Oðinn: An In-Vivo Hypervisor-based Intrusion Detection System for the Cloud

by

Christopher B. Harrison

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 2, 2014

Keywords: virtualization, cloud computing, security, virtual machine introspection, forensic memory analysis, data mining, malware analysis

Approved by

John A. Hamilton, Jr., Chair, Professor of Computer Science and Software Engineering
David Umphress, Associate Professor of Computer Science and Software Engineering
Jeffrey L. Overbey, Assistant Professor of Computer Science and Software Engineering
Dean Hendrix, Associate Professor of Computer Science and Software Engineering

Abstract

Cloud computing has emerged as the de facto service model for modern enterprises; however, security concerns remain a major impediment to full-scale adoption. Concurrent to this paradigm shift looms another concern that dominates the landscape of the security industry - malware proliferation. Leveraging the isolation property of virtualization, Virtual Machine Introspection (VMI) has yielded promising research for cloud security yet adoption of these approaches in production environments remains minimal due to a semantic gap: the extraction of high-level knowledge of the guest operating system's state from low-level artifacts collected out-of-VM. Within the field of Forensic Memory Analysis (FMA), a similar semantic gap existed (low level artifacts found in the reconstruction of physical memory dumps) and was rectified via a number of tools, notably Volatility. Other properties of virtualization have largely been unexplored for the use of Cloud-based Intrusion Detection Systems (IDSs) for use in malware mitigation techniques and post-infection analysis. By merging the these properties of virtualization with the semantic gap solution in FMA, we construct a prototype IDS, Oðinn, at the hypervisor level for detecting, mitigating, and analyzing malicious activity. Using Oðinn, we successfully detect malware in real-time and the accuracy increases when malware attempts to obfuscate itself using standard obfuscation methods. Once detected, Oðinn undoes the effects of the infection for the end user with only a few seconds of downtime and minimizing data loss to five minutes or less. Finally, we use our analysis suite in a novel manner to reduce the search space by at least 95% for the modules, drivers and processes altered or inserted during the malware.

Acknowledgments

I would like to thank my wife, Jaime, for supporting me throughout my long academic journey. I would like to thank my adviser Dr. John Hamilton, as well as Dr. Robert McGraw and Dr. Richard MacDonald at Ram Laboratories for providing me with the initial topic, funding and support that planted the seed from which grew this proposal. I would like to think Devin Cook for his instrumental help when we first explored this topic together in 2011 and finally, C.W. Perr and Chase Rushing for proofreading and providing moral support.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

A major concern dominates the landscape of the security industry - malware proliferation. Host-based Intrusion Detection Systems (HIDSs), which monitor a single computer system against malicious activity were developed to deal with this outgrowth; however, attackers quickly found ways to circumvent the HIDS after a successful intrusion. Consequently, the following tradeoff quickly emerged: IDS residence on the host provided high visibility and low resiliency yet IDS residence separate from the host provided high resiliency and poor visibility.

Additionally, cloud computing (see A.2) has recently emerged as the *de facto* service model for modern enterprises which vastly changes the underlying security requirements and threat models due to its underlying mechanism, virtualization (see A.1). An unintentional consequence imposed by cloud usage is a deterioration of Forensic Memory Analysis (FMA) capabilities which has created the need for a new generation of incident response and forensics analysis tools able to operate effectively in virtual environments. Current methodologies for post-incident analysis require shutting down the impacted physical machine for immediate data gathering. Moreover, the utilization of physical resources by a myriad of users impedes the attribution of malicious activity and raises the probability of losing relevant forensic data. Recent research addressing the technical challenges facing forensic investigators in the cloud concluded that accurately reconstructing the environment for forensic analysis was not feasible.

In researching these problems, isolation, an intrinsic property of virtualization, has been leveraged to provide a highly visible monitoring tool for guest VMs, a technique called Virtual Machine Introspection (VMI) that does not reside on the virtual host machine. VMI;

however, is not a panacea and has a major limitation imposed by the semantic gap. The *semantic gap* is the difference between the low-level artifacts available to the Virtual Machine Monitor or Hypervisor (VMM) (e.g. RAM, registers) and the high-level semantics specific to the OS (e.g. processes, open files, kernel modules). Bridging this gap often results in either a lack of visibility or a lack of flexibility. By utilizing the explicit semantic knowledge of an existing FMA tool, Volatility, and repurposing its use for VMI sensors, a bridge was created to overcome the difficulties posed by the semantic gap to resolve the general challenges facing security in the cloud via a non-quiescent VMI-Intrusion Detection System (IDS). Oðinn is an implementation of a VMI-IDS that exploits these properties to demonstrate efficacy in the current cloud/malware environment. Oðinn detects malware intrusions using standard techniques by the FMA community and a novel technique that deduce instances of orphaned artifacts (e.g. processes, modules, system drivers, object handles, etc. that have been pruned from the Operating System's internal structure lists). Another virtualization property, the ability to store the entire state of the guest machine, called a snapshot, is utilized by Oðinn to ensure that when a malicious intrusion is detected, the user of the now infected guest machine is restored to the most recent uninfected state with only a few seconds of downtime and five or fewer minutes of lost productivity. Finally, by differencing a guest machine's clean state with an infected state, we reduce search space of altered or inserted Operating System (OS) internal structures by 95% or more, quickly allowing a malware analyst to elucidate the malware's functionality with rapidity.

1.1 Research Contribution

This dissertation shows that several properties of virtualization exist that can be used by IDSs to provide unique security benefits to mitigate and overcome many of the challenges imposed by the widespread adoption of cloud computing. It's contribution are:

- Demonstrates the qualitative and quantitative benefits of an integrated explicit semantic bridge when using virtual machine introspection.

- Demonstrates novel mitigation capabilities with minimal user impact.
- Demonstrates novel malware analysis techniques.
- Demonstrates the accuracy of FMA sensors, including a unique manner of detecting modern malware obfuscation strategies.
- Measures and defines the limits of scalability.
- Creates an accurate, portable, and scalable Cloud-based IDS

1.2 Comparative Analysis of Cloud-based Intrusion Detection Systems

1.2.1 Cloud-based IDS with VMI

Saberi [11] introduces a novel performance improvement of the semantic bridge via the use of training memoization to avoid the recomputation of redirectable instruction identification. In doing so, they decouple their slow dynamic taint tracking engine except when the cached meta-data from the training memoization is not present. However, this approach still relies on host sensors (e.g. getpid, proc, ps, etc.) and will fail to detect malware which hides itself from these sensors. Additionally, it provides no mitigation or analysis component and also requires the Guest and Hypervisor Operating System to be the same.

Wu [12] is designed for datacenters and operates on an entirely different introspection concept, using a stealthy hook rather than out-of-VM introspection. As previously noted, this design is troublesome since the hook could be tampered. All of its sensors are focused at the file handle level and when a file handle is opened, the data to be written is first placed into a buffer and its signature is checked against known malware before it is actually written to the guest. This allows Wu to serve as an Intrusion Prevention System but its incredibly limited in scope since its dependent on malware that infects files whereas Oðinn has a myriad of sensors by which to detect and mitigate an infection.

Crawford [13] focuses on insider threats and notably foregoes frequent monitoring for full memory capture every 30 minutes. To detect insider threats, it primarily relies on deciphering

changes to specific registry objects, clipboard information and specific hexadecimal patterns in memory that correlate to previously observed insider threat use cases. The approach had high false positive rates and all observable data is lost if the user logs out.

The approach of Fu [14] is regards to its mitigation response. Unlike Oðinn, where the response is to switch to a period pre-attack, Fu attempts self-healing actions such as removing a rootkit or deleting a hidden malicious kernel module. While novel, this is an unsafe solution as one, it assumes that it has fully captured the extent of the malware's actions and two, by not immediately stopping the guest, it increases the chance for successful data exfiltration. One advantage Fu has is its support for vSphere and HyperV, two platforms where Oðinn lacks support. However, it has a number of drawbacks. First, it is not as guest OS-agnostic as Oðinn, primarily supporting Linux virtual machines. Two, it lacks many memory analysis sensors available in Oðinn, snapshot reversion, and post-infection malware analysis capabilities.

Perhaps the closest in terms of Oðinn's VMI implementation is VMI-Honeymon [15] in that both use libVMI and Volatility. However, the similarities largely end there as instead of being an IDS, it serves as a high interaction honeypot for basic malware analysis. VMI-Honeymon performs no comparison between doubly-linked list and pool tag scans, no high level analysis (e.g. reconstruction of PE headers), and several of their sensors only work on Windows XP - Vista. Additionally, their implementation is entirely unoptimized, which, while suitable for their purposes is entirely inadequate for a scalable cloud IDS solution. Other VMI-IDS solutions [16, 17] are discussed later in Section 3.1 so their relation to Oðinn is only summarized in Table 1.1.

Citation	Year	Scope	Bridge	Mitigation	Analysis	Portability	Weaknesses
Saberi [11]	2014	Memory	Implicit	None	None	QEMU, Guest/VMI must be same OS	Trusts in-VM sensors
Wu [12]	2013	Files	Explicit	None	None	Supports QEMU and Xen	Trusts in-VM sensor
Roberts [18]	2013	Malware	Implicit	None	None	QEMU, Windows Guest only	Not designed for scale
Crawford [13]	2013	Insider	Explicit	None	None	Xen VMI, Windows Guests only	Substantial number of False Positives
Fu [14]	2013	Rootkits	Hybrid	Rootkits	None	Linux Guests only	Removes only most common persistence technique
Jin [19]	2013	Network	Explicit	None	None	Xen VMI; Linux Guest only	Captures only a few features
Dolan-Gavitt [17]	2011	Malware	Implicit	None	None	QEMU VMI; Guest Agnostic	140x Overhead
Azmandian [20]	2011	Malware	None	None	None	Virtual Box VMI; Guest Agnostic	False Positive rates as high as 14%

Table 1.1: Taxonomy of Cloud-based IDSs with VMI

1.2.2 Cloud-based IDSwithout VMI

The methodology employed in Roschke [21] utilizes existing non-VMI based sensors (SNORT, Samhain, F-Secure) to detect attacks, and its cloud structure serves as a centralized management tool rather than taking advantage of introspection to create new sensors. While its architecture does allow for this, it doesn't take advantage of it and aside from its centralizing the decision making apparatus, the only other unique use of emergent cloud properties is the suspending of guest machines undergoing an attack. The only area that Oðinn would conceivably benefit from is the adoption of Intrusion Detection Message Exchange Format (IDMEF) for greater interoperability with other IDS approaches, especially Network-based Intrusion Detection System (NIDS). Sproull [22] is a non-VMI based cloud Intrusion Prevention System (IPS) using the Field Programmable Port Extender platform on an FPGA to efficiently scan thousands of virus signatures as a SNORT-like model that reconstructs traffic at incredible high data rates (2.5 Gbits/sec). In doing so, it can detect many worms and filter them from the network before they ever reach a vulnerable hosts. This model is entirely network based and thus has no real similarities to Oðinn though both could be used together without conflict. The works of Shyu and Sainani [23], Li et al. [24], Khanum [25], Herrero and Corchado [26], and Su et. al [27] are similar in this regard in that their implementations are practical for the cloud but as their IDS is focused on the network, they have little in common with Oðinn.

Citation	Year	Heuristic	Monitor	Type	Characteristics	Limitations
Khanum [25]	2012	Misuse	Passive	NIDS	Utilize minimum possible network resources.	Vulnerable to unknown attacks.
Li [24]	2010	Immune	Active	NIDS	Handles high-volume network traffic.	Only deals with DoS attacks.
Roschke [21]	2010	Misuse	Active	NIDS	Configuration based Virtual Machine (VM) Security	High overhead due to multiple instances of IDS.
Dastjerdi [28]	2009	Anomaly	Passive	DIDS	Location independent protection.	Overhead grows dramatically as VMs on mobile agent increases.
Shyu [23]	2009	AR	Active		Linear scalability and low response time.	
Su [27]	2009	Hybrid	Passive	NIDS	Fewer false alarms.	
Herrero [26]	2009	ANN	Passive	NIDS	Handles high-volume network traffic.	
Byrski [29]	2008	Immune	Passive	NIDS	Independent of routing protocol and services.	
Sproull [22]	2007	Hybrid	Active	NIDS	Prevention model, low network overhead.	Hardware based, expensive

Table 1.2: Taxonomy of Cloud-based IDSs without VMI

1.3 Layout

This dissertation is structured as follows:

- Chapter 2 covers a detailed list of the problems touched on above. Section 2.1 describes the challenges caused by malware. Section 2.2 details the failures of traditional IDSs. Section 2.3 covers VMM weaknesses. Section 2.4 explores the challenges facing cloud forensics.

- Chapter 3 details the Literature Survey and consists of two subsections: Section 3.1 covers Virtual Machine Introspection. Section 3.2 provides a taxonomy of Cloud-based Intrusion Detection Systems.

- Chapter 4 elucidates the architecture, components and design of Oðinn. Section 4.1 illustrates the infrastructure and underlying platform that encompasses Oðinn, Yggdrasil. Section 4.2 describes the VMI component - Hliðskjalf. Section 4.3 covers the monitoring engine, Einherjar, that extracts memory artifacts from the guest VMs. Section 4.4 diagrams the database component - Mimisbrunnr. Section 4.5 describes the malware dropper - Loki. Section 4.6 outlines the heuristic detection agent - Muninn, and its validation. Section 4.7 reveals the technical details of the mitigation component - Balðr. Section 4.8 describes the malware analysis suite - Huginn. Section 4.9 provides the performance benchmarking of Oðinn. Finally, Section 1.2 juxtaposes Oðinn with other similar works.

- Section 5 outlines the areas of improvement for Oðinn that remain.

- Chapter 6 reiterates the findings and contributions derived from Oðinn.

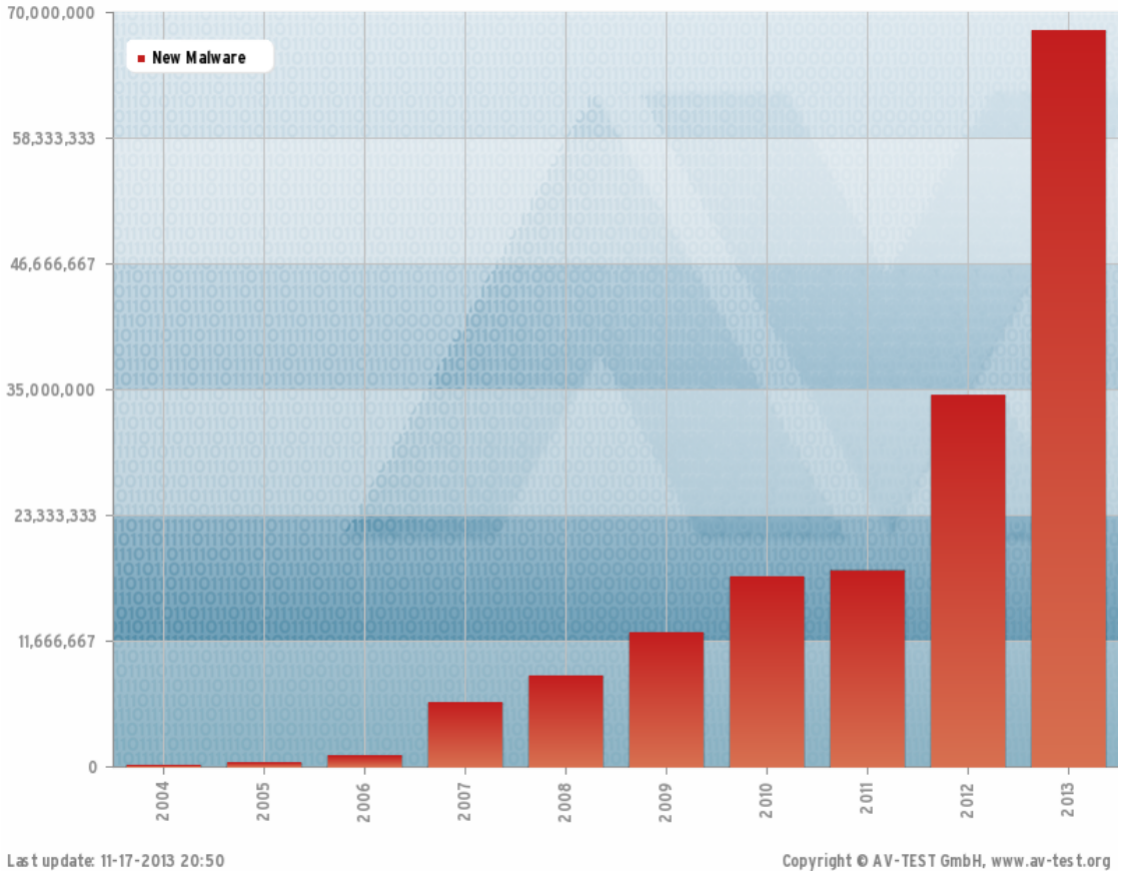
Chapter 2

State of Cloud Security

2.1 Malware

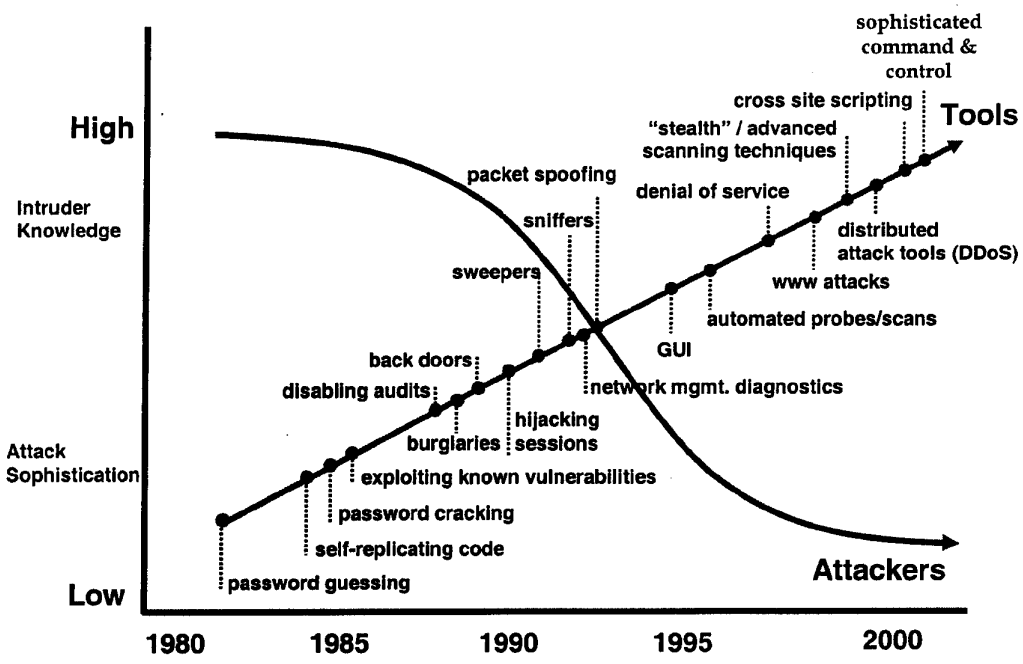
Malware is growing at an exponential rate (see Figure 2.1), with the amount of new malware samples from 2013 alone expected to exceed the combined total of malware samples since the first malware was created up to 2011 [1]. On average, a computer exposed to the internet can expect to be attacked every 39 seconds or less [30]. Concurrent with the general trend of malware growth is malware sophistication.

Figure 2.1: New Malware (last 10 years) [1].



The chart below (Figure 2.2) shows an inverse relationship between the sophistication of an attack and the level of technical knowledge an attacker requires over time. This trend has continued overall [30] but specifically with the vast sophistication in polymorphic and metamorphic malware [31] and the ease at which attackers can acquire off-the-shelf packer technology [32, 33, 34, 35] to encrypt their executable files.

Figure 2.2: Attack Sophistication vs Technical Knowledge over Time [2].



These factors combined along with the growing threat of organized cybercrime [36] has resulted in damages surpassing hundreds of billions of dollars [37, 38]. One major success story from this organized cybercrime was the Zeus botnet which at its peak was responsible for 90% of all banking fraud [39] and still constituted nearly half of all banking fraud in 2012 [38, 40].

2.2 Traditional Intrusion Detection Systems

An intrusion detection system is designed to detect and alert a host that has been compromised. This is accomplished by monitoring the host's behavior and properties (e.g.

I/O activity, network activity, internal state, etc.). The more properties an IDS can be said to observe, the better visibility the IDS possesses; this is significant as the ability of a malicious intruder to mimic normative host behavior is inversely correlated to the IDS's visibility. HIDS have the potential for a high level of visibility as its integrated into the host itself, and a variety of host-based architectures exist that achieve high visibility via system call trace analysis, integrity checking and/or log file analysis [41, 42, 43, 44]. Unfortunately, the very property that provides high visibility renders HIDS irrevocably flawed. Once the host is compromised, the HIDS is subject to malfeasance, which can vary from sensor blinding to misleading reports to using the HIDS itself as an additional vector of attack [45]. An alternative to HIDS is the NIDS. The primary benefit of the NIDS is a strong isolation from the hosts they monitor. As a result, there is a high level of tamper resistance unavailable to HIDS, and they can monitor with integrity long after a host has been corrupted. However, the same isolation that NIDS derive their advantage from also limits their visibility. A malicious adversary can craft network traffic that does not reveal that the host is infected [46].

HIDS have tried a variety of techniques to improve attack resistance, none of which have proven unsurpassable to date. *Kerckhoff's principle* is a series of design principles for security systems that can be surmised as a single axiom: a system is only secure if it is secure when an attacker fully understands its functionality [47]. In ignorance or perhaps in defiance of Kerckhoff's principle, many HIDS architectures attempt to obfuscate the IDS techniques being used via encryption/steganography [48]. A second approach saw the HIDS move from user space to inside the kernel, and while it offered a modicum of resiliency, the technique had a number of drawbacks [45, 49]. Many modern OSes have direct kernel memory access from a user level; which if disabled, results in a loss of functionality but if enabled, provides no protection for the HIDS residing in the kernel. An example involving the Linux OS is the user level modification of the kernel through loadable kernel modules (/dev/kmem) [50]. Even if one ignores the functionality issue, the OS itself very often falls to exploits due to

the sheer complexity of OSes. Regardless of whether it exists in user or kernel space, an IDS will typically fail open (unsecured state) or result in the entire system crashing. This type of attack can affect NIDS if an exploit generates large volumes of traffic to overload the NIDS [46]. Failing open is especially predominant in NIDS as failing closed (secured state) generally would suspend connectivity across hundreds if not thousands of hosts.

2.3 VMM Weaknesses

The properties of virtualization, while advantageous for security (See Appendix A.1) can also be highly detrimental. One core problem is that the x86 architecture was not designed with virtualization in mind and as a result, is not fully virtualizable [51]. Three methods of virtualization dominate the landscape: paravirtualization, binary translations and hardware-assisted virtualization [9]. Paravirtualization involves porting guest operating systems so that they do not use non-privileged sensitive instructions, but instead use ones that better cooperate with the VMM [52]. Binary translation executes guest VM code on an interpreter to handle sensitive instructions [51]. Hardware-assisted virtualization is the implementation of new functionality to hardware to support virtualization (e.g. AMD's AMD-V and Intel's VT-X and VT-I CPU processors). Each methodology has various advantages and drawbacks, but the systemic problem, the lack of inherent virtualization support in x86 and its successor x64, creates a number of VMM flaws.

