2013-09-01

Distributed Agent Cloud-Sourced Malware Reporting Framework

Kellie Elizabeth Kercher
Brigham Young University - Provo

Distributed Agent Cloud-Sourced Malware Reporting Framework

Kellie E. Kercher

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Dale C. Rowe, Chair
Joseph J. Ekstrom
Derek L. Hansen

School of Technology

Brigham Young University

September 2013

ABSTRACT

Distributed Agent Cloud-Sourced Malware
Reporting Framework

Kellie E. Kercher
School of Technology, BYU
Master of Science

Malware is a fast growing threat that consists of a malicious script or piece of software that is used to disrupt the integrity of a user's experience. Antivirus software can help protect a user against these threats and there are numerous vendors users can choose from for their antivirus protection. However, each vendor has their own set of virus definitions varying in resources and capabilities in recognizing new threats. Currently, a persistent system is not in place that measures and displays data on the performance of antivirus vendors in responding to new malware over a continuous period of time. There is a need for a system that can evaluate antivirus performance in order to better inform end users of their security options, in addition to informing clients of prevalent threats occurring in their network. This project is dedicated to assessing the viability of a cloud sourced malware reporting framework that uses distributed agents to evaluate the performance of antivirus software based on malware signatures.

Keywords: malware, agent technology, antivirus

ACKNOWLEDGEMENTS

I am so very grateful for the IT faculty members and staff who helped me in developing this project and going out of their way to ensure its success. I am also very thankful for my family and their support along with the time they took to review my thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

1.1 Nature of the Problem

Malware is a fast growing threat to all end user devices. A popular antivirus vendor, Symantec detected and blocked more than 5.5 billion types of malware in 2011, an 81% increase from 2010 (“Internet Security Threat Report”). Large businesses are slow to detect malware breaches with detection time in the year 2012 averaging 210 days (“2013 Trustwave Global Security Report”). These numbers only include the malware actually discovered.

Intrusion detection systems, firewalls and antivirus software are all used to combat malware attacks and secure devices against intrusion. Currently antivirus detection is part of a commercially competitive market. There are numerous vendors each with their own detection engines and virus databases. These vendors have significant differences in resources, detection capabilities and response times for recognizing new forms of malware. With the increasing number of malware threats, vendors compete to be the first to recognize and respond in order to win a greater share of the security market.

Presently, there is no system in place to measure the real time performance of antivirus software in responding to new malware over an ongoing period. Current evaluations take place at a fixed time point and may provide sufficient uniformity to accurately evaluate ongoing trends in detection response time. Without this data, it is difficult to perform ongoing analysis into

detection efficiency, and thus inform end users of their security options along with threat detection activity.

1.2 Purpose of the Research

The purpose of this research is to assess the viability of a cloud sourced malware reporting framework that utilizes distributed agents to evaluate the performance of antivirus software based on malware signatures.

1.3 Project Approach

The novel aspect of this approach is the use of distributed agents for data collection along with the provisioning of a centralized source of real time antivirus activity. An agent is a primitive form of artificial intelligence (Sycara et al., 1996). It is able to recognize an environment and respond to events. The proposed use of agent technology in this setting will be used to detect new antivirus updates and malware quarantines for a host machine. Information gathered by the agent will be delivered to a cloud-hosted database, where the data will be publically available.

The scope of this research will be limited to the reported information from these distributed agents for a specific set of defined antivirus vendors. The findings will be used to compare the performance of different tools in responding to active threats. The data collected will also be used to identify universal malware descriptors across multiple vendors of antivirus software. It is hoped that this will open avenues to further studies in antivirus technologies along with providing resources to better inform users with their antivirus software decisions.

1.4 Research Questions and Hypotheses

The following questions will be answered from this research:

- (R1) What are the challenges in creating an agent malware reporting framework architecture and what techniques can be used to overcome these issues?
- (R2) What are the key characteristics that are suitable for universally identifying malware strands?
- (R3) Is there a correlation between antivirus malware naming conventions?
- (R4) What quantifiable benefits may be achieved by using multiple vendor products to detect malware?

The proposed research will analyze and compare antivirus trends and examine vendor abilities through the use of a malware reporting framework. This will be accomplished with a centralized data collection server and distributed reporting agents installed on endpoint devices. Each agent will be able to universally identify and correlate malware strands by the malware file hash. It is believed that despite different naming conventions vendors use to label a specific malware threat, there will be patterns and similarities between the companies.

