Securing Web Applications: Web Application Flow Whitelisting to Improve Security

by

Haneen Khalid Alabdulrazzaq

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
May 6, 2017

Keywords: Secure Software Development, Web Application Security, Secure Coding, Insecure Direct Object References, Missing Function Level Access Control, Whitelisting

Abstract

The explosion in the availability of data fueled by mobile devices has pushed security to the forefront. As of 2016, Internet users worldwide are estimated at a staggering 3.47 billion. Such large numbers of users dictate the importance of online presence for organizations across different industries, and with that come security considerations for web-facing applications. Many web development frameworks offer common security features, such as authentication and session management out of the box (e.g. Rails, Django, and CakePHP). However, developers must still direct their efforts toward addressing application specific security issues.

In this dissertation we introduce a novel web application security enhancing practice, called web application flow whitelisting. It targets two of OWASP's top ten web vulnerabilities; A4: Insecure Direct Object References and A7: Missing Function Level Access Control. Initially, we create a workflow based on intended behavior. From the workflow, we define a whitelist as a tuple $\langle C, D, W, S \rangle$, where C is a set of all components within the system boundary, D is a set containing conditions for transitions that occur within an application, W is a set of all ordered pairs representing allowed transitions, and finally S is a matrix containing safe components to redirect to in case a transition fails.

Validation of web application flow whitelisting was carried out in two phases. For phase one, a static whitelist was created and applied to a total of 15 applications developed by students in course COMP4970: Web Development with Django at Auburn University. For phase two, a static whitelist was created and applied on an existing open source Django application in production use. Application specific vulnerabilities were found and quantified through manual testing methods. Our results show that by conforming to a whitelist of flow, all unintended application behavior is eliminated while intended behavior is preserved.

Acknowledgments

"Praise Allah, with whose blessings all good deeds are achieved"

I thank God everyday for giving me the strength and will to go through this long (and emotionally draining!) journey toward a PhD degree. I could not have done this without the support of so many people in my life and I would like to take this opportunity to thank each and every one of them.

My deepest gratitude goes to Dr. David Umphress, my advisor, for his guidance, assistance, and compassion. I am very fortunate to have had the opportunity to work under his mentorship. Dr. Umphress, I finally saw the light at the end of the tunnel! I am forever indebted to you and could not have accomplished this without all your help. Thank you so much! I would like to thank my committee members: Dr. Cross, Dr. Hendrix, and Dr. Skjellum for their valuable feedback on my work, and Dr. Jerry Davis for taking time out of his busy schedule to serve as the University Reader for this dissertation. Also, a warm "Thank you!" goes out to Dr. Jeffrey Overbey who served on my committee prior to leaving the CSSE department at Auburn. His early guidance on this research has significantly shaped its future direction. I would also like to thank all the Software Process research group members who have bounced around ideas with me during the early stages of my research, especially Dr. Bradley Dennis and Matthew Swann.

I would like to thank Mrs. Shoghig Sahakyan, my scholarship advisor at the Embassy of Kuwait for the prompt replies to the many emails I sent over the past 5 years. Shoghig, you are amazing! And, I would like to thank Ms. Anwar Al-Bader at the PAAET Scholarships Department for speeding up the approval process of my extension request.

I would like to thank ALL my friends in both Auburn and Kuwait for being the wonderful people that they are! To Yasmeen and Taha, friends I met during my first semester here at Auburn, I could not have survived without you! Thank you for all the late night study sessions, scrambling to submit assignments on time, and for always offering help and advice when I needed it. And, thank you for all the great laughs, fun times, and beautiful memories I will cherish forever. To all my friends in Kuwait, I will see you soon!

My family means everything to me and I would not be the person I am today without their love and encouragement. To my Dad, who I wish had lived to witness this moment, I miss you. You are always in my thoughts and prayers. Mom, you are my role model. Thank you for being so supportive of my decision to pursue a PhD, even though it meant I was moving your granddaughters halfway across the world! I am so grateful to have a mother like you. To my sisters: Shouq, Besma, Hala and my brother Yousef, thank you for your unconditional love, support, and prayers. To my grandmother Luluwa and my aunts: Hind, Nadia, and Ghada, thank you for your encouragement and prayers. To the rest of the family, I can't wait to be back home with y'all! (that's southern meaning 'you all').

To my husband and best friend, Fahad, whose love, commitment, and support never wavered throughout the years, I love you. I know this journey has tested our relationship at times, but I do believe it made it stronger. Thank you for your reassurance when I doubted my own abilities. Thank you for your (sometimes brutal) honesty whenever I needed a sincere opinion. Thank you for everything! To my daughters, Awatif, Ibtesam, Sarah, and Aishah, thank you for your love, patience, and understanding especially when I could not be there for all your activities and school events. I appreciate all the things you've done for me during the past 5 years. I love you all so much!

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Importance of Secure Software Development

The explosion in the availability of data fueled by mobile devices has pushed security to the forefront. Yet, the software engineering community lags in identifying and using processes that produce secure software. Davis [Davis et al., 2004] notes:

> "Producing secure software is a multifaceted problem of software engineering, security engineering, and management. Thus, producing secure software starts with outstanding software engineering practices, augmented with sound technical practices, and supported by management practices that promote secure software development."

Only now is the software industry recognizing the relevance of security, much less come to grips with engineering approaches to address it.

Data breaches of big name companies such as JP Morgan Chase, Target, and Sony Pictures Entertainment making headline news in 2014 appear to have pushed the importance of security to the point where it is gaining traction. Wysopal [Wysopal, 2015] points out that within the next 3 years, the expenditure on software security will increase to about 26.8% of what it is currently, with more focus on securing the application-layer rather than the network layer.

While traditional heavy weight secure software development processes and practices aid in producing more secure software they do not fit in with today's agile development environments. A survey conducted by [Ayalew et al., 2013] has found that none of the practices in traditional secure software development processes were compatible and beneficial in agile projects. Lane [Hazrati, 2012] goes so far as to posit that agile development teams are not capable of producing secure software . Agile techniques focus on producing features and functionality, leaving security as an after thought. Furthermore, developers shy away from security processes and practices simply because they require extra time, extensive knowledge, and training to implement. Training requires monetary investment, and for many management teams, it is something they are not willing to accept. Putting aside training costs, the extra effort needed to incorporate security into software practices has given the perception of being very expensive [Davis et al., 2004]. Even when security is explicitly expressed as a requirement, small and medium -sized software development organizations have difficulty achieving security goals for web facing applications [Nicolaysen et al., 2010].

1.2 The Problem with Secure Software Development Processes

Several development processes and best practices that incorporate security do, in fact, exist. Most are processes that have been tailored to include security, because, "building processes from scratch is risky and involves high overhead, so developers often tailor existing processes and standards" [Xu and Ramesh, 2008]. An example of this is Microsoft® Security Development Lifecycle for Agile Development, which takes the waterfall-like Security Development Lifecycle and breaks down its requirements into every sprint requirements, bucket requirements (performed regularly over the lifetime of the project), and one time requirements [Microsoft, 2009a]. Other examples include those for tailoring Extreme Programming to support security requirements [Boström et al., 2006] [Beznosov, 2003].

In organizations requiring that a software process be followed during development, there tends to be a negative connotation associated with following it [Wiegers, 2005]. As mentioned earlier, many existing processes for developing secure software are considered heavyweight, and with the existing attitude towards process in general, one can easily conclude that this attitude also spreads to include secure software development processes. The alternative to using a complete process for developing secure software, is the adaptation of security best practices into one's own software development process. By doing so, a development team is essentially tailoring its process to satisfy security as a nonfunctional requirement. There are practices in secure software development that are considered vital and are usually done in the early phases of development. An example is Threat Modeling which is a design phase activity that fits very well in a waterfall-like process. Usually in a waterfall-like approach, a threat model diagram is produced, and when it is completed, the project can move on to the next phase. However, no such clear cut distinction exists in agile sprints. This does not necessarily mean that Threat Modeling cannot be incorporated in an agile manner. It just requires some extra effort from agile teams whereby Threat Models are treated as “living artifacts that should be updated and enhanced during every iteration or sprint” [Jeffries, 2012]. Other ways of combining agile and security are the use of security sprints [Bird, 2012], abuser stories [Boström et al., 2006] [Peeters, 2005], and a security version of Planning Poker called Protection Poker [Williams et al., 2009].

1.3 Web Application Security

The number of Internet users in 2016 worldwide was estimated at a staggering 3.47 billion [internetlivestats.com, 2016]. Contrasting this with 502 million users fifteen years ago clearly shows the growth of Internet usage worldwide. In the United States alone, the US Census Bureau reported that 74.4% of all households in the country use the Internet [File and Ryan, 2014]. Such large numbers of users dictate the importance of online presence for organizations across different industries, and with that comes security considerations for web-facing applications.

The World Wide Web has extensively evolved from a handful of static HTML web pages when it emerged in the early 1990’s to the form we know today. Most of the websites online today are applications producing content dynamically [Stuttard and Pinto, 2011]. With Web 2.0 technologies dominating the Internet and allowing for user generated content, web application developers need to focus on securing their applications against malicious use. The software engineering community in both academia and industry has focused the majority of its research efforts on securing the network layer of the web [Bhimani, 1996] [Oppliger, 2003] [Cheswick et al., 2003]. Several technologies were born from this research and some became mainstream practices, such as using Secure Sockets Layer (SSL) and Hypertext Transfer Protocol Secure (HTTPS). However, securing the application layer, from an application development perspective, has not gained similar attention. While SSL encrypts data in transit, it does not prevent its leakage, disclosure, or misuse at the source or destination. This is something that can only be dealt with by the developer at the application layer.

A number of web development frameworks offer common website security features, such as authentication and session management out of the box (e.g. Rails, Django, and CakePHP). They address general security concerns that can be programmatically detected allowing the developer to direct development efforts towards the application's functionality, features, and special case security. They do not address security issues that are domain or application specific.

Stories about high profile data breaches abound, but many breaches are not publicized, as pointed out, with some hyperbole, by FBI director James Comey: "There are two kinds of big companies in the United States. There are those who've been hacked ... and those who don't know they've been hacked" [Walters, 2014]. Hackers have become more sophisticated in their ways, with many working within organized groups and some funded by foreign governments to gain unauthorized access to valuable information on the web. The Software Engineering Institute estimates that 90% of reported security incidents resulted from exploiting defects in the design or code of software [U.S. CERT, 2013]. Developers must pay special attention to security during the design and implementation phases of software development. Many industry and government agencies recognize now the importance of building security in and are offering tools, practices, guidelines, and other resources to help software developers. The Computer Emergency Readiness Team (CERT), the National Cyber Security Division, and the National Institute of Standards and Technology (NIST), all support initiatives and projects that help developers build security in [U.S. DHS, 2011] [NIST, 2014]. The industry recognizes that “if we are going to fix the field of computer security, the only hope we have is building security in” [McGraw, 2012].

In this dissertation a novel web application security practice, called web application flow whitelisting, is proposed. It follows the concept of whitelisting which is increasingly being used to protect against malware and spam. A whitelist is a list of elements (e.g. data, applications, email addresses, etc.) that have the privilege to execute with granted special access permissions. In contrast to a blacklist, which would contain those elements that do not have execution privileges or access permissions, anything not on the whitelist would be denied to run or would have constrained access permissions. Whitelists and blacklists have both been used in network security [Schneier and Ranum, 2011], malware [Cobb, 2013], and as a technique for input validation [SAFECode, 2011]. The term *flow* refers to the series and order of navigational steps taken to complete a certain task on a web application. The idea proposed by this research requires that the developer create a whitelist of allowed flow and transitions between controllers in a web application built using the Model-View-Controller pattern. Any transition or flow not specified in the whitelist would result in the application resorting to a ‘safe state’ which is predefined by the developer.

1.4 Research Scope

The focus of this research will be on the software engineering practices that promote production of secure software with the ability of being integrated into an existing agile process. With the area of information security being vast and application types varied, this research simply targets web application security. Moreover, only the secure coding practices pertaining to web applications constructed using web development frameworks will be examined. The research is also in line with the recent notion and initiative of building security into a software product. The aim is to show the security best practices used today in web application development and to explore the vulnerabilities that target web applications in particular. Finally, the focus will be on the vulnerabilities that the developer is responsible for mitigating; those handled by the web development framework will be disregarded. This research will address the following question: will whitelisting a web application's flow produce more secure web applications?

Chapter 2

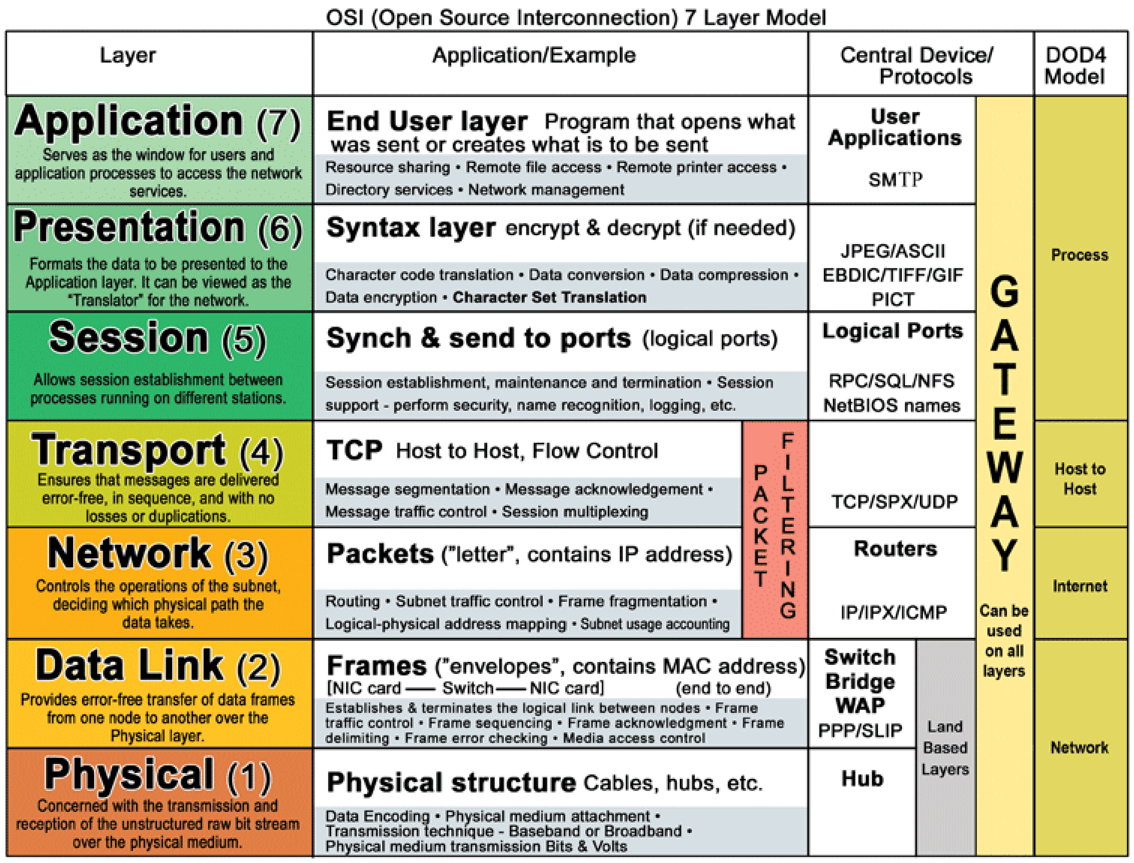
Literature Review

2.1 Addressing Security in the OSI Model

Caelli's observation,

> "It is an accepted principle of computer science and engineering that a computer application can be **no more** secure than the libraries and middleware it incorporates that can themselves be **no more** secure than the operating system and sub-systems that support them which in turn can be **no more** secure than the underlying hardware and firmware of the computer or network system."

[Caelli, 2007] suggests that the most logical way to envision security is through the lens of a layered system. In keeping with this idea, Meunier [Meunier, 2008] proposes the OSI reference model (figure 2.1) as a natural taxonomy for classifying vulnerabilities according to the layer to which they belong. We follow suit by traversing the OSI layers to examine security vulnerabilities from the developer's perspective, the goal being to identify how those vulnerabilities might be eliminated in a light weight process.



OSI (Open Source Interconnection) 7 Layer Model

Layer	Application/Example		Central Device/ Protocols			DOD4 Model
Application (7) Serves as the window for users and application processes to access the network services.	**End User layer** Program that opens what was sent or creates what is to be sent Resource sharing • Remote file access • Remote printer access • Directory services • Network management		**User Applications** SMTP		**GATEWAY** Can be used on all layers	Process
Presentation (6) Formats the data to be presented to the Application layer. It can be viewed as the "Translator" for the network.	**Syntax layer** encrypt & decrypt (if needed) Character code translation • Data conversion • Data compression • Data encryption • **Character Set Translation**		JPEG/ASCII EBDIC/TIFF/GIF PICT			
Session (5) Allows session establishment between processes running on different stations.	**Synch & send to ports** (logical ports) Session establishment, maintenance and termination • Session support - perform security, name recognition, logging, etc.		**Logical Ports** RPC/SQL/NFS NetBIOS names			
Transport (4) Ensures that messages are delivered error-free, in sequence, and with no losses or duplications.	**TCP** Host to Host, Flow Control Message segmentation • Message acknowledgement • Message traffic control • Session multiplexing	PACKET FILTERING	TCP/SPX/UDP			Host to Host
Network (3) Controls the operations of the subnet, deciding which physical path the data takes.	**Packets** ("letter", contains IP address) Routing • Subnet traffic control • Frame fragmentation • Logical-physical address mapping • Subnet usage accounting		**Routers** IP/IPX/ICMP			Internet
Data Link (2) Provides error-free transfer of data frames from one node to another over the Physical layer.	**Frames** ("envelopes", contains MAC address) [NIC card — Switch — NIC card] (end to end) Establishes & terminates the logical link between nodes • Frame traffic control • Frame sequencing • Frame acknowledgment • Frame delimiting • Frame error checking • Media access control		**Switch** **Bridge** **WAP** PPP/SLIP	Land Based Layers		Network
Physical (1) Concerned with the transmission and reception of the unstructured raw bit stream over the physical medium.	**Physical structure** Cables, hubs, etc. Data Encoding • Physical medium attachment • Transmission technique - Baseband or Broadband • Physical medium transmission Bits & Volts		**Hub**			

Figure 2.1: Protocols used in different layers of the OSI model [Ismail, 2012]

2.1.1 Security of Layer 1: The Physical Layer

The physical layer of the OSI model deals with the transmission and reception of raw bits of data over a physical medium [Myhre, 2000]. It, in essence, is electric signals, radio waves, optical pulses, etc. that represent data. Security concerns at this layer are typically two fold: physical security and signal security. The former entails safeguards such as guarding data and transmission facilities, controlling the physical access of personnel; the latter aims to protect connection confidentiality and traffic flow confidentiality [CCITT, 1991]. Common vulnerabilities at this layer are hardware hacking, wiretapping, interception of signals, and physical access attacks [Gregg and Watkins, 2006], as well as signal replay attacks, feature replay attacks, and coercion attacks [Danev et al., 2012].

The extreme low level at which this layer operates means that exploiting vulnerabilities that require domain or application specific knowledge is highly unlikely. The application developer relies instead on low level protections such as, extracting physical fingerprints from a device's circuitry [Lofstrom et al., 2000] [Holcomb et al., 2009] [Su et al., 2007], physically unclonable functions (PUFs) [Suh and Devadas, 2007] [Lim et al., 2005], integrated circuit watermarking [Abdel-Hamid et al., 2003] [Torunoglu and Charbon, 2000] [Koushanfar and Alkabani, 2010], and the physical properties of wireless channels [Faria and Cheriton, 2006] [Patwari and Kasera, 2007].

2.1.2 Security of Layer 2: The Data Link Layer

The data link layer of the OSI model is responsible for converting the data arriving from the upper layers into bits to send across a physical medium (wire) and vice versa [Myhre, 2000]. It is divided into two sub-layers: Logical Link Control and Media Access Control. [CCITT, 1991] recommends that two security services be in place here, connection confidentiality and connectionless confidentiality. Connection confidentiality service requires that all (N) user-data on an (N) connection be confidential. Similarly, connectionless confidentiality service requires protecting the confidentiality of all (N)-user-data in a single connectionless (N)-service data unit. Common vulnerabilities in this layer are active and passive sniffing, MAC spoofing, Wired Equivalent Privacy (WEP) cracking, Address Resolution Protocol (ARP) poisoning [Gregg and Watkins, 2006], Man-in-the-middle (MITM) attacks, and Denial of Service (DoS) attacks [Gregg and Watkins, 2006] [Venkatramulu and Rao, 2013].

The literature proposes several solutions for ARP poisoning. In [Puangpronpitag and Masusai, 2009] a system called Dynamic ARP-spoof Protection Surveillance (DAPS) System is proposed to protect against both MITM and DoS attacks as well as a tool called ARPWATCH [Wikipedia, 2015a]. These mitigation solutions are provided to aid network administrators in monitoring ARP traffic. The developer need not be concerned with mitigation strategies at this layer, as they are out of the application development scope.

2.1.3 Security of Layer 3: The Network Layer

The network layer controls the operation of the subnet. It is concerned with routing packets from their source to the final destination. Here, IP addresses are used to identify nodes, and routing tables are used to identify overall paths and next-hops a packet might take [Reed, 2003]. The vulnerabilities at this layer include: IP attacks, routing attacks, MAC flooding, and ICMP attacks [Gregg and Watkins, 2006]. IP is a connectionless protocol which means that other protocols need to work in conjunction with IP to complete the transfer of packets. Protocols that work with IP at higher layers of the OSI model include ICMP, TCP, UDP, HTTP, HTTPS, SMTP, etc. The more prominent protocols utilized at every layer of the OSI model can be seen in Figure 2.1.

There have been several advances in the area of security at the Network Layer. The main contribution being IPSec. IPSec encrypts and authenticates every IP packet of a communication session [Firewall.cx, 2012]. IPSec VPNs operate at this layer, however, they need certain software to be present on end user devices which makes it harder to maintain. Currently, the preferred choice is it to connect with Secure Socket Layer VPNs which operate at the higher levels of the OSI model [Phifer, 2003].

From an application developer's perspective, attacks at this layer aimed at IP, are not a concern. Securing IPSec VPNs, and maintaining licenses on end user machines are the responsibility of IT departments that allow remote access through IPSec VPNs.

2.1.4 Security of Layer 4: The Transport Layer

The transport layer is responsible for ensuring that messages get delivered error-free, in sequence, and without loss or duplication [Microsoft, 2014]. The focus of the transport layer is on segments. It can either send data quickly or reliably [Gregg and Watkins, 2006].

The two main protocols that work at this layer are: UDP (a connectionless protocol) and TCP (a connection-oriented protocol) [Gregg and Watkins, 2006]. Some of the common vulnerabilities at this layer include: port scanning, DoS attacks, service enumeration and flag manipulation [Gregg and Watkins, 2006].

An example of abusing UDP to achieve a DDoS attack is UDP flooding. This type of attack floods random ports on a host with a large number of UDP packets. This would cause the host to continuously check for the application listening at the port. If no application is found, the host replies with an ICMP Destination Unreachable packet. This attack exhausts the host's resources and eventually deem the host unreachable; ultimately causing a DDoS attack [Incapsula, 2011]. Similarly, an example of abusing TCP to achieve a DDoS attack is SYN flooding. A SYN flooding attack exploits the TCP 3-way handshake mechanism. In a 3-way handshake, the client requests connection by sending a synchronize (SYN) message to the server. The server acknowledges this by sending a synchronize-acknowledge (SYN-ACK) message back. The client would then respond with an acknowledge (ACK) message. The result is an established connection [Incapsula, 2011]. The attacker leverages this process by never sending the ACK message back which results in the host constantly resending the SYN-ACK message on the assumption that previous SYN-ACK messages were damaged or lost. This consumes the host resources resulting in a DDoS [Gregg and Watkins, 2006].

Two major advances in securing the transport layer is the use of cryptography protocols in higher layers, namely, the Secure Socket Layer (SSL) and the Transport Layer Security (TLS). SSL is not an industry standard; it is a proprietary standard under the control of Netscape. TLS is an Internet Engineering Task Force (IETF) standard which is described in RFC 5246 [Dierks and Rescorla, 2008]. However, it should be noted that these protocols operate on top of TCP/IP. Therefore, they belong in layers 5-7 of the OSI model.

Securing the server against SYN flood attacks is the sole responsibility of the network administrator. Because attacks at this layer are aimed at servers and the network infrastructure, the application developer's only concern, if any, is to review and choose which server to deploy the application on (Apache, IIS, Nginx, etc.).

2.1.5 Security of Layer 5: The Session Layer

The session layer takes care of establishing, coordinating, managing, and terminating sessions between 2 applications on different computers. Protocols used at this layer include Structured Query Language (SQL), Remote Procedure Call (RPC), and Network File System (NFS) [Gregg and Watkins, 2006] [Geneiatakis et al., 2006]. Common vulnerabilities at this layer are session hijacking, DNS poisoning, and SSH Downgrade attacks [Pant and Khairnar, 2014] [Gregg and Watkins, 2006]. Session hijacking occurs when an attacker intercepts communication between two machines; the hijacking occurs after the 3-way handshake is completed. DNS poisoning is an attack whereby an adversary successfully diverts traffic from a legitimate server to a fake one [Hoffman, 2015]. A SSH downgrade attack happens when an attacker tricks a SSH server and client into negotiating a lower encryption protocol (SSH1) instead of (SSH2) [Squad, 2015].

Possible mitigation for DNS poisoning and SSH downgrade attacks are DNSSEC (an extension to DNS) [Arends et al., 2005] and Signalling Ciphersuite Value (SCSV) and the Renegotiation Information Extension (RIE) [Giesen et al., 2013]. Of the attacks mentioned above, session hijacking, in particular needs special attention from the developer. Most protocols that deal with it, however, work at the application layer. Therefore, it will be presented and discussed in the Application Layer section.

2.1.6 Security of Layer 6: The Presentation Layer

The purpose of this layer is to present and deliver data to the application layer. Protocol conversions, encryption/decryption of messages, compression/expansion of messages, and manipulation of XML objects all occur at this layer [Gregg and Watkins, 2006]. Common vulnerabilities at the presentation layer include: NetBIOS enumeration, clear text extraction, and protocol attacks [Gregg and Watkins, 2006]. The protocol attacks that can occur at the presentation layer are attacks against NetBIOS and Server Message Block (SMB) protocols, both of these protocols facilitate resource sharing. NetBIOS relies on name resolution through local host files and DNS. While the NetBIOS enumeration attack was well known since the early years of 2000, legacy systems are still vulnerable to it [Gregg and Watkins, 2006]. Weak encryption techniques may also introduce different vulnerabilities at the presentation layer. SMB attacks are still conducted with the latest example being the attack on Sony Pictures Entertainment [Lennon, 2014].

The literature proposes several mechanisms for encryption which can be classified as public-key encryption methods (asymmetric), identity based encryption methods (symmetric), and certificate-less public key encryption [Dent, 2008]. From the perspective of the developer, any sensitive data that the application handles must be protected with encryption. Session data containing user credentials or financial data must be secured with a well known and tested encryption algorithm. The developer should choose from hashing algorithms such as SHA-2, PBKDF2, or Bcrypt for encryption purposes depending on the application's own requirements.

2.1.7 Security of Layer 7: The Application Layer

The application layer is the entry and exit point of information in the OSI model. It is the channel through which applications communicate. Application layer protocols were used long before security was considered an issue, and their focus is more on functionality rather than security [Gregg and Watkins, 2006]. The protocols that operate at the application layer include: File Transfer Protocol (FTP), Telnet, Hyper Text Transfer Protocol (HTTP), Post Office Protocol (POP3), and Internet Mail Access Protocol (IMAP4) [Gregg and Watkins, 2006].

Each protocol comes with its own set of security issues. The main security issue with FTP is that traffic is transmitted without encryption. This makes FTP vulnerable to sniffing attacks, FTP bounce attacks, and FTP brute force attack (whereby an FTP server's password is guessed by brute force) [Khandelwal, 2013].

As with FTP, Telnet also does not provide encryption, nor does it provide server authentication mechanisms [Wikipedia, 2015d] [Gregg and Watkins, 2006]. Therefore, Telnet suffers from the same vulnerabilities as FTP. Common attacks on Telnet include eavesdropping, sniffing, and telnet brute force attack [Popeskic, 2011]. Usage of Telnet is currently not recommended. As an alternative, the recommendation is to switch to Secure Shell (SSH) [Dye et al., 2007].

The HTTP protocol is utilized by web applications to transfer the files of web pages. Most attacks on HTTP target those web applications [Gregg and Watkins, 2006]. The main problem with HTTP is in the POST messages that upload information in plain text to the server. This means that these messages can be captured and read [Dye et al., 2007]. The types of attacks on web applications are discussed below in the OWASP Top Ten Vulnerabilities section.

POP3 is an internet standard protocol used to retrieve e-mail from a remote server to local e-mail clients [Wikipedia, 2015c]. The security problem with POP3 is that email messages are removed from the server and stored locally. Moreover, any attachments in the email are downloaded with the message [Butler, 2008] [Liquidweb, 2011].

IMAP, which is also an email protocol, differs from POP3 in that messages are stored on the server only. Mail servers IMAP/SMTP are vulnerable to what is referred to as IMAP/SMTP injection. This vulnerability facilitates access to a mail server by bypassing the controls in the webmail application and directly accessing the server [OWASP, 2014].

As security has become more of a concern, the technologies are being re-inforced with mechanisms to enhance security. For example, Telnet was replaced with SSH, which provides encrytption of data transmitted between clients and servers. Similarly, the development of the Secure Socket Layer (SSL) provided encryption of HTTP traffic. Transport Layer Security (TLS) is a standard for encrypting client/server data developed by the Internet Engineering Task Force (IETF). SSL and TLS are currently being used interchangeably. The two encryption standards are used to secure HTTP traffic, FTP traffic, POP3, and IMAP, resulting in what is referred to as Hypertext Transfer Protocol Over SSL/TLS, File Transfer Protocol Secure, Secure POP3, and IMAP4 secure [Rescorla, 2000] [Ford-Hutchinson, 2005] [Microsoft, 2009b].

As an illustration of how complex secure software can be, even the security enhanced replacements for older technology have been known to contain security flaws. The latest versions of SSL and TLS have both been reported vulnerable to certain types of attacks such as the recent POODLE bug that allows Man-in-the-middle Attacks to successfully decipher messages [Möller et al., 2014]. OpenSSL also suffered recently from what is known as the Heartbleed vulnerability, which exploits missing bound checks in TLS heartbeat extension [Wikipedia, 2015b].

2.1.8 Summary

The discussion has thus far focused on network specific counter measures to resolve security issues at every layer of the OSI model. Information security can be examined from the perspective of network security, application security, user security, and systems security. In layers 1 through 4, most security controls and countermeasures focus on the network protocols and infrastructure. Securing the upper layers (Layers 5 through 7) however, requires more effort from the application developer. Secure coding practices are recommended during software development. Figure 2.2 shows that secure coding principles and practices are applied in layers 5 through 7 of the OSI model.

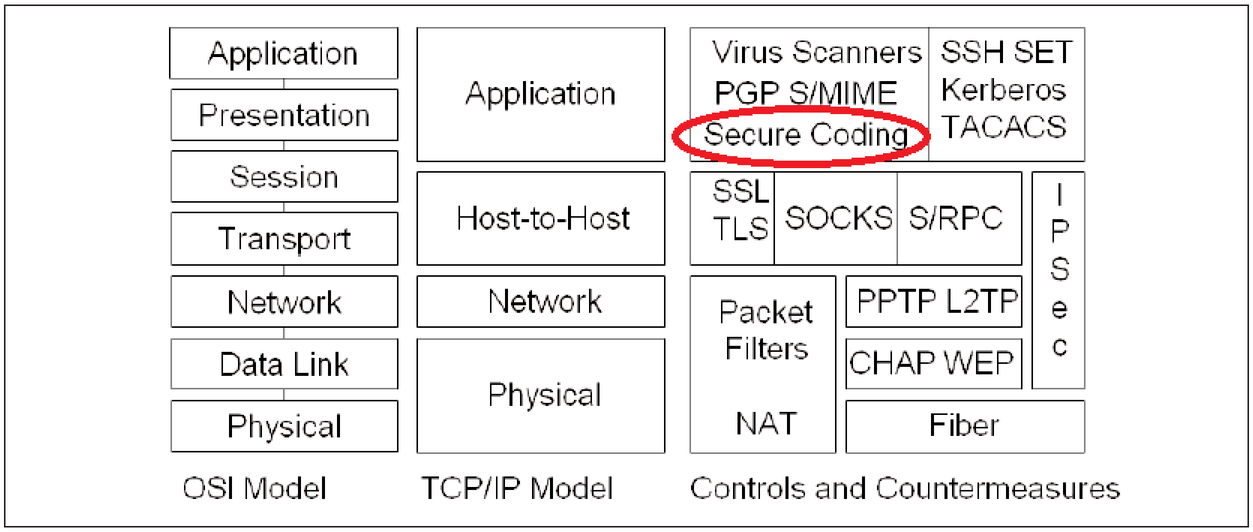


Figure 2.2: Common countermeasures in the OSI & TCP/IP Models. [Gregg and Watkins, 2006]

2.2 Secure Coding Practices and Common Web Vulnerabilities

2.2.1 Introduction

Practices in secure coding can help mitigate common vulnerabilities found in the application layer. Secure coding refers to the practices and processes employed by application developers to avoid introducing security flaws in the production code. There are a number of secure coding guidelines published for different application types and platforms such as Apple [Apple, 2014], Oracle [Oracle, 2014], and Microsoft's .NET framework [Microsoft, 2012]. The Computer Emergency Readiness Team (CERT) has published a general top ten secure coding practices list [CERT, 2011], as did the Open Web Application Security Project (OWASP) [OWASP, 2010]. Since the focus of this research is on securing web applications, OWASP's secure coding guidelines are chosen. OWASP's guidelines are a better candidate for consideration because they are more inclusive and target web applications in specific.

2.2.2 The OWASP Secure Coding Practices

OWASP is a non profit worldwide organization focusing on security of web applications [OWASP, 2015]. In [OWASP, 2010], OWASP describes a checklist of practices to be followed by developers in order to produce secure software that ensures Confidentiality, Integrity, and Availability of information resources. The secure coding practices checklist includes extensive recommendations in several categories. Below are sample recommendations for each category:

1. Input Validation: Input data must be validated against the source type (whether it originated from a trusted or untrusted source). Input data must be encoded into a common character set, an act known as canonicalization, before validation takes place. Furthermore, expected data type, range, and length checking must be done. If special characters such as > < " ' % ( ) & + \ \' \" are allowed in the input, a proper escaping mechanism should be employed.