2.3.1 Transparency

When all three properties - efficiency, equivalence and resource control - identified by Popek [53] are achieved in a "perfect" manner, then a VMM is fully transparent. In this state, there exists no method from within a guest VM to identify the VMM. However, due to the above problem, it is infeasible to make a fully transparent VMM [54]. This is unfortunate as VMM detection measures exist utilizing hardware attestion, instruction timing, and inconsistent CPU emulation can reveal a VMM [54, 55, 56]. A side effect of

this is the ability to make VMM-based rootkits [57]. This also impacts introspection itself because a fully transparent VMM can perform reverse engineering, recovering encryption keys embedded in malware and circumventing anti-debugging measures. Furthermore, a fundamental theoretical advantage of VMMs is their invisibility and thus, implicit trust that can be placed in their sanctity.

2.3.2 VM Escapes

The isolation property of virtualized systems greatly improves VMM security, but the isolation is not absolute. When full transparency does not exist and the VMM is exposed, the possibility of a VM escape exists. A *VM escape* is code executed by the guest VM that breaches the isolation barrier and executes outside of the VM environment lacking any oversight or control by the VMM [9]. Vectors for such attacks are due to the attack surface offered by the additional code base/complexity imposed by the VMM or by attacking virtualized devices, CPU caches or utilizing VMI passthrough mechanisms to access host resources directly (e.g. Graphic Processing Unit). VM escapes remain exceedingly rare in the wild but instances do exist, such as Bluepill [58] and Cloudburst [59].

2.3.3 Side Channels

Physical devices that do not pass through the VMM or are merely shared with the VMM lower VMM transparency and open up an attack vector known as a *side channel attack*. I/O devices, in particular, are very complex, and securing them is difficult without greatly impacting performance [60]. As such, several I/O side channel attacks have been created (e.g. Direct Memory Access to share information using memory locations [61, 62] and insecure Graphics Processing Unit isolation [63]). Network channels present similar challenges and significantly higher risk due to the remote access and integration into all remote cloud-based architectures.

2.3.4 Information Leaks

Information leaks are a subset of side channels that involve the deterioration or dissolution of the resource control property of VMMs. Information leak side channel based attacks include software methods that exploit convenience features between the VM and the host (i.e. clipboard sharing, shared folders, bridged networks, etc.) [64] as well as hardware methods disclose information about the physical machine, such as CPU state, memory usage, and network details (i.e. Cache-based attacks) [65]. Hybrid methods also exist, such as using information leaked by the timing of memory page operations to discover sections of shared memory pages that have been written to by another guest machine. Information leakage allows attackers to steal encryption keys, passwords, and other confidential information.

2.3.5 Image Standardization

In a non-virtualized architecture, adding an additional user requires additional equipment, installing an operating system and adjoining the newly created system to the network. In the virtual environment, it consists of copying a file and instantiated the copied VM image. The ease of cloning of virtual machines has a number of security flaws regarding management, identity and data retention. As the number of VMs increases, the management of the images becomes exponentially more difficult [66]. A number of significant performance cloud management responsibilities are tightly coupled to VM images:

1. efficient and reliable storage

2. low-latency retrieval and instantiation

3. fast transfer of data for VM migration (especially live migration).

In an attempt to resolve these responsibilities, cloud systems have used image similarity and image dedepulication procedures and inadvertently created security challenges. Deduplication removes duplicate data blocks when handling collections of images [67] and

image similarity enhances the amount of deduplication that occurs. From an optimization standpoint, the insures the total amount of image data to store is minimized; however, from a security perspective, this serves to increase the velocity of any attack vector exposed. Consider a scenario prior to cloud adoption. A large number of disparate physical machines running different operating systems or OS versions. The physical machines have different components (one has 2GB of RAM, another 4GB; one uses an AMD processor, the other Intel). Malware, especially ones that exploit memory regions do not have 100% infection rates. Thus, some machines will avoid infection simply because they lack a particular OS/application version that is being exploited or because of differences between the physical machines. However, in the post-cloud environment, shared memory regions, enforced image similarity and images residing on the same hardware causes infection rates to occur with greater frequency and to occur with greater velocities than were previously possible.

2.4 Cloud Forensic

"[T]o our knowledge, no research has been published on how cloud computing environments affect digital artifacts, and on acquisition logistics and legal issues related to cloud computing environments." - Beebe 2009 [68]

2.4.1 Memory Volatility

In current public clouds, virtual Infrastructure as a Service (IaaS) instances lack persistent storage. For example, in the Amazon Elastic Compute Cloud (EC2) cloud, if the guest machine is shutdown or rebooted, all volatile data is immediately lost. This situation has a number of forensically important issues. Should an intrusion take place and the adversary wish to hide evidence of the intrusion or worse, evidence of the intrusion's persistence, the system could be shutdown, rendering the volatile data unrecoverable by current FMA techniques. Additionally, a malicious entity could complicate forensic efforts further by using the EC2 cloud to create an instance to launch attacks from their guest machine and then

shut the guest machine down to ensure the only forensic data left is the network traffic. A confounding corollary to this would be the same malicious attacker using an obfuscated IP to connect to a guest VM and then claiming that their guest machine was compromised. With the loss of the volatile data, it would still be difficult to prove conclusively that the individual is responsible [69].

2.4.2 Live Analysis

One solution prior to the cloud for scenarios where collecting volatile memory after-the-fact was difficult is to use live analysis. Live analysis is a forensic examination (or logging) of the system during its running state. However, live analysis on volatile data has been long-fraught from a technical perspective. Akin to the observer effect in Physics, the forensic examiner that examines a live system will cause that system to change. The act of examining (e.g. attaching a debugger) can open/close network connection, create temporary files, modify a process, modify registry entries during queries, etc. From a legal perspective, once the system is changed, the evidence is contaminated and may be rendered invalid by a judge [70].

2.4.3 Validating Artifacts

An additional impediment is the difficulty in acquiring an exact copy of the snapshot volume for forensic analysis from current cloud-based systems. An example is the EC2 cloud where virtual hard drives (Elastic Block Storage values) are stored in the Amazon Simple Storage Service (S3). It is not possible to verify the integrity of the forensic disk image because Amazon does not provide checksums for their volumes. Ergo, it is impossible to claim at the legal standard for digital forensic collection that the volume downloaded from Amazon is identical to the one requested. Further complicating the issue is that even if Amazon did verify the volume, the investigator could still not positively assert the images

were the same because no hardware write blocker can be used to verify the integrity of the image [71].

Chapter 3

Survey of Literature

3.1 Virtual Machine Introspection

Guest isolation and its security applications were studied and formalized by [72], [73], and [74]. In response to the proliferation of malware over the last decade, guest isolation was re-purposed for instruction-level monitoring for malware analysis; however, research has been impeded by instruction-level monitoring being unsuitable for analysis predicated on high-level semantics, a problem termed the "Semantic Gap" [75, 76, 77]. Introspection was successively improved by additional works focused on attack replaying [78], control or integrity checking [79], a secure VM monitoring framework and implementation [80], and a host of introspection-based intrusion detection systems (IDSs) [81, 82, 83, 84, 85].

3.1.1 Semantic Gap

And how will you enquire, Socrates, into that which you do not know? What will you put forth as the subject of enquiry? And if you find what you want, how will you ever know that this is the thing which you did not know? - Meno's Paradox [86]

The semantic gap is a reformation of Plato's problem, a term, first used in linguistics to represent the gap between knowledge and the environment (e.g. How do children learn the immensely complex grammatical structure of languages without any formal education?) [87]. In computer science, the semantic gap represents the challenge of taking low level structures (environment) and reconstructing or bridging the high level structures that can be understood (knowledge) [88].

The semantic gap exists due to the isolation properties of the virtual machine interface. The VMM does not have special insight into the guest operating system's abstracted states

(e.g. processes, files, etc.). As a result, standard VMM design does not allow it to query a guest operating system and retrieve details which limit the introspection property of virtualization. Traditional introspection tools rely on extensive knowledge of a guest operating machine to reverse engineer the low level details garnered from introspection to recreate a view of the high level abstracts. This explicit knowledge includes location of kernel structures, memory structures and locations, and global variable locations. This leads to a lack of OS independence among introspection tools as individual tools require significant levels of semantic knowledge [17]. Additionally, when the guest OS is not open-source or its internals are not exposed openly, recreating the operating system semantics is often infeasible [89]. Two bridging methods, explicit and implicit, exist to overcome the semantic gap between the VMM and the OS.

Explicit Bridging

Explicit bridging relies upon interaction with the guest OS either through altering the kernel directly or via significant kernel data structure information. Many VMI architectures use explicit bridging because of its high level of performance and accuracy. However, explicit bridging has two primary weaknesses: One, if the kernel is compromised, the information it provides to the VMI may be incorrect and two, the kernel data structures are different from OS to OS and even from version to version, requiring significant upfront development costs for the VMI developers.

Implicit Bridging

Implicit bridging does not rely upon the guest OS and instead, uses architectural events, disk semantics and memory within the guest OS to infer its internal state. The primary advantage of implicit bridging is its resilience against compromise; however, in a manner similar to the HIDS/NIDS comparision, this resilency comes at the cost of visibility and the implicit bridging method often has performance issues and inaccurate information [90].

3.1.2 Analysis

Replay

The ability to replay, or log, events on a VM is useful not only for debugging OSs (which is why researchers introduced VMs in the late 1960s) but also for replaying compromises [91]. Replay is contingent upon the observation that computer operations are primarily deterministic [92]. As such, there are two critical aspects that the recording function must capture: the initial state and the precise moment of all non-deterministic events. To replay, the replay mechanism restores the VM to its initial state and inserts the non-deterministic events at the correct execution points. While VMI architectures using replay mechanisms are capable of analysing malware, they lack the capability to detect, prevent or mitigate the malware intrusion and are unsuitable as the sole component of production IDSs.

Live

Live or non-quiescent analysis as previously discussed in Section 2.4 suffers from the observer effect. However, when paired with introspection, the observer effect can be mitigated and if full transparency is achieved, eliminated. As most live analysis VMI-IDS architectures are currently reliant upon explicit bridging, they can be manipulated by attackers who compromise the host [93].

3.1.3 Monitoring

Passive monitoring is when the security tool monitors by non-resident scanning or polling. The introspection component is placed in a privileged VM and gathers guest-related information from the hypervisor. The analysis component must bridge the semantic gap to apply knowledge of guest OS (OS data structure semantics and positioning) to extract the high level structures by which intrusions can be detected and the development of those components is often a tedious and time-consuming task. *Active monitoring* occurs when

the introspection sensor is placed in the guest machine and was developed to overcome the difficulties of the semantic gap. Prior to virtualization, HIDSs performed active monitoring by intercepting every system call [94] or kernel hooks to track events and enforcing specific security policies [95]. Active monitoring has the advantage of detecting attacks quickly and preventing certain attacks but unlike passive monitoring, exposes the monitoring tool to subversion by the malware. Post virtualization examples of active monitoring include VMScope [96], a system call tracing monitor and ReVirt [91], an I/O monitor focused on backtracking and replay functionality. Hybrid approaches, utilizing in-guest and out-of-guest monitors has emerged, such as, Lares[97] and SIM [98], in an attempt to retain the semantic rich internal view with in-guest sensors and add resilience to malware tampering with out-of-guest sensors.

3.1.4 Summary

Name	Year	Monitor	Analysis	Bridge	Advantages	Disadvantages
Exterior [14]	2013	Passive	Live	Implicit	Mitigates attacks in real time using secure VM	Requires 1:1 ratio of Secure VMs to guest VM.
SPECTRE [99]	2013	Passive	Live	Explicit	Does not need to trust Hypervisor (BIOS based)	Only detects attacks in memory.
TimeScope [100]	2012	Passive	Replay	Explicit	Low Overhead (62% worst case, I/O based)	Limited (4) analysis tools.
Maitland [101]	2012	Active	Live	Explicit	High effectiveness at unpacking malware.	High overhead when knowledge base lacks signature.
VMST [102]	2012	Passive	Live	Implicit	OS Agnostic approach to sensor creation.	Very high overhead (monitors all data)
Odin 0.5 [103]	2012	Passive	Live	Explicit	Restores corrupted guest OS to virgin state.	Not tested on large sample of malware.
InSight [104]	2011	Passive	Live	Implicit	Opensource, highly moddable.	A framework, sensors must be created by user.
EXAMIN-C [105]	2011	Passive	Live	Implicit	Moderate overhead.	Cannot read guest OS user-space application memory.
Virtuoso [17]	2011	Passive	Live	Implicit	OS agnostic	Cannot trace across multiple virtual address spaces.
HIMA [106]	2009	Active	Live	Explicit	Enforces guest memory protection, low overhead.	Writable/executable memory pages not supported.
Ether [83]	2008	Passive	Replay	Implicit	High effectiveness at deobfuscating malware.	Dependent on Intel VT hardware/OS support.
Lares [97]	2008	Active	Live	Explicit	Low overhead, fine-grained memory protection.	Can be subverted.
VMWatcher [81]	2007	Passive	Live	Implicit	High accuracy when comparison-based scheme works.	Very limited semantic gap narrowing.
Antfarm [84]	2006	Passive	Live	Implicit	Very low overhead (2.5%), supports SPARC.	Only detects process and I/O scheduling.
Livewire [107]	2003	Passive	Live	Explicit	Very expressive policy language.	Lengthy guest interruption when malware detected.
ReVirt [91]	2002	Passive	Replay	Implicit	Very low overhead	High storage use (up to 1.4GB/day)

Table 3.1: Taxonomy of VMI-IDSs

Ultimately, the problem with using the above methodologies is the assumption that the kernel data being introspected adheres to the internal structure templates as prescribed by the operating system. The vast majority of the cited VMI tools implicitly trust that the guest OS is conforming to these internals; however, once OS is compromised, this assumption is rendered invalid [108].

Forensic Memory Analysis has, in isolation to virtualization monitors, focused on bridging a similar semantic gap, the elucidation of high-level artifacts from objects in physical

memory. Examples include tools that locate processes and threads [109], reveal Dynamically Linked Libraries (DLL) injections [110], retrieve files mapped in memory [111], and expose Windows registry information [112]. Most recent is a framework for forensic memory tools called Volatility [10] which allows for the rapid prototyping of FMA and includes plug-ins that replicate the above tools and dozens of others. We previously explored an amalgamation of forensic memory analysis with virtual memory introspection [103] and this dissertation builds on that work.

3.2 Cloud-based IDS Heuristics

3.2.1 Anomaly Detection

Anomaly Detection (AD) applies statistical analysis (e.g. data mining, hidden markov models) on the data generated by legitimate users to identify malicious behavior from normative behavior. The primary advantage of AD is its ability to detect attacks that have not been previously known. [113] Examples of AD in Cloud-based IDS are analysis on protocol based attacks [114], a lightweight monitor for real-time, high performance detection [115] and numerous hybrid solutions, a technique using two or more heuristics, include AD as a component [16, 28, 107, 116]. The critical weakness of AD is the high false alarm rate for both known and unknown attacks compared to other heuristics.

3.2.2 Artificial Neural Network

Like AD, Artificial Neural Network (ANN) attempts to classify data as normative or anamlous [117] with the caveat that it does so with incomplete data through generalization rather than the large training sets usually required by AD [118]. The types of ANN used in IDS are: [117]:Multi-Layer Forward-Feed (MLFF) neural nets, Multilayer Perceptron (MLP) and Backward Propagation (BP). The inclusion of additional hidden layers [119] or a self-organizing map [120] increases accuracy but at the cost of complexity [121].

3.2.3 Association Rule

Association Rules (AR) is a heuristic which is very powerful at finding variants of known attacks by detecting frequent subsets (e.g. shared features across multiple attacks); however, database scanning performance was prohibitive [122]. Performance has been ameliorated with subsequent research at the cost of higher false positive rates [115] and recently the false positive rate was lowered while maintaining the performance improvements [123] making this heuristic a desired avenue to explore.

3.2.4 Fuzzy Logic

Fuzzy logic [118] handles the vague descriptiveness of what an intrusion entails to provide flexibility on whether or not an intrusion has occurred. Fuzzy logic-based IDS have been used to detect network intrusions (i.e. Ping of Death, SYN flooding) [124]. Fuzzy logic has been combined with ANN to improve training time [125]. However, neither of these approaches were capable of real-time intrusion detection. A hybrid technique of Fuzzy logic and ANN has achieved real-time detection capabilities to detect denial of service attacks before the network intrusions cripple the service. Fuzzy logic provides some flexibility to the uncertain problem of intrusion detection [27]. Cloud-based IDS have used fuzzy logic with ANN for real-time detection of unknown attacks [16].

3.2.5 Genetic Algorithm

Genetic Algorithms (GAs) optimize parameters or select the best subset of features for performance/accuracy considerations in other IDS heuristics [126, 127]. GAs typically suffer from the best fit problem and have to be carefully tuned to prevent over- or under-fitting [128]. An additional weakness is high complexity of the fitness function which can be a performance consideration [129]. In Cloud-based IDSs, GAs have been used with fuzzy logic [126] wherein the GAs provide the optimal parameters for the fuzzy logic function, resolving the fitting problems discussed previously.

3.2.6 Misuse Detection

Misuse detection requires signatures or a predefined knowledge base to engage its pattern matching algorithms against. When known attacks exist within its predefined set, misuse detection has high accuracy, low response latency and low overhead, making it an ideal solution. However, even small variations in an otherwise known attack can reduce the accuracy of misuse detection [113] and it has no ability to detect unknown attacks [130]. In Cloud-based IDSs, misuse detection has found a fair amount of use as a detection agent for VM intrusions [131, 132, 133, 134].

3.2.7 Support Vector Machine

Support Vector Machine (SVM) [118] is a heuristic best suited for environments with low sample size. In such environments, despite its reliance on binary data and lower feature coverage it generally outperforms other heuristics [135]. Examples of Cloud-based IDS using SVM include a NIDS using SNORT and configurable firewalls [136].

Heuristic	Characteristics	Limitations
Anomaly [126]	Uses statistical model on collected behaviour for detection. Lower false alarm rate for unknown attacks.	Identifying attacks is slow. Accuracy is entirely dependent on collected data.
ANN [137]	Classifies unstructured network packet efficiently. Multiple hidden layers in ANN increase efficiency.	High time complexity during training phase. Large sample size required for effectiveness.
Fuzzy Logic [118]	Used for quantitative features.	Relatively low detection accuracy.
GA [127]	High efficiency Automatically prunes less useful features	Computationally complex. Trends towards overpruning.
Hybrid [138]	Highest accuracy	Highest computational cost.
Misuse [130]	Lowest computational cost. Highest detection for previously cataloged attacks. Matches intrusions to a preconfigured knowledge base.	Cannot detect variants or unknown attacks. False positives very high in unknown attacks. Knowledge base can be polluted.
AR [139]	Detects known and variant attacks in misuse detection.	No ability to detect entirely unknown attacks. Rule generation relies on a myriad of DB scans.
SVM [135]	High classification accuracy with limited sample data. Can handle massive number of features.	Limited to discrete features.

Table 3.2: Taxonomy of IDS Heuristics

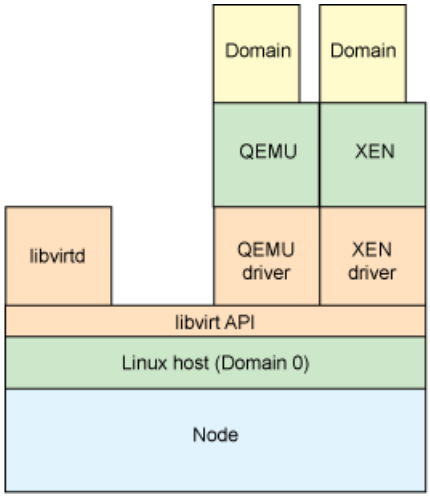
Chapter 4

Oðinn

4.1 Yggdrasil

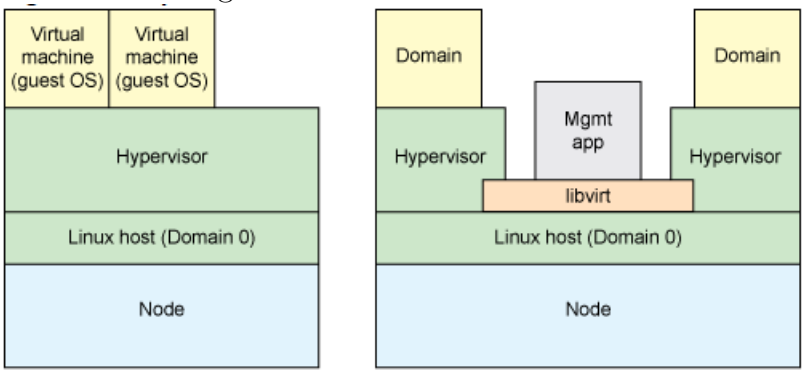
I know that I hung on Yggdrasill
nine whole days and nights,
stabbed with Gungnir, dedicated to Odin,
myself to mine own self given,
high on that Tree of which none hath heard
from what roots it rises to heaven.
None refreshed me ever with food or drink,
I peered right down in the deep;
crying aloud I lifted the Runes
then back I fell from thence [140].

Figure 4.1: Libvirt's driver architecture



Yggdrasil serves as the infrastructure that manages the underlying virtualization technologies comprising Oðinn and is an extension of the libvirt API (4.2). Libvirt [141] is an abstraction layer library for hypervisor management. Conceptually, libvirt initially existed as two main components - a hypervisor agnostic API and a collection of hypervisor specific drivers (4.1), but now includes several additional components to manage storage pools, networks, etc. all written in C.

Figure 4.2: Libvirt's use model



Our primary motivation in using libvirt is the agnostic nature of its API, supporting a number of hypervisor implementations (see Table 4.1). An additional factor was libvirt serving as an underlying technologies for many cloud technologies (e.g. OpenStack) and remains the most popular hypervisor management technology for the cloud [142]. Part of this reason is libvirt's highly scalable nature, able to efficiently monitor over 2,000 active guest nodes at one time [143].