It is known that a single vendor will not always be the first responder to every new piece of malware. At varying times, one vendor responder may be more efficient than another. Each vendor has different capabilities and resources that enable their servers to detect and release new virus definitions. Thus it is plausible that a host device may then benefit from multiple vendor software installations due to this variation in response times and capabilities at identifying malware and updating client devices. Researchers have commented that a single installation of an antivirus alone may not be sufficient to protect a system against malware (Posey). Assuming

certain multiple antivirus installations can operate compatibly in a single environment, this would decrease the time a host is vulnerable to a threat.

1.5 **Definitions**

- Agent Technology – A primitive form of artificial intelligence that is able to recognize an environment and respond to alerts.
- Antivirus Software – An application designed to protect endpoint devices against malware.
- Endpoint Devices – A host computer within a network.
- Hash – A fixed length bit string output resulting from a predefined algorithm on a block of data.
- Malware - A malicious script or software that is used to disrupt, disclose, distort or destroy computer operations.
- Virus Definitions – A set of characteristics that could include a virus signature that uniquely describes a piece of malware.
- Virus Signature – A unique hash that identifies a piece of malware.

2 LITERATURE REVIEW

2.1 Malware

Malware is a malicious script or software that is used to disrupt, disclose, distort or destroy computer operations. There are many different types of malware with different purposes. Malware can be classified into multiple categories. These categories include (Aycock 2006, Tian 2011):

- Adware – A piece of software that automatically delivers advertisements to the client.
- Backdoor - A method that bypasses expected authentication procedures. It can be used to secure entry into a system.
- Bot – An automated process that interacts with other network services. Through bots, a third party can indirectly interact with a system over a network.
- Logic Bomb – A threat that consists of two parts, a payload and trigger. The payload is a specific action to perform and the trigger is a condition that controls the execution of the payload.
- Rabbit – Malware that multiplies rapidly. There are two types of rabbit malware. One attempts to consume all of a resource such as disk space. The other type propagates over a network but deletes its source copy, hence "hopping" from device to device.

- Rootkit – A program designed to take control of a system. It attempts to seize administrative or root system privileges without authorization.
- Spyware – A piece of software that discretely reports user activity to a third party.
- Trojan horse - A malicious script or piece of software disguised as a safe application.
- Virus - A malicious application that self-propagates across devices.
- Worm - A script that self-propagates across a network.

A single piece of malware may perform numerous tasks thus fitting into multiple categories. Malware often hides in the form of media files, advertisements, email attachments or peer-to-peer shared files. Systems can be infected by users downloading the immediate strand of malware or by downloading a piece of software packaged with the malware.

2.2 **Agent Technology**

Agent technology is being implanted in an increasing number of applications ranging in size and capabilities (Jennings et al. 1998). There has been much debate on a universally accepted definition however, research has generally agreed on an agent being a form of primitive intelligence that is able to perform autonomous action in an environment in order to meet programmed objectives. An agent is oriented to act without any human intervention. It is an entity used to observe and environment and identify conditions to act upon (Jennings & Wooldridge 1998, Nwana & Ndumu 1998). An agent development was chosen for this framework because of its ability to watch an environment and take action without any user interaction. For the framework, the agent will be able to detect and respond to antivirus events.

Agents can be integrated together into a distributed network of intelligent agents. The purpose is to have the agents communicate issues or environmental features across the network

so as to inform all parts of each other's state (Sycara et al. 1996). With a distributed array of agents over multiple clients, a larger dataset can be acquired for the research.

2.3 **Antivirus**

Antivirus software is used to protect systems against malware by recognizing threats and either removing or blocking the malware. Figure 1 shows a timeline of when some of the more popular antivirus vendors were founded, the dates were provided by Wikipedia.com.