2. Output Encoding: Output encoding refers to the act of escaping/encoding data for appropriate context in which it will be displayed. For example, in html, a >character would be replaced by its html number (&#62) or html name (&gt). All encoding must take place on a trusted system. Also, data must be sanitized to avoid Cross Site Scripting attacks and SQL injection attacks.

3. Authentication and Password Management: A standard, tested, authentication service must be utilized. Rate limiters on failed log-in attempts must be employed. HTTP's POST requests must be used when transmitting credentials. Before any critical operation is executed, users must be re-authenticated.

4. Session Management: Controls provided by the server or framework to manage sessions must be used. Session ids must be generated with any re-authentication step. URLs must not expose session ids. Moreover, when using TLS, the secure attribute for cookies must be set.

5. Access Control: Access to protected URLs, protected functions, application data, and services should be restricted to authorized users only. Account auditing and disabling of unused accounts is required.

6. Cryptographic Practices: Any cryptographic functions used by the application must be implemented on a trusted server. There should also be a policy declared for managing cryptographic keys.

7. Error Handling and Logging: Error handling should be gracefully done. An example would be replacing generic error messages with custom ones without revealing any sensitive information in them. A log event data must be maintained and it should include information for both successful and failed security controls.

8. Data Protection: Least privilege rule must be implemented. Encryption of sensitive stored information and removal of comments in production code accessible to users are also recommended. Furthermore, sensitive information must not be included in HTTP GET request parameters.

9. Communication Security: Encryption must be implemented for all sensitive information being transmitted. Secure TLS connections must be used during the transmission process.

10. System Configuration: Servers, frameworks, and system components should be running the latest versions. Any existing patches would need to be applied. Safe exception handling must be implemented.

11. Database Security: Variables must be strongly typed. Input validation and output encoding should be carried out. Strongly typed parameterized queries must be used.

12. File Management: Authentication needed before uploading files. Types of uploaded files should be limited. The absolute file path must never be revealed.

13. Memory Management: Buffer size and boundary issues must be tested. Allocated memory should be freed once execution of functions is completed.

14. General Coding Practices: Approved managed code should be utilized for common tasks. User supplied data must not be passed to dynamic execution functions. All variables and data stores should be explicitly initialized during declaration or prior to first usage.

2.2.3 The OWASP Top Ten Web Vulnerabilities

One of the main contributions OWASP offers is a list of top ten web applications vulnerabilities compiled every 3 years. The vulnerabilities presented in the 2013 OWASP top ten list of web application vulnerabilities [OWASP, 2013] are detailed as follows:

1. A1: Injection: Occurs when untrusted data is sent to an interpreter and executed as a command or query. Examples include SQL, OS, and LDAP injections. For example, a flawed query would be:

   ```
   String query = "SELECT * FROM accounts WHERE custID=’" + request.getParameter("id") + "’";
   ```

   An adversary can modify the id parameter in the browser to be ’ or ’1’=’1, and the executed query would return all records in the accounts table. Injection can be mitigated by not allowing untrusted data to execute directly in commands and queries.

2. A2: Broken Authentication and Session Management: Occurs when application functions utilize improper implementations of authentication and session management. An example of improper implementation would be exposing session ids in URLS. An example of an attack that exploits this vulnerability is session hijacking (session fixation) Mitigating attacks of this type requires protecting stored authentication credentials with hashing or encryption. Placing stronger controls in account management functions (e.g. password recovery). Employing session time-outs and ensuring that transmission of credentials happens over TLS.

3. A3: Cross Site Scripting (XSS): Occurs when unsanitized user input is sent to the interpreter in the browser which treats it as active content. [OWASP, 2013], gives the following example:

   ```
   (String) page += "<input name='creditcard' type='TEXT value='" + request.getParameter("CC") + "'>";
   ```

   This code snippet shows that user supplied input is being used in the value attribute. An adversary can modify the CC parameter to be:

   ```
   '><script>document.location= 'http://www.attacker.com/cgi-bin/cookie.cgi? foo='+document.cookie</script>'
   ```

   This would allow the adversary to steal the session id and hijack this user's session.

   To prevent XSS attacks, untrusted data must be escaped.

4. A4: Insecure Direct Object References: Occurs when an adversary successfully changes a parameter's value that directly refers to a system object to another object that he/she is not authorized for. [OWASP, 2013] uses the following example to illustrate this concept:

   ```
   String query = "SELECT * FROM accts WHERE account = ?";
   PreparedStatement pstmt = connection.prepareStatement(query ,  );
   pstmt.setString( 1, request.getParameter("acct"));
   ResultSet results = pstmt.executeQuery( );

   http://example.com/app/accountInfo?acct=notmyacct
   ```

   The code above (third line) shows an adversary can possibly modify the acct parameter in his/her browser and send any account number accessing account information in which he/she is not authorized to view. Preventing such attacks requires the utilization per user or per session *indirect* object references. For instance, instead of using a database key directly; a mapping scheme for authorized resources can be used.

5. A5: Security Misconfiguration: Can occur at any level of the application stack (platform, server(s), database, framework, and any custom code used). Attackers can exploit this vulnerability and gain unauthorized access to unprotected files, unused pages, and directories etc. Mitigating this vulnerability requires joint efforts from developers and system administrators. To protect against an attack of this type, updates and patches need to be applied regularly.

6. A6: Sensitive Data Exposure: Sensitive data such as credentials should never be exposed. Data encryption must be employed for data both at rest and in transit. If applicable, unnecessary sensitive data should be discarded and not saved. Protecting against sensitive data exposure includes encrypting all sensitive data and using strong validated cryptographic algorithms.

7. A7: Missing Function Level Access Control: Occurs when an adversary is able to change a URL or a parameter to a privileged function and is granted access to this function. An attacker exploiting this vulnerability can gain access to administrative functions. Not displaying links and buttons to unauthorized functions in web-pages is not sufficient protection. Mitigating this vulnerability requires implementing checks in the controller.

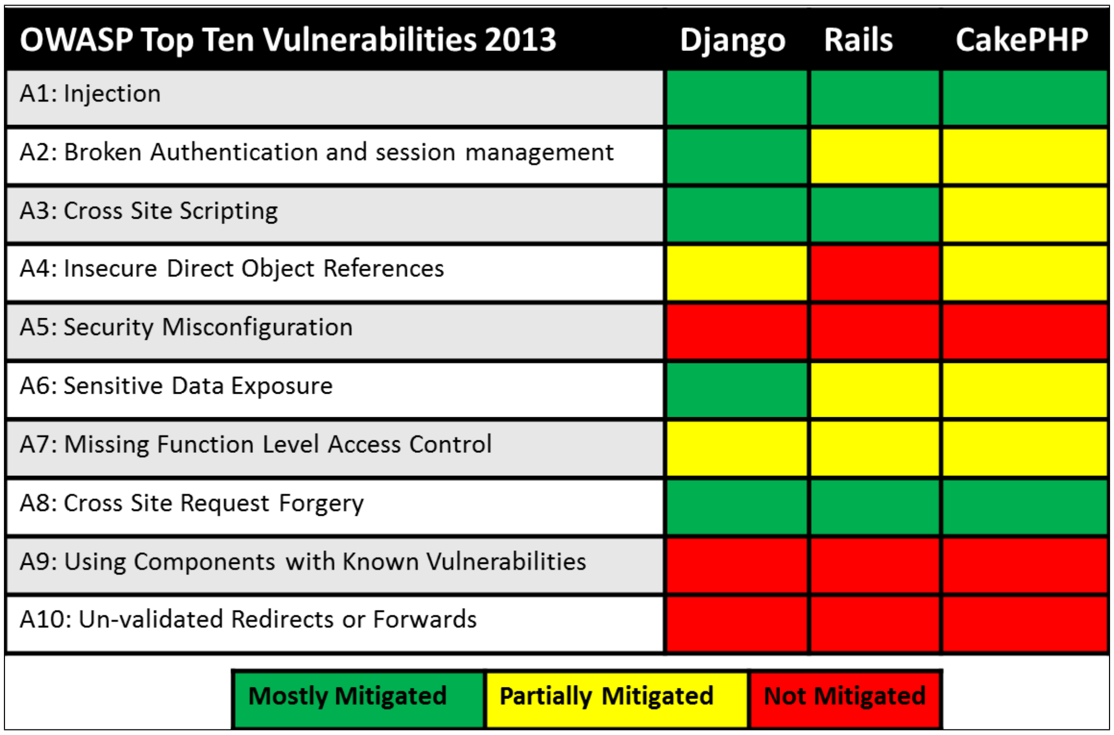
8. A8: Cross Site Request Forgery (CSRF): Occurs when an authenticated user is tricked into executing malicious requests. Since, the user is authenticated, the server will not be able to differentiate between forged and legitimate requests. Preventing CSRF attacks requires the presence of an unpredictable token preferably in a hidden field within the body of the request.

9. A9: Using Components with Known Vulnerabilities: Occurs when an application utilizes libraries, components, or third party applications without checking for existing vulnerabilities in them. Moreover, component dependability makes this vulnerability hard to discover. Preventing this vulnerability requires that all components, libraries, and third party applications used be identified with version numbers. Public vulnerability databases such as the Common Vulnerabilities and Exposures (CVE) and National Vulnerability Database (NVD) must be constantly checked for reported vulnerabilities in these libraries.

10. A10: Unvalidated Redirects and Forwards: Occurs when an attacker takes advantage of a redirect specified in an unvalidated parameter to redirect a legitimate user to a

malicious site (e.g. phishing). Internal application forwards that route requests to various parts of an application could also be targeted in a similar manner. An attacker can utilize a parameter that forwards a user to a page within the application and change the value to admin pages for example, bypassing access control checks.

Modern web development frameworks are able to handle, to a certain degree, the mitigation of some of the vulnerabilities in OWASP's top ten list. For example, Ruby on Rails [Rails, 2014], CakePHP [CakePHP, 2015], and Django [Django, 2015], all offer similar protection levels for common attacks such as SQL injection, Cross Site Scripting (XSS), and Cross Site Request Forgery (CSRF). Figure 2.3 below shows how common open source web development frameworks compare in handling the OWASP top ten vulnerabilities of 2013.



OWASP Top Ten Vulnerabilities 2013	Django	Rails	CakePHP
A1: Injection	🟩	🟩	🟩
A2: Broken Authentication and session management	🟩	🟨	🟨
A3: Cross Site Scripting	🟩	🟩	🟨
A4: Insecure Direct Object References	🟨	🟥	🟨
A5: Security Misconfiguration	🟥	🟥	🟥
A6: Sensitive Data Exposure	🟩	🟨	🟨
A7: Missing Function Level Access Control	🟨	🟨	🟨
A8: Cross Site Request Forgery	🟩	🟩	🟩
A9: Using Components with Known Vulnerabilities	🟥	🟥	🟥
A10: Un-validated Redirects or Forwards	🟥	🟥	🟥

🟩 Mostly Mitigated 🟨 Partially Mitigated 🟥 Not Mitigated

Figure 2.3: Django, Rails, CakePHP, and the OWASP Top Ten 2013 Vulnerabilities

The popularity of web development frameworks depends on several factors. For a developer, the language the framework is built on plays a large role in deciding whether or not to use that framework. Choosing a framework built with a language the developer is comfortable with minimizes the learning curve. An open source well-documented framework with a large community of followers helps the developer resolve any issues that might arise during development. Finally, the choice of a web development framework to use really depends on what a project is trying to achieve. Since Django had the highest rank with 5 of the 10 OWASP vulnerabilities ranked as mostly mitigated, it will be used as the web development framework of choice for implementing Web Application Flow Whitelisting.

Chapter 3
Web Application Flow Whitelisting

3.1 Background and Problem Domain

The increased usage of the Internet has mandated the presence of web-facing applications for many organizations. Following today's nimble development trend, web application developers employ web development frameworks to take advantage of pre-written common features found across many web applications. Using frameworks, thus, allows developers to focus on the unique requirements, functionalities, and security considerations particular to the application under development.

There is a need for agile practices to address security issues in web applications. While web development frameworks handle some of the common web application vulnerabilities such as SQL injection and cross site scripting, they can not handle application specific vulnerabilities; those vulnerabilities must be addressed by the developer. As hackers become more sophisticated in exploiting vulnerabilities, developers need to build application specific security into their web applications.

Web application flow whitelisting provides the developer with a way to make their applications more secure by detecting bad behavior or malicious use, rejecting that behavior, and redirecting the user to a safe view.

3.2 Defining Web Application Flow Whitelisting

The disparity between intended behavior and actual behavior of a web application may be indicative of malicious use or implementation flaws. This research focuses on OWASP's 4^{th} and 7^{th} web vulnerabilities; Insecure Direct Object References and Missing Function

Level Access Control. The objective of this research is to build security into the design of a web application by predetermining its allowed flow. This needs to be done in a lightweight, intuitive manner. When the developer designs the intended flow of a web application and the interactions between the different components within it, he can monitor the actual behavior against a list of allowed interactions. Any sequence of flow not specified in that list would be rejected by the application's logic and the application would then resort to a safe state. The term *interactions* here encompasses user permissions, application flow, and, in MVC terminology, what data each controller has access to. The term *flow* refers to the series and order of navigational steps taken to complete a certain task on a web application.

This research defines, creates, and enforces a list of allowed interactions within a web application's flow. We refer to it as 'web application flow whitelisting'. A 'whitelist' is a list of allowed interactions. It dictates the flow of a web application and which HTTP request/response exchanges occur within it. Defining the elements that comprise a whitelist should be done during the design phase of development

Ideally, a 'whitelist' would be created dynamically by the web application through the use of a behavior monitoring tool. However, for the purpose of proving whether or not a web application's security improves with whitelisting, a statically created 'whitelist' is sufficient. The whitelist resides within a web application framework's middleware to intercept HTTP requests and render the allowed response. Every time a transition between one view and the next occurs, the HTTP request is checked against the 'whitelist' before any response is rendered. If a transition is not allowed by the whitelist, the flow redirects the user to a safe view.

3.2.1 Formal Definition of the Whitelist

The whitelist is defined as a tuple $\langle C, D, W, S \rangle$, where:

- **C** is a set $\{u, c_1, c_2, .., c_n\}$, where $c_1, c_2, .., c_n$ are components within the system boundary and u represents a component outside the system boundary.

- **D** is a set $\{d_1, d_2, .., d_n\}$, where $d_1, d_2, .., d_n$ are conditions for transitions that occur within the application. Every cell in a matrix of size $|C|\, X\, |C|$ contains a distinct subset x, $\{x : x \subseteq D\}$. If the evaluation of the conditions in x returns TRUE, then that ordered pair for the transition from c_{origin} to $c_{destination}$ is added to W.

- **W** is a set of all ordered pairs $\{(c_o, c_d) : c_o, c_d \in C\}$ each pair represents an *allowed transition* from an origin component c_o to a destination component c_d and,

- **S** is a matrix of size $|C|\, X\, |C|$ $S_{c_o c_d} = c_s$ specifies a *safe component* $\{c_s : c_s \in C\}$ when c_d can not follow c_o

A transition between one component to another is dictated by a Transition Function whereby a transition from c_{origin} to $c_{destination}$ occurs if and only if $(c_o, c_d) \in W$, else the transition function is called on $(c_o, S_{c_o c_d})$.

$$T(c_o, c_d) = \begin{cases} c_d, & if (c_o, c_d) \in W \\ & otherwise, \\ T(c_o, S_{c_o c_d}) & \end{cases}$$

There are several operations that are performed on the whitelist. The operations are divided into two categories according to when they take place. The operations that are carried out during development are:

- Create (c_o, c_d) in W: adds an ordered pair to the relation.

- Delete (c_o, c_d) from W: removes an ordered pair from the relation.

- Enter $\langle d_x \rangle$ into D: adds a condition d_x to the set D.

- Delete $\langle d_x \rangle$ from D: removes the condition d_x from the set D.

- Add $\langle d_x \rangle$ to subset x in $W_{c_o c_d}$.

- Remove $\langle d_x \rangle$ from subset x in $W_{c_o c_d}$.

- Enter $\langle c_s \rangle$ into $S_{c_o c_d}$.

- Update $\langle c_s \rangle$ in $S_{c_o c_d}$.

The operations carried out at runtime are:

- Compute $T\left(c_o, c_d\right)$

- Verify $c_o \rightarrow c_d$:

  c_d can follow c_o **IFF** all conditions belonging to subset x in $W_{c_o c_d}$ evaluate to TRUE.

  Else the transition is to c_s

The set of all components (C) and the relation W are represented in a zero-one matrix, where 1 signifies an allowed transition and 0 means that the transition is disallowed. Each cell in the matrix will have a zero or 1 value based on the evaluation of a subset of conditions. Matrix (S) would contain a safe component to transition to in case the transition from c_o to c_d fails. Representations of W and S are shown in Tables 3.1, 3.2 and 3.3 respectively.

	u	c_1	c_2	c_3	c_4	c_5
u	NA	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }
c_1	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }
c_2	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }
c_3	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }
c_4	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }
c_5	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }	distinct subset x $\{x : x \subseteq D\}$ }

Table 3.1: Representation of W with every cell containing a subset of conditions - NA - represents Not Applicable and a 1 is placed for transitions occurring outside the system boundary

	u	c_1	c_2	c_3	c_4	c_5
u	1*	{0,1}	{0,1}	{0,1}	{0,1}	{0,1}
c_1	{0,1}	{0,1}	{0,1}	{0,1}	{0,1}	{0,1}
c_2	{0,1}	{0,1}	{0,1}	{0,1}	{0,1}	{0,1}
c_3	{0,1}	{0,1}	{0,1}	{0,1}	{0,1}	{0,1}
c_4	{0,1}	{0,1}	{0,1}	{0,1}	{0,1}	{0,1}
c_5	{0,1}	{0,1}	{0,1}	{0,1}	{0,1}	{0,1}

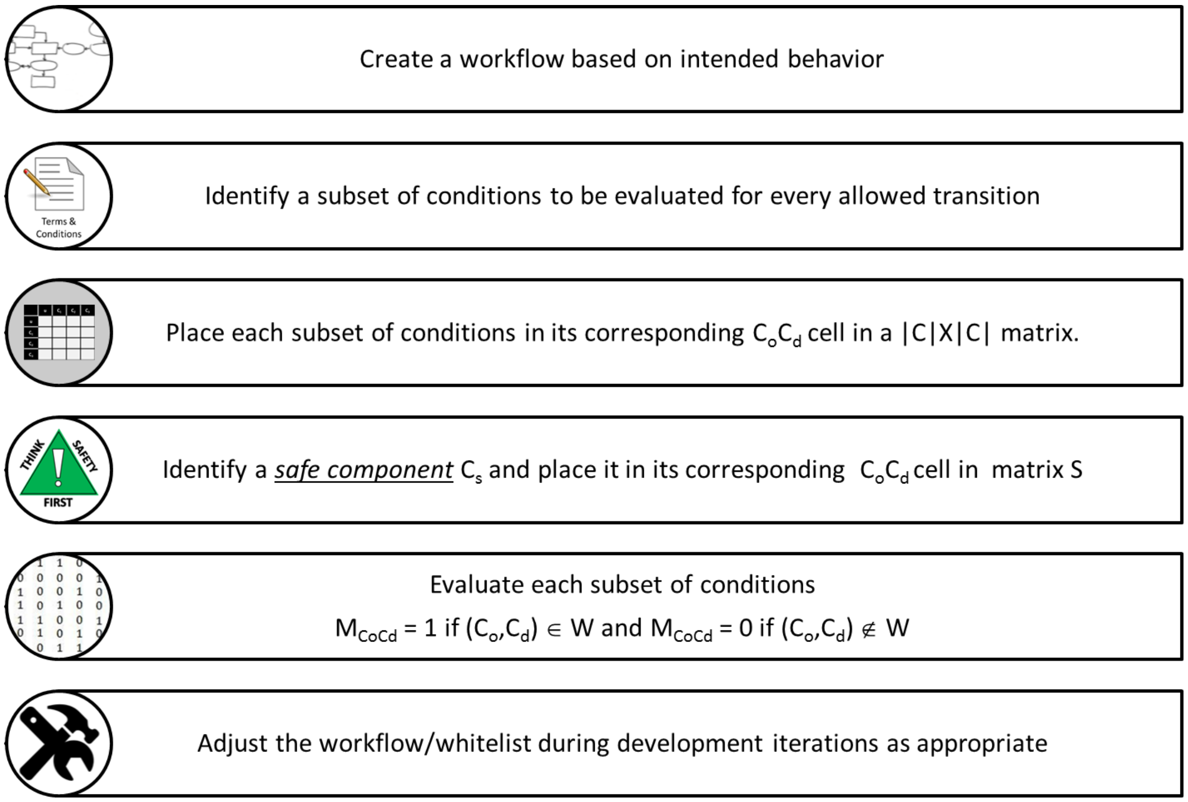
Table 3.2: Representation of W with each cell containing a 0 or 1 from the resulting evaluation of a subset of conditions - For the sake of completion, a 1 is placed for transitions occurring outside the system boundary

	u	c_1	c_2	c_3	c_4	c_5
u	NA	c_s	c_s	c_s	c_s	c_c
c_1	c_s	c_s	c_s	c_s	c_s	c_s
c_2	c_s	c_s	c_s	c_s	c_s	c_s
c_3	c_s	c_s	c_s	c_s	c_s	c_s
c_4	c_s	c_s	c_s	c_s	c_s	c_s
c_5	c_s	c_s	c_s	c_s	c_s	c_s

Table 3.3: Representation of Matrix S - NA - represents Not Applicable and a 1 is placed for transitions occurring outside the system boundary

3.2.2 Building the Whitelist

In order for a developer to build the whitelist, he/she must start with identifying the web application's intended behavior by creating a workflow that illustrates exactly how the application is intended to behave. The workflow should contain all components of an application and any intended transition that occurs from one component to another. Any internal components (subroutines) must also be included in the workflow. Once the workflow is completed, the developer should examine every transition within the workflow and identify a subset of conditions that must be evaluated in order for the transition to succeed. The developer would then place the subset(s) of conditions in the corresponding c_oc_d cell in matrix W. The developer must also identify safe components to redirect to in case any evaluation of the subset(s) of conditions fail. The safe components are placed in the corresponding c_oc_d cell in matrix S. From the initial workflow and the evaluation results of the subset(s) of conditions, the developer extracts a zero-one representation of W. For the sake of simplicity, let's call this representation M, where $|C| \; X \; |C| = M$ with $M_{c_oc_d} = 1$ if $(c_o, c_d) \in W$ and $M_{c_oc_d} = 0$ if $(c_o, c_d) \notin W$. The workflow and/or whitelist may be adjusted during development iterations as deemed appropriate. It should be noted that the conditions in set D should be simple conditions rather than compound. Figure 3.1 below provides a summary of the required steps in building a whitelist.



Create a workflow based on intended behavior
Terms & Conditions
Identify a subset of conditions to be evaluated for every allowed transition
Place each subset of conditions in its corresponding C_oC_d cell in a |C|X|C| matrix.
THINK SAFETY FIRST
Identify a *safe component* C_s and place it in its corresponding C_oC_d cell in matrix S
Evaluate each subset of conditions
$M_{CoCd} = 1$ if $(C_o,C_d) \in W$ and $M_{CoCd} = 0$ if $(C_o,C_d) \notin W$
Adjust the workflow/whitelist during development iterations as appropriate

Figure 3.1: Steps for building a whitelist

3.2.3 A Web Application Flow Whitelisting Example

To further clarify the concepts above, a simple part of a generic web application will be used as an example. Suppose you have a web application that requires a user to be authenticated before being able to use the application. The application allows for 3 attempts at login. If the user fails 3 consecutive login attempts, the application will lock the user out. Once the user is authenticated, the application redirects to a personalized user portal view. Within the user portal, there is a view to edit the user profile and another view that allows the user to contact other users within the application. The user is allowed to logout at any time from the user portal view, the edit profile view, and the contact other users view. A workflow for this example is illustrated in figure 3.2.

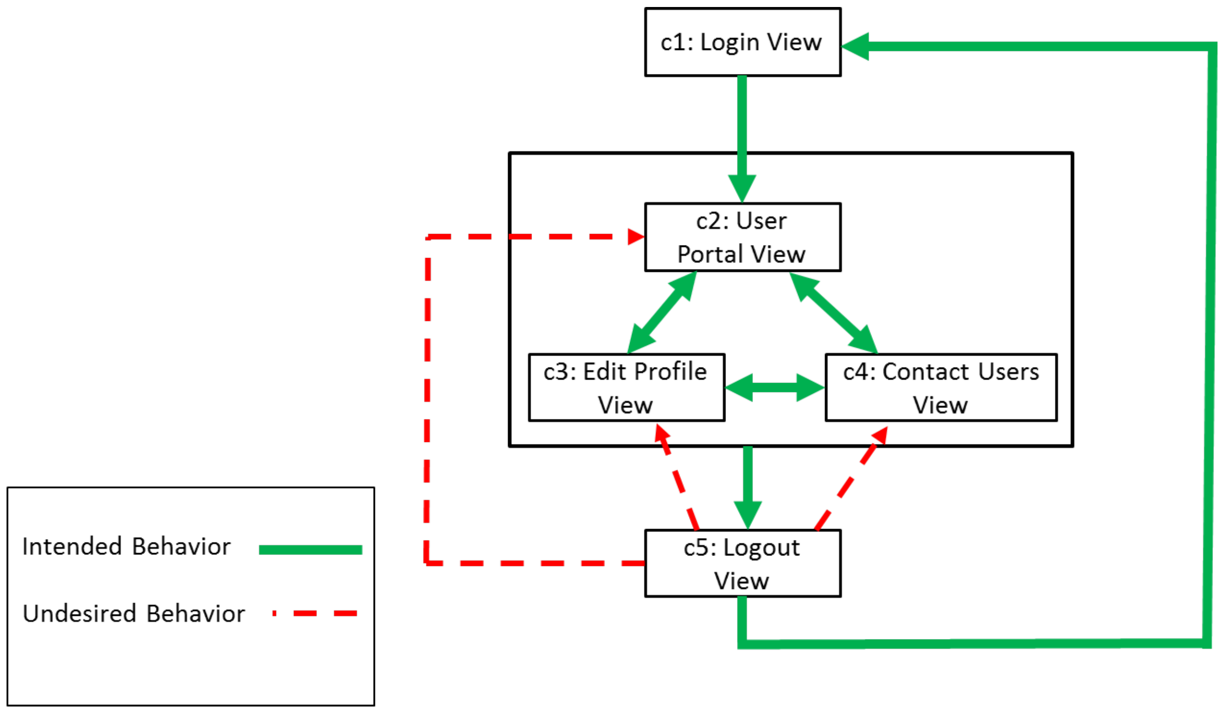


Figure 3.2: Workflow of a generic web application

The workflow above consists of 5 components. Set C will therefore be: $C = \{u, c_1, c_2, c_3, c_4, c_5\}$. Recall that u represents a component outside the system boundary and is included into set C for the sake of completion. As for the global set of conditions D, let's assume it contains the following conditions:

- d_1 : user is anonymous.
- d_2 : user is authenticated.
- d_3 : session expiry time is valid.
- d_4 : previous view
- d_5 : subsequent view

- d_6 : login attempts ≤ 3.

The whitelist would contain a subset of D within each cell of the matrix. Just as an example, the whitelist below shows the subset for an allowed transition from c_1 to c_2 and another subset that resulted in a disallowed transition from c_5 to c_4. For an allowed transition from c_1 to c_2, the subset of conditions is, $x = \{d_2, d_3, d_4 = \text{login view}, d_5 = \text{user portal view}, d6\}$. All the conditions in x must evaluate to TRUE. For a disallowed transition from c_5 to c_4, the subset of conditions to be evaluated is, $x = \{d_2, d_3, d_4 = \text{edit profile view or user portal view}, d6\}$. Clearly, transitioning from c_5 to c_4 will not be allowed as the first condition in the subset (d_2) is not met once the user has logged out. Table 3.4 below shows a partially completed matrix W.

	u	c_1	c_2	c_3	c_4	c_5
u	NA	distinct subset x	distinct subset x	distinct subset x	distinct subset x	distinct subset x
c_1	distinct subset x	distinct subset x	x = {d2,d3,d4 = login, d5 = user portal, d6}	distinct subset x	distinct subset x	distinct subset x
c_2	distinct subset x	distinct subset x	distinct subset x	distinct subset x	distinct subset x	distinct subset x
c_3	distinct subset x	distinct subset x	distinct subset x	distinct subset x	distinct subset x	distinct subset x
c_4	distinct subset x	distinct subset x	distinct subset x	distinct subset x	distinct subset x	distinct subset x
c_5	distinct subset x	distinct subset x	distinct subset x	distinct subset x	x = {d2,d3, d4 = user portal or edit profile, d6}	distinct subset x

Table 3.4: A partially completed matrix W showing only subsets of transitions c_1 to c_2 and c_5 to c_4 - NA - represents Not Applicable and a 1 is placed for transitions occurring outside the system boundary

From the workflow and conditions, the set of ordered pairs in W are:

$$W = \{\,(u,u)\,,(u,c_1)\,,(c_1,c_1)\,,(c_1,c_2)\,,(c_2,c_2)\,,(c_2,c_3)\,,(c_2,c_4)\,,(c_2,c_5)\,,$$
$$(c_3,c_2)\,,(c_3,c_3)\,,(c_3,c_4)\,,(c_3,c_5)\,,(c_4,c_2)\,,(c_4,c_3)\,,(c_4,c_4)\,,(c_4,c_5)\,,(c_5,c_1)\,\}$$

A zero-one representation of W can be used after all subsets of conditions have been evaluated. Table 3.5 shows a depiction of W as a zero-ones matrix with allowed flow represented by 1 and disallowed flow by 0.

	u	c_1	c_2	c_3	c_4	c_5
u	1*	1	0	0	0	0
c_1	0	1	1	0	0	0
c_2	0	0	1	1	1	1
c_3	0	0	1	1	1	1
c_4	0	0	1	1	1	1
c_5	0	1	0	0	0	0

Table 3.5: A zero-one representation of W based on workflow and the evaluation of conditions

The next step would be to populate matrix S with 'safe' components to redirect to in case the evaluation of conditions fails and the transition is disallowed. Table 3.6 shows matrix S for the example above.

	u	c_1	c_2	c_3	c_4	c_5
u	NA	c_1	c_1	c_1	c_1	c_1
c_1	c_5	c_1	c_1	c_1	c_1	c_1
c_2	c_5	c_5	c_5	c_1	c_1	c_5
c_3	c_5	c_5	c_5	c_5	c_5	c_5
c_4	c_5	c_5	c_5	c_5	c_5	c_5
c_5	c_5	c_5	c_5	c_5	c_5	c_5

Table 3.6: Matrix S containing safe components to redirect to if a transition is disallowed or the conditions fail

3.2.4 Perceived Benefits of Web Application Flow Whitelisting

From the theoretical description and definition of web application flow whitelisting above, we perceive the approach to have several benefits. Whitelisting flow ensures that the web application is in compliance with its intended behavior. It also accounts for unintended behavior by redirecting the flow of the application to a safe component. As is the case with whitelists in general, it is easier to comprise a whitelist of interactions between an application's components and check against it rather than allow all and any possible interactions. Ranum [Schneier and Ranum, 2011] states that a whitelist's effectiveness depends on the ability of the person who created it to assess what should be on it. For web application flow whitelisting, the best person to create the whitelist and maintain it is the application's developer. Finally, the simplicity of whitelisting flow makes it possible to adapt it into any existing software process or as a stand-alone security strengthening practice.

Chapter 4
Research Methodology:
Phase1: Small Case Validation

4.1 Assumptions

The approach of whitelisting an application flow is based on the following assumptions:

- The underlying levels of the OSI model are secure.
- The web development framework mitigates common web vulnerabilities and is secure.
- The web application is analyzed in terms of desired flow and behavior.

Given the assumptions stated above, the choice of the web development framework to be used becomes vital. Therefore, part of this chapter is dedicated to discuss Django as the web development framework of choice for validating web application flow whitelisting. Further information on security in Django can be found in appendix A.

4.2 Background

Web sites today are no longer the collection of static pages they were twenty five years ago. Most have turned into web applications with dynamic content offering users a vast array of services. Developers nowadays turn to web development frameworks when constructing web applications because the frameworks offer ready-to-use common features that are found across many web applications. The frameworks also offer protection from common security vulnerabilities, however, they do not mitigate application-specific vulnerabilities.

Django is a web development framework that is built on the the Model-View-Controller (MVC) design pattern [Holovaty and Kaplan-Moss, 2009]. The MVC pattern separates the user interface functionality from the application's functionality without compromising the application's ability to respond to user input. The three components of the MVC pattern are the model, where the application's data resides; the view which displays some of the data and receives user input; and the controller which handles the application's functionality [Bass et al., 2012]. In Django terms, the MVC pattern translates to Model-Template-View (MTV). The template component corresponds to the view in MVC and the view in Django is "the Python callback function for a particular URL, because that callback function describes which data is presented" [Django, 2016]. The controller in Django is essentially the framework itself which processes a user request and returns a response according to the URL configuration [Django, 2016]. The Django framework is a 'batteries included' framework which means that it provides the common functionality required for building web applications without the need to resort to separate libraries or packages [Makai, 2016]. Django is comprised of several main components [Django, 2016] [Makai, 2016] [Wikipedia, 2016b]:

- Lightweight web server for development and testing.
- Template system that can be used to create HTML content on pages.
- System to handle receiving, preparing and processing HTML forms.
- Caching framework that works with different types of caching methods.
- Series of built-in middleware classes (as well as the ability to add custom written ones) that are invoked during the request-response processing cycle.
- Clean URL scheme where URLs are written as simple regular expressions and mapped to Python callback functions.
- Object-relational mapper.
- Built-in commands for database schema migrations.
- Built-in admin interface.

- Support for internationalization.

