2016

A user driven cloud based multisystem malware detection system

Brian Steven Cain
Iowa State University

A user driven cloud based multisystem malware detection system

by

Brian Steven Cain

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Information Assurance

Program of Study Committee:
Doug Jacobson, Major Professor
Manimaran Govindarasu
Diane Thiede Rover

Iowa State University

Ames, Iowa

2016

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
ARPANET	Advanced Research Projects Agency Network
CAPTCHA	Completely Automated Public Turning test to tell Computers and Humans Apart
CPU	Computer Processing Unit
DNS	Domain Name Service
GB	Gigabyte
GHz	Gigahertz
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IE	Internet Explorer
IP	Internet Protocol
ISP	Internet Service Provider
LTS	Long Term Support
MB	Megabyte
MMDS	Multi-system Malware Detection System
OS	Operating System
PDF	Portable Document Format
RAM	Random Access Memory
RPM	Rounds Per Minute
SAM	Security Account Manager
SATA	Serial ATA
SSH	Secure Shell
URL	Uniform Resource Locator
VM	Virtual Machine

ACKNOWLEDGMENTS

I would like thank Patrick Morrissey and Kelli Wolfe for always having an open door, and volunteering hours at the whiteboards helping me solve problems and issues as I encountered them. Both Patrick and Kelli have provided me with mentoring and guidance that have helped me in the journey towards completion of a Master's Degree.

I would also like to thank my committee: Doug Jacobson, Manamaran Govindarasu, and Diane Thede Rover for supporting my graduate studies and serving on my committee.

Finally and foremost, I would like to thank my friend and partner, Meredith Cain. Her support and understanding of the many hours dedicated to this research project over the past year has been one of the factors that kept me motivated and moving towards completion.

ABSTRACT

Using compromised or malicious sites to launch attacks against client systems is a growing attack vector in today's threat landscape. Attackers are able to stand up new sites at an alarming rate while client systems are constantly evolving, and exposing new vulnerabilities that are able to be exploited by an attacker. Additionally, client systems are growing in value for attackers as they often contain personal information, banking information, and passwords. Historically, analyzing new sites for malicious content has been a very manual process or an automated process where the end users' needs were removed from the process. This thesis explores the power of cloud computing technologies capability of real time malware analysis and bringing the user back into the analysis process by using the user's browsing activity to generate URLs for analysis. This paper examines the design of such a system as well as the results of the prototype of the system.

Using a single prototype machine, it is experimentally shown that cloud computing technology is capable of performing an analysis of web sites in near real time. The prototype system performed experiments with two operating systems (Windows 7 and Lubuntu Linux) as well as machine learning algorithms to gather the latency and throughput. The average analysis time for the prototype system was less than 0.5 seconds with a single virtual machine having a throughput of around 1,000 sites per hour. In addition, the technology presented by this thesis is scalable as many virtual machines are capable of being spun up on a single piece of hardware.

.

CHAPTER I: OVERVIEW

Introduction

On October 29, 1969 the first message was sent across ARPANET [9] and is widely considered the birth of the internet. Since that day the internet has grown and evolved into a worldwide network connecting all sorts of devices ranging from personal computers, cell phones, watches, and even refrigerators. On November 2nd 1988 Robert Morris unleashed the Morris worm and became the first person to be convicted under the 1986 Computer Fraud and Abuse Act [11]. Ever since the Morris worm, attackers have evolved along with the evolution of the internet and have grown from single users to entire criminal organizations and nation states, each with their own end goals and agenda. Each attacker brings their own set of skills, varying levels of sophistication, and resources that can be used in an attack scenario. With the ever evolving threat landscape, new technology is required to be developed to protect end users from the various threat actors to ensure client systems confidentiality, integrity, and availability.

Problem statement

Browser based malware is an evolving threat on the internet today with all of the most popular browsers being shipped with vulnerabilities that an attacker can exploit for personal gain. As shown in the Pwn2Own 2015 contest [4], a hacking contest where security researchers attempt to compromise the latest Web Browser technology, even the latest browser technology is vulnerable. Over the course of the 3 day competition the following quantities of bugs were found:

- 5 bugs in the Windows Operating system
- 4 bugs in Internet Explorer 11
- 3 bugs in Mozilla Firefox
- 3 bugs in Adobe Reader
- 3 bugs in Adobe Flash
- 2 bugs in Apple Safari
- 1 bug in Google Chrome

Outside of a competition environment these bugs could be used to add a user's machine to a botnet or to distribute ransomware or other malware. These types of competitions continue to provide fresh evidence every year that writing bug free code is almost impossible even for large, well-funded, corporations such as Google and Apple who have a focus on developing secure solutions.

The problem is compounded when attackers compromise legitimate sites, making it difficult for even security conscious users to know if a site is safe to visit. Pyria et al. [5] notes that sites such as F-Secure and Kaspersky, companies working in the malware analysis sector, have fallen victim and been used by attackers to deliver malicious content.

To date, much of the client malware detection is performed using signatures and blacklists; however this approach is not as effective as it once was. Malware authors have grown proficient at cloning the malware functionality, making minor changes to the code, and producing a completely new signature. To counter this many researchers have looked to machine learning techniques and sandboxing techniques to observe the malware in a safe environment. To date advanced malware identification and analysis requires resources that are beyond standard clients and are not able to be performed in real time.

Thesis Statement

With the advent of cloud computing technology, the ability to offload processing requirements from client machines, such as Android or Windows, opens the door to perform more complex malware analysis in the cloud that would not be feasible on a client device alone. Because client honeypots did not have the processing capacity needed to crawl the entire internet, they were often designed to use a URL list that was based on popular websites and search engine indexes to select which sites to analyze. This causes a problem as it may not account for all possible sites on the internet and it may miss unindexed sites. This thesis proposes to put the end user back in the loop by performing an analysis, in real time, of the end user's browsing activity. This means that as an end user browses the internet all sites that they navigate to will be analyzed with near real time results returned.

With the client back in the URL selection process, a real time analysis is able to be performed as the user requests the URL. This will allow the browser to actively block malicious content before the client system processes it, thus avoiding a malware infection. This will reduce the risk an end user takes when browsing the internet and is able to browse with more confidence that their machine will not be infected even when falling victim to spear phishing attacks or browsing potentially dangerous sites.

A second issue that this solution proposes to solve is the case when a good site is compromised and is serving malicious content. As the end user's browsing is used to generate URLs the sites that an end user browses to are the sites that are analyzed, regardless of the popularity. This means that assumed safe sites, such as Google and Amazon, are analyzed the same as completely untrusted sites.

Finally, by adding the user back into the URL generation process, the possibility of performing analysis on restricted pages or pages hidden behind a Completely Automated Public Turning test to tell Computers and Humans Apart (CAPTCHAS) is now possible. The user is able to provide authentication or validation allowing the analysis to be performed on restricted areas that a standard web crawler might not be able to reach.

Thesis Organization

The remainder of this thesis is organized to give the reader a firm background on the distribution, detection, and prevention of malware. This includes previous research in malware detection, classification and prevention as well as presenting the new ideas proposed by this thesis.

Chapter I: Provides an overview of thesis this including introduction and problem statement.

Chapter II: Provides background information on the threat environment to establish context for why browsers are high value targets.

Chapter III: Describes the related research and how it applies to this thesis topic, looking at the pros and cons of the related research in comparison to this thesis.

Chapter IV: Presents the approach this thesis purposes to detect malware on the various systems investigated.

Chapter V: Presents the system developed by this thesis and describes how it works.