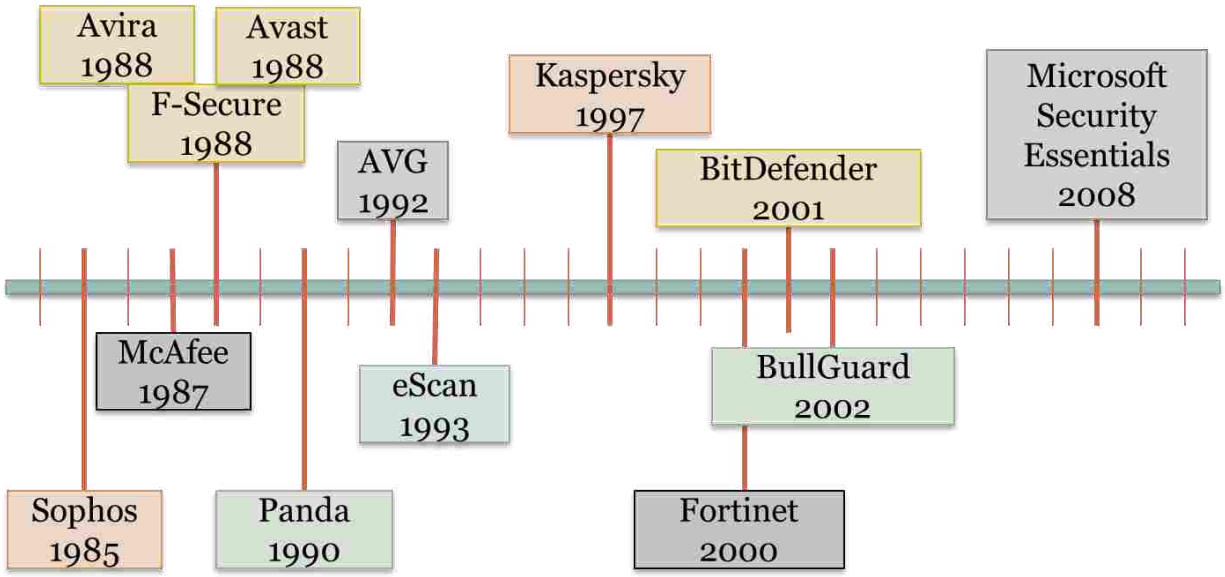

Figure 1: Antivirus Vendor Timeline

Antivirus software utilizes different approaches to update clients with the latest virus information. This information is then used to protect a device against known threats. Cambridge proposed a patent for a method of updating antivirus definitions over a network (Cambridge 2006). This design incorporates a centralized antivirus server with connected end user devices. When the server receives an update of a new virus signature from an end user, its antivirus

database is then updated. Other end user systems can then be compared against the server in order to update their signature files to the latest version as necessary. These signatures identify malware and alert an antivirus of their presence in a network. Cambridge's patent is used or similarly copied in many antivirus applications such as McAfee.

A further advance on updating antivirus definitions includes the use of push agent technology in order to update end users with the latest virus definitions. In application, when a new signature has been found and added to a centralized server, updates from that machine are then pushed out to client hosts. This service is performed in the background unannounced to the user during normal use, as long as the host is connected to the internet (Hodges et al. 2001). These different technologies have been implemented to improve antivirus performance in relaying updates to endpoint clients. The proposed research will look to examine antivirus software and the time it takes to utilize the updating technology in responding to new malware.

2.4 **Antivirus Variations**

According to Maggi et al. antivirus vendors are inconsistent with their naming convention for malware specimens (Maggi et al. 2011). The same piece of malware may have multiple names across antivirus vendors. The proposed project, along with analyzing the performance of different antivirus vendors, will look to provide an interface that is able to compare the different names vendors use to describe the same piece of malware. This visual will reduce confusion and aid further research in analyzing malware across vendors.

Sanok examines the techniques of signature detection, heuristics and general decryption that different antivirus applications use to detect and quarantine viruses (Sanok et al. 2005). His research illuminates a method on how to detect and read virus signatures. This information is

beneficial in understanding how antiviruses treat signatures and where they are stored in order to allow an agent to discover and disclose a signature.

2.5 Capabilities of Antivirus Protection

A study was performed by Rob Lee to measure the capabilities of antivirus software in detecting popular network threats (Lee 2013). He created a lab environment running the popular antivirus McAfee. With a team of college students, he devised a combination of crafted and well-known pieces of malware to exploit the protected environment. From his experiment; it was discovered that antivirus software is mainly used to defend against low-skilled attackers. Lee's findings stress the importance of a new security model to fortify end-users against popular threats in today's networks. His research supports the claim that a single antivirus on its own is not enough to prevent malware attacks on a host machine.