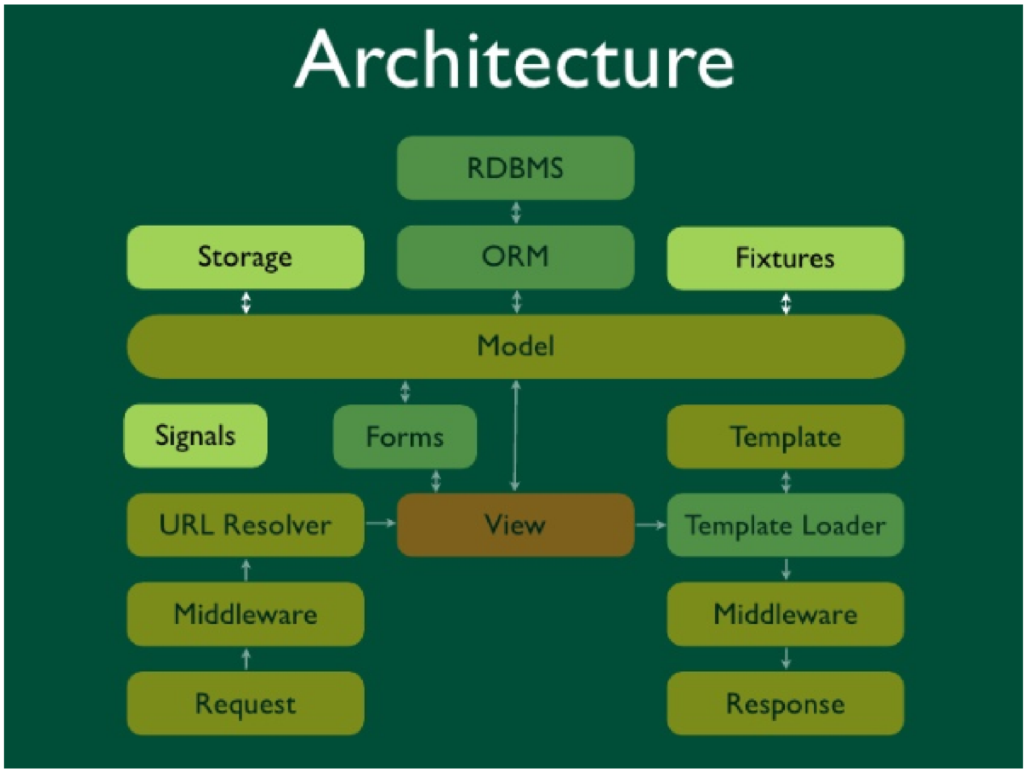


Figure 4.1: Overview of the Django framework [Yates, 2009]

4.2.1 The Request-Response Cycle in Django

Django's middleware classes are a set of hooks that are plugged into the request-response life cycle. Each class in the built-in middleware classes provides certain functionality. For instance, the SecurityMiddleware is used to enhance the application's security, the Session-Middleware provides support for sessions, and the AuthenticationMiddleware adds a user attribute representing the currently logged in user to every HTTP request [Django, 2016]. Django also allows developers to write their own custom middleware classes. This feature in Django facilitates the implementation of the whitelist. The basic structure of a Django project is as follows:

- mysite/
    - manage.py
    - mysite/

    - __init__.py
    - settings.py
    - urls.py
    - wsgi.py
  - myapp/
    - __init__.py
    - admin.py
    - migrations/
      - __init__.py
    - models.py
    - tests.py
    - views.py

The settings.py file contains the global settings of a Django project which includes database configuration, middleware classes, and other application-specific settings. Custom-written middleware needs to be declared within the MIDDLEWARE_CLASSES attribute in settings.py.

4.3 Writing Custom Middleware Class for Web Application Flow Whitelisting

The order in which the middleware classes are declared in settings.py is significant because Django applies the middleware in that order. Figure 4.2 below shows the order of middleware classes as well as the hooks for the request-response cycle.

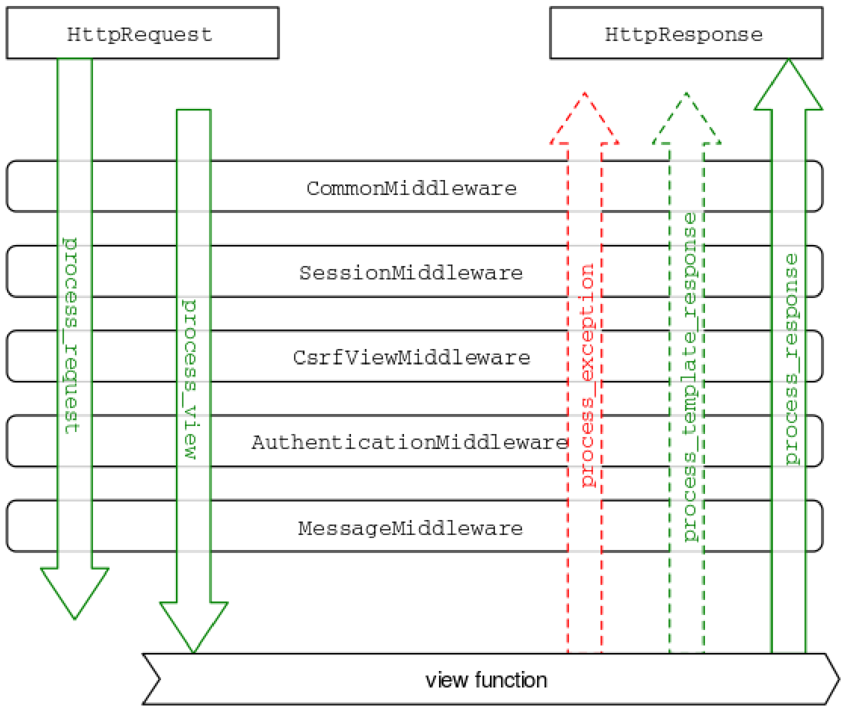


Figure 4.2: Middleware processing order [Django, 2016]

There are two hooks for the request phase of the cycle (process_request() and process_view()) and three for the response phase (process_exception(), process_template_response(), and process_response()). The functions of the hooks provided by Django are as follows [Django, 2016] [Mele, 2015]:

- process_request(request): takes an HttpRequest object (request) and is called on each request before Django decides which view to execute.

- process_view(request, view_func, view_args, view_kwargs): is called just before a view executes. It has access to view_func which is the Python function Django is about to use as well as the arguments it receives.

- process_exception(request, exception): is called only if a view throws an exception, where (request) is an HttpRequest object and (exception) is an Exception object raised by the view function.

- process_template_response(request, response): is called once a view completes execution and only if the response instance has a render() method which indicates that it is a TemplateResponse object.
- proces_response(request, response): takes an HttpRequest object (request) and HttpResponse or StreamingHttpResponse object (response). It is called on all responses before they are returned to the browser.

Whitelisting flow can be implemented in Django using custom written middleware. The purpose of the whitelist middleware is to intercept an incoming request, inspect its compliance (or lack thereof) with intended behavior, and invoke the corresponding response. The whitelist middleware customizes the process_request() and process_view() hooks provided by Django's request/response cycle. The psuedo code in whitelist middleware for process_request() is:

1. State any URLs that must be excluded from whitelisting (such as admin site URLs)
2. Set flag Notallowed to True (Initially all behavior is disallowed)
3. Check for allowed behavior by:
   (a) Get the HTTP_REFERER attribute.
   (b) If flow from HTTP_REFERER to a requested URL is allowed (the evaluation of the subset of conditions for that particular transition resulted in 1), then set flag Notallowed to False.
   (c) Return None for the allowed flow to continue to process_view() function.

The psuedo code for process_view() is:

1. If Notallowed is True, then look for the safe view for that particular transitions
2. Return the safe view.

3. Else, Return None (for the allowed transition to continue the request/response cycle and render a response).

Since the processing order of middleware is of utmost significance, the custom whitelist middleware should be declared at the end of the MIDDLEWARE_CLASSES attribute so that it is the last middleware called when a request is being processed, and the first middleware called when a response is being rendered. Furthermore, custom middleware is placed in a folder named 'middleware' and should be in the same folder that contains settings.py. Custom written middleware fits within the structure of a Django project in the following way:

- mysite/
    - manage.py
    - mysite/
        - middleware/
            - __init__.py
            - mycustommiddleware.py
        - __init__.py
        - settings.py
        - urls.py
        - wsgi.py
    - myapp/
        - __init__.py
        - admin.py
        - migrations/
            - __init__.py
        - models.py
        - tests.py
        - views.py

4.4 Hypothesis

Based on the description of web application flow whitelisting and how to implement it on apps written in Django, we derived the following hypothesis:

- H_0: Django applications that have a whitelisted flow show a greater than or equal to number of security vulnerabilities than Django applications that have not had their flow whitelisted.

- H_1: Django applications that have a whitelisted flow show a lesser number of security vulnerabilities than Django applications that have not had their flow whitelisted.

The aim of this research is to reject the null hypothesis H_0 in favor of the alternative hypothesis H_1.

4.5 Validation of Web Application Flow Whitelisting: Need-A-Nerd student apps

The validation of web application flow whitelisting was carried out in two phases. Phase one applies whitelisting on a set of applications created by students as a requirement for course COMP4970: Web Development with Django at Auburn University. Phase two applies whitelisting on an open source Django application in production use. Before whitelisting the applications, we needed to investigate whether or not they had any vulnerabilities to begin with and if the vulnerabilities were application- or non-application-specific. Since whitelisting addresses only application-specific vulnerabilities, we elected to test for and eliminate all other non-application-specific vulnerabilities in the applications. In order to achieve that, we tested the applications using an open source security scanner - OWASP's Zed Attack Proxy (ZAP).

4.5.1 Zed Attack Proxy (ZAP)

ZAP is a free, open source, cross-platform web application security scanner that provides the means to conduct both automated and manual security testing [OWASP, 2016]. The reason ZAP was selected as the security scanner of choice is because it specifically tests for OWASP's top ten web vulnerabilities. The automated testing scans provided by ZAP are divided into two types: passive scanning and active scanning. ZAP also provides several manual testing tools such as Fuzzer, Spider, Diviner, and Plug-n-Hack. Table 4.1 below provides details about ZAP tools used to test for OWASP's top ten web vulnerabilities.

Method of Testing	**Common Components**
	The 'common components' can be used for pretty much everything, so can be used to help detect all of the Top 10
Manual	Intercepting proxy
Manual	Manual request / resend
Manual	Scripts
Manual	Search
A1	**Injection**
Automated	Active Scan Rules (Release, Beta* and Alpha*)
Automated	SQL Map Injection Engine (Beta*)
Manual	Fuzzer, combined with the FuzzDb (Release)* and SVN Digger (Beta)* files
Manual	Diviner (Alpha)*
A2	**Broken Authentication and Session Management**
Manual	Http Sessions
Manual	Spider
Manual	Forced Browse (Beta)
Manual	Token Generator (Beta)*
Manual	Diviner (Alpha)*
Manual	Vehicle (Alpha)*
A3	**Cross-Site Scripting (XSS)**
Automated	Active Scan Rules (Release)
Manual	Fuzzer, combined with the FuzzDb (Release)* and SVN Digger (Beta)* files
Manual	Plug-n-Hack (Beta)
Manual	Diviner (Alpha)*
A4	**Insecure Direct Object References**
Manual	Params tab
Manual	Diviner (Alpha)*
A5	**Security Misconfiguration**
Automated	Active Scan Rules (Release, Beta* and Alpha*)
Automated	Passive Scan Rules (Release, Beta* and Alpha*)
Manual	HttpsInfo (Alpha)*
Manual	Port Scanner (Beta)*
Manual	Technology detection (Alpha)*
A6	**Sensitive Data Exposure**
Automated	Active Scan Rules (Release, Beta* and Alpha*)
Automated	Passive Scan Rules (Release, Beta* and Alpha*)
A7	**Missing Function Level Access Control**
Manual	Spider
Manual	Ajax Spider (Beta)
Manual	Session comparison
Manual	Access Control (Currently only available in Weekly release)
A8	**Cross-Site Request Forgery**
Automated	Active Scan Rules (Beta)*
Automated	Passive Scan Rules (Beta)*
Manual	Generate Anti CSRF Test Form
A9	**Using Components with Known Vulnerabilities**
Automated	Passive Scan Rules (Alpha)* and Retire (Alpha)*
Manual	Technology detection (Alpha)*
A10	**Unvalidated Redirects and Forwards**
Automated	Active Scan Rules (Release)
Manual	Fuzzer, combined with the FuzzDb (Release)* and SVN Digger (Beta)* files
Manual	Diviner (Alpha)*

* The starred add-ons are not included by default in the full ZAP release
but can be downloaded from the ZAP Marketplace via the Manage add-ons button on the ZAP main toolbar.

Table 4.1: Automatic and manual components of ZAP recommended for testing OWASP top 10 2013 vulnerabilities [OWASP, 2016].

ZAP's automatic scans, both passive and active, were used to test all the apps for non-application-specific vulnerabilities. The vulnerabilities found and the mitigation policy applied are discussed in section 4.6.

4.5.2 Apps from COMP4970: Web Development with Django

As part of the course requirements, students taking COMP4970 were asked to develop a working Django web app. The app, called Need-A-Nerd (NaN), is used by students looking for software development jobs as well as employers who can post job offerings. The goal of NaN is to link students that have a certain set of skills with jobs that require that skill set.

Overview of NaN

The main functional requirements of NaN can be categorized as follows:

1. User Authorization:
   (a) NaN is restricted to registered users.
   (b) NaN users are either Students or Employers. Employers can either be on-campus (affiliated with Auburn University) or off-campus (not affiliated with Auburn University).
   (c) Students and on-campus employers are registered immediately upon request.
   (d) For off-campus employers, NaN administrator approval is required upon registration.
   (e) Registered users may un-register at any time.
2. Student functionality:
   (a) Each student has a profile consisting of his/her name, email address, academic major, and an optional resume. The resume lists the student's skills.
   (b) Students may edit their profile and create, delete, or edit their resume.
3. Employer functionality:

   (a) Each employer has a profile consisting of his/her name, email address, phone number, a description (optional), and off-campus employers must include a mailing address.

4. Student-Employer interaction:

   (a) An employer can post a job description. The description would contain an explanation of the job with the skills required, start date, stop date, and (optional) salary information. Dates can be TBD (To Be Determined).

   (b) The employer who posted a job can edit, delete, or make the posting 'viewable' for students.

   (c) Employers can search for students by name. If no search criteria are provided, all students would be listed in the search result.

   (d) Employers can list all jobs posted by other employers.

   (e) Students can search and list jobs available for viewing.

   (f) Students are notified when a new job is posted.

   (g) Students may NOT see other students' profiles.

   (h) A student can apply for a specific job, and the employer who posted the job would be notified of the application.

   (i) Students may apply for as many jobs as they wish.

   (j) A student can view the jobs to which he/she has applied.

   (k) Employers can view students' profiles and contact a student for a specific job.

In order to whitelist NaN apps, we needed to construct a workflow that illustrates both allowed and disallowed behavior based on the requirements listed above. For the sake of clarity, the workflow is divided into three parts: A workflow for authorization functionality, a workflow for student functionality, and a workflow for employer functionality. Each workflow diagram below (figures 4.3, 5.1, and 4.5) shows both allowed behavior (depicted using solid arrows) and disallowed application behavior (dashed arrows).

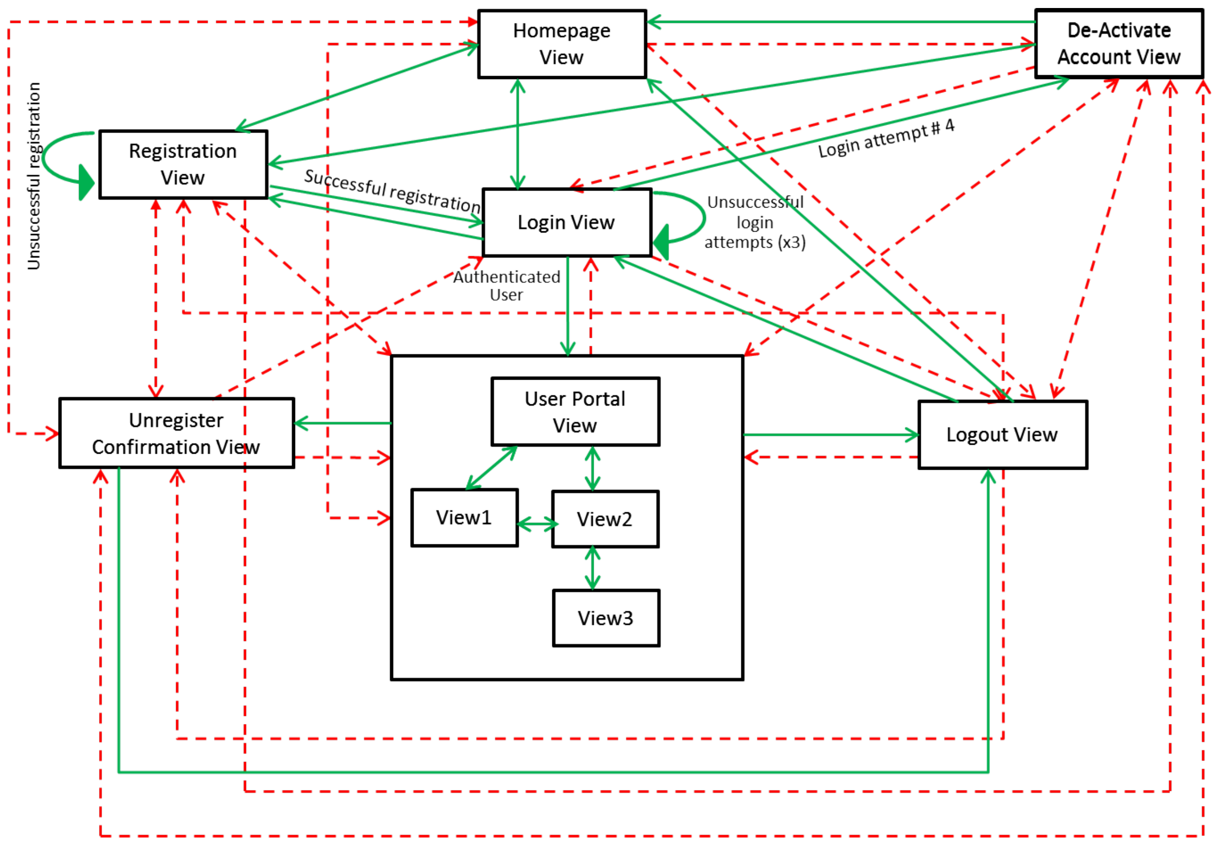


Figure 4.3: NaN Authorization workflow.

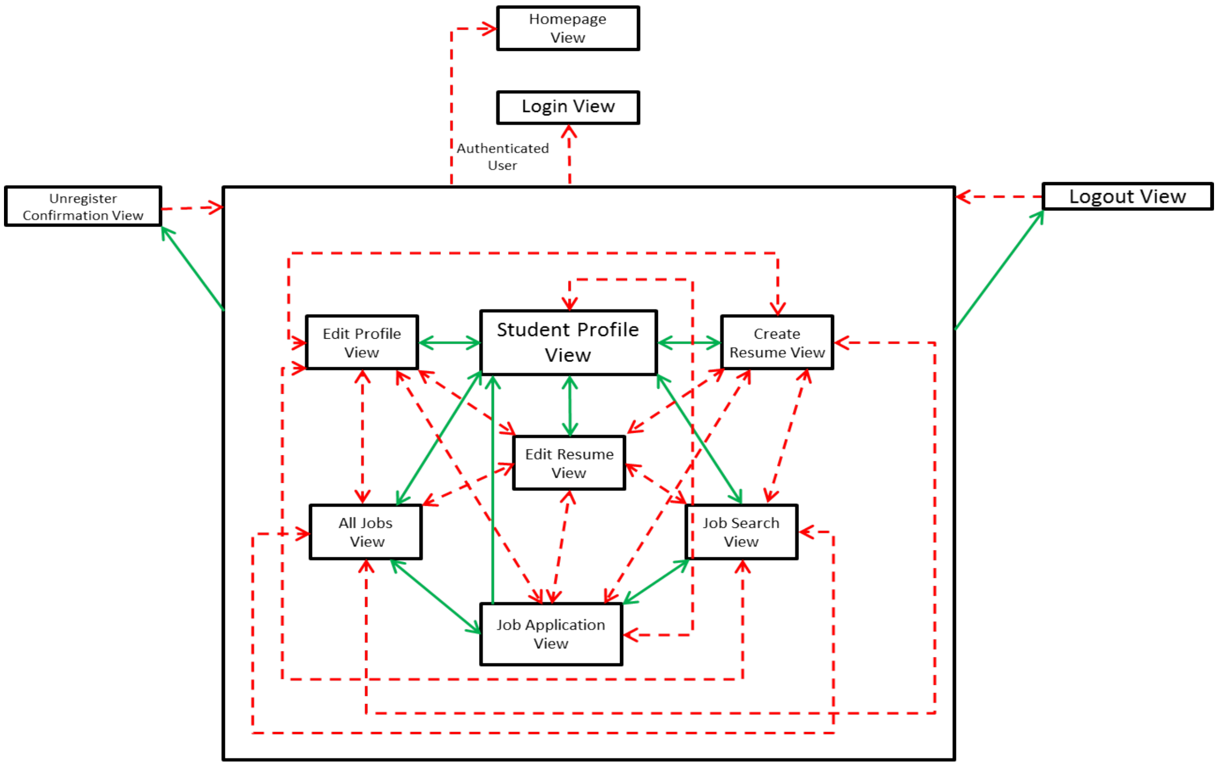


Figure 4.4: NaN Student workflow.

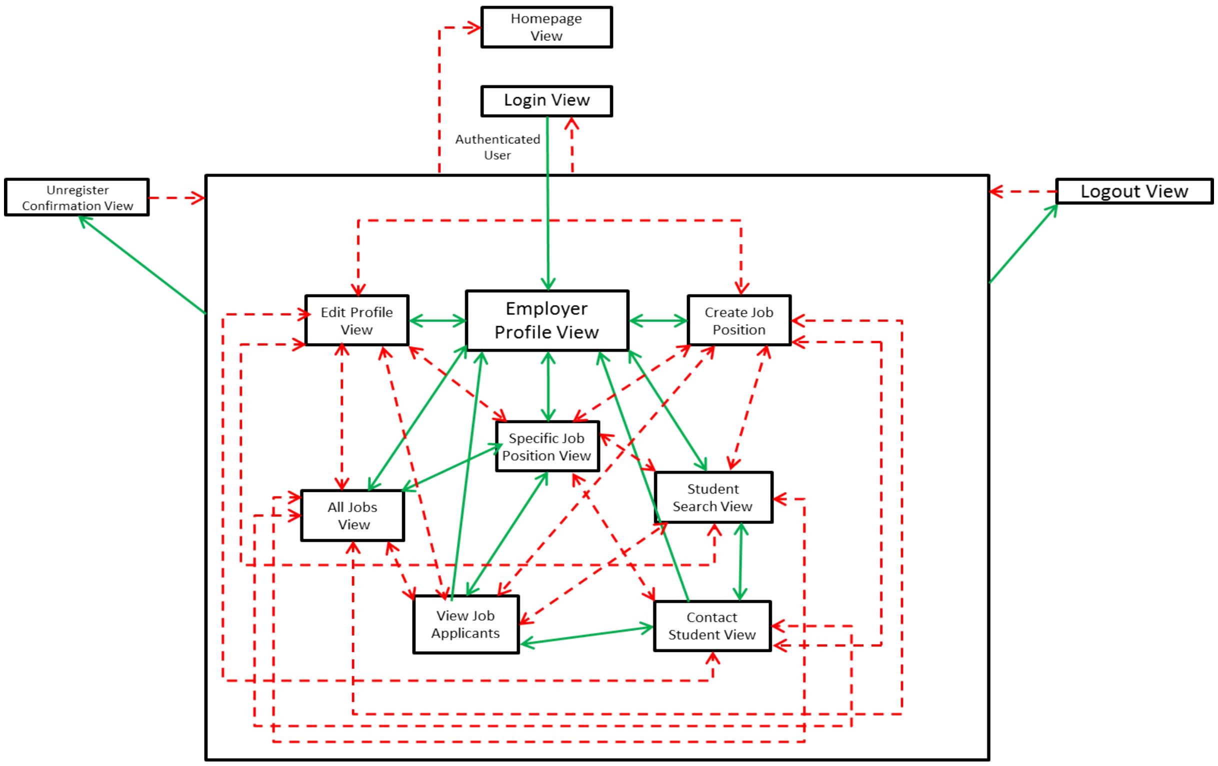


Figure 4.5: NaN Employer workflow.

Based on the authorization workflow of NaN, we derived the whitelist $\langle C, D, W, S \rangle$ as follows:

Set C is the set of all components of NaN authorization and consists of: C = (u, homepage, registration, Login, user portal, Deactivate Account, Unregister, Logout).

As for the global set of conditions D, it contains the following conditions (Table 4.2):

- d_1: anonymous user permissions,
- d_2: current authenticated user permissions and data,
- d_3: valid session expiry time,
- d_4: previous view,
- d_5: login attempts ≤ 3.

	u	homepage	registration	login	user portal	deactivate ac	unregister	logout
u	D={}	D={}	D={d1, d4 = homepage}	D={d1, d4 = homepage}	D={d2, d4 = login}	D={d2, d4 = login}	D={d2, d4 = user portal}	D={d2, d4 = user portal}
homepage	D={d1, d4 = homepage}	D={}	D={d1, d4 = homepage}	D={d1, d4 = homepage}	D={d2, d4 = login}	D={d2, d4 = login}	D={d2, d4 = user portal}	D={d2, d4 = user portal}
registration	D={d1, d4 = homepage}	D={d1, d4 = homepage}	D={}	D={d1, d4 = registration}	D={d2, d4 = login}	D={d2, d4 = login}	D={d2, d4 = user portal}	D={d2, d4 = user portal}
login	D={d1, d4 = logout}	D={d2, d4 = login}	D={d2, d4 = login}	D={d1, d5 <= 3}	D={d2, d4 = login}	D={d2, d4 = login, d5 >3}	D={d2, d4 = user portal}	D={d2, d4 = user portal}
user portal	D={d2, d4 = logout}	D={d2, d4 = logout}	D={d2, d4 = logout}	D={d2, d4 = logout}	D={d2, d3}	D={d2, d4 = login}	D={d2, d4 = user portal}	D={d2, d4 = user portal}
deactivate ac	D={d1, d4 = login}	D={d1, d4 = deactivate ac}	D={d1, d4 = deactivate ac}	D={d1, d4 = homepage}	D={d2, d4 = login}	D={}	D={d2, d4 = user portal}	D={d2, d4 = user portal}
unregister	D={d1, d4 = logout}	D={d2, d4 = logout}	D={d2, d4 = logout}	D={d2, d4 = logout}	D={d2, d4 = login}	D={d2, d4 = login}	D={}	D={d2, d4 = unregister}
logout	D={d1, d4 = logout}	D={d1, d4 = logout}	D={d1, d4 = homepage}	D={d1, d4 = logout}	D={d2, d4 = login}	D={d2, d4 = login}	D={d2, d4 = user portal}	D={}

Table 4.2: Matrix W with a distinct subset of conditions in each cell

From evaluating the subset of conditions for each transition in 4.2, the set of ordered pairs in W are:

$$W = \{\, (u, u)\,, (u, homepage)\,, (homepage, u)\,, (homepage, homepage)\,,$$
$$(homepage, registration)\,, (homepage, login)\,, (registration, homepage)\,,$$
$$(registration, registration)\,, (registration, login)\,, (login, homepage)\,, (login, registration)\,,$$
$$(login, login)\,, (login, user\_portal)\,, (login, deactivate\_account)\,, (user\_portal, user\_portal)\,,$$
$$(user\_portal, unregister)\,, (user\_portal, logout)\,, (deactivate\_account, homepage)\,,$$
$$(deactivate\_account, registration)\,, (deactivate\_account, deactivate\_account)\,,$$
$$(unregister, registration)\,, (unregister, unregister)\,, (unregister, logout)\,, (logout, u)\,,$$
$$(logout, homepage)\,, (logout, login)\,, (logout, logout)\,\}$$

A zero-one representation of W is shown in table 4.3, where allowed flow is represented by 1 and disallowed flow by 0.

	u	homepage	registration	login	user portal	deactivate ac	unregister	logout
u	1*	1	0	0	0	0	0	0
homepage	1	1	1	1	0	0	0	0
registration	0	1	1	1	0	0	0	0
login	0	1	1	1	1	1	0	0
user portal	0	0	0	0	1	0	1	1
deactivate ac	0	1	1	0	0	1	0	0
unregister	0	0	1	0	0	0	1	1
logout	1	1	0	1	0	0	0	1

* For the sake of completion, a 1 is placed for transitions outside the system boundary as they are not under the jurisdiction of the whitelist.

Table 4.3: A zero-one representation of W for NaN Authorization workflow.

We specified matrix S containing the 'safe' components to redirect to in case the evaluation of conditions fails and the transition is disallowed. Table 4.4 shows matrix S for NaN authorization workflow.

	u	homepage	registration	login	user portal	deactivate ac	unregister	logout
u	NA^*	homepage	homepage	homepage	homepage	homepage	homepage	homepage
homepage	u	homepage	homepage	homepage	login	login	login	login
registration	homepage	homepage	registration	homepage	login	login	login	login
login	homepage	homepage	homepage	login	login	login	login	login
user portal	logout	logout	logout	logout	login	logout	login	login
deactivate ac	homepage	homepage	homepage	homepage	registration	homepage	login	login
unregister	homepage	homepage	homepage	homepage	homepage	homepage	homepage	homepage
logout	u	homepage	homepage	login	login	login	login	logout

u represents a component outside the system boundary.
* NA represents Not Applicable and a 1 is placed for transitions outside the system boundary.

Table 4.4: Matrix S for NaN Authorization workflow with safe components to redirect to if a transition is disallowed

Based on the Student workflow of NaN (figure 4.4), we derived the whitelist $\langle C, D, W, S \rangle$ as follows:

Set C is the set of all components of NaN Student workflow and consists of: C = (login, student profile, edit profile, create resume, edit resume, all jobs, job search, job application, unregister, logout).

As for the Student workflow, the set of conditions D contains the following:

- d_1: anonymous user permissions,
- d_2: current authenticated user permissions and data,
- d_3: valid session expiry time,
- d_4: previous view,
- d_5: login attempts ≤ 3.

The conditions that must be checked for each transition to succeed are detailed in (Table 4.5):

	login	student profile	edit profile	create resume	edit resume	all jobs	job search	job applications	logout	unregister
login	D={d1, d5}	D={d2, d4 = login}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = all jobs or job search}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}
student profile	D={d2, d4 = logout}	D={d2, d3}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = all jobs or job search}	D={}	D={d2, d4 = stud profile}
edit profile	D={d2, d4 = logout}	D={d2, d4 = edit profile}	D={d2, d3}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = all jobs or job search}	D={}	D={d2, d4 = stud profile}
create resume	D={d2, d4 = logout}	D={d2, d4 = create resume}	D={d2, d4 = stud profile}	D={d2, d3}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = all jobs or job search}	D={}	D={d2, d4 = stud profile}
edit resume	D={d2, d4 = logout}	D={d2, d4 = edit resume}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d3}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = all jobs or job search}	D={}	D={d2, d4 = stud profile}
All jobs	D={d2, d4 = logout}	D={d2, d4 = all jobs}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d3}	D={d2, d4 = stud profile}	D={d2, d4 = all jobs}	D={}	D={d2, d4 = stud profile}
job search	D={d2, d4 = logout}	D={d2, d4 = job search}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d3}	D={d2, d4 = job search}	D={}	D={d2, d4 = stud profile}
job application	D={d2, d4 = logout}	D={d2, d4 = all jobs or job search}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = job application}	D={d2, d4 = job application}	D={d2, d3}	D={}	D={d2, d4 = stud profile}
logout	D={d2, d4 = logout}	D={d2, d4 = login}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = job search}	D={}	D={d2, d4 = login}
unregister	D={d2, d4 = logout}	D={d2, d4 = login}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = stud profile}	D={d2, d4 = job search}	D={}	D={}

Table 4.5: Matrix W for Student workflow with a distinct subset of conditions in each cell

From evaluating the subset of conditions for each transition in 4.5, the set of ordered pairs in matrix W for Student workflow are:

$$\begin{gathered}
W = \{\, (login, login)\, , (login, stud\_profile)\, , (stud\_profile, stud\_profile)\, , \\
(stud\_profile, edit\_profile)\, , (stud\_profile, create\_resume)\, , (stud\_profile, edit\_resume)\, , \\
(stud\_profile, all\_jobs)\, , (stud\_profile, job\_search)\, , (stud\_profile, logout)\, , \\
(stud\_profile, unregister)\, , (edit\_profile, stud\_profile)\, , (edit\_profile, edit\_profile)\, , \\
(edit\_profile, logout)\, , (create\_resume, stud\_profile)\, , (create\_resume, create\_resume)\, , \\
(create\_resume, logout)\, , (edit\_resume, stud\_profile)\, , (edit\_resume, edit\_resume)\, , \\
(edit\_resume, logout)\, , (all\_jobs, stud\_profile)\, , (all\_jobs, all\_jobs)\, , \\
(all\_jobs, job\_application)\, , (all\_jobs, logout)\, , (job\_search, stud\_profile)\, , \\
(job\_search, job\_search)\, , (job\_search, job\_application)\, , (job\_search, logout)\, , \\
(job\_application, stud\_profile)\, , (job\_application, all\_jobs)\, , (job\_application, job\_search)\, , \\
(job\_application, job\_application)\, , (job\_search, logout)\, , (logout, login)\, , \\
(logout, logout)\, , (unregister, logout)\, , (unregister, unregister)\, \}
\end{gathered}$$

A zero-one representation of W for Student workflow is shown in table 4.6.

	login	student profile	edit profile	create resume	edit resume	all jobs	job search	job applications	logout	unregister
login	1	1	0	0	0	0	0	0	0	0
student profile	0	1	1	1	1	1	1	0	1	1
edit profile	0	1	1	0	0	0	0	0	1	0
create resume	0	1	0	1	0	0	0	0	1	0
edit resume	0	1	0	0	1	0	0	0	1	0
All jobs	0	1	0	0	0	1	0	1	1	0
job search	0	1	0	0	0	0	1	1	1	0
job application	0	1	0	0	0	1	1	1	1	0
logout	1	0	0	0	0	0	0	0	1	0
unregister	0	0	0	0	0	0	0	0	1	1

Table 4.6: A zero-one representation of W for NaN Student workflow.

We populated matrix S with 'safe' components to redirect to when the evaluation of conditions fails and the transition is disallowed. Table 4.7 shows matrix S for NaN Student workflow.

	login	stud profile	edit profile	create resume	edit resume	all jobs	job search	job applications	logout	unregister
login	login	login	login	login	login	login	login	login	login	login
stud profile	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout
edit profile	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout
create resume	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout
edit resume	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout
all jobs	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout
job search	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout
job application	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout
logout	login	login	login	login	login	login	login	login	login	login
unregister	homepage	homepage	homepage	homepage	homepage	homepage	homepage	homepage	homepage	homepage

Table 4.7: Matrix S for NaN Student workflow with safe components to redirect to if a transition is disallowed

Finally, for NaN Employer workflow (figure 4.5), we derived the whitelist $\langle C, D, W, S \rangle$ as follows:

Set C is the set of all components of NaN Employer workflow and consists of: C = (login, employer profile, edit profile, create job, all jobs, specific job, student search, view applicants, contact student, unregister, logout).