Chapter VI: Presents the results gathered from the case studies performed.

Chapter VII: Presents the Countermeasures discovered and presents Future Work proposals.

Chapter VIII: Presents the conclusions for this thesis.

CHAPTER II: BACKGROUND

Introduction to Drive-By Downloads

A Drive-By download occurs when a user visits a website hosting malicious content that attacks the browser. The malicious site will attempt to exploit the browser or browser plugins such as Adobe® Flash, and ActiveX components. The attack takes place silently, without the victim's knowledge, and if successful, allows the malicious site to install software on the victim’s machine. The installed software will attempt to exploit the system resources by joining a Botnet, installing key loggers, or stealing sensitive personal information such as credit card numbers or social security numbers.

Introduction to Honeypots

Honeypots are computer security software and tools that are designed to be probed, exploited, and analyzed. A honeypot allows research to provide a target for an attacker to attempt to exploit while not putting any critical systems at risk. A honeypot provides researchers with logs of the interactions that have occurred between the attacker and the honeypot to learn about the attempted attack and to better understand how to defend against future attacks.

A honeypot can be classified as high interaction, medium interaction, or low interaction. A high interaction honeypot is typically a fully functional system running on dedicated hardware or in a virtualized environment. The high interaction honeypots have the highest resource requirement but tend to have the lowest false positive rates. High interaction honeypots provide the researcher the ability to perform complex analysis of the interactions with the attacker and the Honeypot. High interaction honeypots also provide the ability to detect 0-day exploits. A medium interaction honeypot provides more interaction than that of a low interaction honeypot, but provides less than that of a high interaction honeypot. A medium interaction honeypot is typically a single application, such as SSH, being simulated on dedicated hardware. A low interaction honeypot does not implement the full up operating environment but will instead use lightweight or simulated clients. Low interaction honeypots analyze the traffic received from the interactions and often implement signature based detection. Low interaction honeypot are quicker to analyze content given that they use signatures; however they are also prone false negatives as signatures database require constant updates to catch the latest traffic patterns or threats.

There are two types of honeypots, Server Honeypots and Client Honeypots. A Server Honeypot hosts a service in which attackers connect to, and attempts to perform attacks against the service. One such server honeypot is Kippo [15]. Kippo is a medium interaction SSH honeypot designed to log brute force attacks and, most importantly, the entire shell interaction performed by the attacker.

A Client honeypot is different from a server honeypot in that it doesn’t offer a service and wait for an attacker to connect, but will instead attempt to actively seek out the attackers. A client honeypot is generally constructed of three main components:

- A queue of sites to analyze. The queue is populated through a seed, or in this thesis’ case, user driven interaction.
- The Client itself is used to connect to the server being analyzed. This is either a high interaction client implementation or a low interaction client implementation.

- An Analysis Engine. The Analysis Engine performs analysis on the interactions with the servers in question to determine if malware was distributed or the honeypot system was compromised.

Introduction to malware

Malware is defined by Kaspersky Labs as “malicious software” that is designed to do things that are harmful to or unwanted by a computer’s legitimate user [12].

Malware is developed to achieve an end goal, usually financial or political gain, for its developer. For example, when a computer is infected with malware, it can be added to a botnet where the computing resources are then sold. Once in a botnet the infected machine can partake in distributed computing, execution of Denial of Service attacks, or spam propagation. Malware can be broken down into six basic categories:

- **Adware**: Adware is software that is installed on a client machine with the sole purpose of displaying advertisements to the user. The displayed advertisements are then used to generate revenue for the developer.
- **Ransomware**: Ransomware is software that once installed on a client machine begins to take control of the content on the machine. The Ransomware will often encrypt the user's critical documents and will demand payment for the decryption key.
- **Spyware**: Spyware is any piece of software that is used to track a user's actions while using the computing resource. There are both legitimate and illegitimate spyware instances. For example a tracking cookie, such as those employed by google analytics, is used to track user’s web activity to produce better targeted advertising.

Spyware may also be an executable running on the client system, such as a key logger which is used to steal user credentials and banking information.

- **Virus**: A virus is a piece of software that attaches itself to a legitimate software process (such as explorer.exe in windows) and is executed when the infected process is run. A virus may be used to disrupt normal system operation, or to install other malware (e.g., Adware).
- **Worm**: A worm is a self-replicating piece of malware that does not require a user to help distribute it. A worm can contain destructive content or can be used to install other types of malware onto the machine (e.g., Adware).
- **Trojan**: A Trojan or Trojan horse is an application that is not always what it claims to be. A Trojan application may be a legitimate application, such as a free Portable Document Format (PDF) reader, that also is used to steal private information from the machine, for example by containing a key logger, or may install backdoors into the machine which can be exploited at a later date.

Brief survey of existing malware detection techniques

Malware detection is performed using technologies in two broad categories: anomaly-based detection and signature based detection [6]. Signature based detection methods examine the executable code to pull out unique sequences which can be used to identify the software. This sequence acts as a signature. Commercial antivirus vendors make a business of producing these signatures and distributing them to their customers at regular intervals. Signature based detection is the quickest method of detecting malware once the signature is known, however it is easily bypassed using code obfuscation techniques. Vinod et al. [6]

provides several examples how an executable might be obfuscated using dead-code-insertion, Code Transportation, Instruction Substitution, or encryption.

The second technology for malware detection is anomaly detection methods. Using anomaly detection techniques a profile of the system or software is generated that describes how the system or software behaves normally. When behavior that does not match the profile is detected, a flag is raised that malicious software may be present. Anomaly detection is difficult to implement in the general case and has the possibility of producing a large number of false positives. Just because a system is behaving outside of its profile doesn't prove that malware is present, it may be the case that the profile was incomplete or incorrect.

CHAPTER III: RELATED RESEARCH

Google Safe Browsing API

Google's Safe Browsing [13] is a system set up to crawl the internet looking for malicious sites based on Google's search index. Much like the approach proposed by this research Google utilizes virtual machines to analyze the sites to see if the machine becomes infected. In addition, Google's Safe Browsing system uses statistical models, or machine learning algorithms, to identify phishing sites. Google's system shows that with enough resources, much of the internet can be analyzed with a low false positive rate. Google's approach breaks down for targeted attacks against individuals, such as targeted spear fishing attacks, where the site in question may not be in Google's index.

A Static Approach to Detect Drive-by-download Attacks on Webpages

Priya M et al [5] purposes using a MATLAB based HTML parser to perform static analysis and feature extraction on both benign and malicious web sites. Priya M. et al. extract the following features from the websites analyzed: Number of characters, number of tags like html, body, head, title script elements, number of links in the web page, total number of characters, number of strings, strings with length greater than a particular threshold and whitespace, and more. This research has provided a good starting point for machine learning features to be used for extraction, however Priya M. et al. research does not address the dynamic nature of the internet and how malware has become more complex and intelligent. Priya M. et al. do not take into account that malware may only be downloaded to a machine if the machine presents a vulnerable signature.

Risk Ranker: Scalable and Accurate Zero-day Android Malware Detection

Risk Ranker developed by Grace et al. [2] is an Android Malware detection system used to identify high risk Android applications. While not directly applicable to a real-time browser malware detection system it does provide a perspective on how malware detection takes place. The Risk Ranker solution provides two valuable ideas that can be applied to general malware detection. First is the concept of a multi-tiered approach. Grace et al. [2] purposes a first-order analysis and second-order analysis which applies a risk rating to the application (High, Medium, and Low). The first order-analysis is used to sift through the applications that do not employ obfuscation and can quickly be analyzed to apply a risk assessment. The second-order analysis is used to sift through the complex applications where behavior analysis is required to provide a risk assessment.