Hypervisor	Description
Xen	Hypervisor for IA-32, IA-64, and PowerPC 970 architectures
QEMU	Platform emulator for various architectures
KVM	Linux platform emulator
LXC	Linux (lightweight) containers for operating system virtualization
OpenVZ	Operating system-level virtualization based on the Linux kernel
VirtualBox	Hypervisor for x86 virtualization

Table 4.1: libvirt hypervisor support

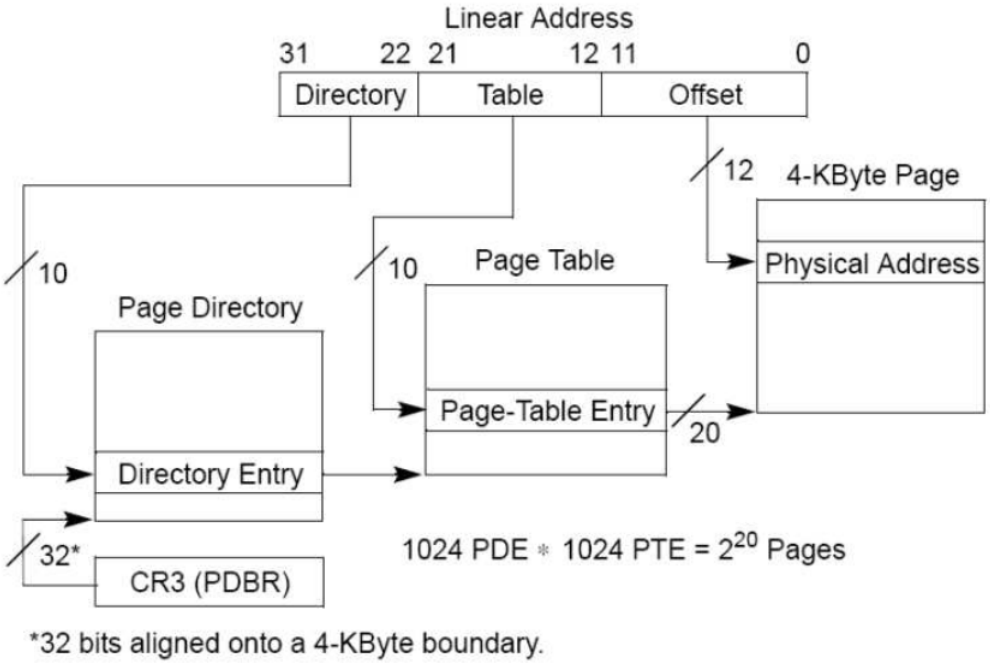
The principle changes we made to libvirt was in the modification/extension of its Python API wrapper. By design, libvirt is intended to support a command line interface rather than

a wrapper and so when encountering something as simple as failing to find a virtual machine requested, it raises an error and kills the entire process. For obvious reasons, this is a poor mechanism when integrating libvirt into an always-on system such as an IDS, so the first task required was to change the manner in which it raised errors. However, this added a number of complications. If the errors were simply ignored, all subsequent commands would be incorrect so Yggdrasil serves primarily as a mechanism to prevent libvirt-based errors from ever occurring. For instance, if one tries to issue a start VM command, libvirt will simply try to start the VM naively whether it has been defined, is already started, etc. Yggdrasil will ensure it has a valid configuration file, that it has already been defined and that its not currently active; if any of these steps fail, Yggdrasil will self-correct and only then start the VM. Additionally, Yggdrasil has a subcomponent that integrates Hliðskjalf with libvirt so that various VMI interactions have this error checking and correction capabilities.

4.2 Hliðskjalf - VMI Library

Hliðskjalf is an implementation and fork of the LibVMI [5] API. LibVMI evolved from the XenAccess [97] project used in our previous implementation [103] of a Hypervisor-based Intrusion Detection System (VMI-IDS). LibVMI was chosen because it is written in C for optimal performance, its API is focused on reading and writing to/from a guest's virtual memory, and it supports the two largest open source virtualization platforms (Xen and KVM). Additionally, it supports both 32/64-bit operating systems. By default, libVMI does not actually provide implicit or explicit semantics for any particular OS support; however, it does provide implicit semantics that, when integrated into Volatility [10], results in explicit semantics for most current Windows, Linux and Mac OS versions. In the following sections, we will explore how we bridge the semantic gap, how we access Volatility's address space and finally, how integrate it into Einherjar.

Figure 4.3: x86 Virtual to Physical Memory [3]



4.2.1 Bridging the Semantic Gap

The implicit information for libVMI is drawn from the x86/x64 virtual memory architecture (see Figure 4.3). When a virtual address is translated to a physical (linear) address, its location is retrieved from a page directory and page table. The register which holds the page directory address is the Control Register 3 (CR3) and the most significant 10 bits of that register (31-22) hold the Page Directory Base Register (PDBR). The PDBR stores the index of the page directory and the location within the page directory of the pointer to the correct page table. The next most significant 10 bits (21-12) contain the index of the page table and a pointer to the proper page table entry which when combined yield the location of a 4-Byte Page. The final 12-bits contain the offset that points to the physical address within the 4-Byte Page. An introspection library capable of tracking the CR3 register is one that is capable of mapping out the physical addresses of a guest virtual machine. Additionally, it is necessary for the introspection library to be able to determine if the machine is or is not

using Physical Address Extension (PAE), this is accomplished by reading Control Register 4 (CR4) , bit 5. If the bit is set, PAE is enabled; otherwise, it is not.

4.2.2 Integrating with Volatility's Address Space

This information alone is insufficient in integrating with memory analysis tools as these tools interface with the system RAM in their own unique format, called Address Space (AS). Volatility utilizes a stacked AS model which is designed to aid plugin developers by abstracting away the underlying formats. As Hlidskalf intends to use this model to structure a guest machine's memory, the model must be understood for integration. Volatility address spaces inherit from the base class (volatility.addrspace.BaseAddressSpace). Address spaces are initiated with a URI (derived from volatility.conf.ConfObject().LOCATION). The returned list is traversed with the base address until it matches and returns the address space to the calling function. For simplicity reasons, we use the lowest level semantic model of Volatility's AS, which is the File. This is intended to be a bit-by-bit copy of virtual memory; however, we will mirror active memory to it later. Thus, we only need to understand the base class, which has the following attributes:

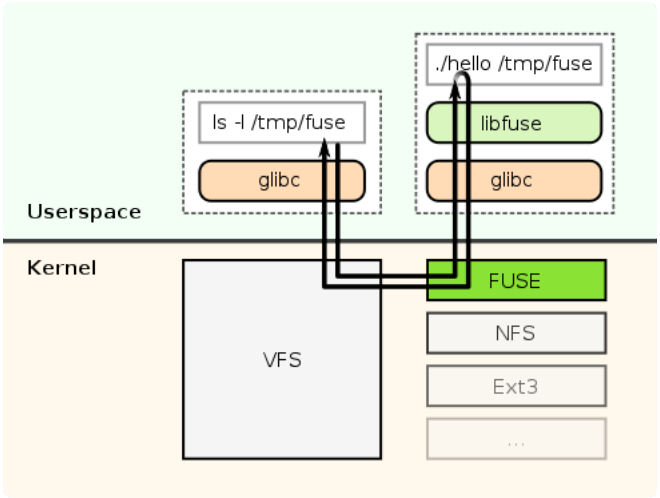
- base: This is the underlying address space given by the constructor.
- profile: This is the explicit semantic information for the Operating System and Version (set automatically by volatility.conf.ConfObject().PROFILE)
- name: Name of the address space
- dtb: The Directory Table Base derived from the CR3 discussed in the previous section.
- pae: A boolean indicated for whether the CR4, bit 5 is set for this AS's paging space.

The base class also requires following functions:

- Constructor (__init__)
- read
- zread
- is_valid_address
- get_available_addresses
- get_available_pages
- vtop

Of those, libVMI supports by all but the zread function. The zread function returns an unavailable page of read memory as a page of available padded zeros of the proper size. This is necessary for Volatility because when reading kernel/process memory from a large range, it becomes increasingly probable that some of the memory pages being read will not be in resident memory (e.g. paged to the disk). Whereas Volatility is written in Python and libVMI is written in C, it is necessary that some wrapper exist to support the already existing functionality natively and support the zread function. The Python language allows the extension of C libraries to have semantically equivalent Python functions so an API wrapper is used.

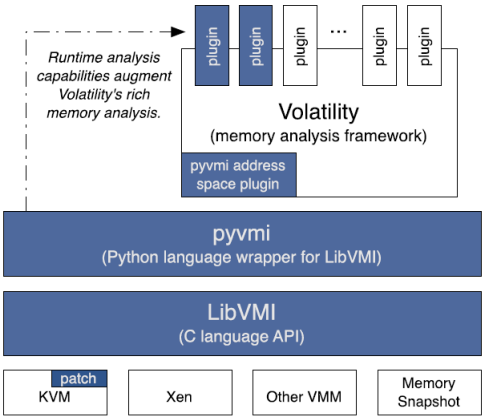
Figure 4.4: FUSE architecture in simple stat (filesystem call) [4].



A Python wrapper isn't sufficient by itself because, despite creating a format (address space) that volatility can understand, the guest machine's memory must be constantly mirrored in order to be used in real-time. A file format alone isn't sufficient and instead, a

file system is necessitated. This is possible through a Linux library called Filesystem in UserSpacE (FUSE). FUSE is a library in which filesystems can be created whose data and metadata can be provided by a userspace process. This is important for Oðinn because guest machines should exist in user space, so foregoing access to the kernel space is of high importance. Figure 4.4 above demonstrates how FUSE works in a basic example - the libfuse library built on glibc provide the API for the FUSE (fuse.ko) kernel module. The kernel module forms a connection between the filesystem daemon, the Virtual File System (VFS), and the kernel [4].

Figure 4.5: Software stack for Volatility address space plugin [5].



Putting it all together, there now exists a seamless way for Volatility sensors to engage a guest virtual machine in real-time without an external developer knowing any explicit OS semantics (see Figure 4.5 above).

4.3 Einherjar - Guest Monitors

4.3.1 Initialize

While other implementations of introspection have utilized Volatility, none thus far provide full integration. There are three attributes that a memory analysis engine needs to optimize its scanning. The first, the Directory Table Base (DTB) traverse the _EPROCESS structure to find the ActiveProcessLinks, PsActiveProcessHead and other doubly-linked list

attributes. Fortunately, the DTB is provided by Hliðskjalf as it is contained in the CR3. Second, the _KDDEBUGGER_DATA64 Kernel Debugger (KDBG) structure which tracks the state of the operating system to provide data when a crash occurs provides quicker access to close to 100 kernel variables. The final value is the Kernel Process Control Region (KPCR) which the windows kernel stores processor information (running processes, network connections, open handles, etc.) As previously mentioned, the DTB is already known, so our approach focuses on discovering and storing the KDBG and KPCR values.

Every version of Windows has a specific virtual address in which the KDBG and KPCR are located. However, the virtual address information alone is not sufficient since introspection accesses the physical address of the virtual machine. To translate this, we need to know whether the page size is 4KB, 2MB, or 4MB and this is dependent on whether PAE, PSE, and PS bits are set. This knowledge can be derived from our existing knowledge of the CR3 value and, upon knowing the page size, we can correctly translate the virtual memory address of the KDBG/KPCR structures to their physical counterpart thereby obtaining the KDBG/KPCR values and caching them for later usage. This results in significant improvement in speed as seen in Table 4.2.

SSDT	Time	Improvement
Unoptimized	3.35s	–
DTB	2.51s	25%
DTB, KDBG, KPCR	1.20s	64%
GDT	Time	Improvement
Unoptimized	0.84s	–
DTB	0.51s	39%
DTB, KDBG, KPCR	0.27s	68%

Table 4.2: Performance increase of Initialize on System Service Dispatch Table (ssdt) and Global Descriptor Table (gdt) modules.

4.3.2 Bifrost

Another area where current implementations using Volatility are inefficient is their reliance upon vol.py. This requires all analysis to be driven via system calls, which spawn off shells of the Python interpreter for each scan. Considering the nature of the cloud where hundreds or even thousands of guest machines exist on a single server and each guest machine requiring a dozen or more simultaneous scans and massive overhead is incurred. Our solution was to make a number of modifications to Volatility and create a new module, called Bifrost, to convert Volatility into a Python library that can be driven by other components. Additionally, we design an entirely new renderer for all of the Volatility plugins we utilize to return data formatted for our database. Volatility only provides a default text renderer which current implementations scrape and then perform substantial preprocessing upon before storing, a highly inefficient design.

4.3.3 Geri - Doubly-linked List Crawlers

Geri	Finds objects by walking doubly-linked lists
pslist	Process objects (PsActiveProcessHead)
modules	Kernel Module objects (PsLoadedModuleList)
threads	Thread objects (ETHREAD)
filelist	Handle objects
dlllist	DLL objects (PsLoadedModuleList)
ssdt	Service objects (ETHREAD.Tcb.ServiceTable)
getsids	SID objects (_TOKEN)
idt	Interrupt Descriptor Table (KiSystemService)
gdt	Global Descriptor Table (KiSystemService)
callbacks	Kernel callback events (DbgkLkmdRegisterCallback, etc.)

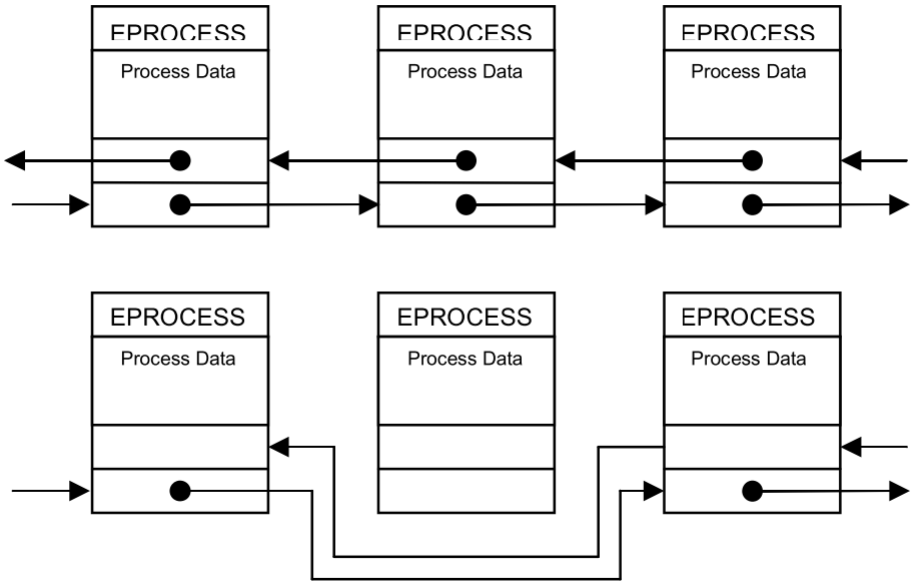
Table 4.3: Volatility modules used by Geri [10].

Having integrated the address space of Volatility using Hliðskjalf and the aforementioned fork of volatility, Einherjar, it is now possible to implement our collection of monitors. We divide our monitors into two categories based on their methodology. The first collection of monitors, called Geri (4.3), walk doubly-linked lists or otherwise access data that can be

subverted by malware. This collection rapidly returns results and thus we poll this set of monitors every few seconds. The second collection, Freki (4.4), scans the memory space for structures and objects that correspond to OS specific nomenclature. These execute much slower, every 10 seconds (time spent is relative to the size of the RAM in the guest OS) but the OS semantics retrieved are difficult and sometimes impossible to alter by malware. This is important since intersecting the data from Geri and Freki, the differences are caused by malware with a high probability. This insight will be explored in detail when we validate and implement detection schemes.

4.3.4 Freki - Pool Tag Scanning Monitor

Figure 4.6: Process hiding using the DKOM technique [6].



Windows keeps track of active processes through a pointer called PsActiveProcessHead, which exists in every process's EPROCESS structure. Within it is a LIST_ENTRY field which contains a doubly-linked list (i.e. each portion has a forward link - F_LINK pointing to the head of the next process's EPROCESS struct and a back link - B_LINK pointing to the previous head). A common malware technique is to alter the doubly-linked list so that they reroute around a structure, such as a process created by the malware, removing it from

the list. This renders traditional live analysis tools, as well as most VMI implementations, from detecting the orphaned process. Forensic memory tools such as Volatility have look for these orphaned structures by scanning the entire memory for sections of memory that match EPROCESS structures. Normally, this can only be done on post-exploitation on memory images; however, with ODinn, we can do this in real-time by intersecting the results of the traditional link crawl method and the forensic memory method (e.g. output of the volatility plugins pslist and pscanner) to immediately reveal the orphaned process that the malware has created.

Freki	Finds memory structs using pool tags (POOL_HEADER)
psscan	Process objects (_EPROCESS)
modscan	Modules objects (LDR_DATA_TABLE_ENTRY)
thrdscan	Thread objects (_ETHREAD)
filescan	Open file objects (FILE_OBJECT)
connscan	TCP/IP objects (_TCPT_OBJECT)
driverscan	Driver objects (DRIVER_OBJECT)
symlinkscan	Symbolic link objects (_REPARSE_DATA_BUFFER)
mutantscan	Mutex objects (KMUTANT)
svcscan	Service objects (SERVICE_TABLE_ENTRY)

Table 4.4: Volatility modules used by Freki [10].

A new optimization to Freki is the combination of all the scanners found in the table as a single scanner. Instead of each scanner crawling through every object, discarding the ones that do not fit its particular scan, we scan the entire pool of memory structs once, processing all of the structures found as one scan. This increases the runtime of the scan slightly compared to a normal scan, but since we obtain the results of all the scans at once, the total time spent scanning is reduced by over 85%.

4.4 Mimisbrunnr - Database

Once a scan is completed, the results are passed to the function in Figure 4.7 below. A series of interactions with Mimisbrunnr then occur. First, the function creates a temporary table that contains the columns and metadata associated with the volatility plugin scan.

Figure 4.7: How modified volatility scans results enter into Mimisbrunnr.

```
def scan(vp):
    """Updates Mimisbrunnr(database) with results from a volatility plugin scan of a guest VM.

    Keyword arguments:
    vp      -- An instance of our volatility plugin class.
    vp.name -- Name of a volatility plugin
    vp.com  -- Command arguments for a volatility plugin.
    vp.tmp  -- Temporary table structure for a volatility plugin.
    vp.attr -- Type (ex. int, str) for each element in result list.

    Note: Volatility plugins have been modified to return a list rather than print to stdout.
    """
    results = subprocess.Popen(vp.com).communicate()[0]
    with db.cursor() as c:
        c.execute("CREATE TEMPORARY TABLE {0}".format(vp.tmp)
        c.executemany(("INSERT INTO {0}_polled ({1}) VALUES ({1})").format(vp.name, vp.attr), results)
        #Update alive status for all existing artifacts that are still active
        c.execute(("UPDATE {0} SET atime=now() WHERE ({1}) IN (SELECT {1} FROM {0} WHERE dead=false "
                   "INTERSECT SELECT * FROM {0}_polled)").format(vp.name, vp.attr))
        #Mark as dead existing artifacts prior that are no longer active
        c.execute(("UPDATE {0} SET dead=true WHERE ({1}) IN (SELECT {1} FROM {0} WHERE dead=false "
                   "EXCEPT SELECT * FROM {0}_polled)").format(vp.name, vp.attr)
        #Insert all new artifacts discovered
        c.execute(("INSERT INTO {0} ({1}) (SELECT * FROM {0}_polled EXCEPT SELECT {1} FROM "
                   "{0} WHERE dead=false)").format(vp.name, vp.attr))
```

Second, the data from the volatility plugin is inserted into this temporary table. It then intersects with the permanent table associated with the plugin and where memory artifacts are still active, updates their atime (alive time) to the current time. It then sets memory artifacts that were no longer present to the dead state (dead=true). Finally, it inserts all newly discovered memory artifacts into the permanent table from the temporary one. With this information, it is possible to compile an accurate timeline of the events occurring within the system as well as easily notate during training when the malware is inserted and flag every row in every table that occurs after that time as infected for the purposes of creating detection rules. The major advantage of this process is that analysis only occurs at each individual node and no data transmission needs to be managed, decreasing the complexity of the system and overhead generated.

4.5 Loki - Malware Dataset Generator

In order to rapidly generate training data to seed Mimisbrunnr with data, we need a tool that would quickly, efficiently and accurately represent real malware attacks and post-exploitation activity. The tool, Autosploit [103], is a custom testing framework that generates exploitation and post-exploitation activity augmented with Metasploit [144]. Metasploit allows the user to exploit known vulnerabilities and inject arbitrary payloads into the vulnerable processes. One of the most widespread payloads is the meterpreter, which is a small shell-like binary that allows a user to run various commands and common tasks on the remote machine that is being exploited. Autosploit gave users access to a GUI that had various buttons on it that performed the meterpreter tasks we were interested in. Common post-exploitations tasks include:

- Initiating three elevations of privilege vulnerabilities
- Killing some anti-virus processes
- Hiding our NT/System access inside of a normal user account
- Launching a remote cmd shell
- Attaching a keylogger to a process
- Creating a backdoor as a service
- Creating a backdoor as a port
- Deleting event logs
- Taking pictures of the victim using their webcam
- Recording the victim via their microphone
- Dumping user password hashes
- Dumping the victim's registry

- Downloading an arbitrary file from the victim's computer

While the tool accurately reflected the modus operandi of many attackers and is broad in its capabilities, it did not fully encapsulate the full attack surface of infrastructure exposed to the Internet. Historically, IDSs were hampered by proprietary test data and thus, results were rarely reproducible. This need was ameliorated by Lincoln Laboratory (LL) when they created the IDEVAL dataset [145] to serve as an evaluation benchmark tool. The IDEVAL dataset is reliant on generated traffic from network services (e.g. ping, finger, dns, smtp, etc.) and despite its wide range of attacks (more than 50), it is unsuitable for testing with a VMI-IDS such as Oðinn. However, another set of testing data exists, that by its very nature is not proprietary, malware.

Instead, we opted to replace Autosploit with a new malware dropper called Loki using live malware retrieved from Sandia National Laboratory (note: author is currently employed by Sandia). Botnets were prioritized due to their sophistication, threat level, and malware obfuscation techniques, but numerous other malware was gathered. Careful steps were taken to limit the network capabilities of the VMs prior to infection, and the VMs were destroyed once sufficient data was gathered. In our training set, 222 malware samples were gathered.

4.6 Munnin - Rule Generation and Detection

Once Loki has filled Mimisbrunnr with sufficient data (222 malware samples), the various tables in the database associated with the guest VM is dumped. Then a preprocessing script is executed on the data to discretize the values as well as map the discrete data to its original form. Then, a number of binary classification techniques and machine learning algorithms are run against the dataset. The Incremental Reduced Error Pruning (IREP++) rule induction classifier used in our prototype required improvements in the areas of rule complexity and temporal analysis. Meta mining techniques for supervised learning, such as those proposed by [146], achieve data model compaction and improved scalability, a necessary factor when transitioning our current prototype to a production cloud. In the development

of an IDS, the ultimate goal is to achieve the best possible accuracy for the task at hand while minimizing false positives and negatives. This objective naturally leads away from our previous methodology of a single classifier. To augment Muninn, the Orange [147] library is utilized. Orange's backend is C++, allowing efficient use of its classifiers while its frontend is written in Python, allowing for near seamless integration into Odin. Orange allows numerous classification schemes, including Naive Bayes, k-Nearest Neighbors (KNN), SVM, ANN and the C4.5 tree classifier [148].

We bind together all the classifiers using an algorithm called ensemble stacking [149]. The term "ensemble" refers to the combination of multiple weak learning algorithms or weak learners [150]. The methodology we seek to use for manipulating our training sets is termed Bagging. Bagging (Bootstrap Aggregation) takes the training set and creates a bootstrap replicate from samples of the original training set drawn randomly with replacement. Each bootstrap replicate contains, on the average, 63.2% of the original training set, with several training examples appearing multiple times [138].