The Employer workflow has the set of conditions D which contain the following:

- d_1: anonymous user permissions,
- d_2: current authenticated user permissions and data,
- d_3: valid session expiry time,
- d_4: previous view,
- d_5: login attempts ≤ 3.

The conditions that must be checked for each transition to succeed are detailed in (Table 4.8):

	login	employer profile	edit profile	create job	all jobs	specific job	stud search	view applicants	contact student	logout	unregister
login	D={d1, d5}	D={d2, d4 = login}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = specific job or contact stud}	D={d2, d4 = stud search or view applicants}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}
employer profile	D={d2, d4 = logout}	D={d2, d3}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = specific job or contact stud}	D={d2, d4 = stud search or view applicants}	D={}	D={d2, d4 = emp profile}
edit profile	D={d2, d4 = logout}	D={d2, d4 = edit profile}	D={d2, d3}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = specific job or contact stud}	D={d2, d4 = stud search or view applicants}	D={}	D={d2, d4 = emp profile}
create job	D={d2, d4 = logout}	D={d2, d4 = create job}	D={d2, d4 = emp profile}	D={d2, d3}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = specific job or contact stud}	D={d2, d4 = stud search or view applicants}	D={}	D={d2, d4 = emp profile}
all jobs	D={d2, d4 = logout}	D={d2, d4 = all jobs}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d3}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = specific job or contact stud}	D={d2, d4 = stud search or view applicants}	D={}	D={d2, d4 = emp profile}
specific job	D={d2, d4 = logout}	D={d2, d4 = specific job}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d3}	D={d2, d4 = emp profile}	D={d2, d4 = specific job }	D={d2, d4 = view applicants}	D={}	D={d2, d4 = emp profile}
stud search	D={d2, d4 = logout}	D={d2, d4 = stud search}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d3}	D={d2, d4 = specific job or contact stud}	D={d2, d4 = stud search}	D={}	D={d2, d4 = emp profile}
view applicants	D={d2, d4 = logout}	D={d2, d4 = view applicants}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = view applicants}	D={d2, d4 = view applicants}	D={d2, d3}	D={d2, d4 = contact stud}	D={}	D={d2, d4 = emp profile}
contact stud	D={d2, d4 = logout}	D={d2, d4 = contact stud}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = contact stud}	D={d2, d4 = contact stud}	D={d2, d3}	D={}	D={d2, d4 = emp profile}
logout	D={d2, d4 = logout}	D={d2, d4 = login}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = specific job or contact stud}	D={d2, d4 = stud search or view applicants}	D={}	D={d2, d4 = login}
unregister	D={d2, d4 = logout}	D={d2, d4 = login}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = emp profile}	D={d2, d4 = specific job or contact stud}	D={d2, d4 = stud search or view applicants}	D={}	D={d2, d4 = login}

Table 4.8: Matrix W for Employer workflow with a distinct subset of conditions in each cell

From evaluating the subset of conditions for each transition in 4.8, the set of ordered pairs in matrix W for Employer workflow are:

$$\begin{gathered}
W = \{\, (login, login)\,, (login, emp\_profile)\,, (emp\_profile, emp\_profile)\,, \\
(emp\_profile, edit\_profile)\,, (emp\_profile, create\_job)\,, (emp\_profile, all\_jobs)\,, \\
(emp\_profile, specific\_job)\,, (emp\_profile, stud\_search)\,, (emp\_profile, logout)\,, \\
(emp\_profile, unregister)\,, (edit\_profile, emp\_profile)\,, (edit\_profile, edit\_profile)\,, \\
(edit\_profile, logout)\,, (create\_job, emp\_profile)\,, (create\_job, create\_job)\,, \\
(create\_job, logout)\,, (all\_jobs, emp\_profile)\,, (all\_jobs, all\_jobs)\,, (all\_jobs, specific\_job)\,, \\
(all\_jobs, logout)\,, (specific\_job, emp\_profile)\,, (specific\_job, emp\_profile)\,, \\
(specific\_job, all\_jobs)\,, (specific\_job, specific\_job)\,, (specific\_job, view\_applicants)\,, \\
(specific\_job, logout)\,, (stud\_search, emp\_profile)\,, (stud\_search, stud\_search)\,, \\
(stud\_search, contact\_stud)\,, (stud\_search, logout)\,, (view\_applicants, emp\_profile)\,, \\
(view\_applicants, specific\_job)\,, (view\_applicants, view\_applicants)\,, \\
(view\_applicants, contact\_stud)\,, (view\_applicants, logout)\,, (contact\_stud, emp\_profile)\,, \\
(contact\_stud, stud\_search)\,, (contact\_stud, view\_applicants)\,, (contact\_stud, contact\_stud)\,, \\
(contact\_stud, logout)\,, (logout, login)\,, (logout, logout)\,, \\
(unregister, logout)\,, (unregister, unregister)\,\}
\end{gathered}$$

A zero-one representation of W for Employer workflow is shown in table 4.9. Matrix S (table 4.10) shows the 'safe' components to redirect to when the evaluation of conditions fails and the transition is disallowed.

	login	employer profile	edit profile	create job	all jobs	specific job	stud search	view applicants	contact stud	logout	unregister
login	1	1	0	0	0	0	0	0	0	0	0
employer profile	0	1	1	1	1	1	1	0	0	1	1
edit profile	0	1	1	0	0	0	0	0	0	1	0
create job	0	1	0	1	0	0	0	0	0	1	0
all jobs	0	1	0	0	1	1	0	0	0	1	0
specific job	0	1	0	0	1	1	0	1	0	1	0
stud search	0	1	0	0	0	0	1	0	1	1	0
view applicants	0	1	0	0	0	1	0	1	1	1	0
contact stud	0	1	0	0	0	0	1	1	1	1	0
logout	1	0	0	0	0	0	0	0	0	1	0
unregister	0	0	0	0	0	0	0	0	0	1	1

Table 4.9: A zero-one representation of W for NaN Employer workflow.

	login	emp profile	edit profile	create job	all jobs	specific job	stud search	view applicants	contact stud	logout	unregister
login	login	login	login	login	login	login	login	login	login	login	login
emp profile	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout
edit profile	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout
create job	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout
all jobs	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout
specific job	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout
stud search	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout
view applicant	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout
contact stud	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout
logout	login	login	login	login	login	login	login	login	login	login	login
unregister	homepage	homepage	homepage	homepage	homepage	homepage	homepage	homepage	homepage	homepage	homepage

Table 4.10: Matrix S for NaN Employer workflow with safe components to redirect to if a transition is disallowed

4.6 Analysis

There were a total of 19 teams who delivered NaN applications to fulfill the COMP4970 project requirement. Four apps were dismissed because they did not achieve a passing grade on the project or they did not function due to their use of a third party package that was no longer available. All apps were upgraded from Django 1.2 to Django 1.8.5 to take advantage of the most recent version of Django. No feature changes were made to the projects. Since the remaining 15 apps did not implemented all of NaN's functionality, we based our whitelist and workflow on the functionalities implemented by most teams. Table 4.11 below shows a list of NaN features and the team projects which implemented them.

		Feature List	Team 1	Team 2	Team 3	Team 4	Team 5	Team 6	Team 7	Team 8	Team 9	Team 10	Team 11	Team 12	Team 13	Team 14	Team 15	**TOTAL COUNT**
Auth	1.1	Restricted to registered users	X	X	X	X	X	X	X	X	X	X	X	X	X	X	NO	14
	1.2.1	User designated as student	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	15
	1.2.2	User designated as on campus employer	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	15
	1.2.3	User designated as off campus employer	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	15
	1.3.1	Registered users may use Nan immediately	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	15
	1.3.2	Off campus employer must be approved b4 registered	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	0
	1.3.3	Registered users may unregister	X	NO	X	NO	NO	X	X	X	X	NO	X	X	NO	NO	X	9
Std	2.1	Each student has a profile containing name, email, major, and (optional) resume	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	15
	2.2	Each student can edit his/her profile and create, delete or edit the resume	X	NO	X	X	NO	X	NO	X	X	NO	NO	NO	NO	X	X	8
Emp	3.1	Each employer has a profile containing name, email, phone #, and (optional) description	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	15
	3.2	off campus employers must include a mailing address	X	NO	X	X	X	X	X	X	X	X	NO	X	X	X	NO	12
St-Emp Interaction	4.1	A job can be posted by an employer containing a description, skills required, start, end and (optional) salary	X	X	X	X	X	X	NO	X	X	X	X	X	X	X	X	14
	4.2	The employer can edit and delete his/her posting and indicate if posting is viewable for students	X	NO	X	X	NO	NO	NO	X	X	X	X	X	X	X	X	11
	4.3	Employers can search for a student by name, if no name provided search returns all registered students	X	NO	NO	X	X	X	NO	X	X	X	X	NO	X	X	X	11
	4.4	Employers can list jobs posted by other employers	X	X	X	X	X	NO	NO	X	X	NO	X	NO	X	X	X	11
	4.5	A student can list all jobs	X	X	X	X	X	NO	NO	X	X	X	X	NO	X	X	X	12
	4.6	A student can only see job descriptions released for viewing	X	NO	X	X	NO	NO	NO	X	X	NO	X	NO	X	NO	X	8
	4.7	Students are notified when a new job is posted	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	0
	4.8	Students cannot see other students profiles	X	NO	NO	X	X	X	X	NO	X	NO	X	NO	NO	X	X	9
	4.9	A student can apply for a job and the employer who posted the job is notified	NO	NO	NO	NO	NO	NO	NO	X	X	NO	NO	X	X	NO	X	5
	4.10	A student can see jobs he/she applied for	NO	X	X	X	X	NO	NO	X	X	X	X	X	X	X	X	12
	4.11	An employer can choose to contact a student regarding a specific job	X	NO	X	X	X	X	NO	X	X	NO	NO	X	NO	NO	X	9

Table 4.11: NaN team projects with implemented (or non-implemented) features

ZAP's passive and active scans were run on each of the team project apps. The non-application-specific vulnerabilities found and the mitigation strategy applied to all 15 projects are summarized in tables 4.12 and 4.13 below:

Issues found by testing using ZAP passive scans				
	Issue	Place of occurrence	Resolved by	No. of Projects
1	CSRF Cookie set w/out HTTPOnly	settings.py	adding: CSRF_COOKIE_HTTPONLY = True adding: SESSION_COOKIE_HTTPONLY = True	15
2	Web browser XSS protection not enabled	settings.py	adding: SECURE_BROWSER_XSS_FILTER = True	15
3	Missing X-Content-Type-Options Header	settings.py	adding: SECURE_CONTENT_TYPE_NOSNIFF = True	15
4	Passowrd autocomplete in browser	templates with forms containing a password field	adding attribute autocomplete=”off” to all html forms containing password field	15

Table 4.12: Results of ZAP passive scans on NaN team projects

Issues found by testing using ZAP active scans				
	Issue	Place of occurrence	Resolved by	No. of Projects
1	Application error disclosure Internal server error	access to Django admin within app or a redirect to a non existing template in views.py	removing access to Django admin from app fixing redirect to an existing template	2
2	Application error disclosure (MultiValueKeyDict) error	views.py	replacing username = request.POST['username'] with request.POST.get(”username”, ””) and password = request.POST['password'] with request.POST.get(”password”,””)	1
3	Application error disclosure uncaught exception	views.py	adding an else clause to an if statement	1
4	Password characters not hidden	views.py	adding password = (widget=forms.PasswordInput()) or changing input type to password instead of text	5

Table 4.13: Results of ZAP active scans on NaN team projects

4.6.1 Application Specific Vulnerabilities

Once the vulnerabilities revealed by ZAP scans were mitigated, we began testing all 15 apps for application-specific vulnerabilities. We found application-specific vulnerabilities in every one of the 15 apps we tested. Since every implementation was unique in terms of structure and component names (view names, URLs, etc.), we used a generic description for each vulnerability found.

1. Anonymous user accessing user portal views directly by typing in a view's URL.

Example: Being able to search the app's database by typing in 127.0.0.1:8000/nan/search/.

Explanation: By examining the workflow of NaN (figures 4.3, 4.4, 4.5), views that are housed within the user portal such as 'Job Search' for Student portal or 'Student Search' for Employer portal can not be accessed without authenticating the user first.

Apps exhibiting this vulnerability: Teams: 3, 4, and 15.

2. <ins>Anonymous user accessing the previous view after successful log out or unregister.</ins>

    Example: A legitimate user logs in and navigates through his/her profile, then decides to log out or unregister. After log out or un-registration is successful, the user clicks the browser's back button and is able to see the profile page he/she was on.

    Explanation: By examining the authorization workflow of NaN (figures 4.3), upon successful log out or un-registration, a user becomes anonymous and therefore must not be able to access any profile view without first being authenticated.

    Apps exhibiting this vulnerability: Teams: 1, 3, 8 and 10.

3. <ins>Authenticated user accessing another user's profile views by typing in a URL containing the other user's name or id.</ins>

    Example: A legitimate user logs in and edits his/her resume through URL: 127.0.0.1:8000/nan/editResume/2/, with 2 being the logged in user's id. If the currently logged in user changes the URL to 127.0.0.1:8000/nan/editResume/5/, he/she is able to access and edit the resume which belongs to user 5. The same behavior was found in Student views and Employer views.

    Explanation: When a user is authenticated, he/she gets access privileges to his/her data records only. The app should not allow any authenticated user to access other users' data.

Apps exhibiting this vulnerability: Teams: 1, 3, 5, 7, 8, 9, 10, 11, 12, 13, 14, and 15.

4. Authenticated user inadvertently (or illegitimately) accessing app functionality that they should otherwise be restricted from.

   Example: A student user logs in and clicks on search jobs link, jobs by employer are listed, and when he/she clicks on a certain employer they are redirected to Employer portal and can now access employer functionality such as viewing other students' profiles.

   Explanation: When a user is authenticated, he/she belongs to a group of users with certain access privileges. The app should not allow any authenticated user from group A, for example, to access functionality that only group B is allowed to access.

   Apps exhibiting this vulnerability: Teams: 2,6, and 14.

Following allowed NaN workflow described in section 4.5.2 above, a whitelist middleware was created for each app. The whitelist middleware for all apps was constructed with the following generic structure:

- In process_request() hook:
  1. Admin site URLs were excluded from whitelisting.
  2. Flag Notallowed was set to True indicating that all behavior is initially disallowed.
  3. We checked for allowed behavior by:
     (a) Getting the HTTP_REFERER attribute and the requested URL.
     (b) Evaluating the subset of conditions for that particular transition (from HTTP_REFERER to requested URL) and if the evaluation resulted in 1, we changed flag Notallowed to False.
     (c) Returning None for the allowed flow to continue to process_view() hook.
- In process_view() hook:

1. We checked if flag Notallowed was True, if it is we flushed the user session and redirected to the safe view for that particular transition.
2. Else, we returned None (for the allowed transition to continue and render the desired response).

It should be noted that every app's whitelist middleware had to be customized according to the component names used by that app.

4.6.2 Results

All NaN apps were tested for behavior using the whitelist middleware. The apps were tested to make sure the whitelist was not preventing any intended behavior or functionality. This means that the intended behavior (allowed flow represented by 1s) was permitted by the whitelist middleware. As for unintended behavior, the whitelisted apps were tested to make sure the whitelist middleware was recognizing the disallowed flow and redirecting to the corresponding safe view(s). Table 4.14 below summarizes the vulnerabilities that were found in every app and were successfully mitigated by the whitelist middleware.

Generic Vulneribility	Number	Specific vulnerability	Team with vulnerability	Mitigated by whitelist
1	1	anonymous user can access http://127.0.0.1:8000/nan/allJobs/	3	Y
1	2	anonymous user can access http://127.0.0.1:8000/employers/	4	Y
1	3	anonymous user can access http://127.0.0.1:8000/search/jobs/	4	Y
1	4	anonymous user can access http://127.0.0.1:8000/employers/	15	Y
1	5	anonymous user can access http://127.0.0.1:8000/students/	15	Y
1	6	anonymous user can access http://127.0.0.1:8000/allJobs/None/	15	Y

Generic Vulneribility	Number	Specific vulnerability	Team with vulnerability	Mitigated by whitelist
2	1	in http://127.0.0.1:8000/profile/, unregister, hit browser back button, profile is displayed	1	Y
2	2	http://127.0.0.1:8000/nan/editResume/5/, log out, hit browser back button, resume can be edited	3	Y
2	3	http://127.0.0.1:8000/nan/editDegree/5/, log out, hit browser back button, degree can be edited	3	Y
2	4	http://127.0.0.1:8000/nan/editUniversity/5/, log out, hit browser back button, university can be edited	3	Y
2	5	http://127.0.0.1:8000/nan/editDegreeMajor/5/, log out, hit browser back button, major can be edited	3	Y
2	6	http://127.0.0.1:8000/nan/editDate/5/, log out, hit browser back button, degree date can be edited	3	Y
2	7	http://127.0.0.1:8000/nan/contactByJob/8/17/, log out, hit browser back button, a student may be contacted	3	Y
2	8	http://127.0.0.1:8000/editResume/5/, log out, hit browser back button, resume can be edited and saved	8	Y
2	9	http://127.0.0.1:8000/editDegree/12/, log out, hit browser back button, degree can be edited and saved	8	Y
2	10	http://127.0.0.1:8000/listAllJobs/, log out, hit browser back button, can view all jobs but can't apply	8	Y
2	11	http://127.0.0.1:8000/JSearch/, log out, hit browser back button, job search page can be viewed	8	Y
2	12	http://127.0.0.1:8000/SSearch/, log out, hit browser back button, student search page can be viewed	8	Y
2	13	http://127.0.0.1:8000/editJob/1/, log out, hit browser back button, job details can be edited and saved	8	Y
2	14	http://127.0.0.1:8000/jobDesc/1/, log out, hit browser back button, job description can be viewed	8	Y
2	15	http://127.0.0.1:8000/home/student/18/resume/edit,	10	Y
		log out, hit back button, resume can be edited and saved	10	Y
2	16	http://127.0.0.1:8000/home/student/18/searchJobs, log out, hit back button, jobs can be searched	10	Y
2	17	http://127.0.0.1:8000/home/employer/1/postedJobs/job/7/edit,	10	Y
		log out, hit back button, job can be edited and saved	10	Y
2	18	http://127.0.0.1:8000/home/employer/searchStudents, log out, hit back button, students can be searched	10	Y
3	1	http://127.0.0.1:8000/job/8/, a student or employer can enter a job URL directly and view a hidden job	1	Y
3	2	http://127.0.0.1:8000/nan/students/1/, change it to /2/ and another student's profile can be viewed	2	Y
3	3	http://127.0.0.1:8000/nan/students/1/jobs/,	2	Y
		change it to /2/jobs/ and another student's job applications can be viewed		Y
3	4	http://127.0.0.1:8000/nan/employers/1/, change it to /2/ and another employer's profile can be viewed	2	Y
3	5	http://127.0.0.1:8000/nan/employers/1/jobs/,	2	Y
		change it to /2/jobs/ and another employer's job applications can be viewed		Y
3	6	http://127.0.0.1:8000/nan/editResume/5/,	3	Y
		change the 5 to 6 and another student's resume can be viewed and edited		Y
3	7	http://127.0.0.1:8000/nan/editDegreee/5/,	3	Y
		change the 5 to 6 and another student's degree can be viewed and edited		Y
3	8	http://127.0.0.1:8000/nan/editUniversity/5/,	3	Y

Generic Vulneribility	Number	Specific vulnerability	Team with vulnerability	Mitigated by whitelist
		change the 5 to 6 and another student's university can be viewed and edited		Y
3	9	http://127.0.0.1:8000/nan/editDegreeMajor/5/,	3	Y
		change the 5 to 6 and another student's major can be viewed and edited		Y
3	10	http://127.0.0.1:8000/nan/editDate/5/,	3	Y
		change the 5 to 6 and another student's degree date can be viewed and edited		Y
3	11	http://127.0.0.1:8000/nan/editJobName/12/6/,	3	Y
		change numbers to /11/17/ and a job posted by other employer can be viewed and edited		Y
3	12	http://127.0.0.1:8000/nan/editJobDescription/12/6/,	3	Y
		change numbers to /11/17/ and a job description posted by other employer can be viewed and edited		Y
3	13	http://127.0.0.1:8000/nan/editJobStart/12/6/,	3	Y
		change numbers to /11/17/ and a job start date posted by other employer can be viewed and edited		Y
3	14	http://127.0.0.1:8000/nan/editJobStop/12/6/,	3	Y
		change numbers to /11/17/ and a job end date posted by other employer can be viewed and edited		Y
3	15	http://127.0.0.1:8000/nan/editJobSalary/12/6/,	3	Y
		change numbers to /11/17/ and a job salary posted by other employer can be viewed and edited		Y
3	16	http://127.0.0.1:8000/nan/editJobVisibility/12/6/,	3	Y
		change numbers to /11/17/ and a job visibility posted by other employer can be viewed and edited		Y
3	17	http://127.0.0.1:8000/nan/removeThisJob/12/6/,	3	Y
		change numbers to /11/17/ and a job posted by other employer can be deleted		Y
3	18	http://127.0.0.1:8000/nan/student/7/applicationlist/, change number to /1/ and another student	5	Y
		applications can be viewed		Y
3	19	http://127.0.0.1:8000/nan/student/7/notifications/, change number to /1/ and another student	5	Y
		notifications can be viewed		Y
3	20	http://127.0.0.1:8000/nan/employer/5/jobcreate/, change number to /6/ and you can create	5	Y
		a job for another employer		Y
3	21	employer logs in and is redirected to http://127.0.0.1:8000/nan/profile/, then types	6	Y
		http://127.0.0.1:8000/nan/jobs/3/delete/, and another employer's posted job is deleted		Y
3	22	employer logs in, types http://127.0.0.1:8000/nan/jobs/7/visible/, and another employer's job is made visible	6	Y
3	23	http://127.0.0.1:8000/employers/JohnDoe/, change it to /JackDoe/ and you access and edit Jack Doe's profile	7	Y
3	24	http://127.0.0.1:8000/JaneDoe/applications/, change it to /JillDoe/ and you access and edit Jill's applications	7	Y
3	25	http://127.0.0.1:8000/editResume/5/, change it to /2/ and another student's resume can be viewed/edited	8	Y
3	26	http://127.0.0.1:8000/editDegree/10/, change it to /12/ and another student's degree can be viewed/edited	8	Y

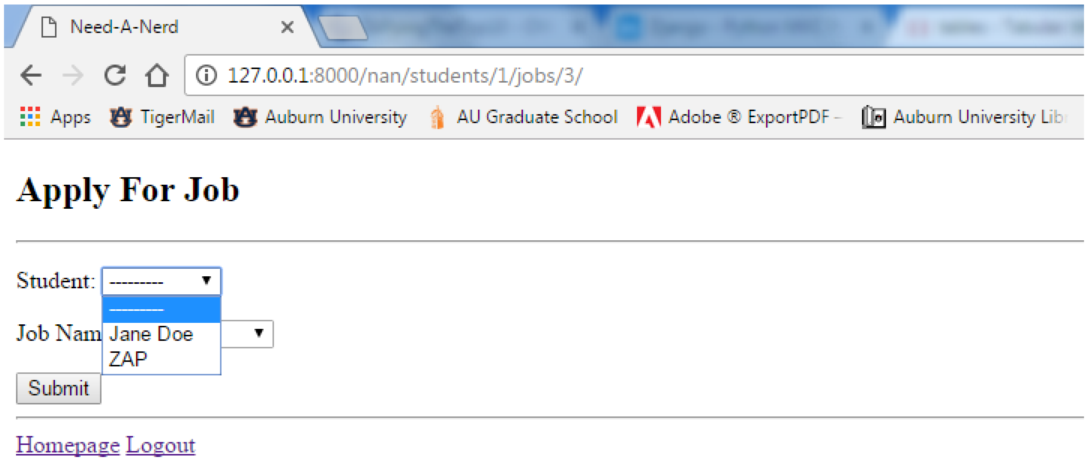
Generic Vulneribility	Number	Specific vulnerability	Team with vulnerability	Mitigated by whitelist
3	27	http://127.0.0.1:8000/editJob/1/, change it to /3/ and edit a job posted by another employer	8	Y
3	28	http://127.0.0.1:8000/jobDesc/1/, change it to /3/ and a job posted by another employer can be viewed	8	Y
3	29	http://127.0.0.1:8000/deleteJob/1/, change it to /3/ and a job posted by another employer can be deleted	8	Y
3	30	http://127.0.0.1:8000/nan/jobs/edit/2/, change it to /7/ and edit a job posted by another employer	9	Y
3	31	http://127.0.0.1:8000/home/student/17/resume/, change it to /18/ and edit another student's resume	10	Y
3	32	http://127.0.0.1:8000/home/student/17/resume/delete/,	10	Y
		change it to /18/ and another student's resume can be deleted		Y
3	33	http://127.0.0.1:8000/home/student/17/resume/addDegree/,	10	Y
		change it to /18/ and add a degree to another student's resume		Y
3	34	http://127.0.0.1:8000/home/student/17/resume/degree/5/delete/,	10	Y
		change it to /18/ and another student's degree can be deleted		Y
3	35	http://127.0.0.1:8000/home/employer/1/postedJobs/job/13/, change it to /2/postedJobs/job/14/	10	Y
		a job posted by another employer can be viewed/edited		Y
3	36	http://127.0.0.1:8000/student/4/degree/add/, change it to /2/degree/add/	11	Y
		and add a degree to another student's resume		Y
3	37	http://127.0.0.1:8000/student/JaneDoe/resume/edit, change it to /JillDoe/ and Jill's resume can be edited	12	Y
3	38	http://127.0.0.1:8000/Student/7/, change it to /3/ and another student's resume can be viewed	13	Y
3	39	http://127.0.0.1:8000/student/degree/update/8/,	14	Y
		change it to /9/ and another student's degree can be edited		Y
3	40	upon completing a job application, a student it redirected to http://127.0.0.1:8000/student/10/jobs/,	14	Y
		change it to http://127.0.0.1:8000/student/11/jobs/ and another students applications can be viewed		Y
3	41	http://127.0.0.1:8000/employer/5/, change it to /4/ and another employer's profile is viewed and jobs can	14	Y
		be edited		Y
3	42	http://127.0.0.1:8000/resume/5/, change it to /4/ and another student's resume can be viewed	15	Y
3	43	http://127.0.0.1:8000/deleteResume/4/10/, change it to /5/13/ and another student's resume can be deleted	15	Y
3	44	http://127.0.0.1:8000/jobApplications/5/, change it to /4/ and another student's applications can be viewed	15	Y
3	45	http://127.0.0.1:8000/employer/2/, change it to /3/ and another employer's profile can be accessed	15	Y
3	46	http://127.0.0.1:8000/deleteJob/1/6/, change it to /3/2/ and another employer's job can be deleted	15	Y
3	47	a user logs in and enters http://127.0.0.1:8000/clearNewApps/2/ and the notifications of employer 2 will	15	Y
		be cleared		Y
4	1	a student can log in, type in http://127.0.0.1:8000/nan/employers/1/search_for_students/	2	Y
		and can search other students (a functionality restricted to employers only)	2	Y

Generic Vulneribility	Number	Specific vulnerability	Team with vulnerability	Mitigated by whitelist
4	2	a student can log in, type in http://127.0.0.1:8000/nan/search/students/ and can view other students'	6	Y
		profiles (a functionality restricted to employers only)	6	Y
4	3	a student can log in, search for employers, select an employer, the app redirects to employers portal	14	Y
		and the student can search and access other students' profiles (a functionality restricted to employers only)	14	Y

Table 4.14: NaN vulnerabilitites mitigated by whitelist middleware

Discussion

From the results illustrated in table 4.14, we reject the null hypothesis in favor of the alternative hypothesis. We observed that the whitelist was successful in mitigating all the vulnerabilities found. The reason for that is the whitelist by default sets an initial flag for all transitions as NotAllowed = True. Only when a transition is intended, which means the evaluation of a specific subset of conditions returned True for every condition in the subset, did the flag change to NotAllowed = False. This method guarantees that any behavior not formally defined within the whitelist is rejected. Three NaN project apps demonstrated serious logical flaws in their implementation. Team 2's project allows a student to apply to a job on behalf of another student by selecting the other student's name from a drop-down list of student names (figure 4.6). Team 5's project did not implement logout functionality at all. Team 12's project allows a student to edit another student's resume by selecting the other student from a drop-down list of students. It also allows an employer to create a job for another employer by selecting the other employer from a drop-down list of employers.



Apply For Job

Student: ---------

Job Nam

Jane Doe
ZAP

Submit

Homepage Logout

Figure 4.6: Sample screenshot of a logical flaw in Team 2's NaN app

The whitelist is unable to rectify these logical flaws without major alteration to the code. However, for the projects mentioned above, the whitelist prevents a currently logged in user from changing the user-id or username portion of the URL directly. Ideally, the concept of web flow whitelisting, should be applied during design and/or development phase(s) of an application and would therefore be regarded as a way for building security into the application. For validation purposes, we needed to use apps that were already built and contained the types of vulnerabilities that the whitelist solves. In this case, web application flow whitelisting was used as a vulnerability mitigation strategy after testing for and finding application specific vulnerabilities. Furthermore, the whitelist helped in identifying and mitigating OWASP's vulnerabilities A4: Insecure Direct Object References and A7: Missing function level access control.

4.6.3 Limitations

Some limitations of the whitelist were found during our validation process. We noticed that the size of the whitelist is directly related to the number of components in an application. If the number of components in an application is equal to n, then a whitelist of size n X n would be generated for this application. The whitelist takes into account the possibility of any transition occurring between two components in an application and it responds by either allowing it to go through or redirecting to a safe component. Another limitation of the whitelist is that it can not resolve any logical flaws in an application's code. However, if the whitelist is constructed during development, it might help the developer realize if any major logical flaws exist in the application's design. Furthermore, web application flow whitelisting should be revisited any time the application changes. Any new components that are added (or any that are removed) must be accounted for and the whitelist should be adjusted to reflect the changes made.

Chapter 5

Research Methodology:

Phase2: Large Case Validation

5.1 Background

In order to validate web application flow whitelisting, we demonstrated it using a Django application that was in production use. The selection process involved searching for all the open source Django applications on Github. There were 8751 repositories that met the search criteria. We further refined the search to Django apps with Python as the primary programming language used and the results were further reduced to 5783 repositories. We manually explored the results and eliminated any apps that were merely reproducing the Django polls tutorial (the official tutorial of the Django website) as well as apps that were add-ons and not stand-alone applications. Also, we eliminated applications that had not been maintained in the last 2 years. The results were reduced to 52 applications. Appendix B contains a table detailing the 52 results in terms of size, best match, most stars, most forks, and most recently updated.

We selected a stand-alone application that is fully deployed and has been maintained recently for validating web application flow whitelisting. The selected application is a dating website written in Django version 1.8. The size of the application is 1106 LOC. Despite multiple requests, we were unable to obtain explicit permission from the application's developer to use its real name in our research. Because the app is on GitHub in a public account, it is open to public scrutiny. Nevertheless, in order to protect the identity of the developer and the application, we gave it a nonexistent domain name and for the remainder of this discussion will refer to it as seekinglove.com.

5.1.1 Overview of seekinglove.com

The main functional requirements of seekinglove.com were found by exploring the code base. The functionality extracted from the code base can be described as follows:

1. seekinglove.com is restricted to registered users only.
2. Users are registered immediately upon request.
3. Each user has a profile consisting of his/her name, email address, gender, and the gender he/she is seeking. Other profile information may be included optionally, such as birthday, home town, university, and interests.
4. A user may edit his/her profile and upload an optional picture.
5. A user may search for other users.
6. A user may add another user as a crush
7. A user may interact with other users by posting messages on his/her wall.
8. A user may unlock different icons depending on how active they are on the website.

In order to whitelist seekinglove.com we had to first construct a workflow that illustrates the allowed and disallowed behavior based on the requirements above. In our workflow, allowed behavior is depicted using solid arrows and disallowed application behavior using dashed arrows. Furthermore, rectangles with double borders are used to depict internal components or subroutines. The workflow for seekinglove is shown in figure 5.1 below.

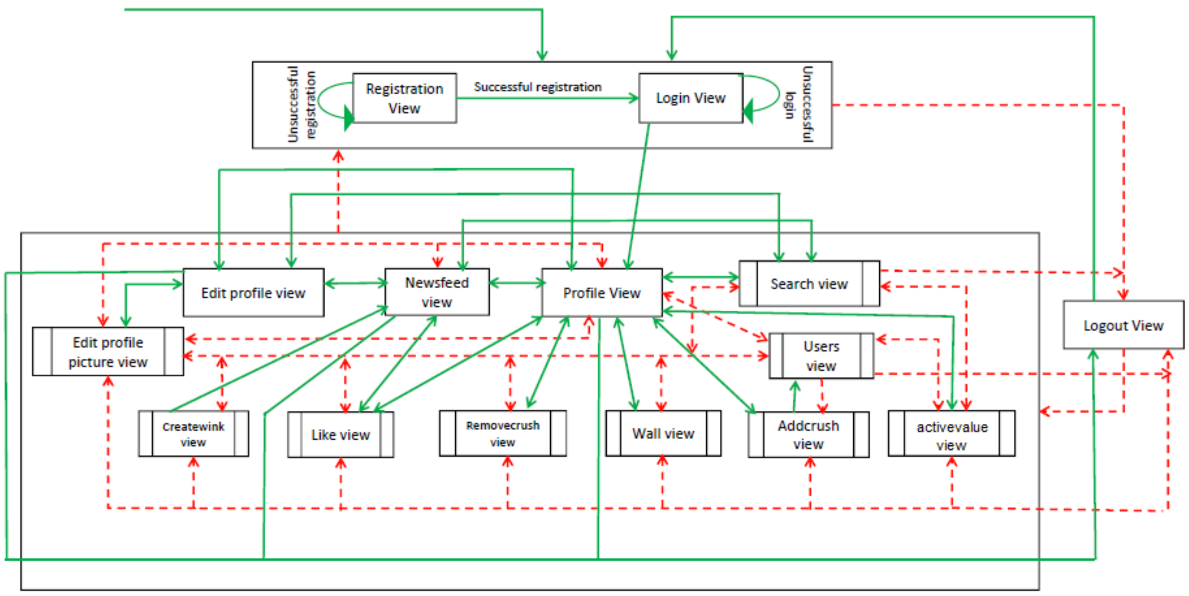


Figure 5.1: Workflow diagram for seekinglove.com

Based on the workflow above, we derived the whitelist $\langle C, D, W, S \rangle$ as follows:

Set C is the set of all components of seekinglove.com and consists of: C = (u, register, login, profile, edit_profile, edit_profilepicture, newsfeed, addcrush, removecrush, like, activevalue, wall, createwink, search, users, logout).