The outcome of the first-order and second-order analysis is a list of applications and risk assessments (high, medium, and low). This provides an analyst with a sorted list of applications which merit further investigation. This prioritized list represents a small subset of the overall set of applications, thus helping to address the scalability issue.

Several of the concepts presented by Grace et al. are adopted by this paper's approach to malware detection; multiple analysis venues and risk based approach.

Using Honey Clients to Detect Malicious Websites

Mahmoud T. Qassrawi et al [3] discuss the practical benefits of server side honeypots and client side honeypots. Mahmoud T. Qassrawi et al proposes using a seed to feed the malware analysis system but admits that even this approach is not the best as search engine popularity can have major effects on the pool of websites with malicious websites being ranked lower in the search results typically. The approach outlined in this thesis purposes a slight tweak to the method used by Mahmoud T. Qassrawi et al in the seed generation method by using real time user driven input instead of a search engine results. Using Honey clients to Detect Malicious Websites also points out several other issues with client side honey pots, for example many malware developers may hide behind a CAPTCHA or a password protected area that an automated system is unable to penetrate.

The Ghost in the Browser Analysis of Web-based Malware

Niels Provos et al [1] were one of the early researchers to experiment with using the Google index to perform offline post processing of web content. Niels Provos et al performed the analysis in three phases. In the first phase the complete list of candidate URLs are filtered, using the Map Reduce algorithm, to sites most likely to be malicious. Second an in-depth analysis is performed using an instrumented Internet Explorer browser to fetch the site and determine if was malicious. Finally, all of the subpages on a site are combined into a domain rating. This research has laid the ground work on drive-by download analysis and many queues such as virtualization and process monitoring are derived from this work.

CHAPTER IV: DETECTING MALWARE

General Malware Detection Approach

The general high interaction honeypot analysis utilizes two approaches to determine if malware has been installed on the client machine during the webpage retrieval, a shallow analysis and a deep analysis. The shallow analysis method relies on the system state changing when malware is installed on the victim machine. Priya et al. [5] states that when a machine is successfully compromised the following system state changes can occur:

- Modification of Windows Registry Entries
- Increased number of processes executing on the system
- File Creation / Deletion

The shallow analysis uses the above information to build a profile of the test system. A snapshot of the system before and after web page retrieval occurs and the two snapshots are analyzed to determine if any unexpected changes occurred. The shallow analysis uses the following system profile parameters:

- Average CPU use
- Virtual memory use
- Logged in users
- Processes and process state

The shallow analysis is performed on the target machine using the Python package psutil [20] to gather the system state. Psutil is an operating system independent tool that gathers the statistics of the system using a Python API.

The deep analysis is performed by recursively performing a hash on the entire file system before the page is retrieved and after the page is retrieved. When the two calculated hashes don't match it is known that files were added or changed on the system and malware was installed. Note: Some files cannot be analyzed because of the nature of the file. For example on a Linux system the file /dev/random appears like a file but is not actually a file, and will return data forever when queried. For specific non analyzed files see the operating system specific sections.

Machine Learning Analysis

The Multi-System Malware Detection System (MMDS) System implements an operating system independent analysis of the web content using machine learning analysis. The use of machine learning to identify malicious sites has been done before [5] and shown to have a True Positive Rate of up to 91%. However as the True Positive rate has a statistical miss rate of 9% or higher, it should not be an end all classification. As such, this thesis purposes to use the machine learning algorithms alongside of the shallow and deep analysis presented in this chapter to provide additional evidence of the malice of a site.

Taking cues from [5] [10] the OS independent analysis extracts several features from and about the webpage in question. The features can be broken into three basic categories: features of the site in question, physical location of the IP Address, and DNS information.

Site Features:

- Length of the hostname
- Site has iframe redirection
- Site has 301 redirection

- Site has 302 redirection
- Number of script elements in the HTML
- Port number site is hosted on

Physical Location Features:

- IP address location metro code
- IP address location city name
- IP address location continent code
- IP address location country code
- IP address location local code
- IP address location time zone
- IP address latitude
- IP address longitude

DNS Information:

- Time-to-live value of the DNS record extracted using [17]
- Domain is an IP address

Once all of the features have been extracted and calculated the python machine learning tool suite scikit-learn [7] is used to perform the analysis. Scikit-learn is an open source python based machine learning system which implements many. From scikit-learn the Support Vector Classification algorithm was selected for analysis. The Support Vector Classification algorithm was selected as it is a supervised learning algorithm that is capable of assigning categories (Safe / Malicious) when presented with site features. This algorithm is trained using 2000 known good sites from the Alexia Top Million sites [8] and 1300 sites from a known malicious list. Each of the sites used for training were passed to the Google

Safe Browsing API as a point of reference to verify if they were classified as benign or malicious before being used in the algorithm training.

Windows 7 analysis

Windows 7 uses both processes launched by the user as well as processes launched by the Windows 7 system for general maintenance activities. One such process that windows utilizes is the svchost.exe. The svchost.exe is a generic host process name for services that are started and used by the operating system; services such as networking, backups, human machine interfaces, virus scanners, and more. As the svchost.exe is an expected process that is required for core operation of the Windows 7 operating system it is ignored during the shallow analysis. In addition, Windows 7 has several file locations that require Administrator or even System level access to view and are unable to be analyzed during the deep analysis. For example, the SAM registry hive is used to store password hashes and is locked from viewing by user handles. The Windows 7 operating system verifies that the current user handle is the same user handle that originally accessed the file when the system was powered on. As such, during a deep analysis, the SAM files are unable to be accessed by the standard user and are skipped.

Linux analysis

As with the Windows based systems, Linux also implements a background worker thread that is expected to create and destroy processes as part of the standard operation of the Linux platform. The process, kworker, is the placeholder process for kernel worker threads that perform most of the processing for the kernel. The kworker process manages interrupts, timers, and Input / Output. As such, in the event that the kworker process is started as a result of the web page gathering it is not necessarily an indication of compromise and will be ignored during the shallow analysis. In addition to process creation and destruction there are files in the Linux operating system that are expected to change as part of normal system operation and are skipped during a deep analysis. These files include:

- Log Files (/var/log)
- Device File (/dev/)
- Pseudo-file system files (/proc)

Browser Cache Files

All modern web browsers utilize local file storage while browsing the internet to cache files and store session information (browser cookies). The cached files are used to speed up browsing by pulling the frequently requested resources from local. The browser cookies are used to store state information about a session with a server because the HTTP protocol is designed to be a stateless protocol. This is considered normal operation for the browser and is not considered an attack and as such, the browser's cache files are not considered in the analysis of the system and are ignored during the shallow analysis.

CHAPTER V: MULTI-SYSTEM MALWARE DETECTION SYSTEM

System Overview

The MMDS is built to support scalability and variability with an emphasis on the ability to start and stop resources as demand changes. The core of the system is built upon the Advanced Message Queuing Protocol (AMQP) which allows for many producers and many consumers to process the required data. As shown in Figure 1, the clients (Windows, Linux, Android, Mac OS, etc.) all send analysis requests to the core server where they are distributed to individual analysis clusters. Once analysis is complete, the cluster responds back to the core server which ultimately responds back to the requesting client. The MMDS is comprised of several subcomponents: client systems, core server, control servers and analysis clusters.