4.6.1 Binary Classification

In machine learning classification approaches, a classifier is generated from training data to predict the outcome of a class attribute $C \in \{1, ..., r\}$, given the domain features $X = (X_1, ..., X_d)$, of a hidden class instance $x = (x_1, ..., x_d)$. In our approach, we use discrete domains $X_i \in \{1, ..., r_i\}$ with (X, C) representing a random vector with a probablity distribution of $p(x, c)$. A classifier maps X into C:

$$classfier : \{1....., r_1\} \times \cdots \times \{1, ..., r_d\} \rightarrow \{1, ..., r\}$$

and is a deterministic function of algorithm (A) operating on a training set
$S_n = \{(x^{(1)}, c^{(1)}, ..., (x^{(n)}, c^{(n)}\}$ [151]. Classifiers in machine learning are judged primarily on their prediction error estimation due to the inability to calculate the exact prediction error in most real world problems. The k-fold cross-validation method [152] is one of the leading methodologies for prediction error estimation and our choice of 10-fold cross-validation is the

optimal solution between the two goals of minimizing bias and computational complexity [151].

First-order Metrics

In binary classification model (classes: true and false), there are two possible prediction errors derived from a confusion matrix as shown in Table 4.5:

		Predicted Class		
		p	**n**	
Actual Class	**p′**	True Positive	False Negative	PPV
	n′	False Positive	True Negative	NPV
		SEN	SPC	ACC

Table 4.5: Confusion Matrix for Binary Classification

False positives (FP), also known as Type I errors, represent the first type of prediction error and false negatives (FN), also known as Type II errors, represent the second type of prediction errors. Derived from these values are a number of properties that are valuable in assessing the classifier. Prevalence (PRV) is the probability of any given sample having an actual positive value ($(TP + FN)/Total$) . Sensitivity (SEN) or True Positive Rate, is the probability that a positive prediction is correct ($TP/(TP + FN)$). A high sensitivity value is reliable when the result is negative because a negative result with 100% sensitivity rules out a positive outcome whereas a positive result could still contain a false positive. Specificity(SPC) or True Negative Rate, is the probability that a negative prediction is correct ($TN/(FP+TN)$). A high specificity value is reliable when the result is positive. This differs from the Positive Predictive Value (PPV) ($TP/(TP + FP)$) which is the probability that a positive prediction is correct. A high PPV indicates that a predicted positive value is

actually positive. Likewise, the Negative Predictive Value (NPV) represents the probability that a negative prediction is correct ($TN/(TN + FN)$). Finally, accuracy (ACC) is the nearness of calculation of a predicted value to its actual value regardless of type ($(SEN \times PRV) + (SPC \times (1 - PRV))$).

First-order metrics are useful in validating rules generated by the model. For example, suppose a tree model generates as its first branch on whether or not a variable is negative. If the model has a high sensitivity score, then this model is worth exploring further; however, if the sensitivity was low, we immediately know the model is flawed.

Second-order Metrics

While these first-order properties are useful in validating specific aspects of the model, they do encapsulate the overall quality of the model. For this, we use second-order metrics. The F_1 score or F-measure is a rating given by the harmonic mean of the PPV and sensitivity $2 \cdot \frac{(PPV \cdot SEN)}{(PPV + SEN)}$. This metric is biased towards the positive class, and hence negative features, even if inverted, are devalued compared to positive features. Due to this, it is preferable to assess this rating in conjunction with others [153].

The Matthews Correlation Coefficient (MCC) is the correlation coefficient between the observed and predicted binary classifications:

$$\frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

and ranges between -1 to 1, with a value of 1 being perfect prediction, a value of 0 being equivalent to random prediction and -1 the perfect inverse of prediction [154].

The Receiver Operating Characteristics (ROC) curve is a two-dimensional graph in which the sensitivity is plotted on the Y-axis and the False Positive Rate (1 - specificity) is plotted on the X-axis. The primary value of the ROC is that one can graph multiple classifiers and juxtapose their performance visually. However, one can reduce ROC performance to a single scalar value by calculating the Area under the curve (AUC). As the AUC is a section of an area of the unit square, it will always be between 0 and 1.0. Moreover, because

random values will produce a diagonal line on an ROC curve, we can further discriminate the performance of a given classifier as being between 0.5 and 1.0. The AUC of a classifier is equivalent to Wilcoxon test of ranks and represents the probability that the classifier will rank a randomly chosen positive instance higher than randomly chosen negative instance [154].

The Chi-squared (χ^2) test measures the divergence from the distribution expected under the assumption that the feature occurrence is independent of the class value and is calculated as:

$$\chi^2 = \sum_{i=1}^{n} \frac{(a_i - e_i)^2}{e_i}$$

where n is the number of cells in the confusion matrix, a_i is the actual value and e_i is the expected value asserted by the null hypothesis. From the value generated by the test, one can deduce the goodness of fit, which describes how well the model fits a set of observations.

We round off our metrics with two statistical measurements of error, Root Mean Squared Error (RMSE), which measures the error rate of a regression model:

$$(\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{P} - A_i)^2})$$

where $\hat{P}$ is the vector of predicted values and A_i is the vector of actual values and Coefficient of determination (R^2) which summarizes the explanatory power of the regression model and is computed from the sums-of-squares terms:

$$\frac{\sum_i (a_i - p_i)^2}{\sum_i (a_i - \bar{a})^2}$$

where a_i is the actual value, p_i is its predicted value and $\bar{a}$ is its mean. R^2 is the proportion of variance of the dependent variable explained by the regression model and if the model is perfect, its value is 1 and if the model has no explanatory power, its value is 0.

Due to the diversity of heuristics used and the inherent bias of all second-order metrics, we rank each model relative to the other models for each metric and then take the average of those scores to generate a rough indicator of model quality.

4.6.2 Feature Selection

Feature selection determines the most relevant features of the forensic artifacts collected that have a significant correlation with abnormal/malicious behavior. Proper feature selection minimizes the cardinality of the set of selected features without sacrificing indicators of abnormal behavior. In doing so, a good feature selection algorithm prevents the over fitting of the model, improving the classification accuracy on future data sets and reducing the performance cost of the model by decreasing the features it must process. Another advantage of feature selection is that by reducing the number of features, the remaining features provide deeper insight into the underlying mechanisms and properties that results in successful classification and can be critical in rule validation. Oðinn relies upon two feature selection algorithms; Stepwise selection on Multivariable Logistic Regression analysis and Kullbeck-Leibler divergence (Information Gain).

Stepwise Selection in Multivariate Logistic Regression

Stepwise selection is often used as a solution to reduce the number of covariables in prediction-based regression models such as binary classification and remains a standard method in virtually all commercial statistical software programs [155]. Stepwise selection works by combining forward-selection (the adding of features) and backward-selection (the reduction of features) based on their statistical significance. Each time a feature is added or pruned, the statistical significance of each covariable is recalculated and thus, through a process of adding and subtracting all combination of features, it can find the minimum number of features that produce the highest significance to the model. Stepwise selection algorithms accuracy improves as the dataset increases and performance increases as the number of features decreases and, as a result, this method has received criticism for its tendency to overfit when the dataset is small and the number of features are large [155].

Multivariate logistic regression measures the effect of a specific covariate in the presence of the other covariates as related to the predicted binary classes. Given a sample training set

with n samples on C classes such that $y = (y_1, ..., y_c)\prime$ with p explanatory variables/features such that $x = (x_1, ..., x_f)\prime$ [156]. From this, we can calculate:

$$Y = XB + E$$

where $Y = (y_1, ..., y_n)\prime$ is an $n \times q$ matrix, $X = (x_1, ..., x_n)\prime$ is an $n \times p$ matrix, B is a $p \times q$ coefficient matrix and $E = (e_1, ..., e_n)\prime$ is the regression noise. Dimension reduction is then used to ensure that the coefficient matrix B consists of explanatory variables that are regressed together rather than individually [156]. The resulting values in the coefficient matrix represent the correlation coefficient of each feature measured against each other and the class. The measurement between the feature and the class is the coefficient estimation measures, which ranges from -1 to 1 and represents the linear relationship between the two, with a 0 being no relationship and 1 being a perfect relationship and a -1 being a perfect inverse relationship. The t-value represents the likelihood that the explanatory feature actual value is not 0. Finally, the p-value is the probability of obtaining a result assuming the null hypothesis is true. For our purposes, we only reject the null hypothesis when $p < 0.01$.

Information Gain

Information Gain is the expectation value of the Kullback-Leibler divergence of a conditional probability distribution [157]. The expected information to classify a given sample is:

$$C(s_1, s_2, ..., s_n) = -\sum_{i=1}^{n} \frac{s_i}{s} log_2(\frac{s_i}{s})$$

where a training set S containing samples s_i of class C where s is the total number of samples in the training set and n is the number of classes. The entropy of a given feature F is:

$$Entropy(F) = \sum_{j=1}^{v} \frac{S_1 j + ... + S_m j}{s} \times I(s_1 j, ..., S_m j)$$

where the feature F with values $\{f_1, f_2, ..., f_v\}$ is divided into a training set with v subsets $\{S_1, S_2, ..., S_v\}$ where S_j is the subset which has the value f_j for feature F. Let S_j contain $s_c j$ samples of class c. Once entropy is known, information gain can be derived with:

$$Gain(F) = C(s_1, ..., s_n) - E(F)$$

Oðinn uses two methods for demonstrating information gain. The first is direct calculation from the formulas above, shown in Table 4.10. The second method the C4.5 decision tree algorithm whose splitting criterion is the attribute with the highest normalized information gain. The C4.5 tree provides the IDS analyst a human readable method to visualize the information gain for rule generation and validation purposes.

4.6.3 Model Evaluation

This section is analysis of three data sets derived from three Einherjar scans analyzed by numerous classifier models using the metrics discussed above. The models are then evaluating using the previously discussed binary classification metrics (see A.3 are constructed along with their Confusion Matrix.

Rankings

In order to evaluate our classifiers against one another, each model is ranked in ascending order for each second-order metric. The model's relative ranking for each metric is then averaged to generate a single metric that ranks each classifier.

Learner	AUC	ACC	F1	MCC	RMSE	R2	X2	AVG
stack	4	1	1	1	6	1	1	2.14
logreg	6	1	1	1	6	1	1	2.43
svm	7	1	1	1	6	1	1	2.57
c45	8	1	4	4	6	1	4	4.0
bayes	1	5	5	5	3	5	5	4.14
neural	2	5	5	5	3	5	5	4.29
forest	2	5	7	7	3	5	7	5.14
knn	5	8	8	8	2	8	8	6.71
rules	9	9	9	9	1	9	9	7.86

Table 4.6: Overall Rankings for Classifiers with Process Scan

The ensemble technique of stacking the various other classifiers is the best performing classifier on process scan with logistic regression and SVM performing almost as well and C4.5 rounding out the top four.

Due to the overwhelming classifying power of modules feature, most classifiers perform optimally and notably, the stacking technique and logistic regression classifiers are in the top position. The SVM classifier fails to generate a proper confusion matrix and is excluded from the module scan.

The association rule classifier and C4.5 algorithm perform in the top 33 percentile in every metric but the F1 score. The stacking algorithm performs roughly average in each

Learner	AUC	ACC	F1	MCC	RMSE	R2	X2	AVG
stack	1	1	1	1	4	1	1	1.43
c45	1	1	1	1	4	1	1	1.43
logreg	1	1	1	1	4	1	1	1.43
rules	1	1	1	1	4	1	1	1.43
forest	1	1	1	1	4	1	1	1.43
bayes	1	6	6	6	1	6	6	4.57
knn	1	6	6	6	1	6	6	4.57
neural	1	6	6	6	1	6	6	4.57

Table 4.7: Overall Rankings for Classifiers with Module Scan

Learner	AUC	ACC	F1	MCC	RMSE	R2	X2	AVG
rules	4	2	1	1	8	2	1	2.71
c45	6	1	3	2	9	1	2	3.43
stack	1	5	2	4	5	5	4	3.71
neural	5	4	7	5	6	4	5	5.14
svm	9	6	5	6	4	6	6	6.0
bayes	7	7	6	7	3	7	7	6.29
forest	2	9	9	9	1	9	9	6.86
logreg	8	8	8	8	2	8	8	7.14

Table 4.8: Overall Rankings for Classifiers with File Scan

metric. Logistic Regression, which had previously performed incredibly well fails thoroughly on the File Scan. However, this is not particularly concerning because, as will be covered later, the File Scan's features simply are not predictive enough to generate a usable model.

Overall, the rankings strongly indicate using the stacking or logistic regression classifier with the C4.5 classifier. However, its possible that our classifier models suffer from overfitting and before we adopt these classifiers wholesale, the features of the data sets should be evaluated.

Feature Selection

In the first two data sets, logistic regression with stepwise selection (Table 4.9) prunes the features to a single feature with extremely high coefficient estimation (0.735 and 0.958 respectively) and highly significant P-value (0 for both). In the third dataset, only when the feature is unknown is there both a significant coefficient and a significant P-value. The

Process Scan	Coeff Est	Std Error	t-value	P-value
pslist=True	0.735	0.097	7.594	0.0001
Module Scan	Coeff Est	Std Error	t-value	P-value
modules=False	0.958	0.023	41.601	0.0001
File Scan	Coeff Est	Std Error	t-value	P-value
dead=Unknown	0.556	0.016	34.195	0.0001
pointers	-0.015	0.002	-6.489	0.0001
access	-0.008	0.003	-3.133	0.002
fname	0.000	0.000	2.694	0.007

Table 4.9: Stepwise Selection for each scan's features

remaining features have zero or negative coefficient estimations and while pointers possesses a highly significant P-value, its coefficient estimation precludes inclusion as a significant feature. The fact that stepwise selection finds in the process and module scan that can explain the entire model with a significant P-value is enough to reject the null hypothesis for those models; however, to be certain, we validate these findings with the information gain of the features.

Process Scan	Info. Gain
pslist	0.62
ppid	0.06
pname	0.03
Module Scan	Info. Gain
modules	0.54
File Scan	Info. Gain
access	0.06
pointers	0.04
fname	0.02

Table 4.10: Information Gain (> 0) for each scan's features

The information gain (Table 4.10) corroborates the stepwise selection algorithm for the first two data sets, showing that our features (pslist and modules) provide enough information gain to reject the null hypothesis. In the Process scan data set, the information gain of pslist is an entire order of magnitude higher than any other feature and in the Module scan data set, the module feature is the only one of relevance. However, information gain does not corroborate the File scan data set because while stepwise selection explores each

features individual outcomes and finds that when the dead feature is Unknown significant, information gain evaluates the feature in its entirety and since dead=Unknown occurs so rarely in the overall data, the information gain of the dead feature is essentially zero. This allows us to reject File scan as a sensor and to avoid using its models as part of our rule generation.

4.6.4 Rule Generation

There are two methods for generating rules from our classifiers. The first is to use the C4.5 algorithm to generate tree-based rules that can be easily validated by a human and the rules generated can be tested after each scan runs with almost no additional overhead. In this scenario, the Balðr component scans Mimisbrunnr and when the tree rule is found, raises an alert. Below is an example of the C4.5 Trees generated by the data sets previously evaluated.

Process Scan Tree

pslist = False:False (34.0)

pslist = True:

— pname > Run.exe:True (10.0)

— pname <= Run.exe:False (4.0)

— — pname >= Cmd.exe: True (3.0)

— — pname < Cmd.exe: False (4.0)

Module Scan Tree

modules = True:True (29.0)

modules = False:False (129.0)

As the values are mapped, Process names (pname) are floating point values stored in the order that processes were discovered by the Einherjar scans. Thus, despite process names (pname) representing a string value (e.g. Run.exe), it can express them as greater than or less than other process names. Because all the non-malicious use-case data set is loaded into

the database prior to running the malware samples, the decision tree found that even when a process is in the doubly-linked list, there is a high probability it is still malicious if the process occurs after a hidden Run.exe or Cmd.exe process. This discovery was something we did not anticipate though such a discovery was easily validated, an aforementioned advantage of the tree model. In this case, we know via the forensic and malware analysis communities that hidden uses of Run.exe have virtually no legitimate purposes and Cmd.exe, while occasionally hidden for legitimate purposes is still a hallmark of many malware infection and persistence routines. However, if the C4.5 algorithm is significantly outperformed, Muninn uses an alternative method.

In these cases, the highest ranking classifier model is stored as a file using object serialized, the Python pickle module. When the protect mode is active, the chosen serialized model is loaded from the file and waits for the Einherjar scans to start storing data into the database, Mimisbrunnr. Data from Mimisbrunnr is mapped into values the classifier can interpret and if a True Positive is found, an alert is raised. The stacking classifier is the most accurate model by the average rankings across all the various scans, beyond the three evaluated in this section though logistic regression is often competitive. When logistic regression is superior, it is chosen do to its performance advantage over the stacking model. This modular approach to rule generation allows for the IDS owner to make appropriate trade-offs between accuracy and speed based upon their individual needs.

4.7 Balðr - Protection and Mitigation

```
In wondrous beauty once again
Shall the golden tables stand mid the grass,
Which the gods had owned in the days of old,
Then fields unsowed bear ripened fruit,
All ills grow better, Baldr comes back [140].
```

Balðr is perhaps the most important component of Oðinn. As previously noted by the Muninn component, a rule generated by our heuristics is validated. Once the rule is accepted, it is stored into a dictionary that exists within Balðr. Balðr only runs during production mode and continuously polls the tables in Mimisbrunnr, looking for matches to its ruleset. If one is found, the guest is considered infected. Traditionally once malware has infected a machine, the machine is rendered unusable until a technician can remove the malware. Often, the infection is so severe or difficult to remove that the machine's OS must be completely reinstalled with significant loss of data and/or time. Furthermore, if forensics and/or malware analysis is to be carried out, a bit-by-bit copy must be made before the infection is removed, increasing the productivity loss to the user and organization. Baldr virtually eliminates this productivity downtime, providing almost immediate use of an infection free machine to the user with only minor data loss and providing copies for concurrent forensic investigation and malware analysis. It achieves this through the use of a virtualization technique known as live snapshots.

4.7.1 Live Snapshots

In order for Oðinn to support live snapshots, we must limit our cloud integration to KVM [158]. In our previous work [103], we used Xen with images residing on an Logical Volume Manager (LVM) drive to achieve low-downtime snapshots. This methodology minimized downtime, but users were impacted briefly during every snapshot and if the snapshots occurred frequently (necessary to minimize data loss during rollback), it was disorienting to users. Live snapshots require the following actions:

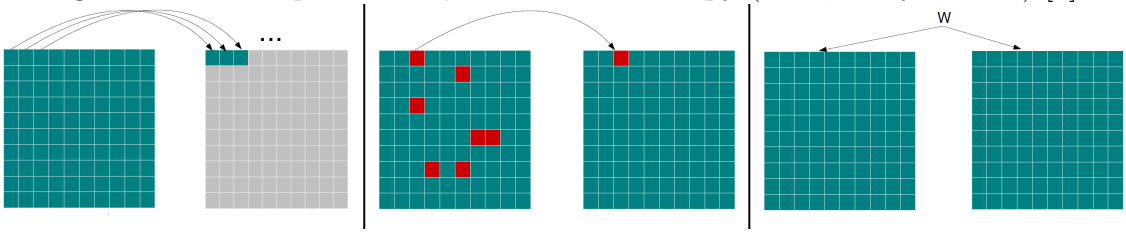
1. Guest Machine is placed in a transient state.
2. A new image is created and added to the snapshot stack
3. All pending I/O from the guest is immediately flushed.
4. The new image file is opened.

5. The current image file is set as the backing file

6. Live block copy is initiated.

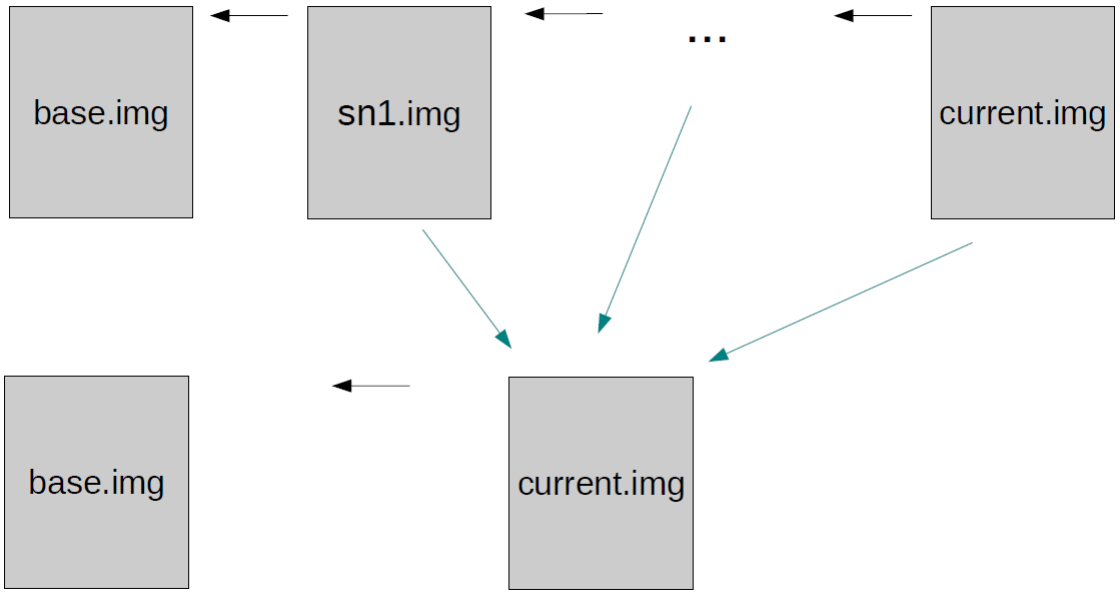
Before the live block copy can occur, a limitation of libvirt first requires the guest to be placed in a transient state. This is because libvirt assumes that all block actions are modifications directly to the image and thus, should a running guest crash while a block job is active, it has a high probability of corrupting the image. However, QEMU's block copy performs mirroring and does not modify the active image. Live block copy consists of three stages: bulk, dirty and mirrored writes (see Figure 4.8). During the bulk phase, all new writes to the guest machine undergoing snapshot are flagged as dirty blocks. Concurrently, all sectors from the source block device are copied to the destination block device. Next, all dirty blocks (blocks that were written to during the bulk phase) are copied. Finally, mirrored writes occur to both source and destination simultaneously until the QEMU finalizes the snapshot. Then the guest is restored to a persistent state or destroyed, pending whether it was a standard snapshot or malware infected snapshot.

Figure 4.8: Three phases of QEMU live block copy (Bulk, Dirty, Mirror) [7].



It is important to note that the snapshot only consists of changes that have occurred to the base image and is not a complete copy of the base image. As this technique is usually an imperceptible performance impact to the user, Balður aggressively performs live snapshots (default is every five minutes), and while this has minimal storage impact, it becomes necessary at the end of the session to merge the snapshots into the original image to consolidate administrative overhead. QEMU initiates live merging as followed (see Figure 4.9):

Figure 4.9: Merging Snapshots. [7].