As for the global set of conditions D, it contains the following conditions:

- d_1: anonymous user permissions,
- d_2: current authenticated user permissions and data,
- d_3: valid session expiry time,
- d_4: previous view,
- d_5: subsequent view.
- d_6: switch = true.

- d_7: user is crush.
- d_8: unsuccessful login attempt.

For clarity, we have assigned numbers for each subset of conditions as follows (Table 5.1):

Using the legend in table 5.1, we constructed a matrix containing the numbers of the subset(s) of conditions that must be checked for each transition to succeed (table 5.2).

By evaluating the subset of conditions for each transition in 5.2, the set of ordered pairs in W are:

$$
\begin{gathered}
W = \{\, (u, u)\,, (u, register)\,, (u, login)\,, (register, u)\,, (register, register)\,, \\
(register, login)\,, (login, u)\,, (login, register)\,, (login, login)\,, \\
(login, profile)\,, (profile, profile)\,, (profile, edit\_profile)\,, (profile, newsfeed)\,, \\
(profile, Addcrush)\,, (profile, Removecrush)\,, (profile, like)\,, \\
(profile, Activevalue)\,, (profile, wall)\,, (profile, createwink)\,, (profile, search)\,, \\
(profile, logout)\,, (edit\_profile, profile)\,, (edit\_profile, edit\_profile)\,, \\
(edit\_profile, edit\_profilepicture)\,, (edit\_profile, newsfeed)\,, (edit\_profile, search)\,, \\
(edit\_profile, logout)\,, (edit\_profilepicture, edit\_profile)\,, \\
(edit\_profilepicture, edit\_profilepicture)\,, (newsfeed, profile)\,, \\
(newsfeed, edit\_profile)\,, (newsfeed, newsfeed)\,, (newsfeed, like)\,, (newsfeed, search)\,, \\
(newsfeed, logout)\,, (Addcrush, profile)\,, (Addcrush, users)\,, (removecrush, profile)\,, \\
(like, profile)\,, (like, newsfeed)\,, (Activevalue, profile)\,, (wall, profile)\,, \\
(createwink, profile)\,, (search, profile)\,, (search, edit\_profile)\,, (search, newsfeed) \\
(logout, u)\,, (logout, register)\,, (logout, login)\,\}
\end{gathered}
$$

subset #	containing
1	{d2, d3, d4 = logout}
2	{d1, d8}
3	{d2, d3}
4	{d2,d3, d4 = login}
5	{d2, d3, d4 = profile or edit_profilepicture, or newsfeed or search}
6	{d2, d3, d4 = profile or edit_profile or like or createwink or search}
7	{d2, d3, d4 = edit_profile}
8	{d2, d3, d4 = profile}
9	{d2, d3, d4 = profile, d7}
10	{d2, d3, d4 = profile or newsfeed}
11	{d2, d3, d4 = profile, d6}
12	{d2, d3, d4 = profile or edit_profile or newsfeed or like or createwink or search}
13	{d2, d3, d4 = addcrush}
14	{d2, d4 = profile or edit_profile or newsfeed}
15	{d2, d3, d4 = newsfeed}
16	{d2, d3, d4 = profile or edit_profile or newsfeed or like or search or addcrush or wall or removecrush or createwink or activevalue}
17	{d2, d3, d4 = edit_profilepicture, d5 = edit_profile}
18	{d2, d3, d4 = removecrush}
19	{d2, d3, d4 = like}
20	{d2, d3, d4 = activevalue}
21	{d2, d3, d4 = wall, d6}
22	{d2, d3, d4 = createwink, d7}
23	{d2, d3, d4 = search}

Table 5.1: Legend for numbering the subsets of conditions

	U	**Register**	**Login**	**Profile**	**Edit_profile**	**Edit_profilepicture**	**Newsfeed**	**Addcrush**	**Removecrush**	**Like**	**Activevalue**	**Wall**	**Createwink**	**Search**	**Users**	**Logout**
U		d1	d1	4	5	7	6	8	9	10	8	11	9	14	13	14
Register	d1	d1	d1	4	5	7	6	8	9	10	8	11	9	14	13	14
Login	1	2	2	4	5	7	6	8	9	10	8	11	9	14	13	14
Profile	1	d1	d1	3	8	7	8	8	9	8	8	11	9	8	13	8
Edit_profile	1	d1	d1	7	5	7	7	8	9	10	8	11	9	7	13	7
Edit_profilepicture	1	d1	d1	16	17	7	6	8	9	10	8	11	9	14	13	14
Newsfeed	1	d1	d1	15	15	7	3	8	9	15	8	11	9	15	13	15
Addcrush	1	d1	d1	13	5	7	12	8	9	10	8	11	9	14	13	14
Removecrush	1	d1	d1	18	5	7	12	8	9	10	8	11	9	14	13	14
Like	1	d1	d1	19	5	7	12	8	9	10	8	11	9	14	13	14
Activevalue	1	d1	d1	20	5	7	12	8	9	10	8	11	9	14	13	14
Wall	1	d1	d1	21	5	7	12	8	9	10	8	11	9	14	13	14
Createwink	1	d1	d1	22	5	7	12	8	9	10	8	11	9	14	13	14
Search	1	d1	d1	23	5	7	12	8	9	10	8	11	9	14	13	14
Users	1	d1	d1	13	5	7	12	8	9	10	8	11	9	14	13	14
Logout	1	d1	d1	d2	5	7	12	8	9	10	8	11	9	14	13	14

Table 5.2: Matrix W for seekinglove.com containing distinct subset(s) of conditions in each cell

Table 5.3 below shows a zero-one representation of W. Allowed flow is represented by 1 and disallowed flow is represented by 0.

	U	**Register**	**Login**	**Profile**	**Edit_profile**	**Edit_profilepicture**	**Newsfeed**	**Addcrush**	**Removecrush**	**Like**	**Activevalue**	**Wall**	**Createwink**	**Search**	**Users**	**Logout**
U	1*	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Register	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Login	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
Profile	0	0	0	1	1	0	1	1	1	1	1	1	1	1	0	1
Edit_profile	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0	1
Edit_profilepicture	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
Newsfeed	0	0	0	1	1	0	1	0	0	1	0	0	0	1	0	1
Addcrush	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0
Removecrush	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Like	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0
Activevalue	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Wall	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Createwink	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Search	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0
Users	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Logout	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5.3: A zero-one representation of W for seekinglove.com

Table 5.4 below contains the safe components to redirect to in case the evaluation of conditions fails and the transition is disallowed.

	U	Register	Login	Profile	Edit_profile	Edit_profilepicture	Newsfeed	Addcrush	Removecrush	Like	activevalue	Wall	Createwink	Search	Users	Logout
U	NA	register	login	login	login	login	login	login	login	login	login	login	login	login	login	login
Register	u	register	login	login	login	login	login	login	login	login	login	login	login	login	login	login
Login	logout	logout	login	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout
Profile	logout	logout	logout	login	login	logout	login	login	login	login	login	login	login	login	login	login
Edit_profile	logout	logout	logout	login	login	login	login	login	login	login	login	login	login	login	login	login
Edit_profilepicture	logout	logout	logout	logout	login	login	logout	logout	logout	logout	logout	logout	logout	logout	logout	login
Newsfeed	logout	logout	logout	login	login	logout	login	logout	logout	logout	logout	logout	logout	login	logout	login
Addcrush	logout	logout	logout	login	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout	login	login
Removecrush	logout	logout	logout	login	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout	login
Like	logout	logout	logout	login	logout	logout	login	logout	logout	logout	logout	logout	logout	logout	logout	login
activevalue	logout	logout	logout	login	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout	login
Wall	logout	logout	logout	login	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout	login
Createwink	logout	logout	logout	login	logout	logout	login	logout	logout	logout	logout	logout	logout	logout	logout	login
Search	logout	logout	logout	login	login	logout	login	logout	logout	logout	logout	logout	logout	logout	logout	login
Users	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout	logout	login
Logout	logout	logout	logout	login	login	logout	login	logout	logout	logout	logout	logout	logout	logout	logout	login

NA represents Not Applicable and a 1 is placed for transitions occurring outside the system boundary.
u - represents a component outside the system boundary

Table 5.4: Matrix S for seekinglove.com with safe components

5.2 Analysis

In order to whitelist seekinglove.com, we needed to first find out whether or not it had any vulnerabilities and if the vulnerabilities found were application- or non-application-specific. Because whitelisting tests for and mitigates application-specific vulnerabilities only, we used OWASP's Zed Attack Proxy (ZAP) tool to test for non-application-specific vulnerabilities. ZAP's passive scans were run on seekinglove.com. The non-application-specific vulnerabilities found are summarized table 5.5. ZAP vulnerability reports for seekinglove.com can be found in Appendix C.

Issues found by testing using ZAP passive scans

	Issue	Place of occurrence	Resolved by
1	CSRF Cookie set w/out HTTPOnly	settings.py	adding: CSRF_COOKIE_HTTPONLY = True
2	Session Cookie set w/out HTTPOnly	settings.py	adding: SESSION_COOKIE_HTTPONLY = True
3	Web browser XSS protection not enabled	settings.py	adding: SECURE_BROWSER_XSS_FILTER = True
4	Missing X-Content-Type-Options Header	settings.py	adding: SECURE_CONTENT_TYPE_NOSNIFF = True
5	X-Frame-Options Header Not Set	settings.py	adding: X_FRAME_OPTIONS = 'SAMEORIGIN'

Table 5.5: Results of ZAP passive scans on seekinglove.com

5.2.1 Application Specific Vulnerabilities in seekinglove.com

We tested seekinglove.com for application specific vulnerabilities. A total of 14 vulnerabilities were found. The vulnerabilities can be classified as follows:

1. Anonymous user accessing seekinglove.com views directly by typing in a view's URL

   Vulnerability: Being able to access the search functionality of the application by directly typing in
   127.0.0.1:8000/search/.

   Explanation: By examining the workflow of seekinglove.com (figure 5.2), views that are restricted to authenticated users such as 'search' cannot be accessed without authenticating the user first.

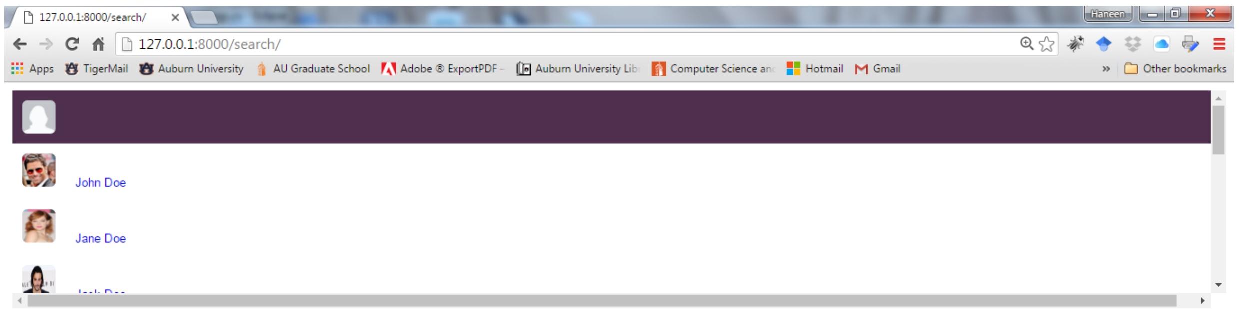


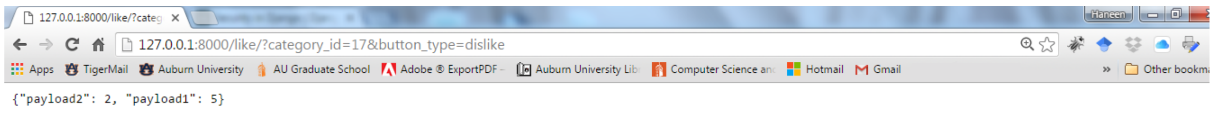
Figure 5.2: App behavior when typing in 127.0.0.1:8000/search/.

2. Authenticated user inadvertently (or illegitimately) accessing application functionality that they should otherwise be restricted from.

   Vulnerability: Being able to access functionality of internal subroutines by directly typing in a URL. When a legitimate user, for example John Doe, logs into the application, he can type in URLs directly into the browser's address bar and access internal application functionality. The vulnerabilities found in this category include the following:

(a) 127.0.0.1:8000/like/?category_id=17&button_type=dislike.

Explanation: category_id 17 is the message with number 17 in the application's database. By typing the URL directly, the application displays the following message (figure 5.3):

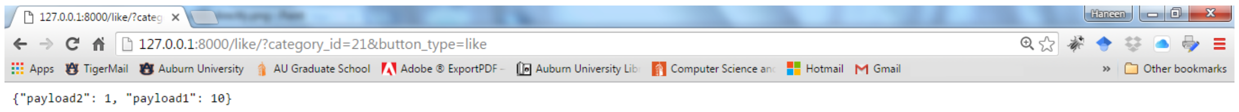


```
{"payload2": 2, "payload1": 5}
```

Figure 5.3: App behavior when typing in 127.0.0.1:8000/like/?category_id=17&button_type=dislike.

(b) 127.0.0.1:8000/like/?category_id=21&button_type=like.

Explanation: category_id 21 is the message with number 21 in the application's database. By typing the URL directly, the application displays the following message (figure 5.4) and creates a notification object that would be sent to the author of the message saying "John Doe liked your message":

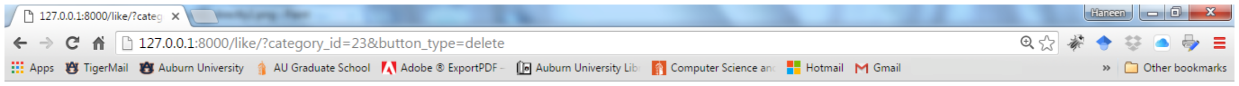


```
{"payload2": 1, "payload1": 10}
```

Figure 5.4: App behavior when typing in 127.0.0.1:8000/like/?category_id=21&button_type=like.

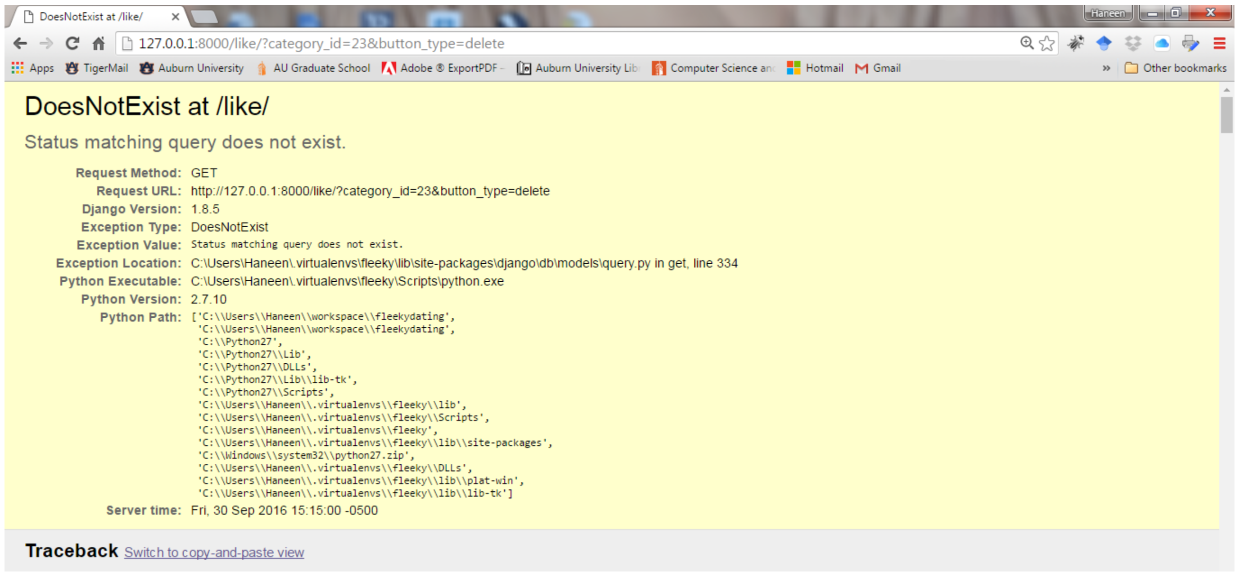
(c) 127.0.0.1:8000/like/?category_id=23&button_type=delete.

Explanation: category_id 23 is the message with number 23 in the application's database. By typing the URL directly, the application renders a blank page (figure 5.5) and when its reloaded, an error page appears (figure 5.6). The application deletes message 23, even though the currently logged in user (John Doe) did not author this message:



127.0.0.1:8000/like/?category_id=23&button_type=delete

Figure 5.5: App behavior when typing in 127.0.0.1:8000/like/?category_id=23&button_type=delete.



127.0.0.1:8000/like/?category_id=23&button_type=delete

```
DoesNotExist at /like/

Status matching query does not exist.

    Request Method: GET
       Request URL: http://127.0.0.1:8000/like/?category_id=23&button_type=delete
    Django Version: 1.8.5
    Exception Type: DoesNotExist
   Exception Value: Status matching query does not exist.
Exception Location: C:\Users\Haneen\.virtualenvs\fleeky\lib\site-packages\django\db\models\query.py in get, line 334
 Python Executable: C:\Users\Haneen\.virtualenvs\fleeky\Scripts\python.exe
    Python Version: 2.7.10
       Python Path: ['C:\\Users\\Haneen\\workspace\\fleekydating',
                     'C:\\Users\\Haneen\\workspace\\fleekydating',
                     'C:\\Python27',
                     'C:\\Python27\\Lib',
                     'C:\\Python27\\DLLs',
                     'C:\\Python27\\Lib\\lib-tk',
                     'C:\\Python27\\Scripts',
                     'C:\\Users\\Haneen\\.virtualenvs\\fleeky\\lib',
                     'C:\\Users\\Haneen\\.virtualenvs\\fleeky\\Scripts',
                     'C:\\Users\\Haneen\\.virtualenvs\\fleeky',
                     'C:\\Users\\Haneen\\.virtualenvs\\fleeky\\lib\\site-packages',
                     'C:\\Windows\\system32\\python27.zip',
                     'C:\\Users\\Haneen\\.virtualenvs\\fleeky\\DLLs',
                     'C:\\Users\\Haneen\\.virtualenvs\\fleeky\\lib\\plat-win',
                     'C:\\Users\\Haneen\\.virtualenvs\\fleeky\\lib\\lib-tk']
       Server time: Fri, 30 Sep 2016 15:15:00 -0500

Traceback Switch to copy-and-paste view
```

Figure 5.6: App behavior after reloading 127.0.0.1:8000/like/?category_id=23&button_type=delete.

(d) 127.0.0.1:8000/like/?category_id=13&button_type=wink.

Explanation: category_id 13 is the user (Jane Doe) with id number 13 in the application's database. By typing the URL directly, the application renders a blank page (figure 5.7). The application does not send a notification to user 13

(Jane Doe). It does, however, create a wink object in the database. If the user with id 13 is a ‘crush’ of John Doe’s, he/she would receive a notification that says: “John Doe winked at you!”.

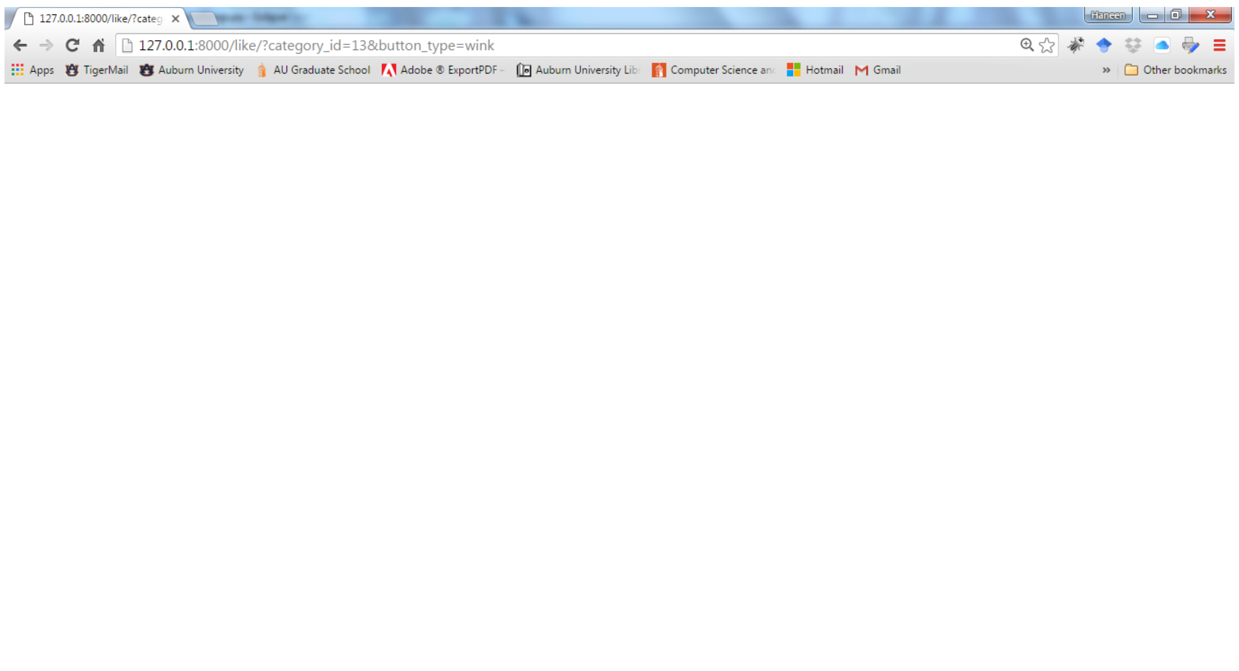
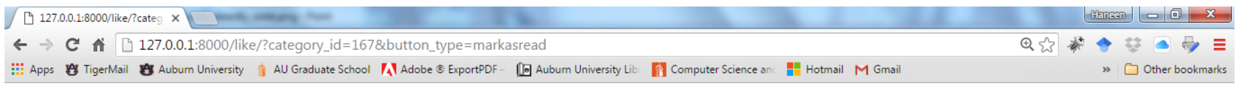


Figure 5.7: App behavior when typing in 127.0.0.1:8000/like/?category_id=13&button_type=wink.

(e) 127.0.0.1:8000/like/?category_id=167&button_type=markasread.

Explanation: category_id 167 is a notification object with id number 167 in the application’s database. By typing the URL directly, the application renders a blank page (figure 5.8). The application removes notification 167 from the notifications database table. This vulnerability can be combined with vulnerability (1) as it does not require a user to be logged in. Any adversary can type in the URL directly from the main page and a notification object would be deleted.



127.0.0.1:8000/like/?category_id=167&button_type=markasread

Figure 5.8: App behavior when typing in 127.0.0.1:8000/like/?category_id=167&button_type=markasread.

(f) 127.0.0.1:8000/like/?category_id=19&button_type=crush.

Explanation: category_id 19 is the user with id number 19 in the application's database. By typing the URL directly, the application will add user with id 19 as a crush of John Doe (see figure 5.9). Reloading the page with the same URL over and over again will continuously add user with id 19 as a crush (see figure 5.10).

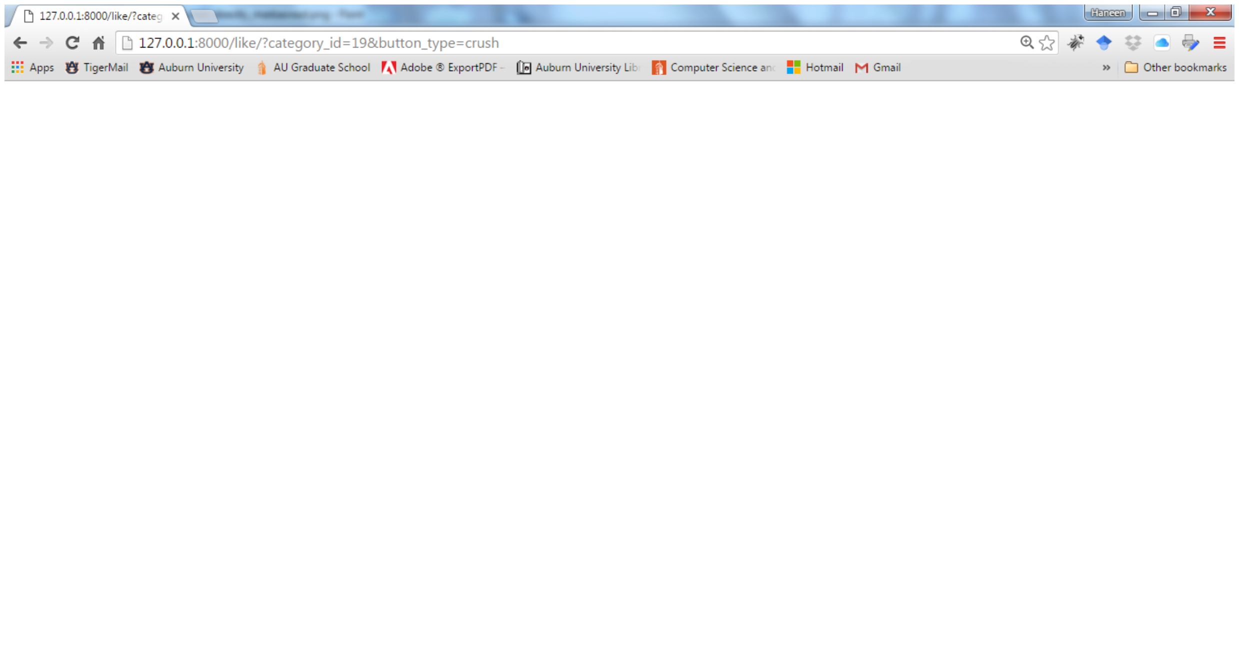
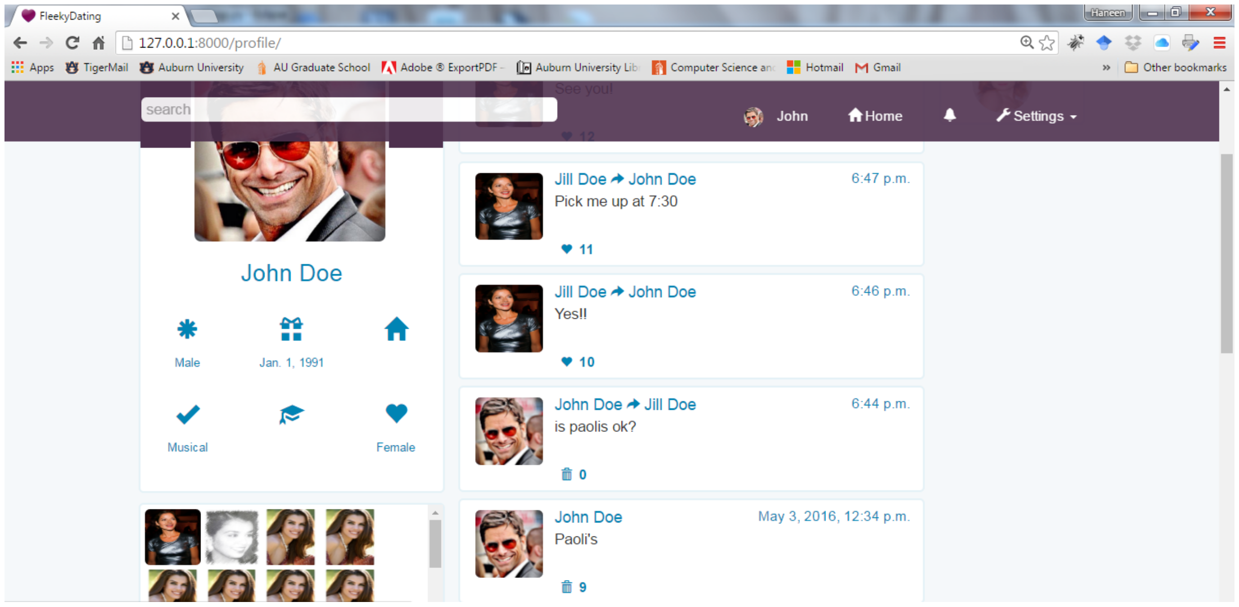


Figure 5.9: App behavior when typing in 127.0.0.1:8000/like/?category_id=19&button_type=crush.



See you!
12

Jill Doe → John Doe 6:47 p.m.
Pick me up at 7:30
11

Jill Doe → John Doe 6:46 p.m.
Yes!!
10

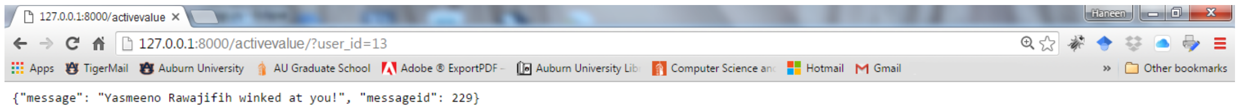
John Doe → Jill Doe 6:44 p.m.
is paolis ok?
0

John Doe May 3, 2016, 12:34 p.m.
Paoli's
9

Figure 5.10: App behavior after reloading URL 127.0.0.1:8000/like/?category_id=19&button_type=crush. several times

(g) 127.0.0.1:8000/activevalue/?category_id=13

Explanation: category_id 13 is the user with id number 13 (Jane Doe) in the application's database. By typing the URL directly, the application will show message number and content of messages that have not been viewed yet and are intended for Jane Doe (figure 5.11). Changing the category_id number will result in viewing other users' unseen messages.



```
{"message": "Yasmeeno Rawajifih winked at you!", "messageid": 229}
```

Figure 5.11: App behavior when typing in 127.0.0.1:8000/activevalue/?category_id=13.

(h) 127.0.0.1:8000/createwink/13/

Explanation: The number 13 in the URL refers to user with id number 13 (Jane Doe) in the application's database. By logging in as John Doe and typing the URL directly, the application will create a wink object in the database and add a wink icon from user John Doe, even though John Doe should not be able to send a wink to anyone who is not currently a crush of his. This is an internal subroutine that should not be accessed directly. Figures 5.12 and 5.13 below show the application's behavior before and after typing in the URL.

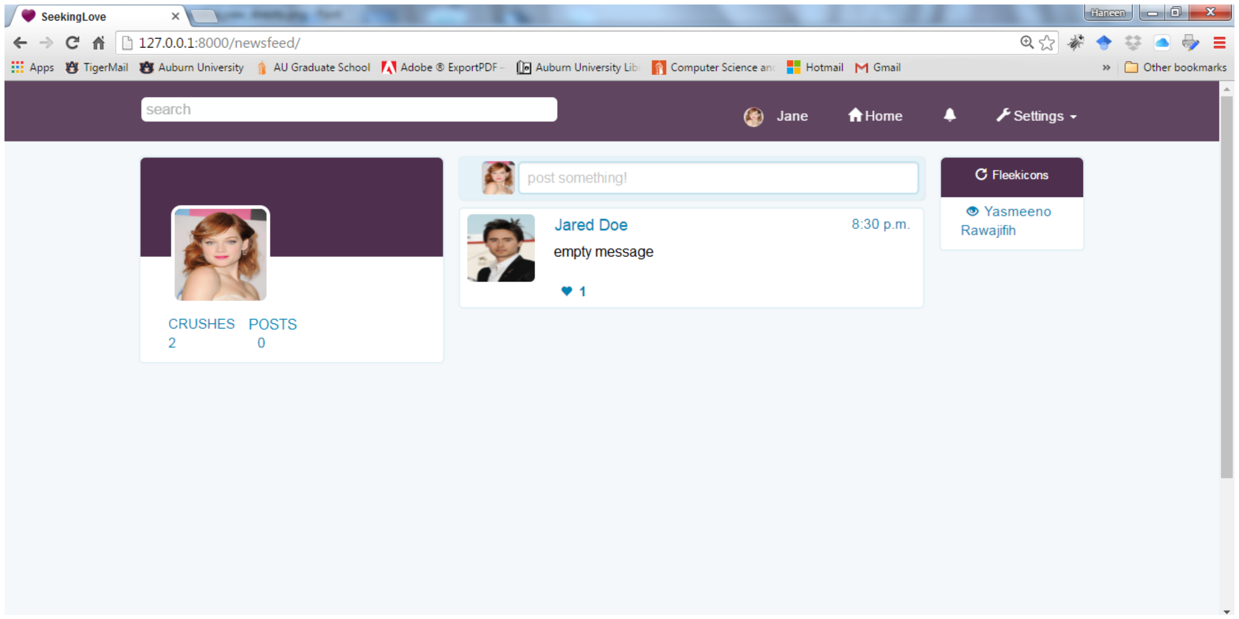


Figure 5.12: App behavior before typing in 127.0.0.1:8000/createwink/13/.