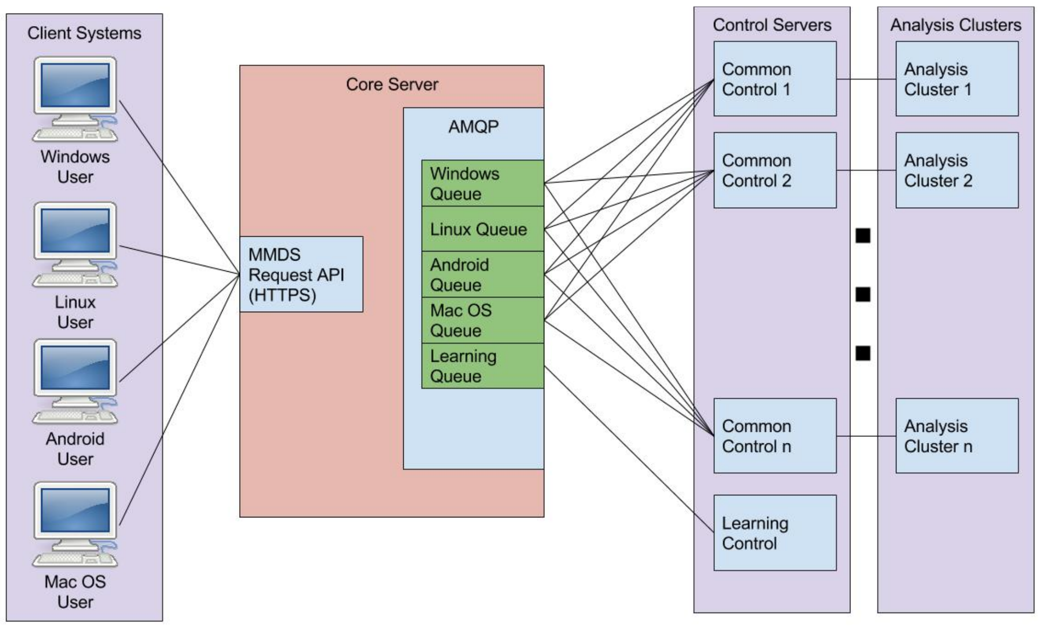

Figure 1: MMDS Overview

Client Systems

The client systems are the end user's browser, such as Chrome or Firefox. The client systems are the primary URL generation mechanism for the MMDS. As a client system is used to browse the internet, the client system sends each URL accessed to the MMDS for analysis. In addition, the client system can wait for the response from the MMDS to determine if the user should be warned before allowing any processing of content from the site.

Core Server

The core server hosts the main results database as well as the primary AMQP Server. In addition, the core server hosts a web server that provides the MMDS web API. The MMDS web API is an interface that client systems can submit URLs to for analysis. The MMDS web API is implemented by a PHP page that places the URL along with client system identifying information, such as IP address, into the correct AMQP queue for the requested operating system and browser configuration. The MMDS web API will then, if requested, wait for the response to return from the analysis clusters either via an AMQP return channel or by directly querying the database. The response is then returned to the client system.

Control Servers

There are two types of control servers, the common control server and the learning control server. The common control server and the learning control server are responsible for querying the AMQP server to retrieve URLs to be processed by the associated machine type (Linux, Windows 7, machine learning), updating the database with the results of the analysis, and responding to the core server with the results.

The common control server uses an analysis cluster to perform the analysis on a site. When the common control server receives a URL the URL is forwarded to a machine in the analysis cluster that is ready to perform an analysis. The common control server will then monitor the analysis machine for the results. Once the results have been determined by the machine in the analysis cluster, the common control server makes a decision on if the analysis cluster machine needs to be restored to a known good state, or if the machine is ready to perform another analysis. When the analysis cluster machine detects malware the state of the analysis cluster machine is restored using a known good snapshot by the common control server.

The learning control server does not utilize an analysis cluster but instead performs the analysis directly. The learning control server gathers the raw URL data, extracts the features of the URL as described in the Machine Learning Analysis section and uses the scikit-learn tool kit to make a determination if the URL is malicious or safe.

Analysis Clusters

The analysis cluster is comprised of various virtual machine types and configurations. The analysis cluster machines are responsible for gathering the URLs and performing an analysis to determine if malware was downloaded. In addition the analysis cluster machines are responsible for clearing browser caches and restoring the system to a ready state in the event that no malware was discovered. When malware is detected, these machines take no further action as they are restored to a known good snapshot by the common control servers. The analysis cluster machines are implemented using various operating systems such as

Linux, Windows, iOS, and Android, which should match the configuration of the client systems as much as possible.

Detailed Control Flow

The control flow of the system is broken into two main processes, the client systems web API control flow and the MMDS control flow. The client systems and web API control flow is shown in Figure 2. The client system and web API process starts by user' browsing to a URL. The client system extracts the URL and sends it to the MMDS web API which in turn adds it to the core server AMQP queues. The MMDS web API then determines if a response is requested by the client system. If a response is requested the MMDS web API waits for a response to be received or for a timeout to occur. When the response is received, or the timeout occurs, the MMDS web API sends the response (Safe, Malicious, or Unknown) to the Client System and the process is ready to start again.

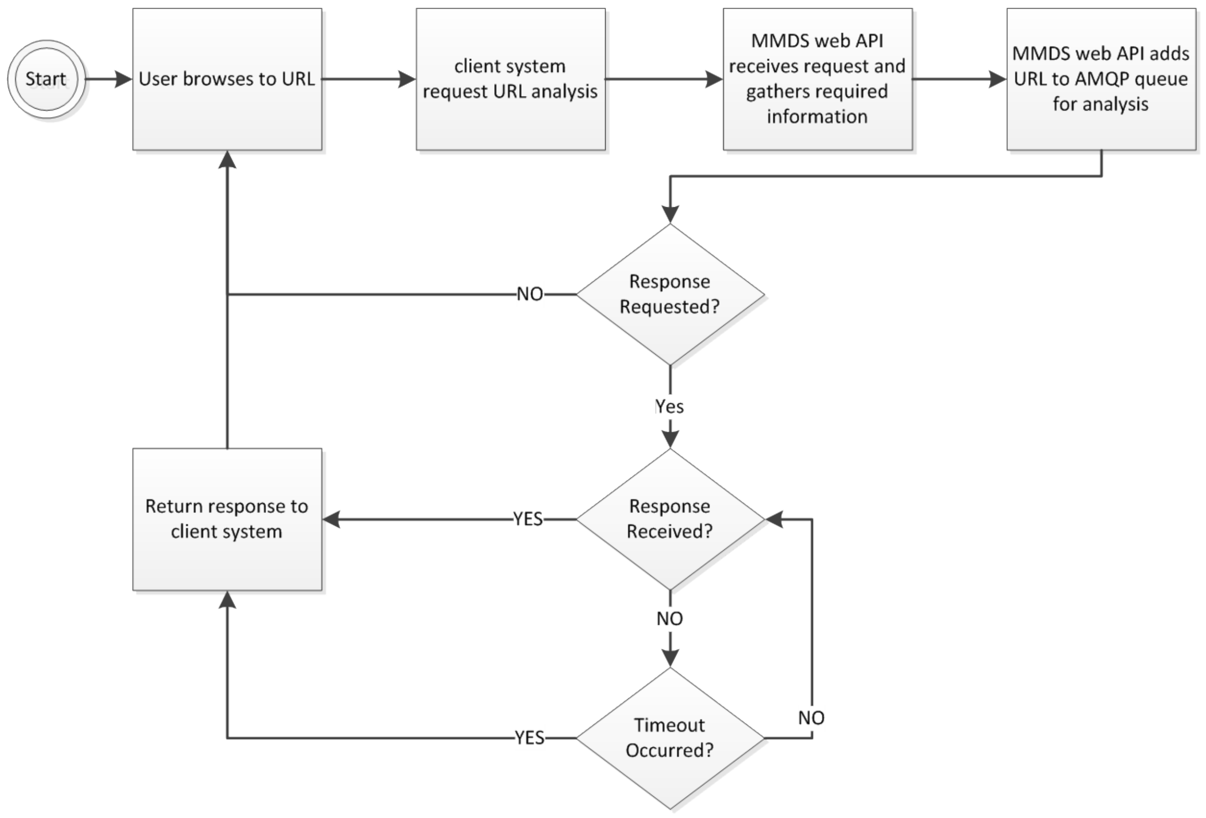
Start
User browses to URL
client system request URL analysis
MMDS web API receives request and gathers required information
MMDS web API adds URL to AMQP queue for analysis
Response Requested?
NO
Yes
Response Received?
YES
Return response to client system
NO
Timeout Occurred?
YES
NO