1. The snapshot stack is consolidated into the topmost (newest) snapshot.

2. A copy-on-read is performed from the beginning to the end of the file.

3. Unallocated areas in the backing file are ignored.

4. One or more images can be pruned from the stack.

Balðr only consolidates at each hour and then does a final consolidation of each hourly snapshot when the user's session has ended. This ensures that, in the worst case scenario, a rule is violated during the hourly snapshot merger, the user only loses an hour of productivity. When malware is detected, the guest machine is paused to prevent any post-exploitation activities by the intrusion that would damage assets. During this pause, the user is informed that an intrusion was found and that the guest machine will be rolled back to the most recent safe snapshot. Concurrent to this messaging, the guest machine undergoes a complete snapshot that is then migrated to a new host that has no network access or external disk access for the forensic and/or malware analyst to explore with Huginn. Once this snapshot is complete, the guest machine is reverted and the guest machine resumes operations. If an intrusion is detected a second time during the same session, the guest machine is disabled until an analyst can manually verify the machine is safe for operation.

4.8 Huginn - Malware Analysis Suite

Huginn is a malware analysis suite that utilizes virtualization properties to provide forensic investigators with a number of tools that do not exist elsewhere for cloud forensics. First, Huginn provides automated verification of DLL and drivers, providing forensic analysts just those DLL and drivers that have been created or altered by the malware. Second, Huginn allows a forensic analyst to automatically reconstruct a stripped Microsoft Portable Executable (PE) header and integrate the reconstructed header into IDA Pro [159] for reverse engineering. Finally, Huginn provides a way to manually augment Muninn with new rules on-the-fly to protect against future intrusions. Additionally, Huginn has access to all the Volatility monitors used by Geri and Freki as well as a number of unique Volatility plugins aimed towards manual forensic analysis.

4.8.1 Verification of DLL and Drivers

Figure 4.10: Hashes from moddump

```
05bfa399f156719d62d949ba1543c485  /home/cbharri/Dissertation/Research/module.1040.661e6b8.1000000.dll
8ed54ff7fe4398e62315d96966160980  /home/cbharri/Dissertation/Research/module.1040.661e6b8.20000000.dll
526a9216e4705af4d5e1588a38d69d66  /home/cbharri/Dissertation/Research/module.1040.661e6b8.5ad70000.dll
26b31e469c7cf720c83d50b956ae0cd5  /home/cbharri/Dissertation/Research/module.1040.661e6b8.5b860000.dll
d25e1c323198b7207527e8c7204a5568  /home/cbharri/Dissertation/Research/module.1040.661e6b8.5cb70000.dll
44848e42a7458375b5c4d90f575ab328  /home/cbharri/Dissertation/Research/module.1040.661e6b8.5d090000.dll
676d4be99ffe7df6018e00a7e0a8cead  /home/cbharri/Dissertation/Research/module.1040.661e6b8.6f880000.dll
e3a75cc4bb7f38e4b2c94d75a89af3b9  /home/cbharri/Dissertation/Research/module.1040.661e6b8.71a50000.dll
29c58b51cdcbfeb48b145b8f03bf4ed3  /home/cbharri/Dissertation/Research/module.1040.661e6b8.71aa0000.dll
6e0a74669e4d4557510bb53dafcbfc96  /home/cbharri/Dissertation/Research/module.1040.661e6b8.71ab0000.dll
d7f109481e9ae2f3cc854cc30321eec6  /home/cbharri/Dissertation/Research/module.1040.661e6b8.71c80000.dll
acb8034a1cf6fe2bde3c5aef85826e1d  /home/cbharri/Dissertation/Research/module.1040.661e6b8.723f0000.dll
448eb20ceb8eadf31c034e02f7de499c  /home/cbharri/Dissertation/Research/module.1040.661e6b8.72400000.dll
2656637bbfb22ffa4b85f5e7b1524db1  /home/cbharri/Dissertation/Research/module.1040.661e6b8.73000000.dll
3d03fc714463696dfde20e3548b2ab7f  /home/cbharri/Dissertation/Research/module.1040.661e6b8.74280000.dll
11bf089ebf320e7db9cc1219a989b20b  /home/cbharri/Dissertation/Research/module.1040.661e6b8.742a0000.dll
43f58b1d0bce25e6292464ce8627ef2e  /home/cbharri/Dissertation/Research/module.1040.661e6b8.742e0000.dll
4f88660aa95fcb3b060c18e167e5176c  /home/cbharri/Dissertation/Research/module.1040.661e6b8.74300000.dll
8f170c4ffbdfc89f9222267a22f8b8cb  /home/cbharri/Dissertation/Research/module.1040.661e6b8.75bb0000.dll
ec2593b3626de4fac4d7f1d6b17884a2  /home/cbharri/Dissertation/Research/module.1040.661e6b8.75c10000.dll
2699661ece8a42edf8684703769875b4  /home/cbharri/Dissertation/Research/module.1040.661e6b8.767a0000.dll
7505c5c223e693f06ddd8a1d9c289856  /home/cbharri/Dissertation/Research/module.1040.661e6b8.769c0000.dll
e012dfcd035e701b578f5192f5f15160  /home/cbharri/Dissertation/Research/module.1040.661e6b8.76b40000.dll
02102453f894469edd323e3c3c2a9b96  /home/cbharri/Dissertation/Research/module.1040.661e6b8.76c30000.dll
078e952154f41c049edbea6759c303a6  /home/cbharri/Dissertation/Research/module.1040.661e6b8.76c60000.dll
5b705bb6c2165047bbef0d4b01cd098c  /home/cbharri/Dissertation/Research/module.1040.661e6b8.76c90000.dll
ea74f364175a7af14483c3f2650dae36  /home/cbharri/Dissertation/Research/module.1040.661e6b8.76d60000.dll
742e9324fa07e5ba8f67679f68f63944  /home/cbharri/Dissertation/Research/module.1040.661e6b8.76f20000.dll
33b9298b06107954b6e866448cfb1959  /home/cbharri/Dissertation/Research/module.1040.661e6b8.76f60000.dll
```

Huginn provides a major advantage over contemporary forensic methodologies by providing the analyst with a fully working copy of the guest machine in a pre-infected state. Building on this intrinsic benefit, Huginn can automatically detect modified DLL and Kernel drivers. To do this, it executes two Volatility monitors - dlldump and moddump (see Table 4.12) - on both images, performs an MD5 hash (see Figure 4.10) on each driver or DLL and then juxtaposes the resulting hashes, outputting only new or modified DLL/drivers. In this manner, intrusions reliant on DLL/driver hooks can be identified almost immediately for further analysis as the total number of modified files is a fraction of the original (see Table 4.11). The analyst can use snapshots dating all the way back to the creation of the image should they suspect that the infection dates to before the most recently thought clean snapshot was taken.

DLL	Files	Reduction
Total	1305	–
Modified	60	95%
Modules	Files	Reduction
Total	172	–
Modified	3	98%

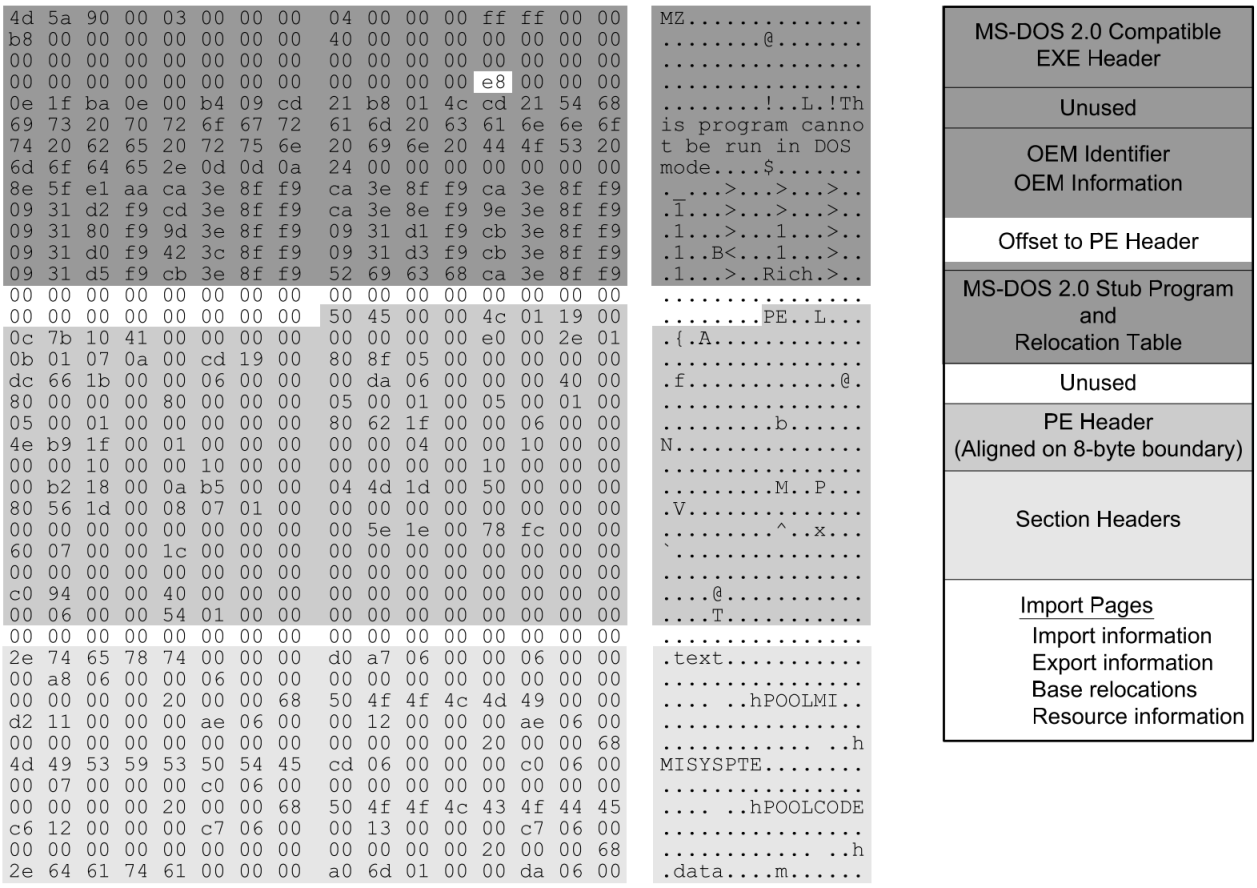
Table 4.11: Number of total DLL files and module files compared to the number modified

4.8.2 Stripped PE Header reconstruction

The PE is used by executable files, DLL, and device drivers for Windows operating systems. Files that follow this specification have a defined structure that can be used to interpret them both on disk and in memory. Figure 4.11 illustrates the structure of a PE file. The base of the image header begins with the hex bytes 0x4d5a, or "MZ" in ascii signifying the PE file's MS-DOS Compatibility. Then an offset, always located at 0x3c points to the actual PE header; which, when valid, always starts with a 4-Byte signature "PE". The PE Header is of contains useful information such as the section information, symbol information, and machine type information of the executable file. The optional header, indicated by 0x10b in 32-bit PE files, is the most important header for malware analysis. This header contains within it the export table which contains all the names and pointers to all Windows API functions exported by the image. Because of this, malware often completely erases the PE header.

In Figure 4.12, the lines of '....' indicate the PE file has been stripped of its headers. At the very least, as in the canonical example previously discussed, this PE header should possess an 'MZ' flag. The malware author's goal is in stripping a PE binary's header is to hamper reverse engineering of the malware code by denying access to which Windows API functions are imported that would otherwise reside in the optional header. Further complicated matters is that some binaries that are dumped may lack the Import Address Table (IAT) due to one or more pages of the PE Header not existing in resident (paged) memory,

Figure 4.11: PE Structure [8].



so sometimes inadvertent PE header stripping occurs without malicious intervention. Regardless of how, Huginn can automatically reconstruct stripped PE headers and additionally, create an IDC file for IDA Pro use. This process takes a couple seconds before outputting an idc file as shown in Figure 4.13 below.

