytes in size when actually it is 1024 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 2 standard C functions susceptible to buffer overflow attacks: sprintf (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: tpasdk.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 156 bytes exists starting at address 22528; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 100 bytes in size when actually it is 719 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1056 bytes in size when actually it is 1536 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 1372 bytes in size when actually it is 1536 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 3 standard C functions susceptible to buffer overflow attacks: fgets (Low risk), sprintf (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: tsasdk.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 140 bytes exists starting at address 71168; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 544 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1056 bytes in size when actually it is 1536 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 1096 bytes in size when actually it is 4608 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 3 standard C functions susceptible to buffer overflow attacks: read (Medium risk), sprintf (Very high risk), vsprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: twebbrowse.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 324 bytes exists starting at address 35840; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 180 bytes in size when actually it is 1533 bytes in size

```
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 1048 bytes in size when actually it
is 1536 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 2084 bytes in size when actually
it is 2560 bytes in size

**** End of Anomalies ****


!!!! Security Vulnerabilities and Risks!!!!

  - Uses 2 standard C functions susceptible to buffer overflow
attacks: sprintf (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 15

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)
```

A	V/R	Filename
3	0	dunzip32.dll
5	2	ierjplug.dll
5	2	mmcdda32.dll
5	1	rjbres.dll
5	1	rjbxfade.dll
5	6	rjdlg.dll
5	1	rjprog.dll
5	4	rmbe3260.dll
5	1	rpau3260.dll
5	2	rpwa3260.dll
5	2	tmdedit.dll
5	2	tnetdtct.dll
5	3	tpasdk.dll
5	3	tsasdk.dll
5	2	twebbrowse.dll

15.13 Executable (EXE) Files for Real One Player (ME/XP)

```
FILE NAME: fixrjb.exe


**** Anomalies ****
```

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 100 bytes exists starting at address 2560; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 100 bytes in size when actually it is 528 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: realjbox.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 104 bytes exists starting at address 2560; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 100 bytes in size when actually it is 568 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1936 bytes in size when actually it is 2048 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: realplay.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 360 bytes exists starting at address 28160; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 120 bytes in size when actually it is 1782 bytes in size

```
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 94672 bytes in size when actually it
is 94720 bytes in size

**** End of Anomalies ****


!!!! Security Vulnerabilities and Risks!!!!

  - Uses 2 standard C functions susceptible to buffer overflow
attacks: sprintf (Very high risk), strncpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!
```

```
FILE NAME: rphelperapp.exe


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 128 bytes
exists starting at address 4096; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 100 bytes in size when actually it is
639 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 4

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     3     0   fixrjb.exe
     4     0   realjbox.exe
     4     2   realplay.exe
     3     0   rphelperapp.exe
```

15.14 Image Files for Veritas Update Manager

FILE NAME: AniGifDisplay.ocx

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1212 bytes exists starting at address 24576; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 140 bytes in size when actually it is 2007 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 6384 bytes in size when actually it is 8192 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 2236 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: Archived.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 624 bytes exists starting at address 81920; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 140 bytes in size when actually it is 2448 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 13272 bytes in size when actually it is 16384 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 5028 bytes in size when actually it is 8192 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: Graph.ocx

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1076 bytes exists starting at address 16384; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 140 bytes in size when actually it is 1611 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 4128 bytes in size when actually it is 8192 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 1732 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: sfcwall31.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 1220 bytes exists starting at address 188416; this table often does not appear in an image file so it was not read and it was also not mapped
- The length of 143 export name(s) in the export table exceeded the buffer size of 255 bytes
- The data directory table in the optional header states that the Import Table (.idata section) is 160 bytes in size when actually it is 9813 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 39152 bytes in size when actually it is 40960 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 11896 bytes in size when actually it is 16384 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 3 standard C functions susceptible to buffer overflow attacks: memcpy (Low risk), sprintf (Very high risk), strcpy (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: sgpropsht.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 752 bytes exists starting at address 40960; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 260 bytes in size when actually it is 4068 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 4072 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 3244 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses one standard C function susceptible to buffer overflow attack: memcpy (Low risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: sgtray.exe

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 2240 bytes exists starting at address 106496; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 320 bytes in size when actually it is 10366 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 664 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses one standard C function susceptible to buffer overflow attack: sprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: sus.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 848 bytes exists starting at address 98304; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 180 bytes in size when actually it is 9369 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 928 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 6360 bytes in size when actually it is 8192 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 2 standard C functions susceptible to buffer overflow attacks: getc (Medium risk), vsprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

FILE NAME: trayrenu.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 709408 bytes in size when actually it is 712704 bytes in size

- The data directory table in the optional header states that the Relocation Table (.reloc section) is 8 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: vxhttp.dll

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 436 bytes exists starting at address 32768; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 140 bytes in size when actually it is 3802 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 912 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 2488 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

!!!! Security Vulnerabilities and Risks!!!!