Figure 2: Client system and web API control flow

The MMDS control flow, as shown in Figure 3, is further broken down into two components: the control server and the analysis cluster. The control flow begins when the control server receives a URL from the AMQP queue. If the control server is a learning control server, the URL raw data is gathered, the features are extracted and an analysis is performed using the scikit-learn tool suite. If the control server is a common control server the URL is forwarded to the analysis cluster machine. The analysis cluster machine takes a snapshot, for use in shallow analysis, and browses to the requested URL. Next the analysis cluster performs a shallow analysis as described in Chapter IV and reports the results back to the common control server. When no malware is detected the analysis cluster restores the browser to an initial state to get ready for the next URL. If malware was detected the common control server commands a deep analysis by the analysis cluster machine. If malware is still detected the analysis machine is restored to a known good snapshot. Finally the common control server sends the results back to the MMDS web API and updates the database.

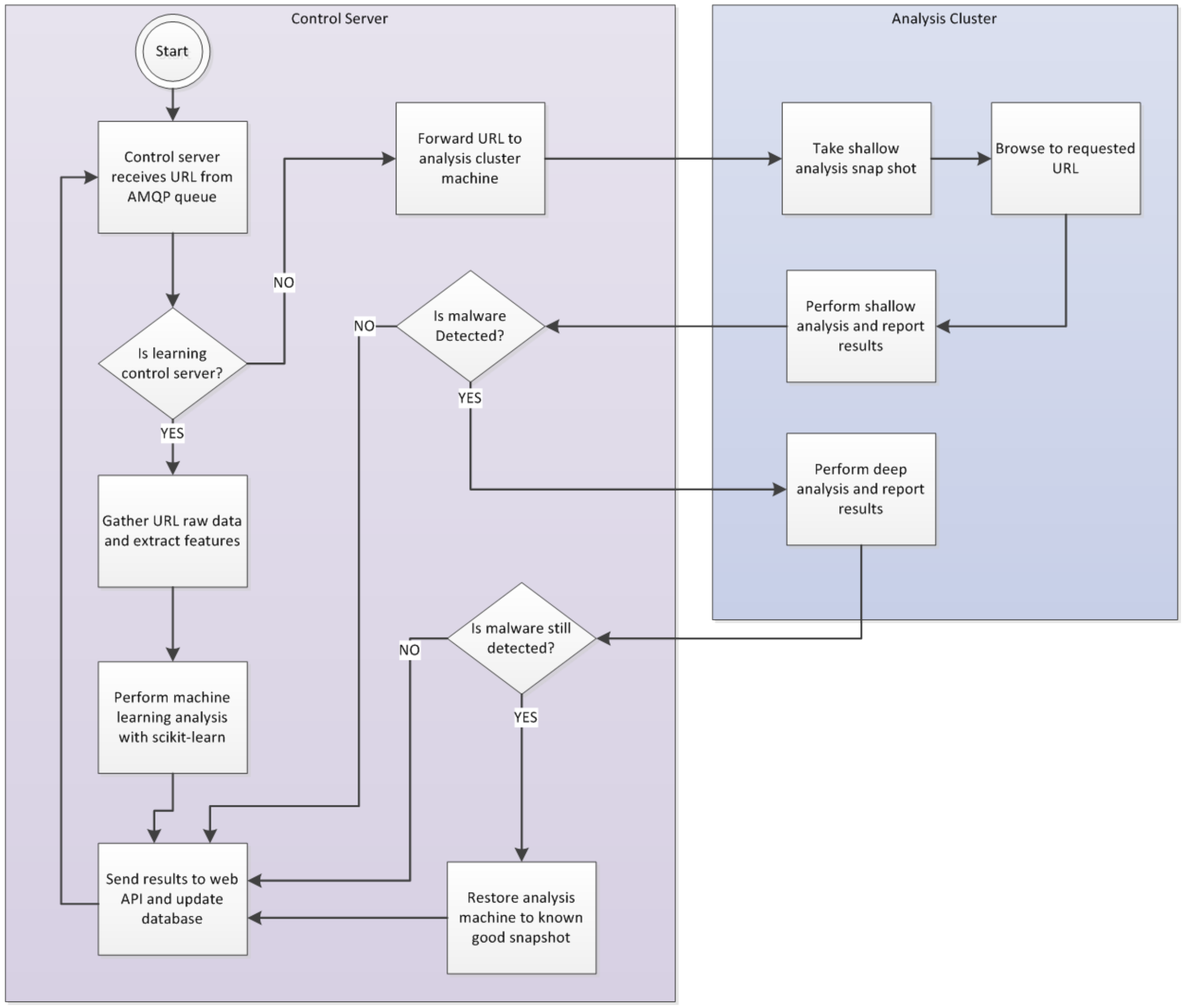
Control Server
Analysis Cluster
Start
Control server receives URL from AMQP queue
Is learning control server?
NO
YES
Forward URL to analysis cluster machine
Take shallow analysis snap shot
Browse to requested URL
Perform shallow analysis and report results
Is malware Detected?
NO
YES
Perform deep analysis and report results
Gather URL raw data and extract features
Perform machine learning analysis with scikit-learn
Is malware still detected?
NO
YES
Send results to web API and update database
Restore analysis machine to known good snapshot

Figure 3: MMDS control flow

CHAPTER VI: RESULTS

Prototype System Overview

The prototype MMDS was implemented on a machine with 2 Intel® E5606 2.13 GHz CPUs, 24489 MB of RAM, and a 1 Terabyte 5900 RPM with 8MB cache SATA Disk. The machine was running the VMWare vSphere 5 Hypervisor. On the test machine there were eight virtual machines executing plus one additional machine running in the google compute engine as shown in Figure 4.