Figure 4.12: Example of stripped PE Header

```
Process: IEXPLORE.EXE Pid: 2044 Address: 0x7ff80000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 45, PrivateMemory: 1, Protection: 6

0x7ff80000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x7ff80010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x7ff80020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x7ff80030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

Figure 4.13: Reconstructed IDC file

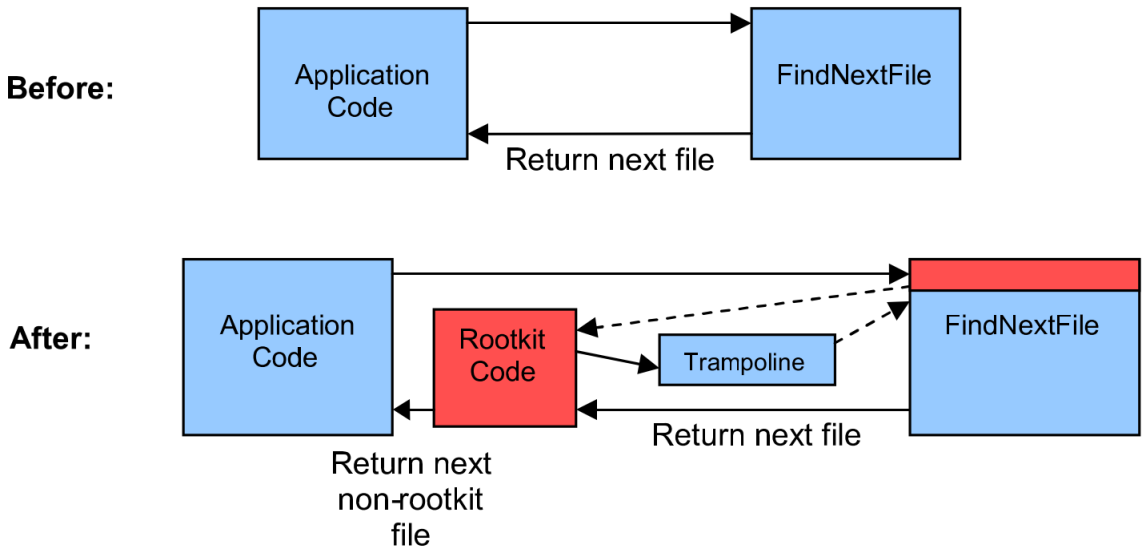
```
#include <idc.idc>
static main(void) {
   MakeDword(0x7FF9E000);
   MakeName(0x7FF9E000, "SetSecurityDescriptorDacl");
   MakeDword(0x7FF9E004);
   MakeName(0x7FF9E004, "GetUserNameA");
   MakeDword(0x7FF9E008);
   MakeName(0x7FF9E008, "RegCloseKey");
   MakeDword(0x7FF9E00C);
   MakeName(0x7FF9E00C, "RegCreateKeyExA");
   MakeDword(0x7FF9E010);
   MakeName(0x7FF9E010, "RegDeleteKeyA");
   MakeDword(0x7FF9E014);
   MakeName(0x7FF9E014, "RegDeleteValueA");
   MakeDword(0x7FF9E018);
   MakeName(0x7FF9E018, "RegNotifyChangeKeyValue");
   MakeDword(0x7FF9E01C);
   MakeName(0x7FF9E01C, "RegOpenKeyExA");
   MakeDword(0x7FF9E020);
   MakeName(0x7FF9E020, "RegQueryValueExA");
   MakeDword(0x7FF9E024);
   MakeName(0x7FF9E024, "RegSetValueExA");
   MakeDword(0x7FF9E028);
   MakeName(0x7FF9E028, "AdjustTokenPrivileges");
   MakeDword(0x7FF9E02C);
   MakeName(0x7FF9E02C, "InitiateSystemShutdownA");
   MakeDword(0x7FF9E030);
   MakeName(0x7FF9E030, "LookupPrivilegeValueA");
   MakeDword(0x7FF9E034);
   MakeName(0x7FF9E034, "OpenProcessToken");
   MakeDword(0x7FF9E038);
   MakeName(0x7FF9E038, "RegEnumKeyExA");
   MakeDword(0x7FF9E03C);
   MakeName(0x7FF9E03C, "InitializeSecurityDescriptor");
   MakeDword(0x7FF9E044);
```

4.8.3 Rule Augmentation

The final novel benefit provided for cloud forensics is the ability to insert new rules on-the-fly found during analysis. Consider IAT-based rootkit shown below (Figure 4.14). In Windows functions after XP SP2, all functions have a five-byte preamble that upon the function accomplishing its task, returns execution to the calling code. This exploit technique works by saving, then overwriting the five-byte preamble of an otherwise innocently called function with a jump whose address points to the malicious code. Upon completion,

the malicious retrieves the proper five-byte preamble, loads it, and returns functionality to the calling application [8]. In this manner, the rootkit can reinsert control whenever this Windows function is called.

Figure 4.14: Hooking the Import Address Table [6].



Suppose an attacker using this technique in a way Muninn's current ruleset doesn't account for is discovered by an analyst using Huginn (refer to apihooks or callbacks in Table 4.12). If the overwritten version of this five-byte preamble were "8B 10 61 A8 54", the analyst can set in a Mimisbrunnr table reserved for this function those bytes, and along with Muninn's normal polling, a subpoller utilizing yarascan will execute, looking for that five-byte YARA signature in the opcodes. If found, normal detection and mitigation protocol outlined in Balðr will occur.

Huginn	Description
dlldump	Extracts all DLL from all processes.
vaddump	Extracts pages belonging to each VAD node are placed in separate files.
moddump	Extract kernel driver to file.
dumpfiles	Extracts all files that are mapped as DataSectionObject, ImageSectionObject or SharedCacheMap.
procmemdump	Extracts process's executable (with slack space).
procexedump	Extracts process's executable (without slack space).
hivedump	Recursively list all subkeys in a hive.
apihooks	Finds IAT, EAT, Inline style hooks, and several special types of hooks.
callbacks	Lits important notification routines and kernel callbacks.
vadinfo	Displays extended information about a process's VAD nodes.
ldrmodules	Cross-references DLLs with the 3 PEB lists looking for inconsistencies.
malfind	Finds hidden or injected code/DLLs based on characteristics such as VAD tag and page permissions.
yarascan	Locate any sequence of bytes, regular expressions, ANSI strings, or Unicode strings in memory.
timeliner	Arranges memory artifacts in order of occurrence to provide a timeline of events.

Table 4.12: List of volatility plugins utilized by Huginn [10].

4.9 Urdarbrunnr - Oðinn's Benchmarks

It is important to demonstrate the scaling of Oðinn as the number of virtual machines increases. As the first three tables below demonstrate, memory usage remains low as it scales. When compared to a single guest machine which uses 50% memory overhead, the memory overhead for 10 guests Oðinn is 33%. This is because only one component, Einherjar, significantly affects memory scaling and a significant portion of its overhead for one guest becomes shared memory when scaled. Guest Machines are configured with a single CPU and 512 MB of RAM. They run the 32-bit Windows 7 Professional with Service Pack 1. Memory usage is measured every second using the ps command with the v flag (memory centric-view).

Figure 4.15: Memory Usage for 1 Guest Machine

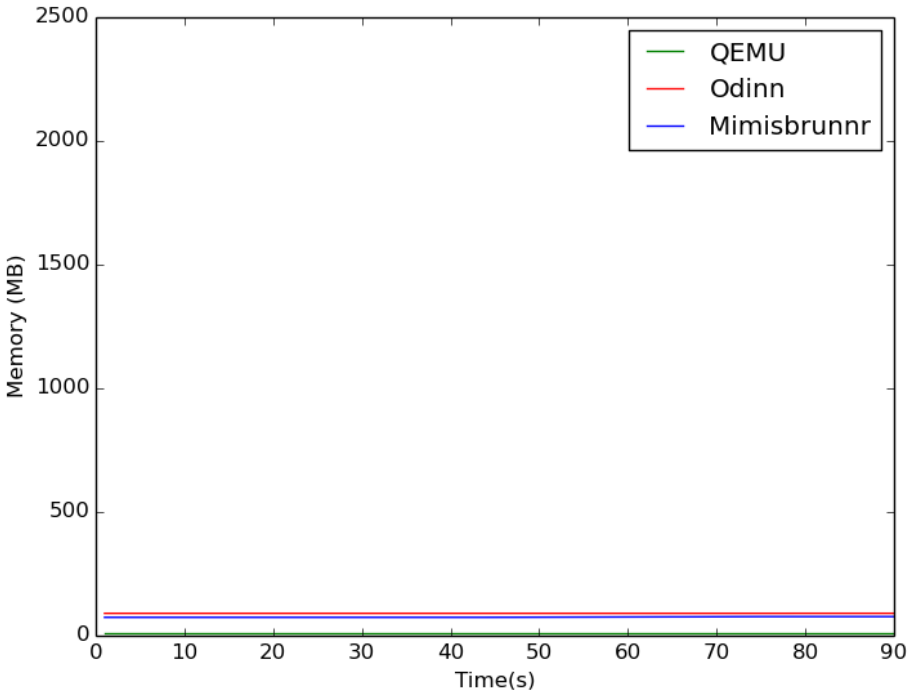


Figure 4.16: Memory Usage for 1 Guest Machine with Oðinn

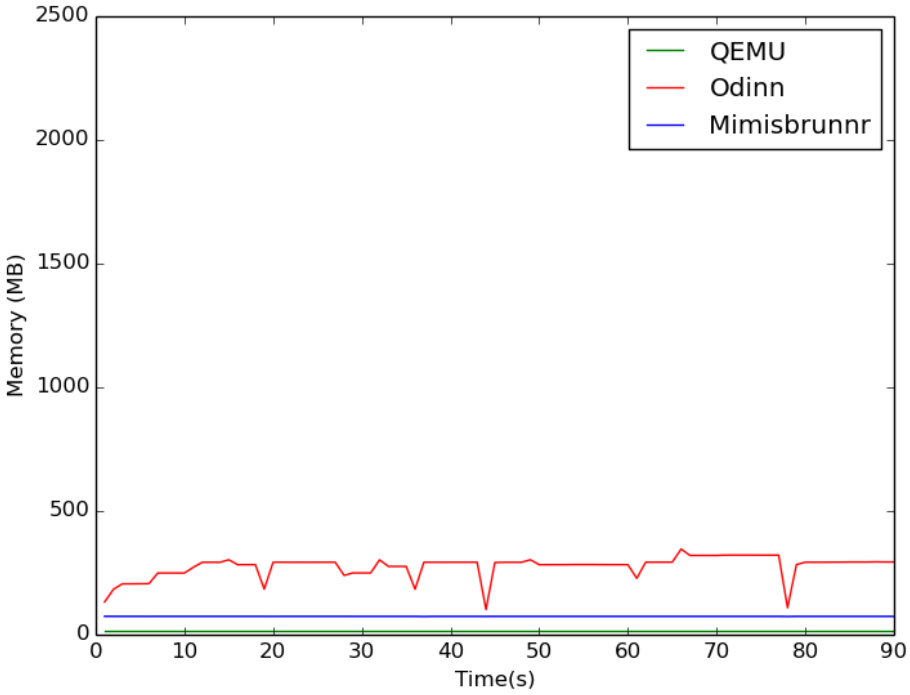
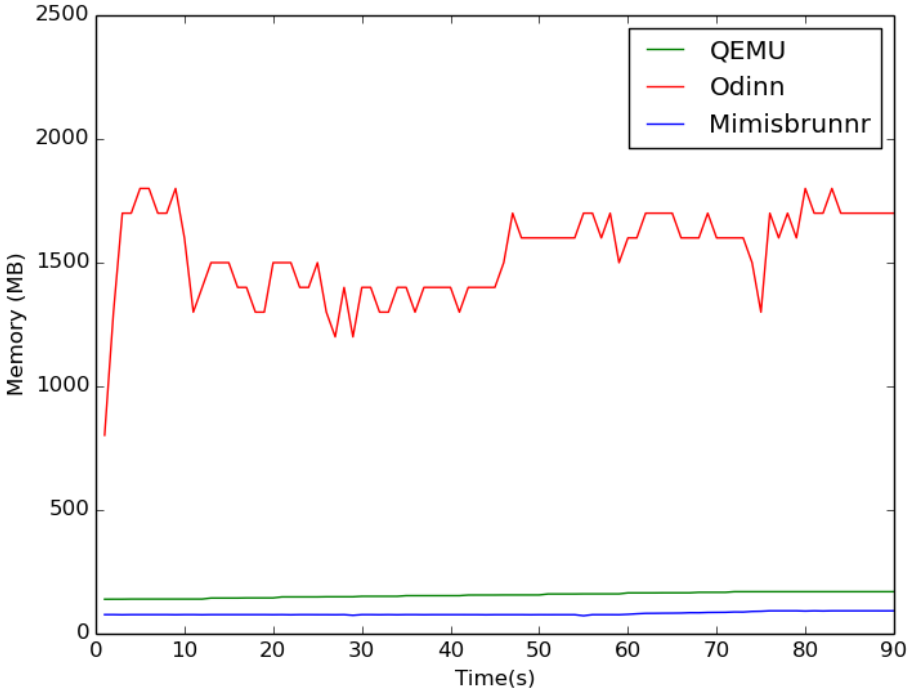


Figure 4.17: Memory Usage for 10 Guest Machines with Oðinn



The CPU scaling (Figure 4.20); however, does improve as scaling increases but not significantly as is the case with memory, going from an 18% average CPU utilization for one guest machine, to nearly 80% average CPU utilization with 10 CPUs. CPU usage is measured every second with the ps command. This overhead is the result of aggressive scanning (scans are restarted as soon as they end) by the Einherjar component and a reduction in the number of modules used or the reduction in scanning frequency will allow for CPU usage curve that is acceptable for the cloud.

Figure 4.18: CPU Usage for 1 Guest Machine

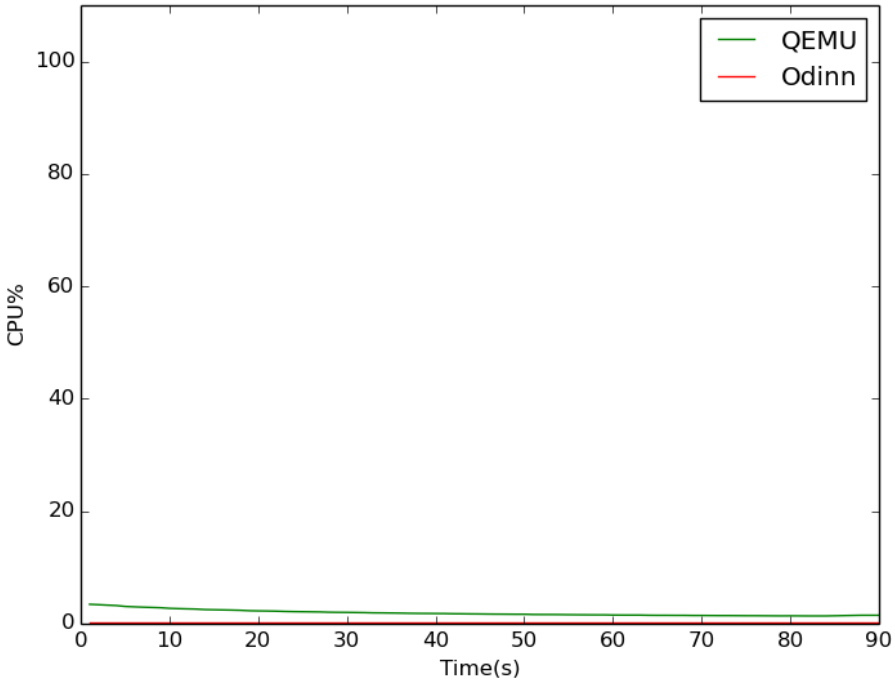


Figure 4.19: CPU Usage for 1 Guest Machine with Oðinn

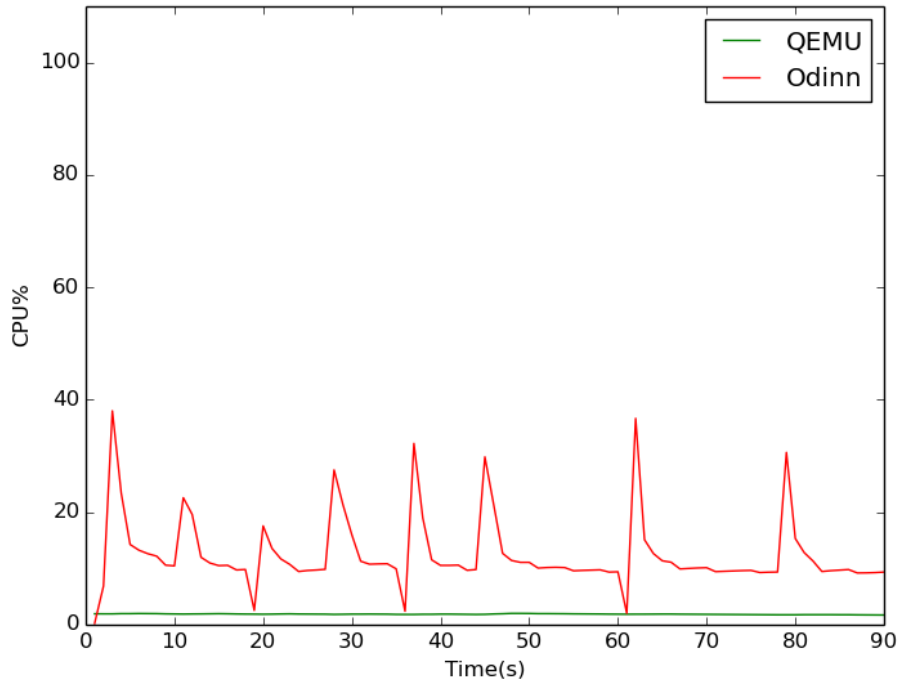
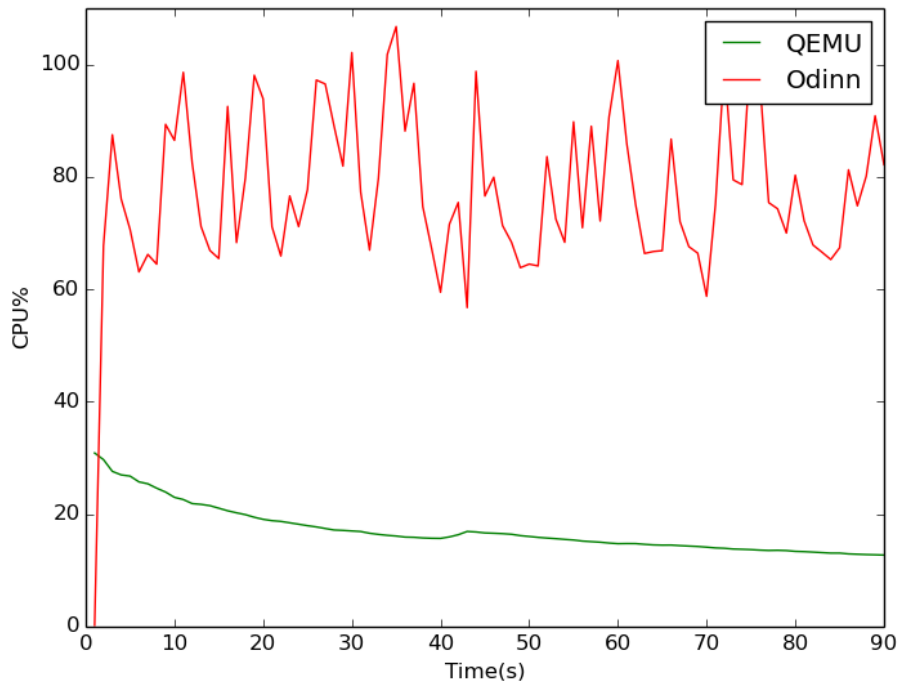


Figure 4.20: CPU Usage for 10 Guest Machines with Oðinn



Chapter 5

Future Work

5.1 Yggdrasil

Yggdrasil's dependency on libvirt, while an asset in many ways, has some limitations that will need to be addressed by directly modifying libvirt or replacing it entirely. The first major concern is the introduction of Non-Uniform Memory Access (NUMA) memory architecture implemented by AMD (HyperTransport) and Intel (QuickPath Interconnect) in multi-processor and multi-core systems. Unlike Symmetric Processing Servers, the NUMA architecture has its memory decentralized and shared by the pool of processors and/or cores. This is highly advantageous in reducing the bottlenecks (saturation) between the processor and its memory but exponentially increases the complexity of efficiently managing optimized virtual machines. This plays out in virtualization in two separate areas. The first is optimizing the server itself to efficiently allocate resources, and in this area, libvirt was designed correctly and memory access can be interleaved. This allocates memory to each guest machine in chunks in a round-robin like manner and can even allow a one-to-one mapping of virtual CPUs to physical CPUs where dedicated resource usage is needed. However, the second area libvirt lacks maturity in is exposing the guest machine to the NUMA architecture at its disposal. The problem comes from its service numad, which automatically optimizes the placement of containers (cgroups) to separate the context of applications on a single guest machine. Here, manual placement is still far superior to the automated version yet the time required to manually configure every guest is too burdensome for use in a typical cloud environment. Additionally, the API has a number of features previously discussed that are deleterious to fully automating the management of Odinn. Sandia has an in-house alternative to libvirt that may be utilized (and has successfully launched over one million

guest machines at once [160] or the author may attempt to contribute directly to libvirt to improve these areas.

5.2 Hliðskjalf and Einherjar

Einherjar will continue to exist as a development tool for Oðinn; however, Volatility's performance limits its utility at the upper threshold of scalability due to its reliance upon Python's Global Intepreter Lock and its design goals (as a memory analysis tool instead of live analysis tool; ease of development rather than optimal performance). Future work will use Einherjar to prototype scans and then, if our model validation finds them useful to malware classification, implement them directly into the Hliðskjalf's API directly in C or ideally, we will create an API wrapper in Go and utilize its highly desirable design focus on concurrency so that thousands of scans can be run simultaneously with low overhead.

5.3 Mimisbrunnr

With the recent development of single modules of RAM with 128GB of memory, individual cloud nodes will be capable of exceeding 1 terabyte of RAM. When first envisioned three years ago, the memory availability of servers running large numbers of virtual machines was primarily blocked by the memory availability rather than the number of CPU cores/threads. As this limitation changes, the database should be modified so that it fully operates in memory rather than on disk for optimal performance and this may necessitate a switch from postgres to NoSQL, requiring a rewrite of its underlying API or sufficient performance may be achieved simply by transitioning Mimisbrunnr directly to memory with no other modifications.

5.4 Muninn

As previously mentioned, Sandia's Forensic Analysis Repository for Malware (FARM) is being utilized to train Oðinn. Future work will incorporate parts of Oðinn directly into FARM and the resulting analysis of its millions of malware can be leveraged alongside a large non-malicious dataset garnered from Sandia's internal cloud [161] to vastly improve the training and validations sets available to Muninn. Additional research also needs to be performed to identify all the legitimate instances when a windows artifact (process, thread, file handle) is hidden and eliminate those from becoming false positives.

5.5 Balðr

The highest priority is to improve the robustness of our QEMU low level block actions as well as adopt to the block job architecture (asynchronous block actions) instead of our synchronous solution. Ideally, the live snapshot command in libvirt will actually work with QEMU properly in an upcoming version allowing us to forego this necessity as our solution exhibits a modest amount of overhead while it monitors the block-copy action until it has finished instead of being notified directly by libvirt or the virtualization tool itself. Additionally, there is value in weighting the various rules such that a single rule is not sufficient in isolation to trigger a malware detection unless that particular rule never exhibits a false positive. The current implementation is somewhat brutish and thus too prone to false positives (though due to the relative low cost to the user, this is not entirely unwise since the tradeoff is less false negatives). Finally, it would be advantageous to do two block jobs simultaneously to improve the speed at which a copy can be made for Huginn before reverting the user to the most recent clean snapshot.

5.6 Huginn

The previous referenced possibility of a massive non-malicious dataset, facilitates the adoption of context piecewise triggered hashes instead of the standard hashing method currently used. This hashing generates a percentage metric that indicates the shared content that two input files share in common. Using this score, we can ascertain with a high degree of confidence the amount of modification a memory artifact has undergone. Currently, this type of fuzzy hashing has been used for malware triage to identify malware variants given a known signature; however, in the context of Huginn, we can do something unique. By generating a collection of hashes for each module and driver found in the non-malicious dataset, we can prune many of the false positives the analyst must review to determine which modifications were actually malicious and which naturally occurred. While the number of modules is already reduced by 95%, reducing this to 99.9% or higher would be quite valuable. Additionally, providing stronger integration into Ida Pro or alternative debuggers (e.g. Immunity Debugger) will facilitate quicker development of temporary yara rules and assess the post-exploitation/data exfiltration goals of the malware.

Chapter 6

Conclusion

In this dissertation, several novel contributions were presented to leverage properties of virtualization. First, Oðinn implements a fully integrated explicit semantic bridge via the Einherjar componenet. This component, implemented as a fork of Volatility, provides out-of-guest sensors pre-validated by the forensic memory analysis community in a performance conscious manner. This is accomplished by caching important values, creating a single scan for all memory structures, and by converting Volatility into a library. By intergrating an existing forensic memory analysis suite, we demonstrate the ability to perform introspection without explicit knowledge of a particular Operating System's internal structures.

Second, Muninn provides exceptional accuracy via ensembles of multiple binary classifiers to efficiently detect malware and mitigate infections before they can engage or complete in post-exploitation activities. Muninn's classifiers undergo stringent model validation to prevent overfitting and maximize accuracy. Muninn was validated against a myriad of highly sophisticated malware via the Loki component and the models generated were selected for their overall quality via numerous binary classification metrics.

Third, a mitigation capability is demonstrated via the Balðr component, a snapshot managing agent with a live snapshot mechanism. This mitigation component rapidly restores a user's guest machine to its pre-infected state, minimizing productivity/data loss, once an infection is detected.

Fourth, a malware analysis engine is demonstrated, Huginn, which takes advantage of a virtualization property, access to the pre-infected state, to greatly the reduce the number of DLLs/drivers/modules that need to be examined so that rapid determination of malware

functionality can be exposed. In many cases, this occurs regardless if it was initially a packed/encrypted PE file.

Furthermore, this solution is portable, dependent only of KVM, operating on both 32-bit and 64-bit architectures and supports the three most common OS-families (Windows, Macintosh, and Linux). Oðinn has numerous benefits compared to its brethren cloud-IDSs. The end result is an IDS that is prepared to meet the challenges imposed by the adoption of the cloud, and the exponential growth/sophistication of malware.

Bibliography

[1] "av-test." [Online]. Available: http://www.av-test.org/typo3temp/avtestreports/malware-last-10-years_en.png

[2] H. Lipson, "Tracking and tracing cyber-attacks: Technical challenges and global policy issues," Carnegie Mellon Software Engineering Institute, Pittsburgh, Tech. Rep., 2002. [Online]. Available: http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA408853

[3] Intel, "Intel 64 and IA-32 Architectures Software Developers Manual," Intel, Tech. Rep. September, 2013.

[4] "FUSE: Filesystem in Userspace." [Online]. Available: http://fuse.sourceforge.net/

[5] B. Payne, "Simplifying virtual machine introspection using LibVMI." *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, Jan. 2012. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1084804512001944http://prod.sandia.gov/techlib/access-control.cgi/2012/127818.pdf

[6] C. Ries, "Inside windows rootkits," *VigilantMinds Inc*, 2006. [Online]. Available: http://thehackademy.net/madchat/vxdevl/library/InsideWindowsRootkits.pdf

[7] F. Bellard, "QEMU, a Fast and Portable Dynamic Translator." *USENIX Annual Technical Conference, FREENIX ...*, pp. 41–46, 2005. [Online]. Available: http://static.usenix.org/events/usenix05/tech/freenix/full_papers/bellard/bellard_html/

[8] B. Pagel, "Automated virtual machine introspection for host-based intrusion detection," Ph.D. dissertation, Air Force Institute of Technology, 2009. [Online]. Available: http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA499499

[9] M. Pearce, S. Zeadally, and R. Hunt, "Virtualization: Issues, security threats, and solutions," *ACM Computing Surveys (CSUR)*, vol. 45, no. 2, p. 17, 2013.

[10] "Volatility Wiki." [Online]. Available: https://code.google.com/p/volatility/wiki/CommandReference23

[11] A. Saberi and Z. Lin, "H YBRID-BRIDGE : Efficiently Bridging the Semantic Gap in Virtual Machine Introspection via Decoupled Execution and Training Memoization," *Proceedings Network and Distributed Systems Security Symposium (NDSS14)*, 2014.

[12] Y.-S. Wu, P.-K. Sun, C.-C. Huang, S.-J. Lu, S.-F. Lai, and Y.-Y. Chen, "EagleEye: Towards mandatory security monitoring in virtualized datacenter environment," *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 1–12, Jun. 2013. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6575300

[13] M. Crawford and G. Peterson, "Insider Threat Detection using Virtual Machine Introspection," *System Sciences (HICSS), 2013 46th ...*, 2013. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6480061

[14] Y. Fu and Z. Lin, "EXTERIOR: using a dual-VM based external shell for guest-OS introspection, configuration, and recovery," *Proceedings of the 9th ACM SIGPLAN/SIGOPS ...*, pp. 97–109, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?id=2451534

[15] T. Lengyel and J. Neumann, "Virtual machine introspection in a hybrid honeypot architecture," *Proceedings of the 5th ...*, p. 5, 2012. [Online]. Available: https://www.usenix.org/system/files/conference/cset12/cset12-final14.pdf

[16] K. Vieira and A. Schulter, "Intrusion detection for grid and cloud computing," *It ...*, no. August, 2010. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5232794

[17] B. Dolan-Gavitt and T. Leek, "Virtuoso: Narrowing the semantic gap in virtual machine introspection," *Security and Privacy (...*, pp. 297–312, 2011. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5958036

[18] A. Roberts, R. Mcclatchey, S. Liaquat, N. Edwards, M. Wray, A. Roberts, R. Mcclatchey, N. Edwards, and M. Wray, "Introducing Pathogen : A Real-Time Virtual Machine Introspection Framework Abstract : POSTER : Introducing Pathogen : A Real-Time Virtual Machine Introspection Framework," *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013.

[19] H. Jin, G. Xiang, D. Zou, S. Wu, F. Zhao, M. Li, and W. Zheng, "A VMM-based intrusion prevention system in cloud computing environment," *The Journal of Supercomputing*, vol. 66, no. 3, pp. 1133–1151, 2013.

[20] F. Azmandian, M. Moffie, M. Alshawabkeh, J. Dy, J. Aslam, and D. Kaeli, "Virtual machine monitor-based lightweight intrusion detection," *ACM SIGOPS Operating Systems Review*, vol. 45, no. 2, pp. 38–53, 2011.

[21] S. Roschke, F. Cheng, and C. Meinel, "An advanced ids management architecture," *Journal of Information Assurance and ...*, vol. 5, pp. 246–255, 2010. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.182.6223&rep=rep1&type=pdf

[22] T. Sproull and J. Lockwood, "Distributed instrusion prevention in active and extensible networks," *Active Networks*, 2007. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-540-71500-9_4

[23] M.-L. Shyu and V. Sainani, "A multiagent-based intrusion detection system with the support of multi-class supervised classification," in *Data Mining and Multi-agent Integration*. Springer, 2009, pp. 127–142.

[24] Z. Li, Y. Gao, and Y. Chen, "HiFIND: A high-speed flow-level intrusion detection approach with DoS resiliency," *Computer Networks*, 2010. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2009.10.016http://www.sciencedirect.com/science/article/pii/S1389128609003375

[25] S. Khanum, M. Usman, and A. Alwabel, "Mobile Agent Based Hierarchical Intrusion Detection System in Wireless Sensor Networks," *International Journal of Computer ...*, vol. 9, no. 1, pp. 101–108, 2012. [Online]. Available: http://ijcsi.org/papers/IJCSI-9-1-3-101-108.pdf

[26] A. Herrero and E. Corchado, "Hybrid multi agent-neural network intrusion detection with mobile visualization," *Innovations in Hybrid ...*, pp. 320–328, 2007. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-540-74972-1_42

[27] M.-Y. Su, G.-J. Yu, and C.-Y. Lin, "A real-time network intrusion detection system for large-scale attacks based on an incremental mining approach," *Computers & security*, vol. 28, no. 5, pp. 301–309, 2009.