2.6 Existing Antivirus Comparisons

Sukwong et al. examined six popular antivirus products and how they respond to 1,115 distinct malware samples (Sukwong et al.). The duration of this study took place over a 5 month period of time. The antivirus software analyzed included:

- Avast 4.8 Professional v.4.8.1335
- Kaspersky Internet Security 2009
- McAfee Total Protection with Security Center v.9.15
- Norton Internet Security 2009 v.16.5.0.135

- Symantec AntiVirus v.10.1.7.7000
- Trend Micro Internet Security Pro v.17.1.1250

This study compared the times it takes these antivirus software to learn of an unknown piece of malware with daily scheduled virus updates. The results of the study concluded that the antivirus vendors varied in being first responders to a specific threat. This framework aims to build upon this study and analyze a similar sample set of antivirus providers in real time. Eventually this framework proposes the capabilities to analyze the ongoing ability of vendors and their resources to update signature viruses and catch new threats.

Another comparison study was performed on 32 different antivirus programs against 1,599 samples of malware. This study analyzed the effects of diversity on the detection capability as well as the time it takes an antivirus to evolve and update virus definitions. It also found that each type of antivirus software has different capabilities of catching and updating systems against a threat (Gashi et al. 2009). The study identified trends and displayed varying antivirus first responders to a specific malware threat.

In addition to these published studies, there are companies that frequently execute antivirus performance comparisons. AV-Comparatives regularly evaluates different antivirus software vendors. Some of these vendors include:

- Avast
- AVG
- AVIRA
- Bitdefender
- F-Secure
- Kaspersky
- McAfee
- Panda
- BullGuard
- eScan
- Fortinet
- Sophos
- Microsoft Security Essentials

They release their findings after summarizing results for end users to review ("Comparatives||tests - Reviews – Reports"). This evaluation is not in real time, but is rather an accumulation of findings. Another company, AV-Test, ranks the protection, usability and capabilities of antivirus software to quarantine malware from an infected device on a six-point scale ("AV-TEST"). However, AV-Test does not provide detailed information on the exact response times to a specific piece of malware.

There are numerous other organizations that regularly compare antivirus performance. Despite these companies and their reports, users are not provided real time information on active threats and antivirus response. In order to perform effective antivirus comparisons, a system is needed that can catch and release comparative data on antivirus software in real time.

2.7 **Multiple Antivirus Installations**

Despite the improvements in antivirus technology, Sukwong determined that malware is still in existence and spreading rapidly (Sukwong et al.). Though many solutions exist, there is not a one antivirus product that is consistently the first responder to a new threat. A single antivirus is not always the most effective in identifying the varying types of malware ("Why one virus engine is not enough…"). It is unpredictable which antivirus vendor will be the first to release a virus definition for a new strand of malware. For this reason, it is advised that a user includes various antivirus installations to reduce the time a system is vulnerable to a threat (Ibid). This is why products such as Microsoft's Forefront Security license numerous scanning engines from third-party vendors (Posey).

Another tool that takes advantage of multiple antivirus vendor resources is VirusTotal. This online resource is used to analyze suspicious files in order to identify malware. It aggregates

resources and uses multiple antivirus engines to identify threats ("About VirusTotal."). Using more than one antivirus can greatly improve the chances of detecting and removing new strands of malware.

3 METHODOLOGY

This section outlines the plans for the study and describes how answers for each purposed research question will be found.

3.1 (R1) Framework Design

The purpose of this research is to assess the viability of a cloud sourced malware reporting framework that uses distributed agents to evaluate the performance of antivirus software based on malware signatures. A prototype proof of concept will be built in order to examine the effectiveness of such a framework. The development of this prototype will verify what key components and design features are necessary along with the challenges involved in creating an agent malware reporting framework.

This development will consist of two parts, client side agents that will be installed on host devices and a cloud hosted centralized server. The agents will monitor antivirus events as they are received on the hosts, securely sending any information of importance to the server. The server will display the real time agent data concerning antivirus performance along with localizing malware threats with publically available IP geo-location databases.