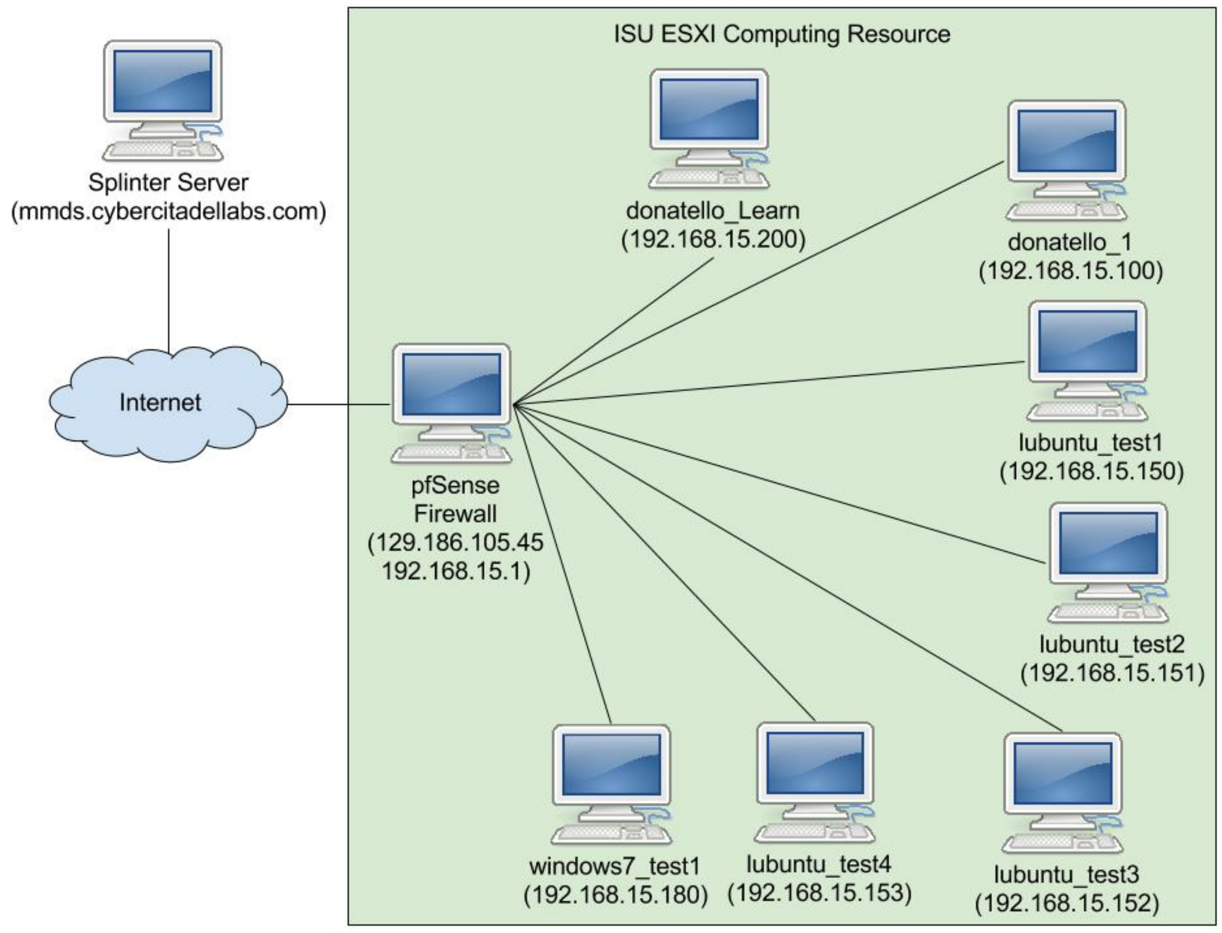

Figure 4: Prototype Network Diagram

Table 1 shows the details of the individual virtual machines resources that are utilized by the prototype.

Table 1: Prototype Network Machine Details

Machine Name	Machine Resources
Splinter Server (core server)	Google Cloud compute engine: **Machine type** n1-standard-1 (1 vCPU, 3.75 GB memory) **Operating System:** Ubuntu 16.04
donatello_Learn (control learning)	VMWare vSphere 5 hypervisor: **Memory:** 1024 MB **CPUs:** 1 **HDD:** 10 GB **Operating System:** Ubuntu 14.04
donatello_1	VMWare vSphere 5 hypervisor: **Memory:** 8192 MB **CPUs:** 2 **HDD:** 100 GB **Operating System:** Ubuntu 14.04
lubuntu_test1 lubuntu_test2 lubuntu_test3 lubuntu_test4	VMWare vSphere 5 hypervisor: **Memory:** 1024 MB **CPUs:** 1 **HDD:** 5 GB **Operating System:** Lubuntu 15.10 Desktop
windows7_test1	VMWare vSphere 5 hypervisor: **Memory:** 1024 MB **CPUs:** 1 **HDD:** 24 GB **Operating System:** Windows 7

Analysis Time

The analysis time is further broken down into the 3 main machine types: Lubuntu Linux, machine learning, and Windows 7.

The Lubuntu Linux timing analysis was performed over approximately 34,000 sites. The analysis was performed twice, once without any previous analysis data available, and the second time using the existing data from the first analysis run. The second analysis resulted in only database lookup time and simulates the average time for popular sites that are analyzed more frequently. The analysis time, in seconds, for the Lubuntu machines is shown in Table 2.

Table 2: Lubuntu Analysis Details

No Previous Analysis		**Previous Analysis**	
Standard Deviation	0.373825	Standard Deviation	0.137656572
Minimum	0.016163	Minimum	0.018162
Maximum	26.33493	Maximum	4.484766
Mean	0.21345	Mean	0.148831745

The Windows 7 analysis was performed in the same manner as the Lubuntu Linux analysis except fewer sites were used in the analysis. Fewer sites were used because only a single Windows 7 test license was available for use during analysis while the Linux analysis used 4 virtual machines. For the analysis 17,500 sites were analyzed by the Windows 7 machine. The analysis time, in seconds, is shown in Table 3.

Table 3: Windows 7 Analysis Details

No Previous Analysis		**Previous Analysis**	
Standard Deviation	0.013474	Standard Deviation	0.008226
Minimum	0.012877	Minimum	0.011891
Maximum	0.274436	Maximum	0.213367
Mean	0.050127	Mean	0.030202

The Learning analysis was performed in the same manner as the Lubuntu Linux analysis. The analysis was performed using a single virtual machine running four instances of the machine learning algorithm. The learning analysis was able to analyze 100,000 sites as it has the highest throughput of the algorithms given that it takes less time to reset the state between each new analysis. The analysis time, in seconds, is shown in Table 4.

Table 4: Machine Learning Analysis Details

No Previous Analysis		**Previous Analysis**	
Standard Deviation	0.331602	Standard Deviation	0.118903
Minimum	0.034008	Minimum	0.013612
Maximum	21.02717	Maximum	4.584732
Mean	0.617779	Mean	0.17271

These throughput rates do not reflect the analysis time for malicious sites, with the exception of the machine learning algorithm. A site is able to be identified as potentially malicious in the listed times using the shallow analysis outlined in Chapter IV. However when a site is identified as potentially malicious, the deep analysis is performed to provide further evidence. The deep analysis is a much more time intensive process which examines all of the files on the file system. The prototype machine uses a spinning disk which on average for the Lubuntu system 5 GB hard disk takes 8 minutes.

A second set of experiments were performed using a Dell Latitude E6540 running Windows 7 and VMWare Workstation 11 and the same Lubuntu system with a 5 GB hard disk. The second analysis machine contains a solid state drive which was able to improve the performance by a factor of 10, performing the deep analysis in an average time of 60 seconds.

Throughput Rates

To determine the throughput rate of each of the three analysis systems a single instance / virtual machine was used to analyze the first 1000 sites from the Alexa domain list [8] against an empty database. This configuration requires the site to perform a full analysis of the site, and restore the system to a state where it is ready to analyze the next site. As shown in Table 5 the machine learning algorithm was the quickest followed by Windows 7 and finally Lubuntu Linux. An important note about throughput analysis is that it only examines safe sites as this represents the majority of end users internet traffic.