[28] A. V. Dastjerdi, K. A. Bakar, and S. G. H. Tabatabaei, "Distributed Intrusion Detection in Clouds Using Mobile Agents," *2009 Third International Conference on Advanced Engineering Computing and Applications in Sciences*, pp. 175–180, Oct. 2009. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5359505

[29] A. Byrski and M. Carvalho, "Agent-based immunological intrusion detection system for mobile ad-hoc networks," in *Computational Science–ICCS 2008*. Springer, 2008, pp. 584–593.

[30] M. Alazab, S. Venkatraman, and P. Watters, "Cybercrime: the case of obfuscated malware," *Global Security, Safety ...*, pp. 204–211, 2012. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-33448-1_28

[31] S. Paul and B. K. Mishra, "Honeypot based signature generation for defense against polymorphic worm attacks in networks," in *Advance Computing Conference (IACC), 2013 IEEE 3rd International*. IEEE, 2013, pp. 159–163.

[32] "Armadillo." [Online]. Available: www.siliconrealms.com

[33] "ASpack." [Online]. Available: www.aspack.com

[34] "Obsidium." [Online]. Available: www.obsidium.de

[35] "Themida." [Online]. Available: www.oreans.com

[36] RSA, "The Current State of Cybercrime and What to Expect in 2011," RSA, Tech. Rep. December 2010, 2011.

[37] ——, "The Current State of Cybercrime: An Inside Look at the Changing Threat Landscape," RSA, Tech. Rep. January, 2013.

[38] Symantec, "Internet Security Threat Report," Symantec Corporation, Tech. Rep. April, 2013.

[39] Mcafee, "2011 Threats Predictions," McAfee, Santa Clara, Tech. Rep., 2011.

[40] ——, "2013 Threats Predictions," McAfee, Santa Clara, Tech. Rep., 2013.

[41] S. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," *Journal of computer security*, 1998. [Online]. Available: http://iospress.metapress.com/index/M19JJ5LNHBEB0BVF.pdf

[42] J. Li, M. Krohn, D. Mazières, and D. Shasha, "Secure untrusted data repository (SUNDR)," *OSDI*, 2004. [Online]. Available: http://www.usenix.org/event/osdi04/tech/full_papers/li_j/li_j.pdf

[43] D. Wagner and R. Dean, "Intrusion detection via static analysis," *Security and Privacy, 2001. S&P 2001. ...*, pp. 156–168, 2001. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=924296

[44] A. Wespi, M. Dacier, and H. Debar, "Intrusion detection using variable-length audit trail patterns," *Recent advances in intrusion detection*, pp. 110–129, 2000. [Online]. Available: http://link.springer.com/chapter/10.1007/3-540-39945-3_8

[45] T. Garfinkel, "Paradigms for Virtualization Based Host Security," Ph.D. dissertation, Stanford University, 2010.

[46] T. Ptacek and T. Newsham, "Insertion, evasion, and denial of service: Eluding network intrusion detection," Information Assurance Technology Analysis Center, Falls Church, Tech. Rep., 1998. [Online]. Available: http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA391565

[47] A. Kerckhoffs, *La cryptographie militaire*. University Microfilms, 1978.

[48] B. Wotring, *Host Integrity Monitoring Using Osiris and Samhain*. Syngress, 2005.

[49] C. Ko, T. Fraser, L. Badger, and D. Kilpatrick, "Detecting and countering system intrusions using software wrappers," in *Proceedings of the 9th conference on USENIX Security Symposium-Volume 9*. USENIX Association, 2000, p. 11.

[50] S. Cesare, "Runtime kernel kmem patching," *VX Heavens*, vol. 5, pp. 63–78, 1998.

[51] K. Adams and O. Agesen, "A comparison of software and hardware techniques for x86 virtualization," *ACM SIGOPS Operating Systems Review*, p. 2, 2006. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1168857.1168860http://dl.acm.org/citation.cfm?id=1168860

[52] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=945462

[53] G. Popek and R. Goldberg, "Formal requirements for virtualizable third generation architectures," *Communications of the ACM*, 1974. [Online]. Available: http://dl.acm.org/citation.cfm?id=361073

[54] T. Garfinkel, K. Adams, A. Warfield, and J. Franklin, "Compatibility Is Not Transparency: VMM Detection Myths and Realities." *HotOS*, 2007. [Online]. Available: http://www.usenix.org/event/hotos07/tech/full_papers/garfinkel/garfinkel_html/

[55] J. Franklin, A. Seshadri, N. Qu, S. Chaki, and A. Datta, "Attacking, repairing, and verifying SecVisor: A retrospective on the security of a hypervisor," Carnegie Mellon University, Pittsburgh, Tech. Rep., 2008. [Online]. Available: http://www.cylab.cmu.edu/files/cmucylab08008.pdf

[56] T. Raffetseder, C. Krügel, and E. Kirda, "Detecting system emulators," *Information Security*, 2007. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-540-75496-1_1

[57] S. King and P. Chen, "SubVirt: Implementing malware with virtual machines," *... and Privacy, 2006 IEEE Symposium on*, 2006. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1624022

[58] J. Rutkowska and A. Tereshkin, "Bluepilling the xen hypervisor," *Black Hat USA*, 2008. [Online]. Available: ftp://crimson.ihg.uni-duisburg.de/pub/pub/Linux/Xen/Xen_0wning_Trilogy/part3.pdf

[59] K. Kortchinsky, "Cloudburst: A VMware guest to host escape story," *Black Hat USA*, 2009.

[60] P. A. Karger and D. R. Safford, "I/O for virtual machine monitors: Security and performance issues," *Security & Privacy, IEEE*, vol. 6, no. 5, pp. 16–23, 2008.

[61] C. Dalton, D. Plaquin, and W. Weidner, "Trusted virtual platforms: a key enabler for converged client devices," *ACM SIGOPS ...*, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1496918

[62] P. Dewan, D. Durham, and H. Khosravi, "A hypervisor-based system for protecting software runtime memory and persistent storage," *Proceedings of the ...*, 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1400685

[63] M. Dowty and J. Sugerman, "GPU virtualization on VMware's hosted I/O architecture," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 3, p. 73, Jul. 2009. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1618525.1618534

[64] T. Jaeger, R. Sailer, and Y. Sreenivasan, "Managing the risk of covert information flows in virtual machine systems," *... of the 12th ACM symposium on ...*, vol. 24154, 2007. [Online]. Available: http://dl.acm.org/citation.cfm?id=1266853

[65] T. Ristenpart and E. Tromer, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," *... conference on Computer ...*, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1653687

[66] M. Rosenblum and T. Garfinkel, "Virtual machine monitors: Current technology and future trends," *Computer*, vol. 38, no. 5, pp. 39 – 47, 2005. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1430630

[67] K. Jin and E. L. Miller, "The effectiveness of deduplication on virtual machine disk images," *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference on - SYSTOR '09*, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1534530.1534540

[68] N. Beebe, "Digital forensic research: The good, the bad and the unaddressed," in *Advances in digital forensics V*. Springer, 2009, pp. 17–36.

[69] D. Birk, "Technical challenges of forensic investigations in cloud computing environments," *Workshop on Cryptography and Security in Clouds*, pp. 1–6, 2011. [Online]. Available: https://www.idc-online.com/technical_references/pdfs/information_technology/TechnicalChallengesofForensicInvestigationsinCloud.pdf

[70] B. Hay and K. Nance, "Forensics examination of volatile system data using virtual introspection," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 3, p. 74, Apr. 2008. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1368506.1368517

[71] J. Dykstra and A. T. Sherman, "Acquiring forensic evidence from infrastructure-as-a-service cloud computing: Exploring and evaluating tools, trust, and techniques," *Digital Investigation*, vol. 9, pp. S90–S98, Aug. 2012. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1742287612000266

[72] L. Kelem, J. Feiertag, and T. Information, "Model for Virtual Machine Monitors," in *Research in Security and Privacy*. IEEE Computer Society Symposium, 1991, pp. 78–86.

[73] J. M. Rushby, "Design and verification of secure systems," *Proceedings of the eighth symposium on Operating systems principles - SOSP '81*, pp. 12–21, 1981. [Online]. Available: http://portal.acm.org/citation.cfm?doid=800216.806586

[74] S. Madnick and J. Donovan, "Application and analysis of the virtual machine approach to information system security and isolation," *... of the workshop on virtual computer systems*, vol. 4102, no. 01, pp. 210–224, 1973. [Online]. Available: http://dl.acm.org/citation.cfm?id=803961

[75] A. Moser, C. Kruegel, and E. Kirda, "Exploring Multiple Execution Paths for Malware Analysis," *2007 IEEE Symposium on Security and Privacy (SP '07)*, pp. 231–245, May 2007. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4223228

[76] Z. Liang, H. Yin, and D. Song, "HookFinder: Identifying and understanding malware hooking behaviors," *Department of Electrical and ...*, 2008. [Online]. Available: http://repository.cmu.edu/cgi/viewcontent.cgi?article=1004&context=ece

[77] H. Yin, D. Song, and M. Egele, "Panorama: capturing system-wide information flow for malware detection and analysis," *Proceedings of the 14th ...*, 2007. [Online]. Available: http://dl.acm.org/citation.cfm?id=1315261

[78] A. Joshi, S. King, G. Dunlap, and P. Chen, "Detecting past and present intrusions through vulnerability-specific predicates," *ACM SIGOPS Operating ...*, 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1095820

[79] N. Petroni and T. Fraser, "Copilot-a Coprocessor-based Kernel Runtime Integrity Monitor." *USENIX Security ...*, 2004. [Online]. Available: http://www.jesusmolina.com/publications/2004NPTF.pdf

[80] B. D. Payne, M. D. P. de Carbone, and W. Lee, "Secure and flexible monitoring of virtual machines," in *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*. IEEE, 2007, pp. 385–397.

[81] X. Jiang, X. Wang, and D. Xu, "Stealthy malware detection through vmm-based "out-of-the-box" semantic view reconstruction," *Proceedings of the 14th ACM conference on Computer and communications security - CCS '07*, p. 128, 2007. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1315245.1315262

[82] A. Srivastava and J. Giffin, "Tamper-resistant, application-aware blocking of malicious network connections," *Recent Advances in Intrusion Detection*, pp. 39–58, 2008. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-540-87403-4_3

[83] A. Dinaburg, P. Royal, M. Sharif, and W. Lee, "Ether: malware analysis via hardware virtualization extensions," *... of the 15th ACM conference on ...*, 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1455779

[84] S. Jones, "Antfarm: Tracking Processes in a Virtual Machine Environment." *..., General Track*, 2006. [Online]. Available: https://www.usenix.org/legacy/event/usenix06/tech/full_papers/jones/jones_html/

[85] K. Kourai and S. Chiba, "HyperSpector: virtual distributed monitoring environments for secure intrusion detection," *... conference on Virtual execution environments*, pp. 197–207, 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1065006

[86] Plato and R. W. Sharples, *Plato: Meno*. Aris and Phillips, 1985.

[87] N. Chomsky, *Aspects of the Theory of Syntax*. The MIT press, 1965, vol. 11.

[88] P. Chen and B. Noble, "When virtual is better than real," *Hot Topics in Operating Systems, 2001. ...*, 2001. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=990073

[89] B. Dolan-Gavitt, B. Payne, and W. Lee, "Leveraging forensic tools for virtual machine introspection," Georgia Institute of Technology, Atlanta, Tech. Rep., 2011. [Online]. Available: http://smartech.gatech.edu/handle/1853/38424

[90] S. Jones, "Implicit operating system awareness in a virtual machine monitor," Ph.D. dissertation, University of Wisconsin-Madison, 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.143.6999&rep=rep1&type=pdf

[91] G. Dunlap, S. King, and S. Cinar, "ReVirt: Enabling intrusion analysis through virtual-machine logging and replay," *ACM SIGOPS Operating ...*, 2002. [Online]. Available: http://dl.acm.org/citation.cfm?id=844148

[92] J. B. Grizzard and R. W. Gardner, "Analysis of Virtual Machine Record and Replay for Trustworthy Computing," *Johns Hopkins APL Technical Digest*, vol. 32, no. 2, pp. 528–535, 2013.

[93] T. Garfinkel, B. Pfaff, and J. Chow, "Terra: A virtual machine-based platform for trusted computing," *ACM SIGOPS Operating ...*, 2003. [Online]. Available: http://www.kiayias.com/compsec/CSE4707_Computer_Security/Reading_files/VM-security.pdfhttp://dl.acm.org/citation.cfm?id=945464

[94] C. Wright, C. Cowan, and S. Smalley, "Linux Security Modules: General Security Support for the Linux Kernel." *USENIX Security ...*, vol. 8032, 2002. [Online]. Available: http://www.cse.psu.edu/~tjaeger/cse544-s11/papers/lsm.pdf

[95] N. P. Loscocco, "Integrating flexible support for security policies into the Linux operating system," *Proceedings of the FREENIX Track:... USENIX ...*, 2001. [Online]. Available: ftp://130.251.61.4/pub/person/ChiolaG/sic00-01/slinux-200104121417.pdf

[96] X. Jiang and X. Wang, "Out-of-the-box Monitoring of VM-based High-Interaction Honeypots," *Recent Advances in Intrusion Detection*, 2007. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-540-74320-0_11

[97] B. D. Payne, M. Carbone, M. Sharif, and W. Lee, "Lares: An Architecture for Secure Active Monitoring Using Virtualization," *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pp. 233–247, May 2008. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4531156

[98] M. Sharif, W. Lee, W. Cui, and A. Lanzi, "Secure in-vm monitoring using hardware virtualization," *... of the 16th ACM conference on ...*, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1653720

[99] F. Zhang, K. Leach, K. Sun, and A. Stavrou, "SPECTRE: A Dependable Introspection Framework via System Management Mode," *... on Dependable Systems and ...*, no. Vmi, 2013. [Online]. Available: http://cs.gmu.edu/~astavrou/research/spectre-dsn13.pdf

[100] D. Srinivasan and X. Jiang, "Time-traveling forensic analysis of vm-based high-interaction honeypots," *Security and Privacy in Communication Networks*, 2012. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-31909-9_12

[101] C. Benninger, S. W. Neville, Y. O. Yazir, C. Matthews, and Y. Coady, "Maitland: Lighter-Weight VM Introspection to Support Cyber-security in the Cloud," *2012 IEEE Fifth International Conference on Cloud Computing*, pp. 471–478, Jun. 2012. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6253540

[102] Y. Fu and Z. Lin, "Space Traveling across VM: Automatically Bridging the Semantic Gap in Virtual Machine Introspection via Online Kernel Data Redirection," *2012 IEEE Symposium on Security and Privacy*, pp. 586–600, May 2012. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6234438

[103] C. Harrison, D. Cook, R. McGraw, and J. a. Hamilton Jr., "Constructing a Cloud-Based IDS by Merging VMI with FMA," *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 163–169, Jun. 2012. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6295971

[104] C. Schneider, J. Pfoh, and C. Eckert, "A universal semantic bridge for virtual machine introspection," *Information Systems Security*, 2011. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-25560-1_25

[105] H. Inoue and F. Adelstein, "Automatically Bridging the Semantic Gap using C Interpreter," *Annual Symposium on ...*, 2011. [Online]. Available: http://www.albany.edu/iasymposium/proceedings/2011/ASIA11Proceedings.pdf#page=60

[106] A. Azab and P. Ning, "HIMA: A hypervisor-based integrity measurement agent," *..., 2009. ACSAC'09. Annual*, pp. 461–470, 2009. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5380699

[107] T. Garfinkel and M. Rosenblum, "A Virtual Machine Introspection Based Architecture for Intrusion Detection." *NDSS*, 2003. [Online]. Available: http://www.isoc.org/isoc/conferences/ndss/03/proceedings/papers/13.pdf

[108] S. Bahram, X. Jiang, and Z. Wang, "Dksm: Subverting virtual machine introspection for fun and profit," *...Systems, 2010 29th ...*, pp. 82–91, 2010. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5623380

[109] A. Schuster, "Searching for processes and threads in Microsoft Windows memory dumps," *Digital Investigation*, vol. 3, pp. 10–16, Sep. 2006. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1742287606000727

[110] A. Walters, "FATKit: detecting malicious library injection and upping the "anti"," 4TPhi Forensic Research, Washington, DC, Tech. Rep., 2006. [Online]. Available: http://www.4tphi.net/fatkit/papers/fatkit_dll_rc3.pdf

[111] R. van Baar, W. Alink, and A. van Ballegooij, "Forensic memory analysis: Files mapped in memory," *Digital Investigation*, vol. 5, pp. S52–S57, Sep. 2008. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1742287608000327

[112] B. Dolan-Gavitt, "Forensic analysis of the Windows registry in memory," *Digital Investigation*, vol. 5, pp. S26–S32, Sep. 2008. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1742287608000297

[113] C. Kruegel and T. Toth, "A survey on intrusion detection systems," *TU Vienna, Austria*, 2000. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.72.605

[114] T. Dutkevych, A. Piskozub, and N. Tymoshyk, "Real-time intrusion prevention and anomaly analyze system for corporate networks," in *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2007. IDAACS 2007. 4th IEEE Workshop on.* IEEE, 2007, pp. 599–602.

[115] H. Zhengbing, L. Zhitang, and W. Junqi, "A novel Network Intrusion Detection System (NIDS) based on signatures search of data mining," in *Knowledge Discovery and Data Mining, 2008. WKDD 2008. First International Workshop on.* IEEE, 2008, pp. 10–16.

[116] Y. Guan and J. Bao, "A CP Intrusion Detection Strategy on Cloud Computing," *International Symposium on Web ...*, vol. 8, pp. 84–87, 2009. [Online]. Available: http://academypublisher.com/proc/wisa09/papers/wisa09p84.pdf

[117] L. Ibrahim, "Anomaly network intrusion detection system based on distributed time-delay neural network (DTDNN)," *Journal of Engineering Science and*

Technology, vol. 5, no. 4, pp. 457–471, 2010. [Online]. Available: http://jestec.taylors.edu.my/Vol5Issue4December10/Vol_5_4_457_471_L.M.Ibrahim.pdf

[118] J. Han, M. Kamber, and J. Pei, *Data mining: concepts and techniques*, 2nd ed., J. Gray, Ed. Morgan Kaufmann, 2006. [Online]. Available: http://books.google.com/books?hl=en&lr=&id=AfL0t-YzOrEC&oi=fnd&pg=PP2&dq=Data+Mining+:+Concepts+and+Techniques&ots=Uv_WsS8sB3&sig=I1ELlllCuEEtf17CRy0BCeTtTrs

[119] M. Moradi and M. Zulkernine, "A neural network based system for intrusion detection and classification of attacks," *... on Advances in Intelligent Systems Theory ...*, 2004. [Online]. Available: http://www.cs.queensu.ca/~moradi/148-04-MM-MZ.pdf

[120] A. Grediaga, F. Ibarra, F. García, B. Ledesma, and F. Brotóns, "Application of neural networks in network control and information security," in *Advances in Neural Networks-ISNN 2006*. Springer, 2006, pp. 208–213.

[121] J. Cannady, "Artificial neural networks for misuse detection," *National information systems security conference*, 1998. [Online]. Available: http://webpages.cs.luc.edu/~pld/courses/intrusion/sum08/class9/cannady.1998.artificial_neural_networks_for_misuse_detection.pdf

[122] H. Han, X.-L. Lu, and L.-Y. Ren, "Using data mining to discover signatures in network-based intrusion detection," in *Machine Learning and Cybernetics, 2002. Proceedings. 2002 International Conference on*, vol. 1. IEEE, 2002, pp. 13–17.

[123] L. Li, D.-z. Yang, and F.-c. Shen, "A novel rule-based Intrusion Detection System using data mining," *2010 3rd International Conference on Computer Science and Information Technology*, pp. 169–172, Jul. 2010. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5563714

[124] P. Tillapart, T. Thumthawatworn, and P. Santiprabhob, "Fuzzy intrusion detection system," *AU JT*, vol. 6, no. 2, pp. 109–114, 2002. [Online]. Available: http://www.journal.au.edu/au_techno/2003/jan2003/aujt6-3_article01.pdf

[125] S. Chavan, K. Shah, N. Dave, S. Mukherjee, a. Abraham, and S. Sanyal, "Adaptive neuro-fuzzy intrusion detection systems," *International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004.*, pp. 70–74 Vol.1, 2004. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1286428

[126] Y. Dhanalakshmi and I. Babu, "Intrusion detection using data mining along fuzzy logic and genetic algorithms," *International Journal of Computer Science and ...*, vol. 8, no. 2, pp. 27–32, 2008. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.133.5130&rep=rep1&type=pdf

[127] R. Gong, "A software implementation of a genetic algorithm based approach to network intrusion detection," *...Intelligence, Networking ...*, 2005. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1434896

[128] W. Lu and I. Traore, "Detecting new forms of network intrusion using genetic programming," *Computational Intelligence*, 2004. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1111/j.0824-7935.2004.00247.x/abstract