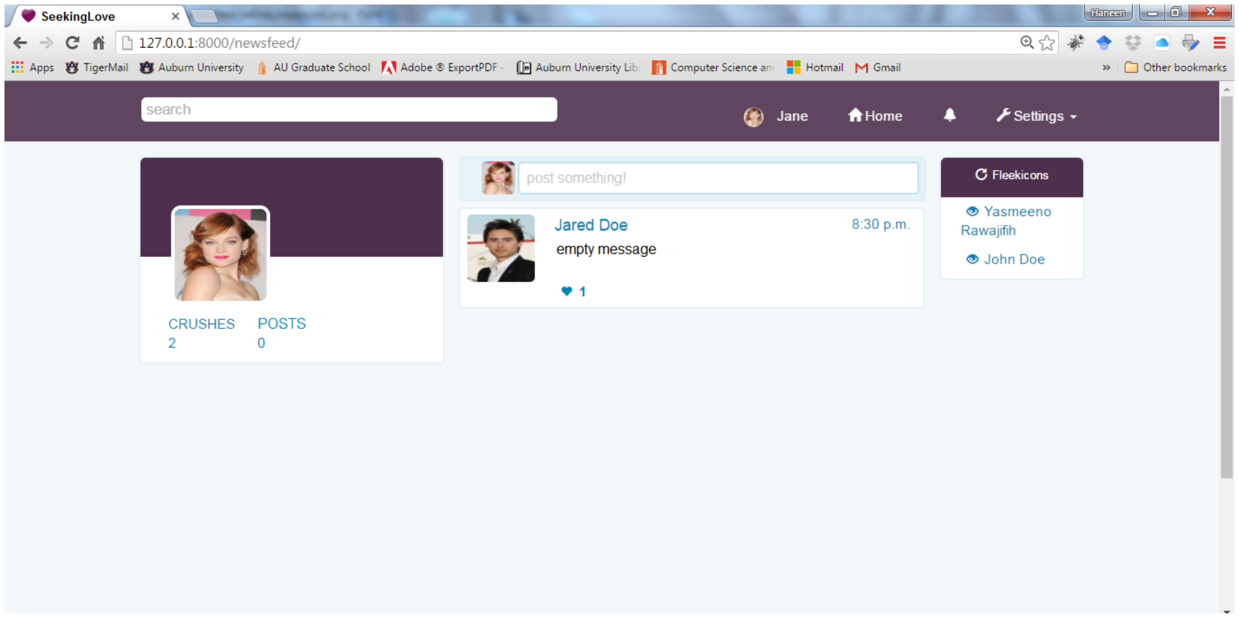
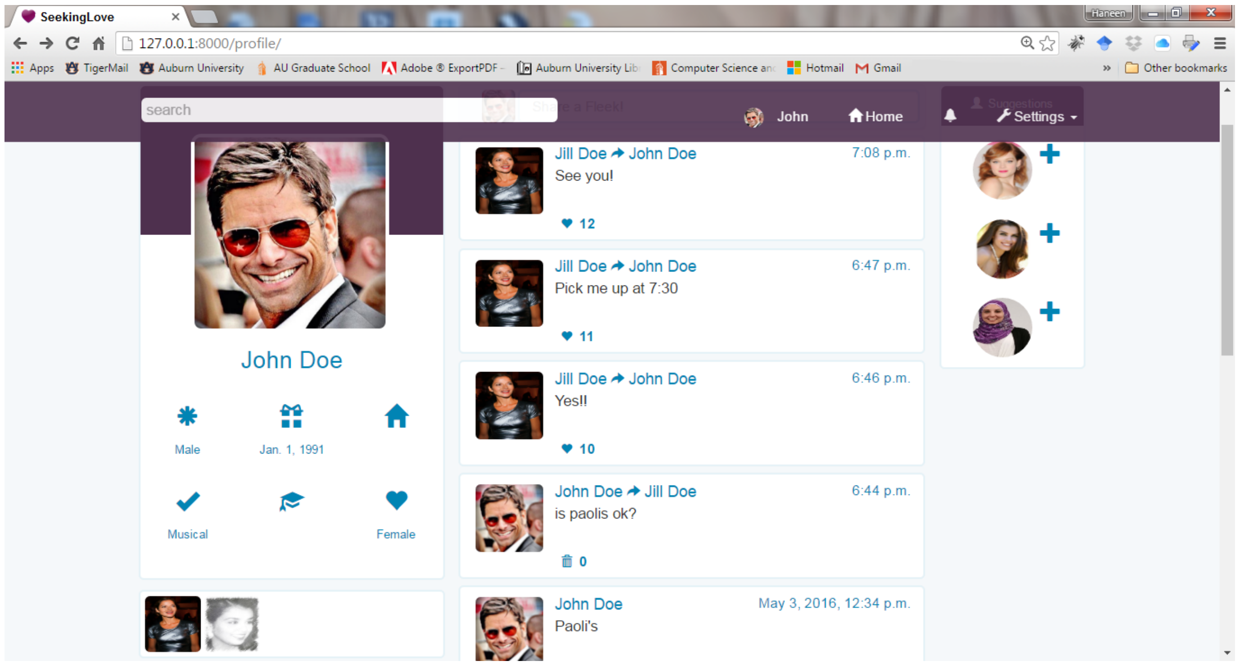


Figure 5.13: App behavior after typing in 127.0.0.1:8000/createwink/13/.

(i) 127.0.0.1:8000/addcrush/18/13/

Explanation: The number 18 in the URL refers to user with id number 18 (Jared Doe) in the application's database. By logging in as John Doe and typing the URL directly, the application will create a new crush object in the database and a new notification object will be created as well. User with id number 18 will get a notification that John Doe added him as a crush. This is an internal subroutine that should not be accessed directly. Figures 5.14, 5.15, 5.16, and 5.17 below show the application's behavior before and after typing in the URL.



John Doe

Jill Doe → John Doe 7:08 p.m.
See you!
12

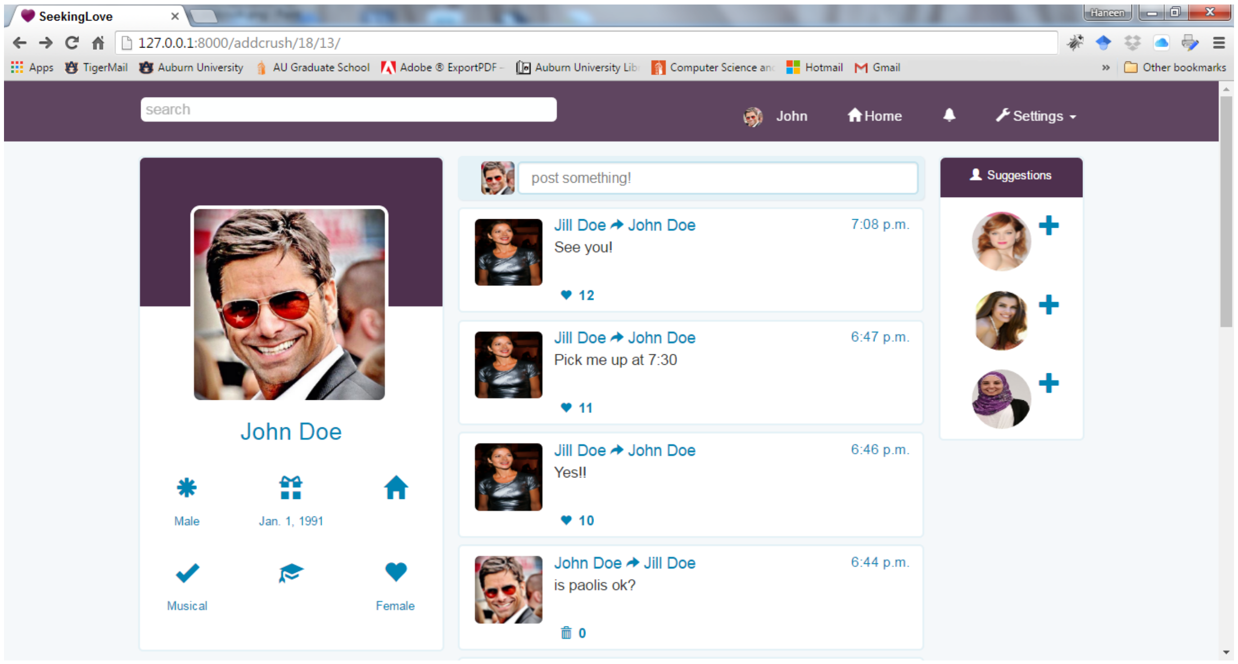
Jill Doe → John Doe 6:47 p.m.
Pick me up at 7:30
11

Jill Doe → John Doe 6:46 p.m.
Yes!!
10

John Doe → Jill Doe 6:44 p.m.
is paolis ok?
0

John Doe May 3, 2016, 12:34 p.m.
Paoli's

Figure 5.14: App behavior before typing in 127.0.0.1:8000/addcrush/18/13/.



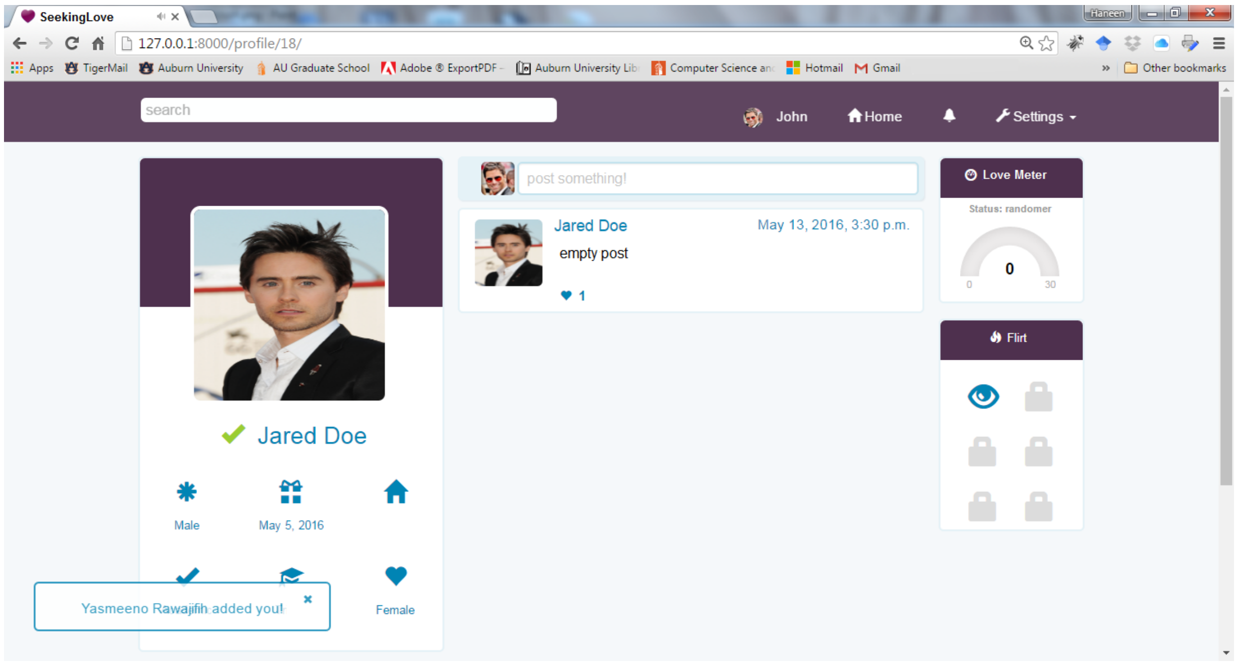
Jill Doe → John Doe 7:08 p.m.
See you!
♥ 12

Jill Doe → John Doe 6:47 p.m.
Pick me up at 7:30
♥ 11

Jill Doe → John Doe 6:46 p.m.
Yes!!
♥ 10

John Doe → Jill Doe 6:44 p.m.
is paolis ok?
0

Figure 5.15: Typing in 127.0.0.1:8000/addcrush/18/13/.



Jared Doe May 13, 2016, 3:30 p.m.
empty post
♥ 1

Yasmeeno Rawajifih added you!

Figure 5.16: App behavior after typing in 127.0.0.1:8000/addcrush/18/13/.

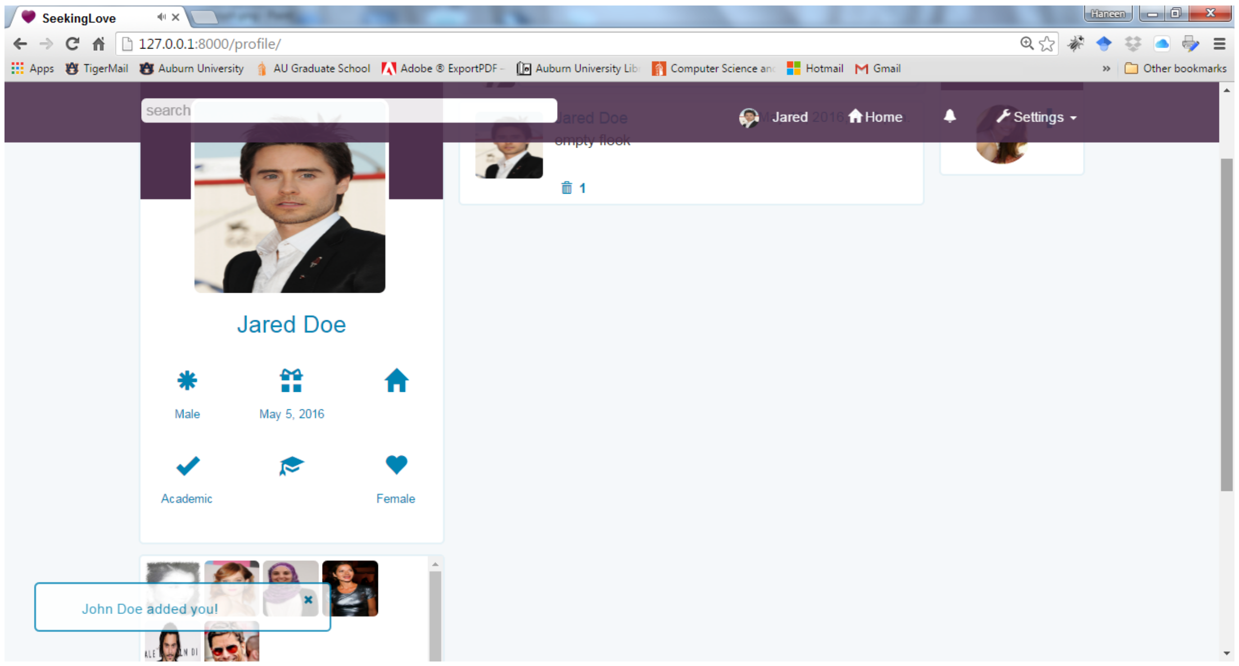
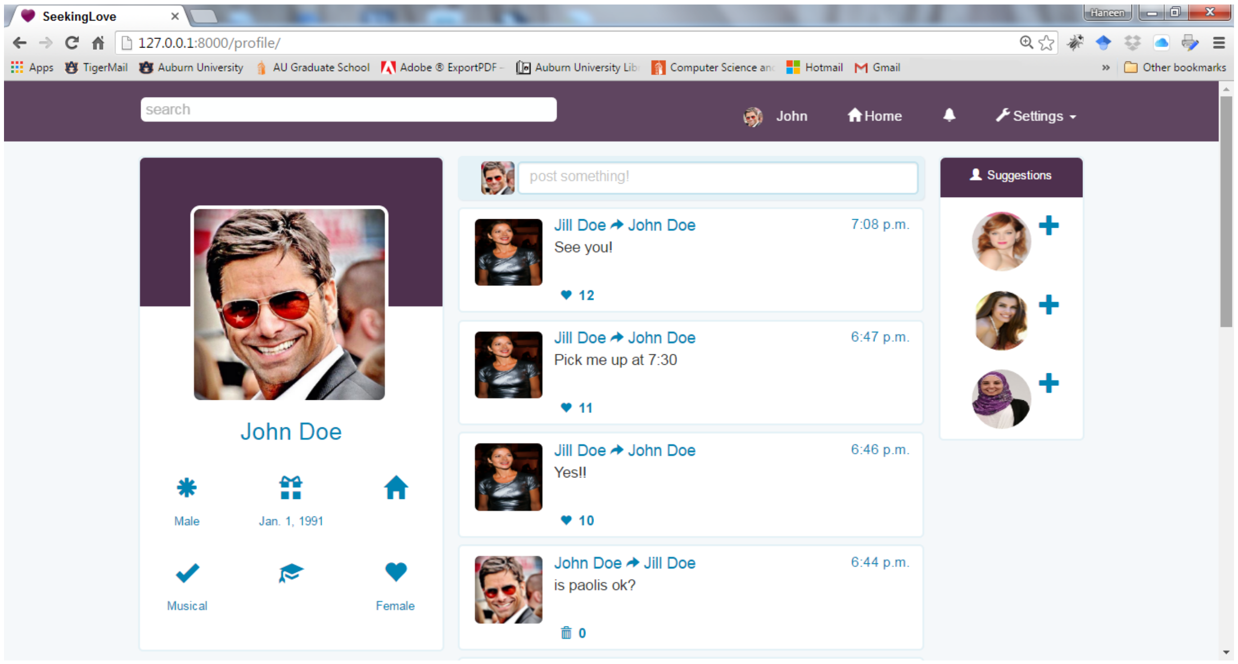


Figure 5.17: Notification sent to user 18 after in 127.0.0.1:8000/addcrush/18/13/.

(j) 127.0.0.1:8000/addcrush/12/

Explanation: The number 12 in the URL refers to user with id number 12 (John Doe) in the application's database. John Doe is also the currently logged in user. By typing the URL directly, the application will create a new crush object in the database and a new notification object will be created as well. John Doe is able to add himself as a crush and will get a notification that says: "John Doe added you as a crush". This is an internal subroutine that should not be accessed directly. Figures 5.18 and 5.19 below show the application's behavior before and after typing in the URL.



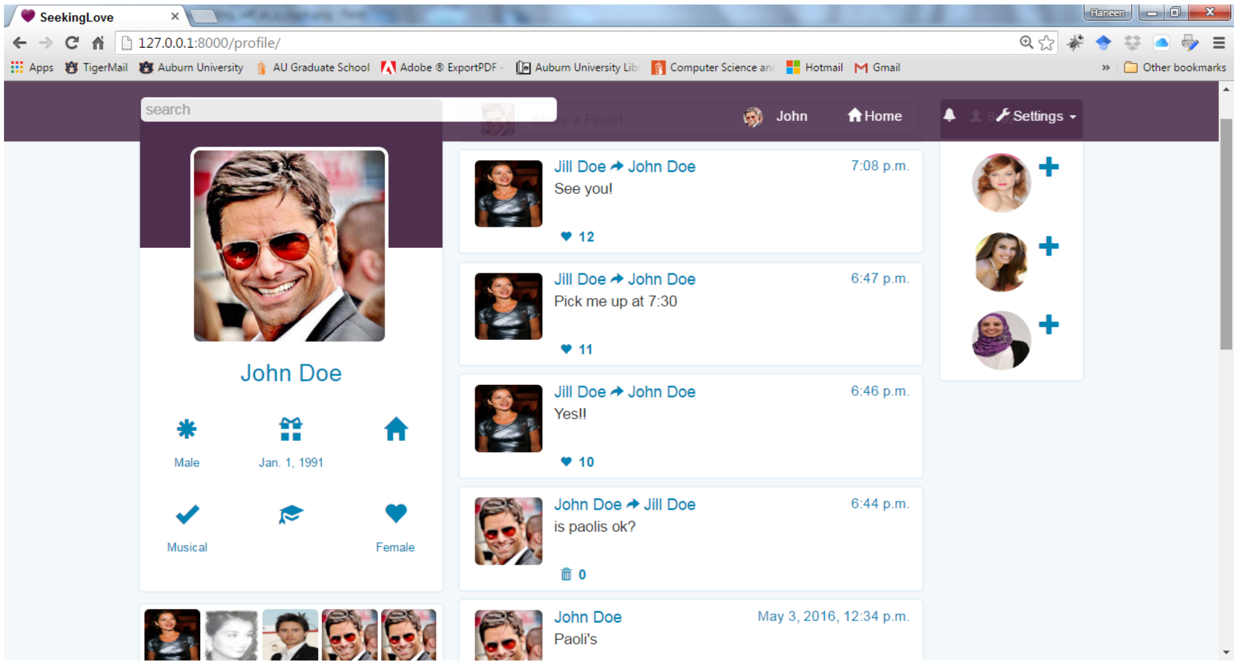
Jill Doe → John Doe 7:08 p.m.
See you!
♥ 12

Jill Doe → John Doe 6:47 p.m.
Pick me up at 7:30
♥ 11

Jill Doe → John Doe 6:46 p.m.
Yes!!
♥ 10

John Doe → Jill Doe 6:44 p.m.
is paolis ok?
🗑 0

Figure 5.18: App behavior before typing in 127.0.0.1:8000/addcrush/12/.



Jill Doe → John Doe 7:08 p.m.
See you!
♥ 12

Jill Doe → John Doe 6:47 p.m.
Pick me up at 7:30
♥ 11

Jill Doe → John Doe 6:46 p.m.
Yes!!
♥ 10

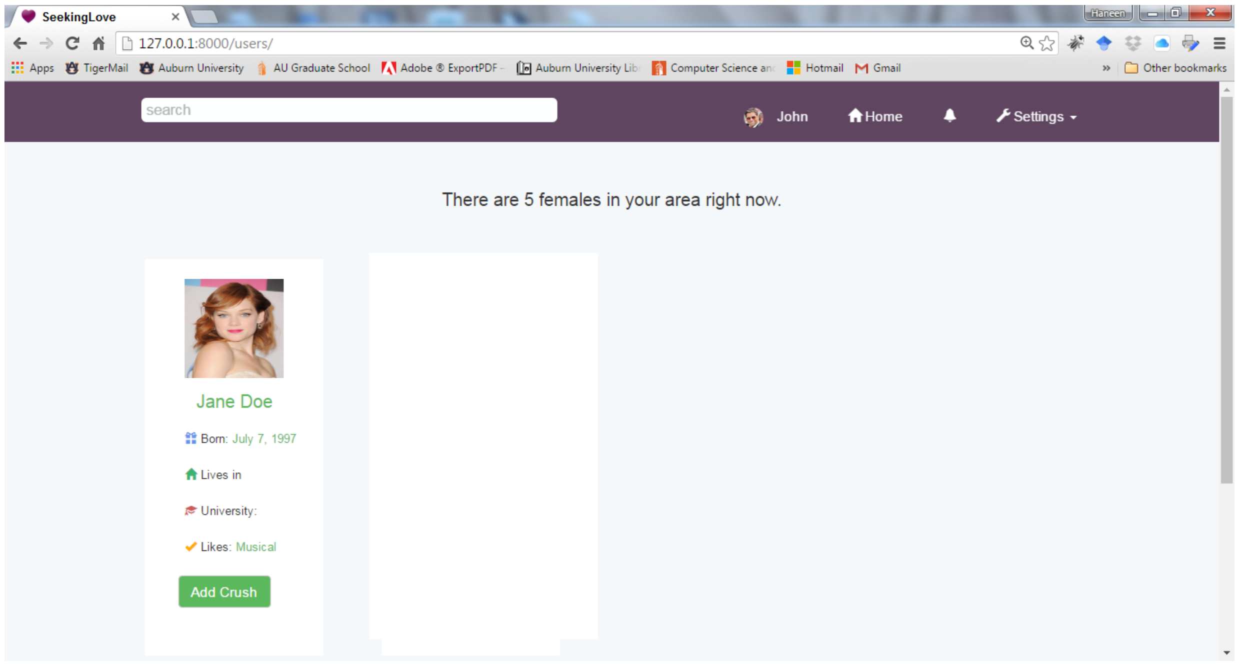
John Doe → Jill Doe 6:44 p.m.
is paolis ok?
🗑 0

John Doe May 3, 2016, 12:34 p.m.
Paoli's

Figure 5.19: App behavior after typing in 127.0.0.1:8000/addcrush/12/.

(k) 127.0.0.1:8000/addcrush/19/1/

Explanation: The number 19 in the URL refers to user with id number 19 in the application's database. The number 1 in the URL is used to redirect the application to the users view (an internal view that can be accessed only from addcrush view). By typing the URL directly, the application will display the users page. There is no direct functionality in the application that displays the users page. It is an internal subroutine that should not be accessed directly. Figure 5.20 below shows the application's behavior after typing in the URL.



There are 5 females in your area right now.

Jane Doe

Born: July 7, 1997

Lives in

University:

Likes: Musical

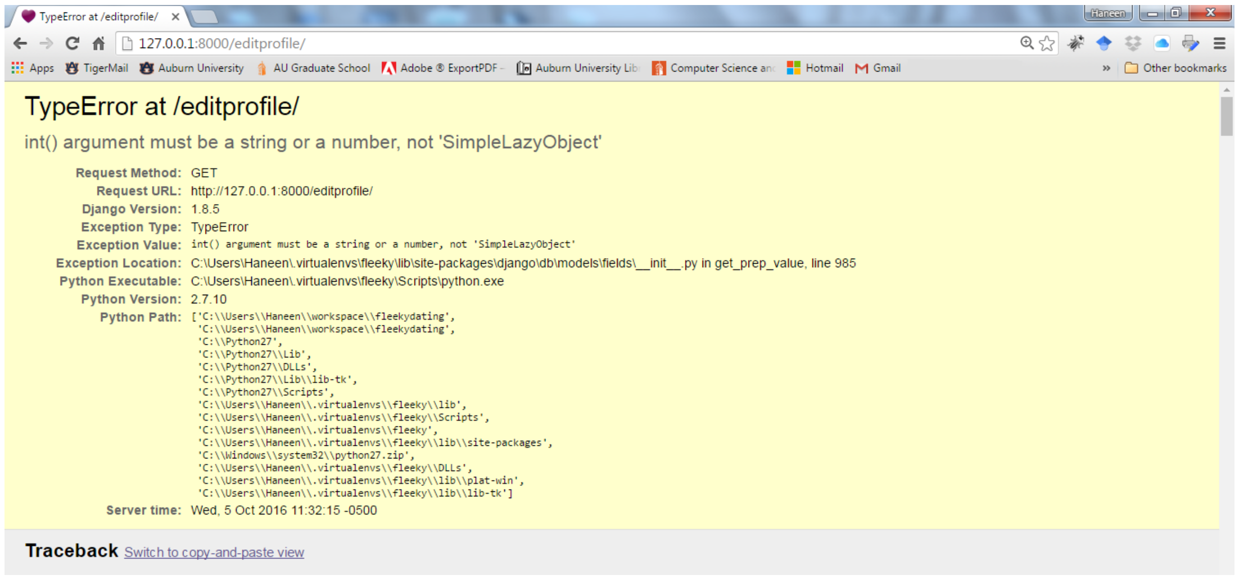
Figure 5.20: App behavior after typing in 127.0.0.1:8000/addcrush/19/1/.

3. Missing error handling

   (a) *Vulnerability:* Typing in 127.0.0.1:8000/editprofile/ when a user is logged out raises a ***TypeError: int() argument must be a string or number, not 'SimpleLazyObject'*** (5.21). This error appears because the application is trying to access the notifications objects. If we comment out the lines that pertain to accessing the notifications objects, another error appears: ***AttributeError: at /editprofile/ 'Anonymoususer' Object has no attribute '_meta'*** (5.22).

   (b) *Vulnerability:* Typing in 127.0.0.1:8000/editprofilepicture/ when a user is logged out raises the same error as in /edit/profile/ above.

Explanation: While this vulnerability does not reveal or alter the application's data, it is unintended behavior and should be addressed by the developer.

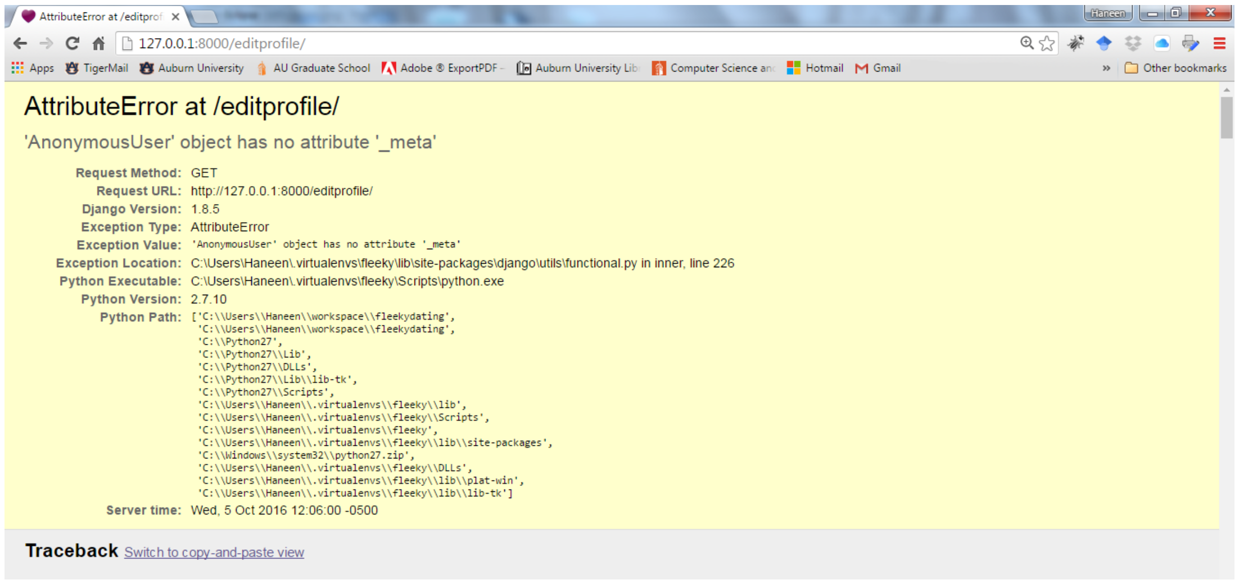


```
TypeError at /editprofile/
int() argument must be a string or a number, not 'SimpleLazyObject'

     Request Method: GET
        Request URL: http://127.0.0.1:8000/editprofile/
     Django Version: 1.8.5
     Exception Type: TypeError
    Exception Value: int() argument must be a string or a number, not 'SimpleLazyObject'
 Exception Location: C:\Users\Haneen\.virtualenvs\fleeky\lib\site-packages\django\db\models\fields\__init__.py in get_prep_value, line 985
  Python Executable: C:\Users\Haneen\.virtualenvs\fleeky\Scripts\python.exe
     Python Version: 2.7.10
        Python Path: ['C:\\Users\\Haneen\\workspace\\fleekydating',
                      'C:\\Users\\Haneen\\workspace\\fleekydating',
                      'C:\\Python27',
                      'C:\\Python27\\Lib',
                      'C:\\Python27\\DLLs',
                      'C:\\Python27\\Lib\\lib-tk',
                      'C:\\Python27\\Scripts',
                      'C:\\Users\\Haneen\\.virtualenvs\\fleeky\\lib',
                      'C:\\Users\\Haneen\\.virtualenvs\\fleeky\\Scripts',
                      'C:\\Users\\Haneen\\.virtualenvs\\fleeky',
                      'C:\\Users\\Haneen\\.virtualenvs\\fleeky\\lib\\site-packages',
                      'C:\\Windows\\system32\\python27.zip',
                      'C:\\Users\\Haneen\\.virtualenvs\\fleeky\\DLLs',
                      'C:\\Users\\Haneen\\.virtualenvs\\fleeky\\lib\\plat-win',
                      'C:\\Users\\Haneen\\.virtualenvs\\fleeky\\lib\\lib-tk']
        Server time: Wed, 5 Oct 2016 11:32:15 -0500

Traceback Switch to copy-and-paste view
```

Figure 5.21: Error before removing notifications object.



```
AttributeError at /editprofile/
'AnonymousUser' object has no attribute '_meta'

     Request Method: GET
        Request URL: http://127.0.0.1:8000/editprofile/
     Django Version: 1.8.5
     Exception Type: AttributeError
    Exception Value: 'AnonymousUser' object has no attribute '_meta'
 Exception Location: C:\Users\Haneen\.virtualenvs\fleeky\lib\site-packages\django\utils\functional.py in inner, line 226
  Python Executable: C:\Users\Haneen\.virtualenvs\fleeky\Scripts\python.exe
     Python Version: 2.7.10
        Python Path: ['C:\\Users\\Haneen\\workspace\\fleekydating',
                      'C:\\Users\\Haneen\\workspace\\fleekydating',
                      'C:\\Python27',
                      'C:\\Python27\\Lib',
                      'C:\\Python27\\DLLs',
                      'C:\\Python27\\Lib\\lib-tk',
                      'C:\\Python27\\Scripts',
                      'C:\\Users\\Haneen\\.virtualenvs\\fleeky\\lib',
                      'C:\\Users\\Haneen\\.virtualenvs\\fleeky\\Scripts',
                      'C:\\Users\\Haneen\\.virtualenvs\\fleeky',
                      'C:\\Users\\Haneen\\.virtualenvs\\fleeky\\lib\\site-packages',
                      'C:\\Windows\\system32\\python27.zip',
                      'C:\\Users\\Haneen\\.virtualenvs\\fleeky\\DLLs',
                      'C:\\Users\\Haneen\\.virtualenvs\\fleeky\\lib\\plat-win',
                      'C:\\Users\\Haneen\\.virtualenvs\\fleeky\\lib\\lib-tk']
        Server time: Wed, 5 Oct 2016 12:06:00 -0500

Traceback Switch to copy-and-paste view
```

Figure 5.22: Error after removing notifications object.

A whitelist middleware was created for seekinglove.com following the allowed workflow described in section 5.1.1 above. Below is the psuedo code for the whitelist middleware of seekinglove.com. For the actual code, please refer to Appendix D.

- In process_request() hook:

    1. Get the HTTP_REFERER attribute and the requested URL.
    2. Flag NotAllowed was set to True indicating that all behavior is initially disallowed.
    3. Admin site URLs were excluded from whitelisting.
    4. We checked for allowed behavior by:
        (a) Evaluating the subset of conditions for each transition (from HTTP_REFERER to requested URL) and if the evaluation resulted in 1, we changed flag NotAllowed to False.
        (b) Returning None for the allowed flow to continue to process_view() hook.

- In process_view() hook:

    1. We checked if flag NotAllowed was True, if it is we flushed the user session and redirected to the safe view.
    2. Else, we returned None (for the allowed transition to continue and render the desired response).

5.2.2 Results

Seekinglove.com was tested for behavior using the whitelist middleware in appendix D. We tested the whitelist to make sure it was not preventing any intended behavior or functionality. Seekinglove.com was also tested using the whitelist middleware for the vulnerabilities described in section 5.2.1 above. Table 5.6 below summarizes the vulnerabilities that were found in seekinglove.com and were successfully mitigated by the whitelist middleware.

Discussion

From the results we obtained in table 5.6, we can clearly reject the null hypothesis (section 4.4) in favor of the alternative hypothesis. We have observed that the whitelist was successful in mitigating all the vulnerabilities found. By initially setting the default value

Number	Specific vulnerability	Mitigated by whitelist
1	anonymous user can access http://127.0.0.1:8000/search/	Y
2	Directly typing in URL: 127.0.0.1:8000/like/?category_id=17&button_type=dislike	Y
3	Directly typing in URL: 127.0.0.1:8000/like/?category_id=21&button_type=like	Y
4	Directly typing in URL: 127.0.0.1:8000/like/?category_id=23&button_type=delete	Y
5	Directly typing in URL: 127.0.0.1:8000/like/?category_id=13&button_type=wink	Y
6	Directly typing in URL: 127.0.0.1:8000/like/?category_id=167&button_type=markasread	Y
7	Directly typing in URL: 127.0.0.1:8000/like/?category_id=19&button_type=crush	Y
8	Directly typing in URL: 127.0.0.1:8000/activevalue/?category_id=13	Y
9	Directly typing in URL: 127.0.0.1:8000/createwink/13/	Y
10	Directly typing in URL: 127.0.0.1:8000/addcrush/18/13/	Y
11	Directly typing in URL: 127.0.0.1:8000/addcrush/12/	Y
12	Directly typing in URL: 127.0.0.1:8000/addcrush/19/1/	Y
13	Typing in 127.0.0.1:8000/editprofile/ when a user is logged out	Y
14	Typing in 127.0.0.1:8000/editprofilepicture/ when a user is logged out	Y

Table 5.6: Vulnerabilities in seekinglove.com mitigated by whitelist middleware

of flag NotAllowed = True, the whitelist guarantees that only intended behavior is allowed and all other behavior is rejected. As was stated before, the whitelist should ideally be created during development, however, for validation purposes, it was applied on applications that were already developed. Furthermore, Django provides built-in decorators that can be added to views such the login_required decorator (please refer to appendix A). While seekinglove.com uses this decorator for some of its views, it did not prevent an authenticated user from being able to access internal functionality that they should have otherwise been restricted from. The whitelist did successfully mitigate those types of vulnerabilities. Furthermore, the whitelist was helpful in identifying and mitigating OWASP vulnerabilities A4: Insecure Direct Object References and A7: Missing Function Level Access Control.