Table 5: MMDS Throughput Analysis

Analysis Technique	Sites Analyzed	Analysis Time
Machine Learning Algorithms	1,000	55 Minutes 19 Seconds
Lubuntu Linux	1,000	88 Minutes 54 Seconds
Windows 7	1,000	68 Minutes 9 Seconds

MMDS Vs Google Safe Browsing API

The Google Safe Browsing API performs a very similar function to that of the MMDS and makes for a good point of comparison. Where the Google Safe Browsing API falls short is in the ability to perform a real time analysis on a requested URL. Google uses their massive search engine index to direct virtual machines to analyze sites, storing the results in a database. When a user requests the status of a URL from the Google Safe Browsing API the result is returned from the database. The approach taken by Google is a faster approach but may return incomplete or out of date information. Table 6 provides a detailed listing of the pros and cons to each approach, the MMDS real time analysis approach and the Google Safe Browsing API approach.

Table 6: MMDS vs Google Safe Browsing API comparison

Feature	MMDS	Google Safe Browsing
Protect users from downloading malware	Yes	Yes
Analysis performed using user requested operating system configuration	Yes	No
Real time analysis of URLs	Yes	No
Ability to bypass CAPTCHA and passwords for analysis of URL	Yes	No
Targeted spear phishing malware protection	Yes	No
Consistent look up times of 0.5 seconds or less	No	Yes
Ability to detect phishing sites that do not install malware	No	Yes

CHAPTER VII: COUNTERMEASURES AND FUTURE WORK

There are several problems yet to be solved by the solution proposed by this research technology which would aid in efficiency and accuracy. First, there are enhancements to the machine learning algorithms that would increase prediction accuracy. Additional features can include but are not limited to:

- Automatic detection of obfuscated or encrypted JavaScript code
- Automatic detection of certificate information including type of certificate, algorithms used, expiration date, issuer, etc.

In addition to additional features, the machine learning field of study has the opportunity to implement many different algorithms. While many of the algorithms available have been examined in various works, no timing analysis has been performed on these algorithms against a system where an end user is waiting for a response.

Next is the user side of the technology; will the end users accept the privacy requirements. The technology presented by this thesis is by nature invasive to a user's privacy as this technology asks that a user submits all of their web traffic to an external actor for analysis. Before a technology such as this could be deployed for widespread use a better understanding of the acceptance of the system as what restrictions would be required for such a technology to be accepted is needed.

Additionally, the evasion and anti-analysis technique being deployed by malware should be researched. Modern malware has grown in complexity and intelligence and has evolved to contain anti-analysis techniques. Modern malware can first check if it is being executed in a virtual machine and alter its behavior when a virtualized environment is discovered. Additional research is required to determine how often anti-virtual machine methods are used, what anti-virtual machine methods are available and what possible solutions to these detection techniques exist to aid in malware analysis.

Finally, additional operating systems and configurations should be analyzed. The prototype for this thesis paper did not fully explore all of the possible operating systems that exist today. It is the desire to further develop the prototype to explore additional operating systems in further detail such as Android, iOS, and additional variants of the Windows operating system.

CHAPTER VII: CONCLUSIONS

Living in an age where internet connectivity is growing at alarming rates and personal information is continually being stored on personal computing devices it is critical for users to be able to trust the sites they visit are safe. It is also critical that the problem is analyzed for not just a single, popular, operating environment, but a large variety of operating environments and configurations with the ability to scale to internet proportions.

This research project has made several contributions to malware detection and protection of client systems. By reducing the number of overall sites to be analyzed by only looking at sites that the end users are visiting means that less time will be spent analyzing sites that people may never visit. In addition this provides enhanced protection by performing analysis on sites that may be missed by the standard seed process by using user's actual browsing activity. Sites such as targeted phishing campaigns which are designed for a single user may not be analyzed when a seed based on popular search engine results is utilized; however using the solution proposed by this research the site would be analyzed and the client system protected.

In addition the solution proposed by this research addresses the problem of malware targeting specific configurations. By allowing multiple operating systems and configurations, the only limitation being the hardware available, the system can scale up or down depending on the demand from the users.

REFERENCES

[1] N. Provos, D. McNamee, P. Mavrommatis, K. Wang and N. Modadugu, "The ghost in the browser analysis of web-based malware", in *HotBots'07 Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, 2007, pp. 4-4.

[2] M. Grace, Y. Zhou, Q. Zhang, S. Zou and X. Jiang, "RiskRanker: scalable and accurate zero-day android malware detection", in *MobiSys '12 Proceedings of the 10th international conference on Mobile systems, applications, and services*, Low Wood Bay, Lake District, United Kingdom, 2012, pp. 281-294.

[3] M. Qassrawi and H. Zhang, "Using Honeyclients to Detect Malicious Websites", in *2010 2nd International Conference on e-Business and Information System Security (EBISS)*, 2010, pp. 1 - 6.

[4] "Pwn2Own 2015: Day Two Results." HP Enterprise Business Community. N.p., 20 Mar. 2015. Web. 25 Oct. 2015.

[5] M., Priya M, Sandhya L., and Ciza Thomas. "A Static Approach To Detect Drive-By-Download Attacks On Webpages". 2013 International Conference On Control Communication And Computing (ICCC). IEEE, 2013. 298-303. Print.

[6] Vinod, P., et al. "Survey on malware detection methods." Proceedings of the 3rd Hackers' Workshop on Computer and Internet Security (IITKHACK'09). 2009.

[7] Pedregosa, F., Varoquaux, G., Gramfort, A. & Michel, V. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

[8] Amazon Web Services, Inc.,. "AWS | Alexa Top Sites - Up-To-Date Lists Of The Top Sites On The Web". N.p., 2016. Web. 10 Jan. 2016.

[9] Chris Sutton, "Internet began 35 years ago at UCLA with first message event sent between two computers". UCLA, 2004.

[10] Ma, Justin et al. "Beyond Blacklists: Learning To Detect Malicious Web Sites From Suspicious Urls". KDD'09. ACM, 2009. 1245-1254. Print.

[11] Loundy.com,. "United States V. Morris". N.p., 2016. Web. 6 Feb. 2016.

[12] Kaspersky Lab United States,. "Kaspersky Personal & Family Security Software". N.p., 2016. Web. 6 Feb. 2016.

[13] Google Developers,. "Safe Browsing API Google Developers". N.p., 2016. Web. 6 Feb. 2016.

[14] Fiori, Alexandre. "Freegeoip.Net". *Freegeoip.net*. N.p., 2016. Web. 6 Feb. 2016.

[15] GitHub,. "Desaster/Kippo". N.p., 2015. Web. 6 Feb. 2016.

[16] Greasespot.net,. "Greasespot". N.p., 2015. Web. 6 Feb. 2016.

[17] Dnspython.org,. "Dnspython Home Page". N.p., 2016. Web. 6 Feb. 2016.

[18] Rabbitmq.com,. "Rabbitmq - Messaging That Just Works". N.p., 2016. Web. 6 Feb. 2016.

[19] Apache.org,. "Welcome To The Apache Software Foundation!". N.p., 2016. Web. 6 Feb. 2016.

[20] Pypi.python.org,. "Psutil 3.4.2 : Python Package Index". N.p., 2016. Web. 6 Feb. 2016.

[21] Mongodb.org,. "Mongodb For GIANT Ideas | Mongodb". N.p., 2016. Web. 9 Feb. 2016.

[22] Selenium-python.readthedocs.org,. "Selenium With Python — Selenium Python Bindings 2 Documentation". N.p., 2016. Web. 9 Feb. 2016.