[129] T. Xia, G. Qu, S. Hariri, and M. Yousif, "An efficient network intrusion detection method based on information theory and genetic algorithm," in *Performance, Computing, and Communications Conference, 2005. IPCCC 2005. 24th IEEE International.* IEEE, 2005, pp. 11–17.

[130] D. Stiawan, A. H. Abdullah, and M. Yazid Idris, "The trends of Intrusion Prevention System network," *2010 2nd International Conference on Education*

Technology and Computer, pp. V4–217–V4–221, Jun. 2010. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5529697

[131] S. Roschke, F. Cheng, and C. Meinel, "An Extensible and Virtualization-Compatible IDS Management Architecture," *2009 Fifth International Conference on Information Assurance and Security*, pp. 130–134, 2009. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5283195

[132] A. Bakshi and B. Yogesh, "Securing cloud from ddos attacks using intrusion detection system in virtual machine," in *Communication Software and Networks, 2010. ICCSN'10. Second International Conference on.* IEEE, 2010, pp. 260–264.

[133] C.-C. Lo, C.-C. Huang, and J. Ku, "A cooperative intrusion detection system framework for cloud computing networks," in *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on.* IEEE, 2010, pp. 280–284.

[134] C. Mazzariello, R. Bifulco, and R. Canonico, "Integrating a network IDS into an open source Cloud Computing environment," *2010 Sixth International Conference on Information Assurance and Security*, pp. 265–270, Aug. 2010. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5604069

[135] W.-H. Chen, S.-H. Hsu, and H.-P. Shen, "Application of SVM and ANN for intrusion detection," *Computers & Operations Research*, vol. 32, no. 10, pp. 2617–2634, Oct. 2005. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0305054804000711

[136] H. Li and D. Liu, "Research on intelligent intrusion prevention system based on snort," in *Computer, Mechatronics, Control and Electronic Engineering (CMCE), 2010 International Conference on*, vol. 1. IEEE, 2010, pp. 251–253.

[137] O. Awodele and O. Jegede, "Neural networks and its application in engineering," *Science & IT*, 2009. [Online]. Available: http://proceedings.informingscience.org/InSITE2009/InSITE09p083-095Awodele542.pdf

[138] A. Rasoulifard, A. Bafghi, and M. Kahani, "Incremental hybrid intrusion detection using ensemble of weak classifiers," *Advances in Computer Science and ...*, 2009. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-540-89985-3_71

[139] W. Cohen, "Fast effective rule induction," *ICML*, 1995. [Online]. Available: http://www.inf.ufrgs.br/~alvares/CMP259DCBD/Ripper.pdf

[140] H. A. Bellows, *The Poetic Eddas: The Mythological Poems*. DoverPublications. com, 2004.

[141] M. T. Jones, "Anatomy of the libvirt virtualization library An API for easy Linux virtualization," *IBM developerWorks*, no. January, pp. 1–11, 2010.

[142] J. Sigvaldsen, "Improving cloud performance by solving scalability limitations in libvirt," Ph.D. dissertation, Oslo University College, 2013. [Online]. Available: https://www.duo.uio.no/handle/10852/37442

[143] D. Salomoni, A. Karen, C. Melcarne, A. Chierici, and G. D. Torre, "Performance Improvements in a Large-Scale Virtualization System," *ISGC*, pp. 19–25, 2011.

[144] L. L. C. Metasploit, "The metasploit project," *Framework*, vol. 3, 2006.

[145] J. McHugh, "Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory," *ACM transactions on Information and system Security*, vol. 3, no. 4, pp. 262–294, 2000. [Online]. Available: http://dl.acm.org/citation.cfm?id=382923

[146] K. Cios and L. Kurgan, "Trends in data mining and knowledge discovery," *. . . techniques in knowledge discovery and data mining*, no. Dm, pp. 1–26, 2005. [Online]. Available: http://link.springer.com/chapter/10.1007/1-84628-183-0_1

[147] "Orange." [Online]. Available: http://orange.biolab.si/

[148] J. R. Quinlan, "J. R. Quinlan," *AAAI/IAAI*, vol. 1, pp. 725–730, 2006.

[149] S. Džeroski and B. Ženko, "Is combining classifiers with stacking better than selecting the best one?" *Machine learning*, pp. 255–273, 2004. [Online]. Available: http://link.springer.com/article/10.1023/B:MACH.0000015881.36452.6e

[150] J. a. B. Cabrera, C. Gutiérrez, and R. K. Mehra, "Ensemble methods for anomaly detection and distributed intrusion detection in Mobile Ad-Hoc Networks," *Information Fusion*, vol. 9, no. 1, pp. 96–119, Jan. 2008. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1566253507000425

[151] J. D. Rodríguez, A. Pérez, and J. A. Lozano, "Sensitivity analysis of kappa-fold cross validation in prediction error estimation." *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 3, pp. 569–75, Mar. 2010. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/20075479

[152] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," *IJCAI*, 1995. [Online]. Available: http://frostiebek.free.fr/docs/MachineLearning/validation-1.pdf

[153] D. Powers, "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness & correlation," *Journal of Machine Learning Technologies*, no. December, 2011. [Online]. Available: http://www.bioinfo.in/uploadfiles/13031311552_1_1_JMLT.pdf

[154] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, Jun. 2006. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S016786550500303X

[155] W. Lee and S. J. Stolfo, "A Framework for Constructing Features and Models for Intrusion Detection Systems," *ACM Transactions on Information and System Security*, vol. 3, no. 4, pp. 227–261, 2001.

[156] M. Yuan, A. Ekici, and Z. Lu, "Dimension reduction and coefficient estimation in," *Royal Statistics Society*, pp. 329–346, 2007.

[157] H. G. Kayack, A. N. Zincir-heywood, and M. I. Heywood, "Selecting Features for Intrusion Detection : A Feature Relevance Analysis on KDD 99 Intrusion Detection Datasets," in *Proceedings of the Third Annual Conference on Privacy Security and Trust*, 2005, pp. 3–8.

[158] A. Kivity, Y. Kamay, and D. Laor, "kvm: the Linux virtual machine monitor," *Proceedings of the Linux ...*, 2007. [Online]. Available: https://www.kernel.org/doc/mirror/ols2007v1.pdf#page=225

[159] C. Eagle, *The IDA pro book: the unofficial guide to the world's most popular disassembler*. No Starch Press, 2008.

[160] M. Janes, "Sandia computer scientists successfully boot one million linux kernels as virtual machines," 2009. [Online]. Available: https://share.sandia.gov/news/resources/news_releases/sandia-computer-scientists-successfully-boot-one-million-linux-kernels-as-virtual-machines/

[161] L. Arellano, S. Arroyo, G. Giese, P. Cox, and G. Rogers, "Cloud computing strategic framework (FY13-FY15)." Sandia National Laboratories, Albuquerque, Tech. Rep. November, 2012. [Online]. Available: http://www.osti.gov/scitech/biblio/1055927

[162] S. Bratus and P. Johnson, "The cake is a lie: privilege rings as a policy resource," *... the 1st ACM workshop ...*, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1655154

[163] T. Ormandy, "An empirical study into the security exposure to hosts of hostile virtualized environments," *Proceedings of CanSecWest Applied Security ...*, 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.105.6943&rep=rep1&type=pdf

[164] P. Ferrie, "Attacks on more virtual machine emulators," *Symantec Technology Exchange*, 2007. [Online]. Available: http://repo.meh.or.id/Windows/attacks2.pdf

[165] A. Seshadri, M. Luk, N. Qu, and A. Perrig, "SecVisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes," *ACM SIGOPS Operating Systems ...*, vol. 41, pp. 335–350, 2007. [Online]. Available: http://dl.acm.org/citation.cfm?id=1294294

[166] J. Pfoh, C. Schneider, and C. Eckert, "A formal model for virtual machine introspection," *... 1st ACM workshop on Virtual machine ...*, p. 1, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1655148.1655150http://dl.acm.org/citation.cfm?id=1655150

[167] F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger, and D. Leaf, "NIST Cloud Computing Reference Architecture Recommendations of the National Institute of Standards and," National Institute of Standards and Technology, Gaithersburg, Tech. Rep., 2011.

[168] W. A. Vogels, "Head in the CloudsThe Power of Infrastructure as a Service," in *First workshop on Cloud Computing and in Applications (CCA08)(October 2008)*, 2008.

Appendices

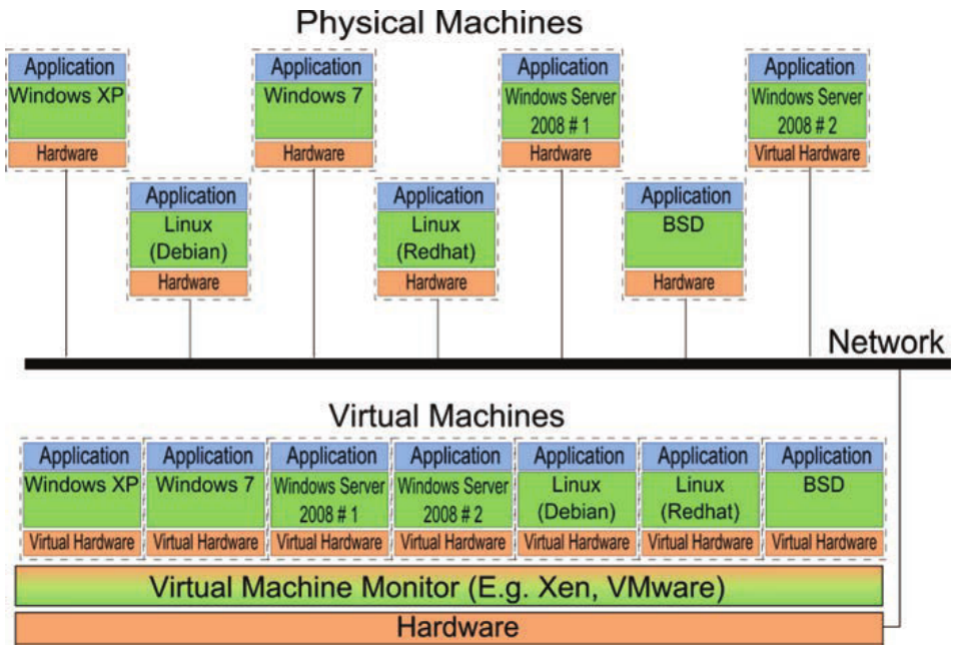
Appendix A

Key Concepts

A.1 Virtualization

A.1.1 Overview

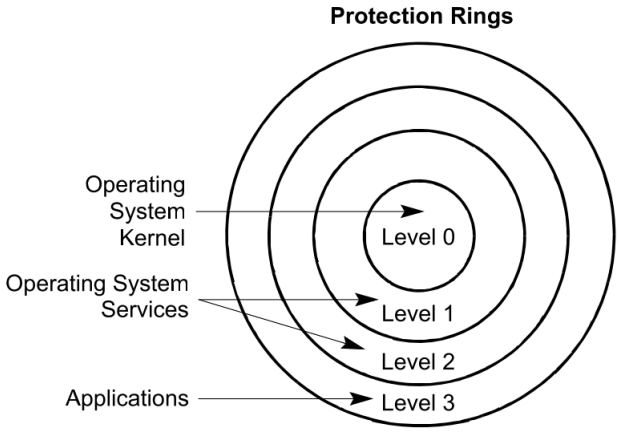
Figure A.1: Physical Machines and their Virtualized Counterpart [9].



Virtualization is the use of highly privileged software to create an encapsulating software layer that decouples the operating system from the physical hardware and provides a layer of indirection to accurately mimic the expected behavior of physical hardware resulting in "an efficient, isolated duplicate of the real machine" [53]. *Privilege* in this context is the authority to perform operations that the hardware supports; privilege levels are typically called rings (see Figure A.2), with lower rings having more privileges than those of a higher number [162]. The operating system traditionally has been given the lowest ring (ring 0); however, in

the context of virtualization, the Virtual Machine Monitor or Hypervisor (VMM) supplants the Operating System (OS) as ring 0. This level of indirection affords key properties that are necessary for Oðinn's success.

Figure A.2: Operating System Kernel Protection Rings [3].



A.1.2 Properties of VMMs

Isolation

The encapsulating layer that allows operating systems to be placed inside VMs births a barrier that isolates not only between OSs running on the same physical hardware, but also between the individual OSs and the physical hardware [163]. This isolation allows for risky services to be on one Virtual Machine (VM) and critical services on another (but both using the same physical hardware) to be insulated from one another such that the critical service is not compromised if the risky service becomes compromised [74]. Furthermore, malware analysts now possess isolated environments that do not require the time-consuming setup of cleaning and reinstalling an OS for every malware sample that requires examination [56, 83, 164]. For this isolation to be breached, the VMM itself must be exploited, which is several orders of magnitude harder, because while an operating system might have millions or even tens of millions of lines of code [55, 165] and cannot be provably secure, the VMM contains a few thousand or tens of thousands of lines of code, which can be provably secure

[166]. Additionally, OSs add a single point of failure for all processes and data within. The VMM can then consolidate a collection of virtual machines with low resources onto a single computer, thereby lowering hardware costs and space requirements. Strong isolation is also valuable for reliability and security.

A.1.3 Inspection

A VMM has complete visibility of the virtual machine. Every CPU, memory and I/O device state can be captured and ignoring the very real constraints of the heuristic utilized and the overhead incurred to the user; by definition, a Hypervisor-based Intrusion Detection System (VMI-IDS) is theoretically impossible to obfuscate its activities as there is no state the Intrusion Detection System (IDS) is denied from viewing.

Encapsulation

As the VMM provides an uniform view of underlying hardware, the virtual machine is incapable of distinguishing the different underlying physical systems beneath it. As a result, different vendors with different subsystems are abstracted away from the view of the virtual machine, affording a high level of compatibility, allowing virtual machines to run on any physical system that supports virtualization. This abstraction works from an administration viewpoint as well, allowing the hardware to be abstracted as a pool of resources to be arbitrarily run on demand for whatever services are desired. Tangential to this is OS independence. The same encapsulation that provides a unified view of physical systems also abstracts away the semantics of the guest OS, allowing trivial support for multiple OSes and multiple OS versions from being supported simultaneously on the same hardware. Additionally, the encapsulation of the VM software state allows for the mapping, remapping and migration (moving from one physical machine to another) of VMs. This simplifies load balancing, increases hardware fault tolerance, and greatly aids in scalability. The final advantage wrought by encapsulation is the ability to arbitrarily pause and resume virtual

machines or save their state in a specific moment of time (called a snapshot) and then roll them back to that specific moment when needed. This has numerous purposes though the most common is to recover from crashes and to aid with configuration.

A.2 Cloud Computing

NIST defines cloud computing as "a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [167]."

Five features are intrinsic to cloud computing [167]:

1. On-demand self-service - Instantaneously, as needed, a consumer can receive computational resources automatically, without interfacing with a human.

2. Broad network access - Resources are provided over the Internet and are utilized by the consumer with heterogeneous platforms (desktop, mobile phones, etc.).

3. Resource Pooling - Resources are pooled together by the cloud service provider using a virtualization model, "with different physical and virtual resources dynamically assigned and reassigned according to consumer demand" to service throngs of consumers.

4. Rapid Elasticity - Computational resources are better viewed as immediate rather than persistent as the resources scale up by need and can be terminated and reacquired as desired.

5. Measured Service - Metering capabilities are such that, despite multiple consumers using the same pooled resources, the cloud infrastructure is sufficiently able to measure their usage at the individual consumer level.

This gives rise to three distinct novel advantages that have lead to widespread adoption of the cloud [168]. First, from the consumer perspective, resource provisioning rapidly

rises to meet peak consumption needs and gives the illusion of infinite resources eliminating consumers need to plan ahead for provisioning. Second, companies no longer have large up-front costs to building large scale infrastructure and need to only increase centralized hardware resources as the need grows. Third, resources are only used when needed and released thereafter, conserving energy advantageous to both energy cost and environmental impact.

A.3 Binary Classification Results

Process Scan

Predicted Class

Actual Class	**p**	**n**	
p′	TP 13	FN 4	PPV: 0.81
n′	FP 3	TN 31	NPV: 0.89
	SEN: 0.76	SPC: 0.91	ACC: 0.86

Evaluation Metrics

AUC: 0.94	MCC: 0.69	RMSE: 0.37
χ^2: 24	F_1: 0.79	R^2: 0.588

Table A.1: Results for Bayes

Predicted Class

Actual Class	**p**	**n**	
p′	TP 14	FN 3	PPV: 0.82
n′	FP 3	TN 31	NPV: 0.92
	SEN: 0.82	SPC: 0.91	ACC: 0.89

Evaluation Metrics

AUC: 0.89	MCC: 0.74	RMSE: 0.343
χ^2: 28	F_1: 0.82	R^2: 0.647

Table A.2: Results for Logistics Regression

Predicted Class

Actual Class	**p**	**n**	
p′	TP 13	FN 4	PPV: 0.81
n′	FP 3	TN 31	NPV: 0.89
	SEN: 0.76	SPC: 0.91	ACC: 0.86

Evaluation Metrics

AUC: 0.85	MCC: 0.69	RMSE: 0.37
χ^2: 24	F_1: 0.79	R^2: 0.588

Table A.3: Results for ANN

		Predicted Class		
		p	**n**	
Actual Class	**p′**	TP 12	FN 5	PPV: 0.63
	n′	FP 7	TN 27	NPV: 0.84
		SEN: 0.71	SPC: 0.79	ACC: 0.76

Evaluation Metrics		
AUC: 0.77	MCC: 0.49	RMSE: 0.485
χ^2: 12	F_1: 0.67	R^2: 0.294

Table A.4: Results for Association Rules

		Predicted Class		
		p	**n**	
Actual Class	**p′**	TP 13	FN 4	PPV: 0.68
	n′	FP 6	TN 28	NPV: 0.88
		SEN: 0.76	SPC: 0.82	ACC: 0.80

Evaluation Metrics		
AUC: 0.90	MCC: 0.57	RMSE: 0.443
χ^2: 17	F_1: 0.72	R^2: 0.412

Table A.5: Results for kNN

Predicted Class

Actual Class	**p**	**n**	
p′	TP 13	FN 4	PPV: 0.87
n′	FP 2	TN 32	NPV: 0.89
	SEN: 0.76	SPC: 0.94	ACC: 0.88

Evaluation Metrics

AUC: 0.8	MCC: 0.73	RMSE: 0.343
χ^2: 27	F_1: 0.81	R^2: 0.647

Table A.6: Results for C4.5

Predicted Class

Actual Class	**p**	**n**	
p′	TP 14	FN 3	PPV: 0.82
n′	FP 3	TN 31	NPV: 0.91
	SEN: 0.82	SPC: 0.91	ACC: 0.88

Evaluation Metrics

AUC: 0.87	MCC: 0.74	RMSE: 0.343
χ^2: 28	F_1: 0.82	R^2: 0.647

Table A.7: Results for SVM

		Predicted Class		
		p	**n**	
Actual Class	**p′**	TP 11	FN 6	PPV: 0.92
	n′	FP 1	TN 33	NPV: 0.85
		SEN: 0.65	SPC: 0.97	ACC: 0.86

Evaluation	**Metrics**	
AUC: 0.92	MCC: 0.69	RMSE: 0.37
χ^2: 24	F_1: 0.76	R^2: 0.588

Table A.8: Results for Random Forest

		Predicted Class		
		p	**n**	
Actual Class	**p′**	TP 14	FN 3	PPV: 0.82
	n′	FP 3	TN 31	NPV: 0.91
		SEN: 0.82	SPC: 0.91	ACC: 0.88

Evaluation	**Metrics**	
AUC: 0.90	MCC: 0.74	RMSE: 0.343
χ^2: 28	F_1: 0.82	R^2: 0.647

Table A.9: Results for Stacking

Modules Scan

		Predicted Class		
		p	**n**	
Actual Class	**p′**	TP 28	FN 1	PPV: 0.90
	n′	FP 3	TN 126	NPV: 0.99
		SEN: 0.97	SPC: 0.98	ACC: 0.97

Evaluation Metrics		
AUC: 0.97	MCC: 0.92	RMSE: 0.159
χ^2: 133	F_1: 0.93	R^2: 0.862

Table A.10: Results for Bayes

Predicted Class

Actual Class	**p**	**n**	
p′	TP 28	FN 1	PPV: 0.97
n′	FP 1	TN 128	NPV: 0.99
	SEN: 0.97	SPC: 0.99	ACC: 0.99

Evaluation Metrics

AUC: 0.97	MCC: 0.96	RMSE: 0.113
χ^2: 145	F_1: 0.97	R^2: 0.931

Table A.11: Results for Logistic Regression

Predicted Class

Actual Class	**p**	**n**	
p′	TP 28	FN 1	PPV: 0.97
n′	FP 1	TN 128	NPV: 0.99
	SEN: 0.97	SPC: 0.99	ACC: 0.99

Evaluation Metrics

AUC: 0.97	MCC: 0.96	RMSE: 0.113
χ^2: 145	F_1: 0.97	R^2: 0.931

Table A.12: Results for ANN

Predicted Class

Actual Class	**p**	**n**	
p′	TP 28	FN 1	PPV: 0.97
n′	FP 1	TN 128	NPV: 0.99
	SEN: 0.97	SPC: 0.99	ACC: 0.99

Evaluation Metrics

AUC: 0.97	MCC: 0.96	RMSE: 0.113
χ^2: 145	F_1: 0.97	R^2: 0.931

Table A.13: Results for Association Rules

Predicted Class

Actual Class	**p**	**n**	
p′	TP 28	FN 1	PPV: 0.97
n′	FP 1	TN 128	NPV: 0.99
	SEN: 0.97	SPC: 0.99	ACC: 0.99

Evaluation Metrics

AUC: 0.97	MCC: 0.96	RMSE: 0.113
χ^2: 145	F_1: 0.97	R^2: 0.931

Table A.14: Results for kNN

Predicted Class

Actual Class	p	n	
p′	TP 28	FN 1	PPV: 0.97
n′	FP 1	TN 128	NPV: 0.99
	SEN: 0.97	SPC: 0.99	ACC: 0.99

Evaluation Metrics

AUC: 0.97	MCC: 0.96	RMSE: 0.113
χ^2: 145	F_1: 0.97	R^2: 0.931

Table A.15: Results for C4.5

Predicted Class

Actual Class	p	n	
p′	TP 28	FN 1	PPV: 0.97
n′	FP 1	TN 128	NPV: 0.99
	SEN: 0.97	SPC: 0.99	ACC: 0.99

Evaluation Metrics

AUC: 0.97	MCC: 0.96	RMSE: 0.113
χ^2: 145	F_1: 0.97	R^2: 0.931

Table A.16: Results for Random Forest

Predicted Class

Actual Class	**p**	**n**	
p′	TP 28	FN 1	PPV: 0.97
n′	FP 1	TN 128	NPV: 0.99
	SEN: 0.97	SPC: 0.99	ACC: 0.99

Evaluation Metrics

AUC: 0.99	MCC: 0.96	RMSE: 0.113
χ^2: 145	F_1: 0.97	R^2: 0.931

Table A.17: Results for Stacking

File Scan

		Predicted Class		
		p	**n**	
Actual Class	**p′**	TP 506	FN 349	PPV: 0.70
	n′	FP 220	TN 2080	NPV: 0.86
		SEN: 0.59	SPC: 0.90	ACC: 0.82

Evaluation Metrics		
AUC: 0.79	MCC: 0.52	RMSE: 0.525
χ^2: 866	F_1: 0.64	R^2: 0.335

Table A.18: Results for Bayes

		Predicted Class		
		p	**n**	
Actual Class	**p′**	TP 464	FN 391	PPV: 0.70
	n′	FP 200	TN 2100	NPV: 0.84
		SEN: 0.54	SPC: 0.91	ACC: 0.81

Evaluation Metrics		
AUC: 0.78	MCC: 0.50	RMSE: 0.433
χ^2: 779	F_1: 0.61	R^2: 0.309

Table A.19: Results for Logistic Regression

		Predicted Class		
		p	**n**	
Actual Class	**p′**	TP 470	FN 385	PPV: 0.76
	n′	FP 149	TN 2151	NPV: 0.85
		SEN: 0.55	SPC: 0.94	ACC: 0.83

Evaluation Metrics		
AUC: 0.84	MCC: 0.54	RMSE: 0.411
χ^2: 929	F_1: 0.64	R^2: 0.375

Table A.20: Results for ANN

		Predicted Class		
		p	**n**	
Actual Class	**p′**	TP 593	FN 262	PPV: 0.72
	n′	FP 228	TN 2072	NPV: 0.89
		SEN: 0.69	SPC: 0.9	ACC: 0.84

Evaluation Metrics		
AUC: 0.85	MCC: 0.60	RMSE: 0.394
χ^2: 1144	F_1: 0.71	R^2: 0.427

Table A.21: Results for Association Ruleset

		Predicted Class		
		p	**n**	
Actual Class	**p′**	TP 473	FN 382	PPV: 0.79
	n′	FP 124	TN 2176	NPV: 0.85
		SEN: 0.55	SPC: 0.95	ACC: 0.84

Evaluation Metrics		
AUC: 0.87	MCC: 0.57	RMSE: 0.400
χ^2: 1013	F_1: 0.65	R^2: 0.408

Table A.22: Results for kNN

Predicted Class

Actual Class	**p**	**n**	
p′	TP 477	FN 378	PPV: 0.82
n′	FP 104	TN 2196	NPV: 0.85
	SEN: 0.56	SPC: 0.95	ACC: 0.85

Evaluation Metrics

AUC: 0.82	MCC: 0.59	RMSE: 0.391
χ^2: 1090	F_1: 0.66	R^2: 0.436

Table A.23: Results for C4.5

Predicted Class

Actual Class	**p**	**n**	
p′	TP 518	FN 337	PPV: 0.69
n′	FP 231	TN 2069	NPV: 0.86
	SEN: 0.61	SPC: 0.90	ACC: 0.82

Evaluation Metrics

AUC: 0.77	MCC: 0.53	RMSE: 0.424
χ^2: 879	F_1: 0.65	R^2: 0.336

Table A.24: Results for SVM

		Predicted Class		
		p	**n**	
Actual Class	**p′**	TP 273	FN 582	PPV: 0.88
	n′	FP 36	TN 2264	NPV: 0.80
		SEN: 0.32	SPC: 0.98	ACC: 0.80

Evaluation Metrics		
AUC: 0.87	MCC: 0.45	RMSE: 0.443
χ^2: 650	F_1: 0.47	R^2: 0.277

Table A.25: Results for Random Forest

		Predicted Class		
		p	**n**	
Actual Class	**p′**	TP 589	FN 266	PPV: 0.67
	n′	FP 285	TF 2015	NPV: 0.88
		SEN: 0.69	SPC: 0.88	ACC: 0.83

Evaluation Metrics		
AUC: 0.90	MCC: 0.56	RMSE: 0.418
χ^2: 993	F_1: 0.68	R^2: 0.356

Table A.26: Results for Stacking