5.2.3 Limitations

The same limitations from section 4.6.3 apply here, with the exception that there were no major logical flaws found in seekinglove.com's code base.

Chapter 6

Conclusions and Future Work

6.1 Summary

The increased usage of the Internet has mandated the presence of web-facing applications for many organizations and with that comes security considerations for those applications. The software engineering industry realizes that to fix the field of computer security, developers need to build security into their applications; a notion supported by many agencies such as CERT, NIST, and NCSD. The problem, however, is that existing secure development processes and practices are heavyweight and do not fit in with today's nimble development trend. There is a need for agile practices to address security issues present in web applications.

Many developers use web development frameworks to take advantage of pre-written common features found across many web applications, such as authentication and session management. However, developers can not rely solely on just those features to mitigate application-specific vulnerabilities. To address application-specific vulnerabilities, developers need to inspect the disparity between intended behavior and actual behavior of an application as it may be indicative of malicious use or implementation flaws. Focusing on identifying and enforcing intended behavior would aid in mitigating application specific vulnerabilities.

Our research provided a novel approach called web application flow whitelisting to build security into a web application. The approach focused on mitigating OWASP's A4 and A7 web vulnerabilities; Insecure Direct Object References and Missing Function Level Access Control. The approach also specified a whitelist of allowed flow from one component to another according to a web application's intended behavior. A formal definition of the whitelist which included operations that would need to be carried out during development and during run-time was presented. Furthermore, a description of the steps required to build the whitelist was provided.

The concepts introduced in this research were applied using the Django framework. The validation process was carried out in two phases. Phase one used a body of applications created by students in COMP4970: Web Development with Django at Auburn University. Phase two was conducted on an open source application in production use. Despite multiple attempts, we were unable to obtain explicit permission from the application's developer to use its real name in our research. Therefore, in order to protect the identity of the developer and the application, we referred to it as seekinglove.com.

The validation process included scanning the applications using a web security scanner called Zed Attack Proxy (ZAP) that targets OWASP's top ten web vulnerabilities. The ZAP scanner was used to find non application specific vulnerabilities. The vulnerabilities revealed by ZAP scans were mitigated according to the scanner's recommendations. Manual testing of the applications was carried out to identify and quantify application specific vulnerabilities. The vulnerabilities found were classified into 5 general categories:

1. Anonymous users accessing app functionality that they should otherwise be restricted from by directly typing in a URL.

2. Anonymous user accessing a previous view after another user successfully logs out by utilizing the browser's back button.

3. Authenticated user accessing another user's profile views by typing in a URL containing the other user's name or id.

4. Authenticated user inadvertently (or illegitimately) accessing application functionality that they should otherwise be restricted from.

5. Missing error handling.

Using the formal definition and steps for building a whitelist, we created static whitelists of flow and enforced them on all the applications we tested.

The results show that whitelisting web application flow was successful in producing more secure web applications and specifically mitigated OWASP's A4 and A7 web vulnerabilities that existed in the applications. We found that whitelisting web application flow addressed and mitigated the five general categories of vulnerabilities above.

There were two limitations observed when using the approach of whitelisting an application's flow:

1. The approach can not resolve any major logical flaws that exist in an application's implementation. However, if it was applied during the design and development phases of an application, it may reveal whether or not logical flaws in the design exist and can therefore be rectified before implementation takes place.

2. The validation process exposed a direct relationship between the number of components within an application and the size of the whitelist. An application with n components would have a whitelist of size n x n.

6.2 Contributions

This research aimed to find a simple approach for developers nowadays to incorporate security into their web application development efforts. The main contributions of this research are:

- **Targeting security at the Application Layer in the OSI model.** While security is in place at many locations in the lower layers of the OSI model, it is often missed at the application layer. The application layer is the layer that needs to be secured the most since it is the entry and exit point of the OSI model. This research targets the application layer and offers a simple approach for developers to build security into their applications.

- **Highlighting the ability to exploit different features in web development frameworks.** Many web development frameworks offer built-in security features, however, developers can not rely solely on these features to secure their applications. An adversary may take advantage of the way that certain features are implemented within a framework or how they are used by the developer and be able to access information that he/she should otherwise be restricted from.

- **Offering a mitigation strategy for two of the OWASP top ten web vulnerabilities.** This research offered a mitigation strategy for OWASP vulnerability A4: Insecure direct object reference; and A7: Missing function level access control.

- **Emphasizing the importance of an application's workflow and behavior.** In order to build secure web applications, developers need to understand an application's workflow and the behavior that it depicts. Intended behavior occurs when certain conditions are met and a transition from one component to another within an application succeeds. Only this behavior should be allowed by the application. Furthermore, it is equally important to instruct the application on how to behave when a transition is disallowed. This is achieved by supplying the application with safe states to redirect to when a transition fails.

6.3 Future Work

Whitelisting web application flow has unveiled several topics of significance that will guide our direction for future research. The topics for future research include:

- Extending the concept of whitelisting flow to a set of authorized web applications is an area that merits further investigation. Many web applications today are comprised of several smaller applications and third party packages. Whitelisting the flow between those applications at a higher level of abstraction may be beneficial in enhancing their security level. Furthermore, the formal definition of whitelisting flow can be applied to other types of applications not just web applications. However, the implementation aspect would need to be adjusted according to the language, platform, and other constraints that a particular application relies on.

- Creating a dynamic whitelist of web application flow is another promising area of research. By monitoring an application's behavior over a period of time, and with the aid of machine learning, a dynamic whitelist of flow can be created and adjusted according to a knowledge base of allowed behavior observed over that period of time.

- Automating the operations for creating the whitelist. The current definition of whitelisting web application flow contains static operations that are done during the development phase, such as adding an ordered pair to matrix W and entering a condition d_x in the global set of conditions D. These operations can be dynamically carried out during the course of an application's development. Moreover, the current set of conditions D, holds simple conditions only. Further research can be done to explore the possibility of making the conditions in set D compound conditions.

- Allowing more than a single option for safe state. The current web application flow whitelisting approach redirects to a single safe state when a transition fails. This feature of the whitelist may be enhanced by providing several options to choose from as a safe state to redirect to. The safe state can redirect to a subset of components, as long as they all have an equal level of access rights and permissions to data and other components.

References

[Abdel-Hamid et al., 2003] Abdel-Hamid, A. T., Tahar, S., and Aboulhamid, E. M. (2003). Ip watermarking techniques: survey and comparison. In *System-on-Chip for Real-Time Applications, 2003. Proceedings. The 3rd IEEE International Workshop on*, pages 60–65. IEEE.

[Apple, 2014] Apple (2014). Secure coding guide. https://developer.apple.com/library/mac/documentation/Security/Conceptual/SecureCodingGuide/Introduction.html. [Online; accessed 14-April-2015].

[Arends et al., 2005] Arends, R., Austein, R., Larson, M., Massey, D., and Rose, S. (2005). Dns security introduction and requirements. Technical report, RFC 4033, March.

[Ayalew et al., 2013] Ayalew, T., Kidane, T., and Carlsson, B. (2013). Identification and evaluation of security activities in agile projects. In *Secure IT Systems*, pages 139–153. Springer.

[Bass et al., 2012] Bass, L., Clements, P., and Kazman, R. (2012). *Software Architecture in Practice*. SEI Series in Software Engineering. Pearson Education.

[Bennett, 2012] Bennett, J. (2012). Django in depth. https://www.youtube.com/watch?v=t_ziKY1ayCo. [Online; accessed 27-October-2016].

[Beznosov, 2003] Beznosov, K. (2003). Extreme security engineering: On employing xp practices to achieve'good enough security'without defining it. In *First ACM Workshop on Business Driven Security Engineering (BizSec), Fairfax, VA*, volume 31.

[Bhimani, 1996] Bhimani, A. (1996). Securing the commercial internet. *Commun. ACM*, 39(6):29–35.

[Bird, 2012] Bird, J. (2012). Agile development teams can build secure software. http://software-security.sans.org/blog/2012/02/22/agile-development-teams-can-build-secure-software/. [Online; accessed 30-January-2015].

[Boström et al., 2006] Boström, G., Wäyrynen, J., Bodén, M., Beznosov, K., and Kruchten, P. (2006). Extending xp practices to support security requirements engineering. In *Proceedings of the 2006 international workshop on Software engineering for secure systems*, pages 11–18. ACM.

[Butler, 2008] Butler, S. (2008). Why you shouldn't enable the pop3 server. http://blog.sembee.co.uk/post/Why-You-Shouldnt-Enable-the-POP3-Server.aspx. [Online; accessed 13-April-2015].

[Caelli, 2007] Caelli, W. J. (2007). Application security–myth or reality? In *Information Security Practice and Experience*, pages 1–10. Springer.

[CakePHP, 2015] CakePHP (2015). Cakephp cookbook 3.x. url-http://book.cakephp.org/3.0/en/index.html. Online; accessed 21-April-2015.

[CCITT, 1991] CCITT (1991). *CCITT Recommendation X. 800: Data Communication Networks: Open Systems Interconnection (OSI); Security, Structure and Applications: Security Architecture for Open Systems Interconnection for CCITT Applications.* International Telecommunication Union. (International Telegraph and Telephone Consultative Committee).

[CERT, 2011] CERT (2011). Top 10 secure coding practices. https://www.securecoding.cert.org/confluence/display/seccode/Top+10+Secure+Coding+Practices. [Online; accessed 14-April-2015].

[Cheswick et al., 2003] Cheswick, W. R., Bellovin, S. M., and Rubin, A. D. (2003). *Firewalls and Internet security: repelling the wily hacker.* Addison-Wesley Longman Publishing Co., Inc.

[Cobb, 2013] Cobb, M. (2013). Application whitelisting vs. blacklisting: Which is the way forward? urlhttp://searchsecurity.techtarget.com/answer/Application-whitelisting-vs-blacklisting-Which-is-the-way-forward. [Online; accessed 20-January-2015].

[Danev et al., 2012] Danev, B., Zanetti, D., and Capkun, S. (2012). On physical-layer identification of wireless devices. *ACM Comput. Surv.*, 45(1):6:1–6:29.

[Davis et al., 2004] Davis, N., Humphrey, W., Redwine Jr, S. T., Zibulski, G., and McGraw, G. (2004). Processes for producing secure software. *Security & Privacy, IEEE*, 2(3):18–25.

[Dent, 2008] Dent, A. W. (2008). A survey of certificateless encryption schemes and security models. *International Journal of Information Security*, 7(5):349–377.

[Dierks and Rescorla, 2008] Dierks, T. and Rescorla, E. (2008). The transport layer security (tls) protocol version 1.2 (rfc5246). http://www.rfc-editor.org/rfc/pdfrfc/rfc5246.txt.pdf. [Online; accessed 10-April-2015].

[Django, 2015] Django (2015). Django 1.8 documentation. https://docs.djangoproject.com/en/1.8/. [Online; accessed 19-April-2015].

[Django, 2016] Django (2016). Django 1.10 documentation. https://docs.djangoproject.com/en/1.10/. [Online; accessed 26-August-2016].

[Dye et al., 2007] Dye, M., McDonald, R., and Rufi, A. (2007). *Network Fundamentals, CCNA Exploration Companion Guide.* Cisco press.

[Faria and Cheriton, 2006] Faria, D. B. and Cheriton, D. R. (2006). Detecting identity-based attacks in wireless networks using signalprints. In *Proceedings of the 5th ACM workshop on Wireless security*, pages 43–52. ACM.

[File and Ryan, 2014] File, T. and Ryan, C. (2014). Computer and internet use in the united states: 2013. http://www.census.gov/content/dam/Census/library/publications/2014/acs/acs-28.pdf. [Online; accessed 6-January-2015].

[Firewall.cx, 2012] Firewall.cx (2012). Understanding vpn ipsec tunnel mode and ipsec transport mode - what's the difference? http://www.firewall.cx/networking-topics/protocols/870-ipsec-modes.html. [Online; accessed 9-April 2015].

[Ford-Hutchinson, 2005] Ford-Hutchinson, P. (2005). Securing ftp with tls. http://tools.ietf.org/html/rfc4217. [Online; accessed 14-April-2015].

[Geneiatakis et al., 2006] Geneiatakis, D., Dagiuklas, T., Kambourakis, G., Lambrinoudakis, C., Gritzalis, S., Ehlert, S., Sisalem, D., et al. (2006). Survey of security vulnerabilities in session initiation protocol. *IEEE Communications Surveys and Tutorials*, 8(1-4):68–81.

[Giesen et al., 2013] Giesen, F., Kohlar, F., and Stebila, D. (2013). On the security of tls renegotiation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 387–398. ACM.

[Gregg and Watkins, 2006] Gregg, M. and Watkins, S. (2006). *Hack the Stack: Using Snort and Ethereal to Master the 8 Layers of an Insecure Network.* Syngress Publishing.

[Hazrati, 2012] Hazrati, V. (2012). Secure code development: A casualty with agile? http://www.infoq.com/news/2012/03/secure-code-with-agile. [Online; accessed 6-January-2015].

[Hoffman, 2015] Hoffman, C. (2015). Htg explains: What is dns cache poisoning? http://www.howtogeek.com/161808/htg-explains-what-is-dns-cache-poisoning/. [Online; accessed 10-April-2015].

[Holcomb et al., 2009] Holcomb, D. E., Burleson, W. P., and Fu, K. (2009). Power-up sram state as an identifying fingerprint and source of true random numbers. *Computers, IEEE Transactions on*, 58(9):1198–1210.

[Holovaty and Kaplan-Moss, 2009] Holovaty, A. and Kaplan-Moss, J. (2009). The django book: Version 2.0. *The Django Book*, 16.

[Incapsula, 2011] Incapsula (2011). Ddos attacks. https://www.incapsula.com/ddos/ddos-attacks/. [Online; accessed 10-April-2015].

[internetlivestats.com, 2016] internetlivestats.com (2016). Internet users. http://www.internetlivestats.com/internet-users/. [Online; accessed 6-October-2016].

[Ismail, 2012] Ismail (2012). Osi (open source interconnection) 7 layer model. https://learningnetwork.cisco.com/docs/DOC-15624. [Online; accessed 9-April-2015].

[Jeffries, 2012] Jeffries, C. (2012). Threat modeling and agile development practices. http://technet.microsoft.com/en-us/security/hh855044.aspx. [Online; accessed 16-January-2015].

[Kaplan-Moss, 2013] Kaplan-Moss, J. (2013). Building secure web apps: Python vs the owasp top 10. https://www.youtube.com/watch?feature=player_embedded&v=sra9x44lXgU. [Online; accessed 27-October-2016].

[Khandelwal, 2013] Khandelwal, S. (2013). Security risks of ftp and benefits of managed file transfer. http://thehackernews.com/2013/12/security-risks-of-ftp-and-benefits-of.html. [Online; accessed 13-April-2015].

[Koushanfar and Alkabani, 2010] Koushanfar, F. and Alkabani, Y. (2010). Provably secure obfuscation of diverse watermarks for sequential circuits. In *Hardware-Oriented Security and Trust (HOST), 2010 IEEE International Symposium on*, pages 42–47. IEEE.

[Lennon, 2014] Lennon, M. (2014). Hackers used sophisticated smb worm tool to attack sony. http://www.securityweek.com/hackers-used-sophisticated-smb-worm-tool-attack-sony. [Online; accessed 12-April-2015].

[Lim et al., 2005] Lim, D., Lee, J. W., Gassend, B., Suh, G. E., Van Dijk, M., and Devadas, S. (2005). Extracting secret keys from integrated circuits. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 13(10):1200–1205.

[Liquidweb, 2011] Liquidweb (2011). Imap vs. pop3 email. http://www.liquidweb.com/kb/imap-vs-pop3-e-mail/. [Online; accessed 13-April-2015].

[Lofstrom et al., 2000] Lofstrom, K., Daasch, W. R., and Taylor, D. (2000). Ic identification circuit using device mismatch. In *Solid-State Circuits Conference, 2000. Digest of Technical Papers. ISSCC. 2000 IEEE International*, pages 372–373. IEEE.

[Makai, 2016] Makai, M. (2016). Django. https://www.fullstackpython.com/django.html. [Online; accessed 1-September-2016].

[McGraw, 2012] McGraw, G. (2012). Gary mcgraw on software security assurance: Build it in, build it right. http://searchsecurity.techtarget.com/opinion/Gary-McGraw-on-software-security-assurance-Build-it-in-build-it-right. [Online; accessed 19-January-2015].

[Mele, 2015] Mele, A. (2015). *Django By Example*. Packt Publishing.

[Meunier, 2008] Meunier, P. (2008). Classes of vulnerabilities and attacks. *Wiley Handbook of Science and Technology for Homeland Security*.

[Microsoft, 2009a] Microsoft (2009a). Security development lifecycle for agile development. https://www.microsoft.com/en-us/SDL/Discover/sdlagile.aspx. [Online; accessed 8-January-2015].

[Microsoft, 2012] Microsoft (2012). Secure coding guidelines. https://msdn.microsoft.com/en-us/library/8a3x2b7f(v=vs.110).aspx. [Online;accessed 14-April-2015].

[Microsoft, 2014] Microsoft (2014). The osi model's seven layers defined and functions explained. https://support.microsoft.com/en-us/kb/103884. [Online; accessed 10-April-2015].

[Microsoft, 2009b] Microsoft, T. (2009b). Configuring tls and ssl for pop3 and imap4 access. https://technet.microsoft.com/en-us/library/aa997149%28v=exchg.141%29.aspx. [Online; accessed 14-April-2015].

[Möller et al., 2014] Möller, B., Duong, T., and Kotowicz, K. (2014). This poodle bites: Exploiting the ssl 3.0 fallback. https://www.openssl.org/~bodo/ssl-poodle.pdf. [Online; accessed 14-April-2015].

[Myhre, 2000] Myhre, R. (2000). *CCNA Certification: Routing Basics for Cisco Certified Network Associates Exam 640-407*. Prentice Hall PTR.

[Nicolaysen et al., 2010] Nicolaysen, T., Sasson, R., Line, M. B., and Jaatun, M. G. (2010). Agile software development: The straight and narrow path to secure software? *IJSSE*, 1(3):71–85.

[NIST, 2014] NIST (2014). New nist guidelines aim to help it system developers build security in from the ground up. http://www.nist.gov/itl/csd/sp800-160-051314.cfm. [Online; accessed 19-February-2015].

[Oppliger, 2003] Oppliger, R. (2003). *Security Technologies for the World Wide Web*. Artech House computer security series. Artech House.

[Oracle, 2014] Oracle (2014). Oracle software security assurance. http://www.oracle.com/us/support/assurance/development/secure-coding-standards/index.html. [Online; accessed 14-April-2015].

[OWASP, 2010] OWASP (2010). Owasp secure coding practices quick reference guide. https://www.owasp.org/images/0/08/OWASP_SCP_Quick_Reference_Guide_v2.pdf. [Online; accessed 14-April-2015].

[OWASP, 2013] OWASP (2013). Owasp top 10 for 2013. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=OWASP_Top_10_for_2013. [Online; accessed 2-February-2015].

[OWASP, 2014] OWASP (2014). Testing for imap/smtp injection (otg-inpval011). https://www.owasp.org/index.php/Testing_for_IMAP/SMTP_Injection_%28OTG-INPVAL-011%29. [Online; accessed 13-April-2015].

[OWASP, 2015] OWASP (2015). Welcome to owasp the free and open software security community. https://www.owasp.org/index.php/Main_Page. [Online; accessed 14-April-2015].

[OWASP, 2016] OWASP (2016). Zed attack proxy project version 2.4.3. https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project. [Online; accessed 08-September-2016].

[Pant and Khairnar, 2014] Pant, R. and Khairnar, C. (2014). A cumulative security metric for an information network. *Network*, 3(4).

[Patwari and Kasera, 2007] Patwari, N. and Kasera, S. K. (2007). Robust location distinction using temporal link signatures. In *Proceedings of the 13th annual ACM international conference on Mobile computing and networking*, pages 111–122. ACM.

[Peeters, 2005] Peeters, J. (2005). Agile security requirements engineering. In *Symposium on Requirements Engineering for Information Security*.

[Phifer, 2003] Phifer, L. (2003). Tunnel vision: Choosing a vpn – ssl vpn vs. ipsec vpn. http://searchsecurity.techtarget.com/feature/Tunnel-vision-Choosing-a-VPN-SSL-VPN-vs-IPSec-VPN. Online; accessed: 26-4-2015.

[Popeskic, 2011] Popeskic, V. (2011). Telnet attacks ways to compromise remote connection. http://howdoesinternetwork.com/2011/telnet-attacks. [Online; accessed 13-April-2015].

[Puangpronpitag and Masusai, 2009] Puangpronpitag, S. and Masusai, N. (2009). An efficient and feasible solution to arp spoof problem. In *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, 2009. ECTI-CON 2009. 6th International Conference on*, volume 2, pages 910–913. IEEE.

[Rails, 2014] Rails, R. O. (2014). Ruby on rails guides (v4.2.1). http://guides.rubyonrails.org/. Online; accessed 21-April-2015.

[Reed, 2003] Reed, D. (2003). Applying the osi seven layer network model to information security. *Sans Institute (November 2003)*.

[Rescorla, 2000] Rescorla, E. (2000). Http over tls. https://tools.ietf.org/html/rfc2818.

[SAFECode, 2011] SAFECode (2011). Fundamental practices for secure software development, 2nd ed. https://www.safecode.org/publication/SAFECode_Dev_Practices0211.pdf. [Online; accessed 20-December-2014].

[Schneier and Ranum, 2011] Schneier, B. and Ranum, M. (2011). Schneier-ranum face-off on whitelisting and blacklisting. http://searchsecurity.techtarget.com/magazineContent/Schneier-Ranum-Face-Off-on-whitelisting-and-blacklisting. [Online; accessed 20-January-2015].

[Squad, 2015] Squad, C. D. (2015). Ssh mitm downgrade. https://sites.google.com/site/clickdeathsquad/Home/cds-ssh-mitmdowngrade. [Online; accessed 10-April-2015.

[Stuttard and Pinto, 2011] Stuttard, D. and Pinto, M. (2011). *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws, 2nd Edition: Finding and Exploiting Security Flaws*. Wiley.

[Su et al., 2007] Su, Y., Holleman, J., and Otis, B. (2007). A 1.6 pj/bit 96% stable chip-id generating circuit using process variations. In *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*, pages 406–611. IEEE.

[Suh and Devadas, 2007] Suh, G. E. and Devadas, S. (2007). Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the 44th Annual Design Automation Conference*, DAC '07, pages 9–14, New York, NY, USA. ACM.

[Torunoglu and Charbon, 2000] Torunoglu, I. and Charbon, E. (2000). Watermarking-based copyright protection of sequential functions. *Solid-State Circuits, IEEE Journal of*, 35(3):434–440.

[U.S. CERT, 2013] U.S. CERT, C. E. R. T. (2013). Software assurance. https://www.us-cert.gov/sites/default/files/publications/infosheet_SoftwareAssurance.pdf. [Online; accessed 01-December-2016].

[U.S. DHS, 2011] U.S. DHS, N. C. S. D. (2011). What is build security in? https://buildsecurityin.us-cert.gov/. [Online; accessed 19-February-2015].

[Venkatramulu and Rao, 2013] Venkatramulu, S. and Rao, C. G. (2013). Various solutions for address resolution protocol spoofing attacks. *International Journal of Scientific and Research Publications*, 3(7).

[Walters, 2014] Walters, R. (2014). Cyber attacks on u.s. companies in 2014. http://www.heritage.org/research/reports/2014/10/cyber-attacks-on-us-companies-in-2014. [Online; accessed 9-January-2015].

[Wiegers, 2005] Wiegers, K. (2005). Software process improvement handbook: A practical guide. *Process Impact, Eigenverlag.*

[Wikipedia, 2015a] Wikipedia (2015a). Arpwatch — wikipedia, the free encyclopedia. [Online; accessed 8-April-2015].

[Wikipedia, 2015b] Wikipedia (2015b). Heartbleed — wikipedia, the free encyclopedia. [Online; accessed 14-April-2015].

[Wikipedia, 2015c] Wikipedia (2015c). Post office protocol — wikipedia, the free encyclopedia. [Online; accessed 13-April-2015].

[Wikipedia, 2015d] Wikipedia (2015d). Telnet — wikipedia, the free encyclopedia. [Online; accessed 13-April-2015].

[Wikipedia, 2016a] Wikipedia (2016a). Clickjacking — wikipedia, the free encyclopedia. [Online; accessed 11-September-2016].

[Wikipedia, 2016b] Wikipedia (2016b). Django (web framework) — wikipedia, the free encyclopedia. [Online; accessed 26-August-2016].

[Williams et al., 2009] Williams, L., Gegick, M., and Meneely, A. (2009). Protection poker: Structuring software security risk assessment and knowledge transfer. In *Engineering Secure Software and Systems*, pages 122–134. Springer.

[Wysopal, 2015] Wysopal, C. (2015). 3 reasons we'll finally get serious about security in 2015. http://venturebeat.com/2015/01/07/3-reasons-well-finally-get-serious-about-security-in-2015/. [Online; accessed 9-January-2015].

[Xu and Ramesh, 2008] Xu, P. and Ramesh, B. (2008). Using process tailoring to manage software development challenges. *IT Professional*, 10(4):39–45.

[Yates, 2009] Yates, R. (2009). Jumpstart django the web framework for perfectionists with deadlines. http://www.slideshare.net/ryates/jumpstart-django. [Online; accessed 06-September-2016].

Appendices

Appendix A

Security in Django

A.1 Django and the OWASP Top 10 Web Vulnerabilities

Django offers protection from several common vulnerabilities in OWASP's top 10 web vulnerabilities. We examine each vulnerability in terms of what Django offers as a mitigation strategy for that vulnerability. The information below was compiled and obtained from [Django, 2016], [Kaplan-Moss, 2013], [Bennett, 2012], :

A.1.1 A1: Injection

Django offers protection against SQL injection attacks by escaping user supplied input used in query construction by default. Django uses an Object-Relational Mapper which is its model backend; django.db.models. It must be noted though that the use of any raw queries would have to be properly escaped and handled by the developer. Developers that use NoSQL and MongoDB while safe from SQL injection attacks, are not safe from other types of injection such as javascript injection.

A.1.2 A2: Broken Authentication and Session Management

Django recommends serving all of the pages in a website over SSL to protect against Man-in-the-Middle (MITM) attacks. Django provides session management via django.contrib.sessions and a built-in middleware called django.contrib.sessions.middleware.SessionMiddleware. Django stores sessions in the application's database by default using the model django.contrib.sessions.models.Sessio
The Django sessions framework is entirely cookie-based and session ids are never displayed in URLs. Cookies contain a session ID and not the data (unless cookie-based sessions are

being utilized). Django also allows storing session data using cache-based sessions, file-based sessions, and cookie-based sessions. The developer needs to be aware that session data is signed but not encrypted when using cookie-based sessions. Another issue that arises when it comes to session security is that sub-domains within a site are able to set cookies on the client for the whole domain. This allows session fixation if cookies are permitted from sub-domains which are not controlled by trusted users. Django also provides guidelines for developers wishing to utilize the protection provided by HTTPS.

A.1.3 A3: Cross Site Scripting - XSS

Django has its own template language called the Django Template Language and comes with built-in backends. Django's template system protects against most XSS attacks by escaping characters considered dangerous to HTML. Developers need to make sure that all attribute values within HTML tags are quoted. In addition, certain tags should be used with care such as is_safe with custom template tags, the safe template tag, and mark_safe. Also, Django 1.10 comes with a boolean called 'autoescape' that controls whether HTML autoescaping is enabled and is set to True by default.

A.1.4 A4: Insecure Direct Object References

Django, out-of-the-box, does not protect against this vulnerability. It is the developer's responsibility to do so. One common approach is to use throw away identifiers known as slugs. Django provides a class called SingleObjectMixin that contains several methods to incorporate slugs such as query_pk_and_slug(). This helps in mitigating Insecure Direct Object references but is not a complete solution as it depends on the developer's implementation. For instance, if a slug is comprised of a user's first name initial and then the last name (http://example.com/profile/JDoe), it would be easy for an adversary to guess such a slug.

A.1.5 A5: Security Misconfiguration

This vulnerability is very difficult to protect from and frameworks can not provide protection for it by default. One major mistake many developers fall into is deploying or publishing their production code with the setting Debug = True. This causes error pages to display sensitive information that an adversary might take advantage of. Django documentation supplies developers with a deployment checklist to utilize before putting a app in production. Developers may automate the process using run manage.py chech –deploy.

A.1.6 A6: Sensitive Data Exposure

Django by default provides protection of sensitive data such as passwords by using the Password-Based Key Derivation Function 2 (PBKDF2), a cryptographic function for password storage and retrieval purposes. Django uses the PBKDF2 algorithm with a SHA256 hash. A developer may opt to choose other password hashing techniques and Django offers an attribute called PASSWORD_HASHERS in settings.py, where a developer can list several hashing algorithms. Django will use the first entry specified in this list.

A.1.7 A7: Missing Level Access Control

Django does not offer protection from this vulnerability. Frameworks by default can not protect against this vulnerability as it is the developer's responsibility to restrict access to certain privileged functions. While Django provides authentication out-of-the-box through django.contrib.auth, a developer should create another layer of abstraction on top of Django's authentication mechanism to handle access control. For example, an adversary might be able to access a function to reset a user's password without that user being currently logged-in. While Django does provide decorators for methods such as the login_required decorator, using it would not stop a legitimate user from resetting another user's password. There is nothing inherent in Django to prevent a view from accessing more information than it should. Access control is an issue that should be addressed in the function's internal logic.

It is the responsibility of the developer to put these checks that need to be performed in the actual code.

A.1.8 A8: Cross Site Request Forgery - CSRF

Django provides built-in protection against many types of CSRF attacks. Django has a CSRF middleware which is activated by default, however, sub-domains within a site are able to set cookies on the client for the whole domain. Sub-domains, can therefore bypass Django's CSRF protection. A CSRF secret token is required for every POST in a form and the CSRF middleware checks for this secret token. Developers have the ability to turn off this check per view using the csrf_exempt decorator, although it is highly inadvisable to do so.

A.1.9 A9: Using Components with Known Vulnerabilities

Developers who choose to utilize packages and third party libraries are responsible for ensuring that they are patched, updated, and do not have any reported vulnerabilities. Django, and other frameworks, do not not provide any protection from this type of vulnerability.

A.1.10 A10: Unvalidated Redirects and Forwards

One possible approach for protecting against this vulnerability is to declare a whitelist of allowed hosts that are authorized for redirects. Django has a setting called ALLOWED_HOSTS in settings.py where a developer can list allowed hosts to redirect to. Furthermore, in order to protect against redirect poisoning, Django offers a utility function called is_safe_url in django.utils.http. The function checks the URL and hostname to ensure the prevention of redirects to random third party sites.

A.2 Mitigation for Common Attacks

Attacks and vulnerabilities are often used interchangeably although there is a distinct difference between the two. A vulnerability refers to a weakness in the application (design flaw or an implementation bug) that might be exploited by an attacker to cause harm to the stakeholders of an application [OWASP, 2015]. An attack is the actual exploitation of a vulnerability.

A.2.1 Clickjacking

Clickjacking, also known as User Interface Redress Attack, is an attack where users are tricked into clicking on something different from what they perceive they're clicking on [Wikipedia, 2016a]. Usually a malicious site wraps the legitimate site in a frame. Django provides a middleware called X-Frame-Options which, in a browser that supports these options, can prevent a side from loading within a frame.

A.2.2 Brute Force Login

This is one of the security issues Django out-of-the-box does not protect against. In order to guard against brute force login attempts at the application layer, the developer would need to be aware of where to apply rate limiters and timeout mechanisms. While Django does not implement rate limiters out-of-the-box, there are some third party plug-ins that can be utilized to throttle authentication requests. Another form of protection would be to incorporate multiple factor authentication methods when appropriate.

A.3 Security Best Practices

Django provides a more thorough protection from OWASP's top ten vulnerabilities. For example, Django protects against SQL injections by escaping the user supplied input used to construct a SQL query. It also provides protection against Cross Site Scripting attacks, CSRF attacks, and clickjacking among others. However, developers must not rely only on what web development frameworks offer to mitigate security issues in their applications. Vulnerabilities exist at each layer of the OSI model and countermeasures to prevent them need to be present at every layer. Network administrators and IT personnel are responsible for securing the lower layers, however, security at the higher layers is the responsibility of the application developer(s). Every application developed has its own set of unique requirements, functionalities, and security considerations and that leaves the developer with various issues to tackle during development efforts. For instance, how does the application handle brute force trial attempts? While a Denial of Service attack (DoS) is possible at each level of the OSI model, the first line of defense against it is in the application. The best practices to guard against brute force trial attempts, suggest that developers use proper security controls such as limiting the number of log-in attempts and possibly utilizing multiple factor authentication methods when appropriate. Another issue of concern is information leaked from the application. Does the application unintentionally leak information? Information might leak in several places within the application such as in error messages, developer comments left in the code, or URLs. Security experts recommend that applications follow the principle of graceful error handling and not reveal unnecessary information in messages presented to the end user. Web based applications, for example, need to have special attention paid to state information carried in cookies and sessions. Pages with sensitive information must use SSL and sessions need to have short timeouts. Access to information should be addressed when designing different components and their interactions in an application. This includes users of the application, internal components to the application, and external components that use the application. The security community advises developers to employ the Least Privilege principle for dealing with access and permissions. User permissions can be managed in a structured manner using a web development framework for example, but this is not the case for component and inter-component access privileges. The developer must explicitly state which data each component has access to and can this data be inadvertently accessed by other components. Dealing with inappropriate access is entirely left at the developer's discretion. One of the OWASP Application Security Principles is that an application should be able to detect intrusions. Current best practices call for the use of Intrusion Detection Systems (IDSs) which rely on blacklisting known threats and keeping logs of security information monitored regularly. Both these approaches are insufficient in facing zero day attacks. The problem with IDSs is that it detects previously known threats and not emerging ones. As for monitoring security logs, they help in understanding how an adversary was able to illegitimately penetrate an application's defenses. They can not help in finding a security hole before it has been exploited. The developer must not rely on these practices alone, but rather build applications that are able to dynamically detect bad behavior and defend against it. In conclusion, there are many issues that need to be addressed when it comes to securing web applications. Web development frameworks such as Django provide protection against some of the most common web application vulnerabilities. However, it remains the responsibility of the developer to correctly utilize the protection methods offered by web development frameworks as well as take into account the frameworks' shortcomings and employ security best practices during development efforts.