APPENDIX A: CORE SERVICES DETAILED DESIGN

Core Server

The core server is a cluster of one or more Ubuntu based machines hosting the Apache [19] web front end as well as the RabbitMQ AMQP server [18]. The apache server hosts the main interface page and converts the input parameters into an AMQP message in the following JSON format:

```
{
        “requester”: “<OS@UID>”,
        “URL”: “<url to parse>”
        “targetOS”: “<Windows 7, Linux, Learning, …>”
        “targetBrowser”: “Chrome, Firefox,
        “osVersion” : <Reserved>,
        “refreshRequested”: <boolean>
}
```

Once the request has been sent, the core server creates a queue in the RabbitMQ server which is used to return the response back to the user once the analysis is complete. The return information is sent back to the user in the form of the HTML body content.

Database Server

The core database server hosts a NoSQL database implemented by MongoDB [21] server that acts as the primary data store for the MMDS. There are two databases stored in the Mongo Database: GeoLocations and MMDS main.

The GeoLocations database is built using [14], a public HTTP API for developers to search the geolocation of an IP Address. The GeoLocations database is a local cache of this information which is utilized to speed up the geolocation lookup process. The GeoLocation database has the following format for each of the IP Addresses:

```
{
  "is_anonymous_proxy" : "0",
  "is_satellite_provider" : "0",
  "postal_code" : jsonData["zip_code"],
  "geonames" : [ {
          "subdivision_1_name" : <region_name>,
          "continent_name" : "",
          "subdivision_2_iso_code" : "",
          "metro_code" : <metro_code>,
          "city_name" : <city>,
          "continent_code" : "",
          "country_iso_code" : <country_code>,
          "subdivision_1_iso_code" : "",
          "local_code" : "",
          "subdivision_2_name" : "",
          "country_name" : <country_name>,
          "time_zone" : <time_zone">
             }],
  "registered_country_geoname_id" : [],
  "IPAddress" : <IPAddress>,
  "represented_country_geoname_id" : [],
  "latitude" : <latitude>,
  "longitude" : <longitude>
}
```

The second database, the MMDS main database, breaks the information into collections based on the Operating System (LinuxSites, LearnSites, Windows7Sites) as well as information about IP addresses that have been analyzed. The operating system collections are in the following format:

```
{ "URL" : <url>
 "Data": {
    "<OperatingSystem>" : {
      "<Browser>" : {
            "LastAnalyzed" : <last_alanlyzed_time>,
            "HasBeenMalicious" : <true / false>,
            "IsMalicious"      : <true / false>
          }
       }
    }
}
```

The IP Address database is in the following format:

```
{ "IPAddress" : "IPAddress Integer",
  "Malicious" : {
     "HasBeenMalicious" : <true / false>,
     "IsMalicious"      : <true / false>
  }
}
```

For the operating system collections as well as the IPAddress collections, the HasBeenMalicious field, once set to true, remains set to true. Each operating system as well as the IP Address collection can have unique values for the HasBeenMalicious values. That is the Linux operating system may have this value set to false, while the Windows 7 operating system may have this set to true.

Client Systems

The client systems are built use the GreaseMonkey framework [16]. GreaseMonkey is a browser extension that allows scripts to be developed and run in a browser sandbox. GreaseMonkey allows for the browser to make a call to the core server by calling the core web server URL. This PHP page takes several base64 encoded parameters:

- url
- targetOS
- targetBrowser
- refreshRequested

APPENDIX B: CONTROL CLUSTER DETAILED DESIGN

Control Server Configuration Items

The control server upon start up parses a configuration file which specifies the type of operating system that will be analyzed, the AMQP server details, the MongoDB server details, the logging server details and the address and credentials of the analysis system. As the MMDS is built using VMWare technology the use of snapshots and restore points is used heavily to assure that a known state can be restored when the system is first launched or when malware is detected. The control server hosts one or more clients.

AMQP Client

The control server clients are implemented using Python running on Linux. Upon startup of the client code a connection is established with the VMWare analysis infrastructure to verify that the analysis machine is present. The first action performed is to restore the analysis machine to a known snapshot, “Start”, to ensure that the system comes up in a known state free of malware. After the analysis machine has been restored, a connection to the AMQP server is established and messages are able to be consumed and analyzed.

When the client receives a message for analysis from the AMQP server it first checks if the requested URL has been analyzed in the last week and if a refresh has been requested. If the analysis time is older than a week, or a refresh was requested, the client machine sends an XML RPC request to the analysis machine instructing the analysis machine to gather the specified URL. Once the URL has been parsed, the client machine sends a command to perform a shallow analysis to the analysis machine and waits for the results. If the results come back as malware suspected, or if the database reports the domain as being malicious a deep analysis is commanded. Once the results are known the client system sends the results back to the user, stores the results in the database for the URL as well as the IP address of the URL.

In the event that malware was detected, the client will command the VMWare machine to revert the analysis system to the known snapshot “Start” before processing any additional URLs. If malware was not detected, the client system commands that the browser is reset, clearing the cache and history.

Test Machine Windows 7

For the Windows 7 test machine the Windows 7 Professional with Service Pack 1 installed was selected. To simplify prototype development the Mozilla Firefox browser was selected to align with the Lubuntu Linux. The Windows 7 test machine for the prototype does not have any special add-ons or extensions for the browser that may be present in many end users’ systems, but instead represents an initial installation state of the operating system.

The Windows 7 test machine does not implement any special performance enhancements. All that is required to for installation of the Windows 7 test machine is installation of the Mozilla Firefox browser and python 3.5.

Test Machine Learning

For the machine learning test machine the Ubuntu 14.04 LTS distribution was selected as it provides a complete set of the necessary tools (e.g., Python) and does not depend on a complex graphical user environment. For the machine learning algorithm a single instance of the OS is capable of launching many instances to perform the analysis. This is possible as the analysis does not depend on the state of the machine learning test machine but instead depends on the features of the URL being analyzed. For the prototype system developed the machine learning test machine hosted 4 instances each performing separate analysis. The machine learning test machine is the most flexible of the machines tested with the only requirements being that it is a Linux machine running the proper python version and packages.

Test Machines Lubuntu

For the Linux test machines the Lubuntu 15.04 distribution was selected as it is a small and lightweight version of Ubuntu. Lubuntu allowed for a complete desktop experience; however with its smaller size was able to be deployed in the cloud environment in higher quantities. Each Lubuntu instances is instantiated with 1GB of RAM, a single processor, and 5GB of hard disk space. This differs from the requirements for the Ubuntu desktop which requires a minimum 1.5 GB of RAM and 7GB of disk space.

To further increase the performance of the Lubuntu systems, all cache and browser profiles are stored in a RAM disk instead of on the primary hard disk for the system. This change helps to mitigate performance issues on systems which do not have a Solid State Drive where resetting the browsers frequently caused delays.

Lubuntu Test System Install Instructions

To stand up a new Lubuntu test system, once the base operating system has been installed it is required to install selenium [22] into the python environment. Selenium allows for the python language to control a full browser instance instead of just a web scraper. This helps to provide assurance that the instance browsing the requested URLs is as close to client configurations as possible. This also allows for plugins to be installed, just as they would for a standard browser installation.

After installing selenium, psutil [20] must be installed as this provides the capability required to gather the operating system process list, logged in users gathering the operating system, disk partitions, network connections and more.

Finally the browser, Firefox for the prototype system, needs to be commanded to run completely from RAM. This is necessary to increase performance as the browser is started and stopped for each URL visited. This can be accomplished editing the */etc/fstab* file and adding the following line:

```
tmpfs /tmp tmpfs defaults,noatime,nosuid,nodev,noexec,mode=1777,size=512M 0 0
```