- Uses 2 standard C functions susceptible to buffer overflow attacks: sprintf (Very high risk), vsprintf (Very high risk)

!!!! End of Security Vulnerabilities and Risks!!!!

---------- Summary of File Security Analysis ----------

Total number of files submitted: 9

List of files containing anomalies (A), vulnerabilities (V) or risks (R)

A	V/R	Filename
5	0	AniGifDisplay.ocx
5	0	Archived.dll
5	0	Graph.ocx

```
6     3   sfcwall31.dll
5     1   sgpropsht.dll
4     1   sgtray.exe
5     2   sus.dll
3     0   trayrenu.dll
5     2   vxhttp.dll
```

15.15 Dynamic Link Library (DLL) Files for WinZIP 8.0

```
FILE NAME: WZ32.DLL


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 500 bytes
exists starting at address 184320; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 80 bytes in size when actually it is
2745 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 35592 bytes in size when actually it
is 36864 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 14808 bytes in size when actually
it is 20480 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
FILE NAME: WZCAB.DLL


**** Anomalies ****

  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 916 bytes in size when actually it
is 1024 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 1136 bytes in size when actually
it is 1536 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

FILE NAME: WZCAB3.DLL

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 312 bytes exists starting at address 36864; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 1617 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 6680 bytes in size when actually it is 8192 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 2244 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: WZSHLEX1.DLL

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 424 bytes exists starting at address 45056; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 120 bytes in size when actually it is 2315 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 7592 bytes in size when actually it is 8192 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 3060 bytes in size when actually it is 8192 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: WZSHLSTB.DLL

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 80 bytes exists starting at address 8192; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 475 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1208 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 144 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: WZVINFO.DLL

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 232 bytes exists starting at address 32768; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 80 bytes in size when actually it is 1267 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 1352 bytes in size when actually it is 4096 bytes in size
- The data directory table in the optional header states that the Relocation Table (.reloc section) is 1660 bytes in size when actually it is 4096 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: WZZPMAIL.DLL

**** Anomalies ****

```
  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 656 bytes
exists starting at address 61440; this table often does not appear in
an image file so it was not read and it was also not mapped
  - The data directory table in the optional header states that the
Import Table (.idata section) is 160 bytes in size when actually it is
3612 bytes in size
  - The data directory table in the optional header states that the
Resource Table (.rsrc section) is 18104 bytes in size when actually it
is 20480 bytes in size
  - The data directory table in the optional header states that the
Relocation Table (.reloc section) is 3660 bytes in size when actually
it is 8192 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found
```

```
---------- Summary of File Security Analysis ----------

Total number of files submitted: 7

List of files containing anomalies (A), vulnerabilities (V) or risks
(R)

     A   V/R   Filename
     -   ---   --------
     5     0   WZ32.DLL
     2     0   WZCAB.DLL
     5     0   WZCAB3.DLL
     5     0   WZSHLEX1.DLL
     5     0   WZSHLSTB.DLL
     5     0   WZVINFO.DLL
     5     0   WZZPMAIL.DLL
```

15.16 Executable (EXE) Files for WinZIP 8.0

```
FILE NAME: WINZIP32.EXE


**** Anomalies ****

  - The normally small zero-filled region following the MS-DOS Stub
contains some kind of information
  - The file indicates an import address table consisting of 1628
bytes exists starting at address 487424; this table often does not
appear in an image file so it was not read and it was also not mapped
```

- The data directory table in the optional header states that the Import Table (.idata section) is 200 bytes in size when actually it is 7538 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 865264 bytes in size when actually it is 868352 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

FILE NAME: WZSEPE32.EXE

**** Anomalies ****

- The normally small zero-filled region following the MS-DOS Stub contains some kind of information
- The file indicates an import address table consisting of 748 bytes exists starting at address 75776; this table often does not appear in an image file so it was not read and it was also not mapped
- The data directory table in the optional header states that the Import Table (.idata section) is 160 bytes in size when actually it is 3910 bytes in size
- The data directory table in the optional header states that the Resource Table (.rsrc section) is 88312 bytes in size when actually it is 88576 bytes in size

**** End of Anomalies ****

No security vulnerabilities or security risks were found

---------- Summary of File Security Analysis ----------

Total number of files submitted: 2

List of files containing anomalies (A), vulnerabilities (V) or risks (R)

A	V/R	Filename
4	0	WINZIP32.EXE
4	0	WZSEPE32.EXE