Appendix B

Open source Django Applications on Github

	best match	most stars	most forks	recently updated	python LOC
Django-tastypie	X	X	X		13265
Django-basic-apps	X	X	X		4242
Django-rest-auth	X	X	X		1483
Django-avatar	X	X	X		912
Django-hvad	X	X			8180
django_crud	X				226
Django-appconf	X				368
Django-appsettings	X				470
Django-categories	X		X		3310
Django-schedule	X	X	X		2625
djangoappengine	X				3168
Django-locking	X				556
Openshift-django17	X				113
Django-recaptcha	X		X		284
Django-tables2	X	X	X		3744
Django-shorturls	X				343
Django-notifications	X	X	X		848

	best match	most stars	most forks	recently updated	python LOC
Django-medusa	X				465
Django-rq	X	X			1217
Django-crispy-forms		X	X		3608
Django-cors-headers		X	X		494
merchant		X	X		7808
wooey		X			4099
fabulous		X			171
Django-secure		X			853
Pinax-stripe		X	X		6084
Django-cron		X			918
djangae		X		X	12542
Flask-xxl		X			2630
Django-oauth2-provider		X			1805
Django-calaccess-raw-data			X	X	17309
Django-form-designer			X		1788
oauth2app			X		2431
Django-scheduler			X		3976
Django-chartit			X		3411
Django-ditto				X	11119
Django-frequently				X	950
django_ecommerce2				X	1712
mealy				X	1165

	best match	most stars	most forks	recently updated	python LOC
Django-canvas-oauth				X	242
Django-htk				X	30699
Django-web-app				X	140020
pipelion				X	744
chembiohub_ws				X	7222
Correctiv-ttip-barometer				X	232
Django-updown				X	442
Djangocms-blogit				X	1211
Django-edx-courseware				X	123
polls				X	234
Edc-sync				X	2800
Django-people				X	3614
SUM LOC					318275
AVG LOC					6241
MAX					140020
MIN					113
AVG W/OUT MAX					3638

Table B.1: Django Applications on Github

seekinglove	python LOC
	1106

Table B.2: Seekinglove size in python LOC

Appendix C

ZAP Passive Scan Reports

ZAP Scanning Report

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	1
Low	5
Informational	0

Alert Detail

Medium (Medium)	X-Frame-Options Header Not Set
Description	X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks.
URL	http://127.0.0.1:8000/static/css/non-responsive.css
URL	http://127.0.0.1:8000/static/css/base.css
URL	http://127.0.0.1:8000/static/css/landing.css
URL	http://127.0.0.1:8000/static/css/profile.css
URL	http://127.0.0.1:8000/static/css/newsfeed.css
URL	http://127.0.0.1:8000/static/css/introjs.css
URL	http://127.0.0.1:8000/static/js/jquery-1.11.3.js
URL	http://127.0.0.1:8000/static/js/ajax.js
URL	http://127.0.0.1:8000/static/js/justgage.js
URL	http://127.0.0.1:8000/static/js/raphael-2.1.4.min.js
URL	http://127.0.0.1:8000/static/js/intro.js
Instances	11
Solution	Most modern Web browsers support the X-Frame-Options HTTP header. Ensure it's set on all web pages returned by your site (if you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. ALLOW-FROM allows specific websites to frame the web page in supported web browsers).
Other information	At "High" threshold this scanner will not alert on client or server error responses.
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx

Low (Medium)	**Cookie set without HttpOnly flag**
Description	A cookie has been set without the HttpOnly flag, which means that the cookie can be accessed by JavaScript. If a malicious script can be run on this page then the cookie

	will be accessible and can be transmitted to another site. If this is a session cookie then session hijacking may be possible.
URL	http://127.0.0.1:8000/
Parameter	csrftoken=631cqjNN59eqWf3yTAP1kiObtv1sgnQs; expires=Mon, 10-Apr-2017 14:38:06 GMT; Max-Age=31449600; Path=/
Evidence	csrftoken=631cqjNN59eqWf3yTAP1kiObtv1sgnQs; expires=Mon, 10-Apr-2017 14:38:06 GMT; Max-Age=31449600; Path=/
URL	http://127.0.0.1:8000/
Parameter	csrftoken=LgTyYmdWZfhd9gWBPDAvGeIVCrTcvVUv; expires=Mon, 10-Apr-2017 14:38:06 GMT; Max-Age=31449600; Path=/
Evidence	csrftoken=LgTyYmdWZfhd9gWBPDAvGeIVCrTcvVUv; expires=Mon, 10-Apr-2017 14:38:06 GMT; Max-Age=31449600; Path=/
Instances	2
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	www.owasp.org/index.php/HttpOnly
WASC Id	13

Low (Medium)	**Cross-Domain JavaScript Source File Inclusion**
Description	The page at the following URL includes one or more script files from a third-party domain
URL	http://127.0.0.1:8000/
Parameter	https://getbootstrap.com/assets/js/ie-emulation-modes-warning.js
Evidence	https://getbootstrap.com/assets/js/ie-emulation-modes-warning.js
URL	http://127.0.0.1:8000/
Parameter	https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js
Evidence	https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js
URL	http://127.0.0.1:8000/
Parameter	https://getbootstrap.com/dist/js/bootstrap.min.js
Evidence	https://getbootstrap.com/dist/js/bootstrap.min.js
URL	http://127.0.0.1:8000/
Parameter	https://getbootstrap.com/assets/js/ie10-viewport-bug-workaround.js
Evidence	https://getbootstrap.com/assets/js/ie10-viewport-bug-workaround.js
Instances	4
Solution	Ensure JavaScript source files are loaded from only trusted sources, and the sources can't be controlled by end users of the application
Reference	

Low (Medium)	**Web Browser XSS Protection Not Enabled**
Description	Web Browser XSS Protection is not enabled, or is disabled by the configuration of the 'X-XSS-Protection' HTTP response header on the web server
URL	http://127.0.0.1:8000/
URL	http://127.0.0.1:8000/robots.txt
URL	http://127.0.0.1:8000/sitemap.xml
URL	http://127.0.0.1:8000/static/css/non-responsive.css
URL	http://127.0.0.1:8000/static/css/base.css
URL	http://127.0.0.1:8000/static/css/landing.css
URL	http://127.0.0.1:8000/static/css/profile.css
URL	http://127.0.0.1:8000/static/css/newsfeed.css
URL	http://127.0.0.1:8000/static/css/introjs.css
URL	http://127.0.0.1:8000/static/js/jquery-1.11.3.js
URL	http://127.0.0.1:8000/static/js/ajax.js
URL	http://127.0.0.1:8000/static/js/justgage.js
URL	http://127.0.0.1:8000/static/js/raphael-2.1.4.min.js
URL	http://127.0.0.1:8000/static/js/intro.js
URL	http://127.0.0.1:8000/login/
Instances	15
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
Other information	The X-XSS-Protection HTTP response header allows the web server to enable or disable the web browser's XSS protection mechanism. The following values would attempt to enable it: X-XSS-Protection: 1; mode=block X-XSS-Protection: 1; report=http://www.example.com/xss The following values would disable it: X-XSS-Protection: 0 The X-XSS-Protection HTTP response header is currently supported on Internet Explorer, Chrome and Safari (WebKit). Note that this alert is only raised if the response body could potentially contain an XSS payload (with a text-based content type, with a non-zero length).
Reference	https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

	https://blog.veracode.com/2014/03/guidelines-for-setting-security-headers/
CWE Id	933
WASC Id	14

Low (Medium)	**Password Autocomplete in browser**
Description	AUTOCOMPLETE attribute is not disabled in HTML FORM/INPUT element containing password type input. Passwords may be stored in browsers and retrieved.
URL	http://127.0.0.1:8000/
Parameter	input
Evidence	<input id="id_password" name="password" placeholder="password" type="password" />
Instances	1
Solution	Turn off AUTOCOMPLETE attribute in form or individual input elements containing password by using AUTOCOMPLETE='OFF'
Reference	http://msdn.microsoft.com/library/default.asp?url=/workshop/author/forms/autocomplete_ovr.asp
CWE Id	525

Low (Medium)	**X-Content-Type-Options Header Missing**
Description	The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.
URL	http://127.0.0.1:8000/
URL	http://127.0.0.1:8000/robots.txt
URL	http://127.0.0.1:8000/sitemap.xml
URL	http://127.0.0.1:8000/static/css/non-responsive.css
URL	http://127.0.0.1:8000/static/css/base.css
URL	http://127.0.0.1:8000/static/css/landing.css
URL	http://127.0.0.1:8000/static/css/profile.css
URL	http://127.0.0.1:8000/static/css/newsfeed.css
URL	http://127.0.0.1:8000/static/css/introjs.css
URL	http://127.0.0.1:8000/static/js/jquery-1.11.3.js
URL	http://127.0.0.1:8000/static/js/ajax.js
URL	http://127.0.0.1:8000/static/js/justgage.js

URL	http://127.0.0.1:8000/static/js/raphael-2.1.4.min.js
URL	http://127.0.0.1:8000/static/js/intro.js
URL	http://127.0.0.1:8000/login/
Instances	15
Solution	Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages. If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.
Other information	This issue still applies to error type pages (401, 403, 500, etc) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scanner will not alert on client or server error responses.
Reference	http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx https://www.owasp.org/index.php/List_of_useful_HTTP_headers
WASC Id	15

ZAP Scanning Report

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	1
Low	3
Informational	0

Alert Detail

Medium (Medium)	X-Frame-Options Header Not Set
Description	X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks.
URL	http://127.0.0.1:8000/static/css/non-responsive.css
URL	http://127.0.0.1:8000/static/css/base.css
URL	http://127.0.0.1:8000/static/css/landing.css
URL	http://127.0.0.1:8000/static/css/profile.css
URL	http://127.0.0.1:8000/static/css/newsfeed.css
URL	http://127.0.0.1:8000/static/css/introjs.css
URL	http://127.0.0.1:8000/static/js/jquery-1.11.3.js
URL	http://127.0.0.1:8000/static/js/ajax.js
URL	http://127.0.0.1:8000/static/js/justgage.js
URL	http://127.0.0.1:8000/static/js/raphael-2.1.4.min.js
URL	http://127.0.0.1:8000/static/js/intro.js
Instances	11
Solution	Most modern Web browsers support the X-Frame-Options HTTP header. Ensure it's set on all web pages returned by your site (if you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. ALLOW-FROM allows specific websites to frame the web page in supported web browsers).
Other information	At "High" threshold this scanner will not alert on client or server error responses.
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx

Low (Medium)	**Cross-Domain JavaScript Source File Inclusion**
Description	The page at the following URL includes one or more script files from a third-party domain
URL	http://127.0.0.1:8000/

URL	http://127.0.0.1:8000/
Parameter	https://getbootstrap.com/assets/js/ie-emulation-modes-warning.js
Evidence	https://getbootstrap.com/assets/js/ie-emulation-modes-warning.js
URL	http://127.0.0.1:8000/
Parameter	https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js
Evidence	https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js
URL	http://127.0.0.1:8000/
Parameter	https://getbootstrap.com/dist/js/bootstrap.min.js
Evidence	https://getbootstrap.com/dist/js/bootstrap.min.js
URL	http://127.0.0.1:8000/
Parameter	https://getbootstrap.com/assets/js/ie10-viewport-bug-workaround.js
Evidence	https://getbootstrap.com/assets/js/ie10-viewport-bug-workaround.js
Instances	4
Solution	Ensure JavaScript source files are loaded from only trusted sources, and the sources can't be controlled by end users of the application
Reference	

Low (Medium)	**Web Browser XSS Protection Not Enabled**
Description	Web Browser XSS Protection is not enabled, or is disabled by the configuration of the 'X-XSS-Protection' HTTP response header on the web server
URL	http://127.0.0.1:8000/static/css/non-responsive.css
URL	http://127.0.0.1:8000/static/css/base.css
URL	http://127.0.0.1:8000/static/css/landing.css
URL	http://127.0.0.1:8000/static/css/profile.css
URL	http://127.0.0.1:8000/static/css/newsfeed.css
URL	http://127.0.0.1:8000/static/css/introjs.css
URL	http://127.0.0.1:8000/static/js/jquery-1.11.3.js
URL	http://127.0.0.1:8000/static/js/ajax.js
URL	http://127.0.0.1:8000/static/js/justgage.js
URL	http://127.0.0.1:8000/static/js/raphael-2.1.4.min.js

URL	http://127.0.0.1:8000/static/js/intro.js
Instances	11
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
Other information	The X-XSS-Protection HTTP response header allows the web server to enable or disable the web browser's XSS protection mechanism. The following values would attempt to enable it: X-XSS-Protection: 1; mode=block X-XSS-Protection: 1; report=http://www.example.com/xss The following values would disable it: X-XSS-Protection: 0 The X-XSS-Protection HTTP response header is currently supported on Internet Explorer, Chrome and Safari (WebKit). Note that this alert is only raised if the response body could potentially contain an XSS payload (with a text-based content type, with a non-zero length).
Reference	https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet https://blog.veracode.com/2014/03/guidelines-for-setting-security-headers/
CWE Id	933
WASC Id	14

Low (Medium)	**X-Content-Type-Options Header Missing**
Description	The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.
URL	http://127.0.0.1:8000/static/css/non-responsive.css
URL	http://127.0.0.1:8000/static/css/base.css
URL	http://127.0.0.1:8000/static/css/landing.css
URL	http://127.0.0.1:8000/static/css/profile.css
URL	http://127.0.0.1:8000/static/css/newsfeed.css
URL	http://127.0.0.1:8000/static/css/introjs.css
URL	http://127.0.0.1:8000/static/js/jquery-1.11.3.js
URL	http://127.0.0.1:8000/static/js/ajax.js
URL	http://127.0.0.1:8000/static/js/justgage.js
URL	http://127.0.0.1:8000/static/js/raphael-2.1.4.min.js

URL	http://127.0.0.1:8000/static/js/intro.js
Instances	11
Solution	Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages. If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.
Other information	This issue still applies to error type pages (401, 403, 500, etc) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scanner will not alert on client or server error responses.
Reference	http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx https://www.owasp.org/index.php/List_of_useful_HTTP_headers
WASC Id	15

Appendix D

Whitelist Middleware For Seekinglove.com

```
'''
Created on May 18, 2016

@author: Haneen
'''
# whitelist middleware to intercept requests and redirect to a safe component if
request is not within
# the application's intended behavior
from django.conf import settings
from django.http import HttpResponseRedirect
from django.core.urlresolvers import reverse
from django.template import RequestContext
from django.shortcuts import render_to_response
from accounts.views import *
from django.template.context_processors import request
from accounts.forms import UserRegistrationForm, UserLoginForm
#from django.contrib.sites.shortcuts import get_current_site

class WhitelistMiddleware(object):

    def process_request(self, request):

        referrer = request.META.get('HTTP_REFERER')
        requested_url = request.path_info


        # assume initially that all requests are not allowed
        request.notallowed = True

        # exclude admin site urls from this middleware
        if request.path.startswith(reverse('admin:index')):
            request.notallowed =  False
            return None

        # to account for error: AttributeError at / ... 'NoneType' object has no
        attribute 'startswith'
        if referrer == None:
            referrer = '|'

        else:

            register_allowed_referrers = ['http://127.0.0.1:8000/']
            login_allowed_referrers = ['http://127.0.0.1:8000/',  '
            http://127.0.0.1:8000/login/']
            login_requested_urls = ['/login/', '/profile/']
            profile_allowed_referrers = ['http://127.0.0.1:8000/', '
            http://127.0.0.1:8000/profile/', 'http://127.0.0.1:8000/editprofile/', '
            http://127.0.0.1:8000/newsfeed/',
                                            'http://127.0.0.1:8000/search/']
            profile_requested_urls = ['/profile/', '/editprofile/', '/newsfeed/',
            '/search/', '/activevalue/', '/like/']


            # to allow access to media pictures
            if request.path.startswith('/media/'):

                request.notallowed = False
                return None
            #allowing flow to registration view from unknown component when user is
            anonymous
            if request.user.is_anonymous():
                if (referrer in register_allowed_referrers) and ((requested_url ==
                '/register/') or (requested_url == '/') or
                                                                (requested_url ==
                                                                '///logout/')):

                    request.notallowed = False
                    return None

                # allowing login from specific views
                elif ((referrer in login_allowed_referrers) and (requested_url in
                login_requested_urls)):

                    request.notallowed = False
                    return None
                #stopping direct access to createwink
                elif request.method == "GET" and 'createwink' in request.GET:

                    request.notallowed = True
                    return None

                else:
                    request.notallowed = True
            else:

                # being able to logout from any view within the profile portal
                if ((referrer in profile_allowed_referrers) and (requested_url ==
                '///logout/' or requested_url == '//logout/')):

                    request.notallowed = False
                    return None
                #allowing access between internal profile views
                elif ((referrer in profile_allowed_referrers or referrer.startswith('
                http://127.0.0.1:8000/profile/')) and (requested_url in
                profile_requested_urls or requested_url.startswith('/profile/')
                            or requested_url.startswith('/addcrush/') or
                            requested_url.startswith('/removecrush/')
                            or requested_url.startswith('/wall/'))):

                    request.notallowed = False
                    return None

                # to allow access to activevalue
                elif referrer.startswith('http://127.0.0.1:8000/activevalue/') and
                requested_url.startswith('/activevalue/'):

                    request.notallowed = False
                    return None
                # to allow access from profile/ activevalue to newsfeed and
                edit_profile and profile
                elif referrer.startswith('http://127.0.0.1:8000/activevalue/') and
                requested_url in profile_requested_urls:

                    request.notallowed = False
                    return None


                else:
                    # if a transition from one view to another is not specified in the above whitelist, then request.notallowed flag to set to True
                    request.notallowed = True


    def process_view(self, request, view_func, view_args, view_kwargs):
        referrer = request.META.get('HTTP_REFERER')
        requested_url = request.path_info

        if request.notallowed:

            request.session.flush()

            #returning the safe view
            return render_to_response('landing.html', {'form': UserRegistrationForm(), 'form2': UserLoginForm()}, context_instance=RequestContext(request))
        else:

            return None




```

Appendix E

PhD. Research Progress Timeline

PhD Research Progress Timeline

The following timeline is for the purpose of documenting and tracking my research progress over the past 3 years.

Hyperlinks in this document are private.

#	Date	Description
1	12-Feb-14	**IDEA:** TDD-Like approach to achieve security as a Non-Functional Requirement **NOTES:** Use a sekeltal SW process and add necessary security practices **APPROACH:** Start with a minimal instance of PCSE and document any necessary tailoring for achieving security **RESOURCE:** Haneen's Quest for Finding A Research Topic
2	24-Feb-14	**IDEA:** Tailor an instance of PCSE for security as a NFR **NOTES:** prepare presentation of the idea for PCSE reasearch group **APPROACH:** MSA - Analyze --> set security goals and identify assets MSA - Architect --> Threat Modeling with STRIDE MSA - Architect --> Threat Modeling with STRIDE MSA - Construct --> TDD-like approach to conduct pen tests alongside functionality tests MSA - Interpret --> How can I evaluate achieving security as a NFR??? **RESOURCE:** Secure TDD Skeleton Process
3	10-Mar-14	**PLAN:** Apply ideas above to a real SW development project **NOTES:** Document the instance of PCSE & indicate reasons behind any deviations from it. **APPROACH:** Use project Need-A-Nerd to apply idea
4	9-May-14	**GOAL:** Learn Django from now to the end of the semester. **RESOURCE:** django_bookmarks (Django Project)
5	27-Aug-14	**DEADEND:** Validating a tailored process for security is currently unfeasible **PLAN:** Brainstorm for a different idea
6	2-Sep-14	**QUESTION:** Can a SW system monitor its own security & detect possible breaches before they happen? **RESOURCE:** DASADA Project Analysis. Mandak and Stowell
7	8-Sep-14	**IDEA:** Asset/Component/Threat Matrix (ACT-Matrix) **NOTES:** From matrix you should be able to elicit: 1) Threat frequency / number per component 2) Asset value **APPROACH:** Construct working example on Need A Nerd projects. Come up with a formula to measure component's likelihood of breach or req'd security attention.
8	12-Sep-14	**PLAN:** Review how can 'good enough' security be measured **RESOURCE:** Methodolgy for evaluating security controls based on key performance indicators and stake holder mission

IDEA: Measure risk per asset factoring in an asset's value
IDEA: link effect of a proposed threat to the component that must mitigate it
NOTES:

1) An asset may have a different value according to different threats it faces
2) A threat may carry a different value for different assets
3) Fill an asset/threat matrix similar to Protection Poker AV(A1, T1) - value of asset 1 in light of existing threat 1
4) Fill a threat/component matrix TC(T1, C1) - values populated by approx. impact

RESOURCES: Adapting secure TROPOS for Security Risk Management
TRIKE

NOTES: Threats are never technology specific but attacks are

1) An asset's intiial value is calculated from taking avg. of value/stakeholder
2) A component's value is taken from effort req'd for development; acquisition cost if the component was outsourced
3) Fill an asset/threat matrix similar to Protection Poker AV(A1, T1) - value of asset 1 in light of existing threat 1
4) Fill a threat/component matrix TC(T1, C1) - values populated by approx. impact

9 7-Oct-14 **QUESTION:** How can vulnerabilties be linked to attacks?
NOTES:

1) is there a gap between vulnerabilities and attacks
2) is there a gap between implementation and design

10 8-Oct-14 **IDEA:** (Reevaluated) Asset/Component/Threat Matrix (ACT-Matrix)
RESOURCE: See ACT Estimation Matrix Document

11 28-Oct-14 **DEADEND:** Validating the efficacy of ACT Extimation Matrix is currently unfeasible

12 4-Nov-14 **PLAN:** Understand innerworkings of Django to extract its capabilities and shortcomings
RESOURCE: See Django in Depth

13 28-Nov-14 **IDEA:** Whitelist a Web Application's flow according to intended behavior
NOTES:

1) Test idea on Need A Nerd Django student apps
2) Identify access control of different Django views to data fields
3) Incorporate user permissions into whitelisted flow
   See What security issues should a developer be concerned with

14 4-Dec-14 **PLAN:** Work on dissertation proposal for whitelisting an app's flow

15 16-Jan-15 **QUESTION:** Can whitelisting a web application's flow improve security?

16 21-Jan-15 **PLAN:** Write Introduction Chapter of Proposal
RESEARCH QUESTIONS:

1) What are the security best practices used nowadays in web app development?
2) What are the vulnerabilities that target web applications?
3) What mitigation strategies are offered in modern web frameworks?
4) What vulnerabilities must be addressed by the developer?
5) How can whitelisting an app's flow be utilized by the developer to address

		vulnerabilities not mitigated by the framework 6) Would whitelisting a web application's flow improve its security? **PROPOSAL WRITING PROGRESS:** See Proposal draft 1/21/2015 See Proposal draft 1/26/2015 See Proposal draft 1/28/2015
17	29-Jan-15	**PLAN:** Outline for dissertation proposal **NOTES:** 1) State the problem 2) Give a big picture for the whitelisting idea 3) State the specific part the dissertation will focus on - Scope 4) State any assumptions made 5) Describe validation method (proof of feasibility) 6) Indicate research goals and how they'll be achieved **PROPOSAL WRITING PROGRESS:** See Proposal draft 2/4/2015
18	12-Feb-15	**PLAN:** Investigate state info of a request in Django **NOTES:** 1) Use OSI model to separate concerns between developer and network admin 2) What are the major threats that occur at each level of the OSI 3) Web frameworks deal with levels 5 and 6; developer deals with level 7 **PROPOSAL WRITING PROGRESS:** See Proposal draft 2/18/2015 See Proposal draft 2/25/2015 See Proposal draft 3/4/2015 See Proposal draft 3/11/2015
19	12-Mar-15	**PLAN:** Add section called Research Description **NOTES:** 1) Include research objectives 2) Describe whitelisting and how it is accomplished 3) Describe how to articulate and enforce the whitelist 4) Indicate what happens when the whitelist is violated 5) Describe validation methodology **PROPOSAL WRITING PROGRESS:** See Proposal draft 3/18/2015
20	19-Mar-15	**PLAN:** Write Literature Review chapter **NOTES:** 1) Describe security at every level in the OSI model 2) Decribe OWASP's top 10 web vulnerabilities 3) Show what web frameworks protect against and what they don't protect against 4) Re-arrange proof of concept example: Show detailed example of a compromised view - what information was leaked? Show same example with whitelisted flow and how it prevents the vulnerability **PROPOSAL WRITING PROGRESS:** See Proposal draft 4/1/2015
21	3-Apr-15	**PLAN:** Send out doodle request for proposal defense date **PROPOSAL WRITING PROGRESS:** See Proposal draft 4/8/2015 See Proposal draft 4/15/2015
22	16-Apr-15	**PLAN:** Map OWASP's secure coding practices with Django security controls **NOTES:**

		1) Construct a table that maps every OWASP secure coding practice to a Django security control 2) Create comparison figure of Django/Rails/CakePHP Vs. OWASP top 10 3) Discuss at end of Literature Review what common vulnerabilities are mitigated by frameworks out of the box 4) Confirmed proposal date and time: May 6th, 2015. 2-3pm **PROPOSAL WRITING PROGRESS:** See Proposal draft 4/21/2015 See Proposal draft 4/22/2015 See Proposal draft 4/26/2015
23	27-Apr-15	**PLAN:** Work on Proposal Defense Presentation **NOTES:** 1) Limit presentation to approx. 20 slides 2) Email final dissertation proposal to committee 3) Email invitation to proposal defense scheduled on May 6th, 2015 at 2pm **PROPOSAL WRITING PROGRESS:** See Final Proposal 4/29/2015 **PROPOSAL DEFENSE SLIDES:** See Presentation Draft 5/1/2015 See Presentation Draft 5/4/2015 See Final Proposal Presentation 5/5/2015
24	6-May-15	**PROPOSAL DEFENSE: 2-3 pm**
25	12-May-15	**PLAN:** Work on a paper about Using Whitelisted Flow Between Application Components
26	25-Aug-15	**NOTES:** Get feedback on paper **RESOURCE:** Employing Whitelists in Interactions Between Application Components
27	3-Sep-15	**PLAN:** Participate in School of Engineering Graduate Research Showcase **NOTES:** Registration deadline 9/14/15
28	7-Sep-15	**PLAN:** Research and Choose Web Vulnerability Scanning Tool **NOTES:** Choice: Zed Attack Proxy (ZAP) **RESOURCES:** See Vulnerability Scanning Tools See Zed Attack Proxy (ZAP)
29	10-Sep-15	**PLAN:** Work on a formal definition for the Whitelist AND Poster for research showcase **FORMAL DEFINITION PROGRESS:** Formal Definition Version 1 Formal Definition Version 2 Formal Definition Version 3 Formal Definition Version 4 Formal Definition Version 5 Formal Definition Version 6 Formal Definition Final Version **RESEARCH SHOWCASE POSTER:** See Poster Draft 1 See Poster Draft 2 See Poster Final Version
30	2-Nov-15	**PLAN:** Scan Need A Nerd student projects with ZAP and document vulnerability reports **NOTES:**

		1) Research what OWASP vulnerabilities ZAP tests for 2) Conduct passive testing using ZAP 3) Conduct active testing using ZAP (proxy through ZAP) **ZAP SCANNING REPORTS:** Need A Nerd Scan Reports for all Team Projects
31	2-Dec-15	**PLAN:** Rework paper on whitelisting a web application's flow **STATUS:** Awaiting advisor's comments **RESOURCE:** Whitelisting Paper Version 1
32	14-Jan-16	**PLAN:** Create Generic Workflow for All Need A Nerd Apps **NOTES:** 1) Make a list of features/ functionality implemented by team projects 2) Construct workflow based on functionality implemented by most teams 3) Apply mitigation policy for non-application specific vulnerabilities found using ZAP **RESOURCES:** NaN Implemented Functionality List ZAP Results for all NaN Projects NaN Workflow Initial Draft NaN Workflow Final Draft
33	8-Feb-16	**PLAN:** Conduct Manual Testing on All NaN Apps to Find Application Specific Vulnerabilities AND Create Whitelisting Middleware for Every NaN Project. **NOTES:** 1) Test for all disallowed flow as indicated in NaN Workflow 2) Document findings for every project tested 3) Create whitelist middleware for every NaN project 4) Retest NaN projects for all disallowed flow as indicated in NaN Workflow 5) Document findings for every project tested with whitelist middleware enabled **RESOURCE:** Whitelist Middleware Classes for All NaN Projects
34	23-Mar-16	**GRADUATION DEFFERED TO SPRING 2017**
35	28-Mar-16	**PLAN:** Find Open Source Django App in Production Use to Whitelist **NOTES:** 1) Search for a Django App in DjangoSites 2) Search for a Django App in Github **POTENTIAL APP:** Codesters **DEADEND:** App not maintained **POTENTIAL APP:** OSQA **DEADEND:** App's codebase is extremely large **POTENTIAL APP:** Parsifal **DEADEND:** App does not contain many features **RESOURCE:** Django Apps on Github LOC Count
36	11-Apr-16	**CHOSEN APP:** seekinglove.com **RESOURCE:** Codebase for seekinglove.com
37	15-Apr-16	**PLAN:** Scan seekinglove.com with ZAP and document vulnerability reports AND Create workflow based on intended behavior. **NOTES:** 1) Test seekinglove.com using ZAP to find non-application specific vulnerabilities 2) Document results. 3) Apply mitigation policy for non-application specific vulnerabilities found using ZAP

		4) Extract features from codebase review
		5) Construct workflow based on app features and functionality
		RESOURCES: seekinglove.com ZAP Scan Report 1
		seekinglove.com ZAP Scan Report 2
		seekinglove.com settings file
		seekinglove.com Workflow
38	20-Apr-16	**PLAN:** Conduct Manual Testing on seekinglove.com to Find Application Specific Vulnerabilities AND Create a whitelisting middleware for it.
		NOTES:
		1) Test for all disallowed flow as indicated in Workflow
		2) Document application specific vulnerabilities found through manual testing
		3) Create whitelist middleware for seekinglove.com
		4) Retest the application for all disallowed flow as indicated in its Workflow
		5) Document results from testing the application with whitelist middleware enabled
		RESOURCES: seekinglove.com application-specific vulnerabilities
		seekinglove.com whitelist middleware class
39	30-May-16	**PLAN:** Contact seekinglove.com developer to get approval on using his work
		NOTES: Email sent to developer 5/31/2016
40	7-Jul-16	**PLAN:** Contact seekinglove.com developer to get approval on using his work
		NOTES: 2nd attempt - Email sent to developer 7/7/2016
41	17-Aug-16	**PLAN:** Rewrite Chapter 3 - Web Application Flow Whitelisting
		NOTES:
		1) Restate problem
		2) Decribe Wwb application flow whitelisting
		3) Include formal definition as well as how a whitelist would be built
		4) Include a generic example to illustrate the concept.
		DISSERTATION WRITING PROGRESS: See Dissertation draft 8/25/2016
42	19-Aug-16	**PLAN:** Ask Dr. Skjellum to join my PhD committee
		REASON: Due to Dr. Overbey's departure from CSSE; Dr. Skjellum will be the 4th committee member
43	25-Aug-16	**PLAN:** Write Chapter 4 - Research Methodology and Validation
		NOTES:
		1) Incorporate Dr. Umphress's suggested changes in Ch.3
		2) Add section: Assumptions
		3) Include background on Django's architecture
		4) Describe writing custom middleware for web application flow whitelisting
		5) Write validation section for Need A Nerd apps
		DISSERTATION WRITING PROGRESS: See Dissertation draft 9/8/2016
		See Dissertation draft 9/12/2016
		See Dissertation draft 9/15/2016
		See Dissertation draft 9/20/2016
		See Dissertation draft 9/21/2016
44	19-Sep-16	**NOTE:** Committee member officially changed.

45	22-Sep-16	**PLAN:** Write Chapter 5 - Validation for seekinglove.com **NOTES:** 1) Change title of Ch.4 to Research Methodology Phase 1: Small Case Validation 2) Write title of Ch.5 as Research Methodology Phase 2: Large Case Validation 3) Describe process for selecting seekinglove.com 4) Give an overview of seekinglove.com functionality 5) Describe applying the formal definition of whitelisting to seekinglove.com **DISSERTATION WRITING PROGRESS:** See Dissertation draft 9/30/2016 See Dissertation draft 10/6/2016
46	11-Oct-16	**PLAN:** Make minor adjustments to: **NOTES:** 1) Emphasize that whitelisting targeted A4 and A7 vulnerabilities 2) Update statistics of internet users in intro and abstract 3) Group vulnerabilities under missing error handing 4) Ch. 6 must havea contributions section
47	25-Oct-16	**PLAN:** Write Chapter 6 - Conclusion and Future Work **NOTES:** 1) Incorporate Dr. Umphress's suggested changes in Ch.4 and 5 2) Start Ch. 6 with a brief summary of this research 3) Write contributions section 4) Write future work section 5) Fix margin issues in bibliography entries with urls **DISSERTATION WRITING PROGRESS:** See Dissertation draft 10/28/2016 See Dissertation draft 10/31/2016
48	15-Nov-16	**NOTES:** Work on security in Django paper/practices table
49	29-Nov-16	**PLAN:** Prepare to submit dissertation to committee for review **NOTES:** 1) University Reader: Contact Dr. Jerry Davis in Industrial Engineering to be the reader 2) Write acknowledgements section 3) Email initial dissertation to committee members and graduate school for review **DISSERTATION WRITING PROGRESS:** See Dissertation for committee review 12/6/2016
50	24-Jan-17	**PLAN:** Set final defense date and time **NOTES:** 1) Dry run set for Mon, 1/30/2017 at 1 pm 2) Send out doodle poll to committee for week of 2/12/17 - 2/17/17
51	7-Feb-17	**PLAN:** Get feedback on whitelisting paper **NOTES:** 1) Incorporate Dr. Umphress's suggested changes in whitelisting paper 2) Make sure paper and figures are inline with submission guidelines
52	9-Mar-17	**PLAN:** Submit whitelisting paper **NOTES:** Submitted 3/20/2017

53	2-Apr-17	**PLAN:** Work on PhD Progress Timeline **NOTES:** 1) Proof read dissertation and prepare for ETD submission 2) Complete PhD Progress Timeline appendix 3) Deadline for ETD final submission: **4/24/2017**