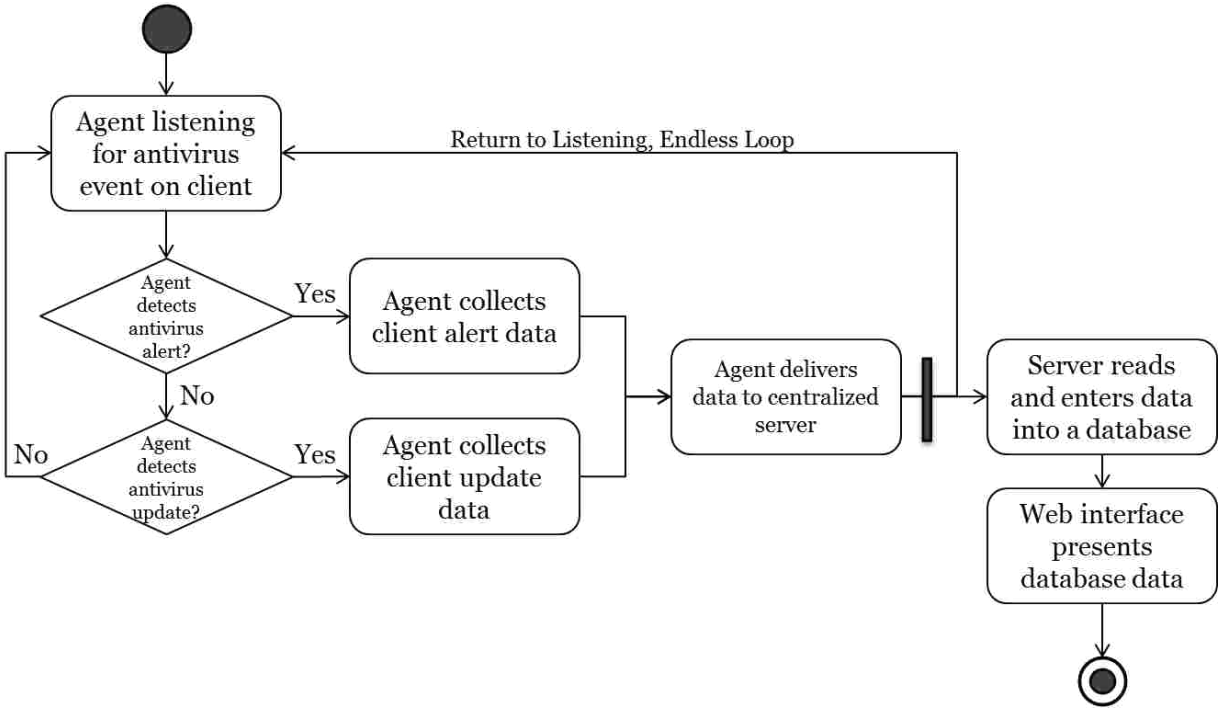

Figure 2: Prototype Flowchart

3.1.1 Development Environment

This prototype consists of the construction of a universal agent that can be distributed across multiple clients ranging in antivirus software using a Windows development environment. The reason for programming in this environment is due to the easily accessible antivirus software available and ample malware samples. The following antivirus vendors are initially proposed to be analyzed by this framework prototype:

- Sophos
- Symantec
- Windows Defender
- AVG

- Kaspersky
- McAfee

These antivirus vendors, excluding Sophos, were chosen because their products were listed among the top ten worldwide contenders for the antivirus market share in the OPSWAT "Market Share Report for Worldwide Antivirus Vendors" released in December 2012 (OPSWAT). Sophos was additionally chosen because of the developer's familiarity with the antivirus and to provide diversity in the antivirus products examined over past studies such as Sukwong's work on vendor comparison.

3.1.2 Prototype

The agent will be programed in Python 2.7. Python is a high-level programming language that simplifies development by requiring fewer lines of code to perform functions. Python also includes a large community of developers who are actively creating and maintaining libraries that may be useful in the prototype development. The current production versions of Python are 2.7.5 and 3.3.2. Version 2.7.5 is still being maintained and is widely used throughout the programming community. The original release of version of 3.0 contained numerous bugs which discouraged developers from immediate adoption. While versions 3.0 and higher are actively maintained, many third party packages have not yet released candidates compatible with this version (Python Programming Language). Python 2.7 was chosen for the prototype development because of the greater library support and developer familiarity.

The proof of concept will be programmed to look for antivirus update notifications and malware detections. If an update is perceived on the client device, the agent will search and retrieve the virus definition, update timestamp and hashes. If a malware alert is detected the

agent will report the timestamp, malware name, hash, vendor, location and signature version. Additional data will be collected as found necessary during the development of the prototype.

A server will be built to receive data and host antivirus statistics from the distributive agents. The server machine will consist of a Linux machine with the Apache HTTP server installed along with a MySQL database to store data. These tools have been chosen because they are open sourced, require little configuration, simplistic in installation processes and can quickly render a web user interface for data collection. The agent will reside on a host machine and be tested for its capabilities to communicate with the web server. Once compatibility is confirmed, the agents will be installed across multiple devices.

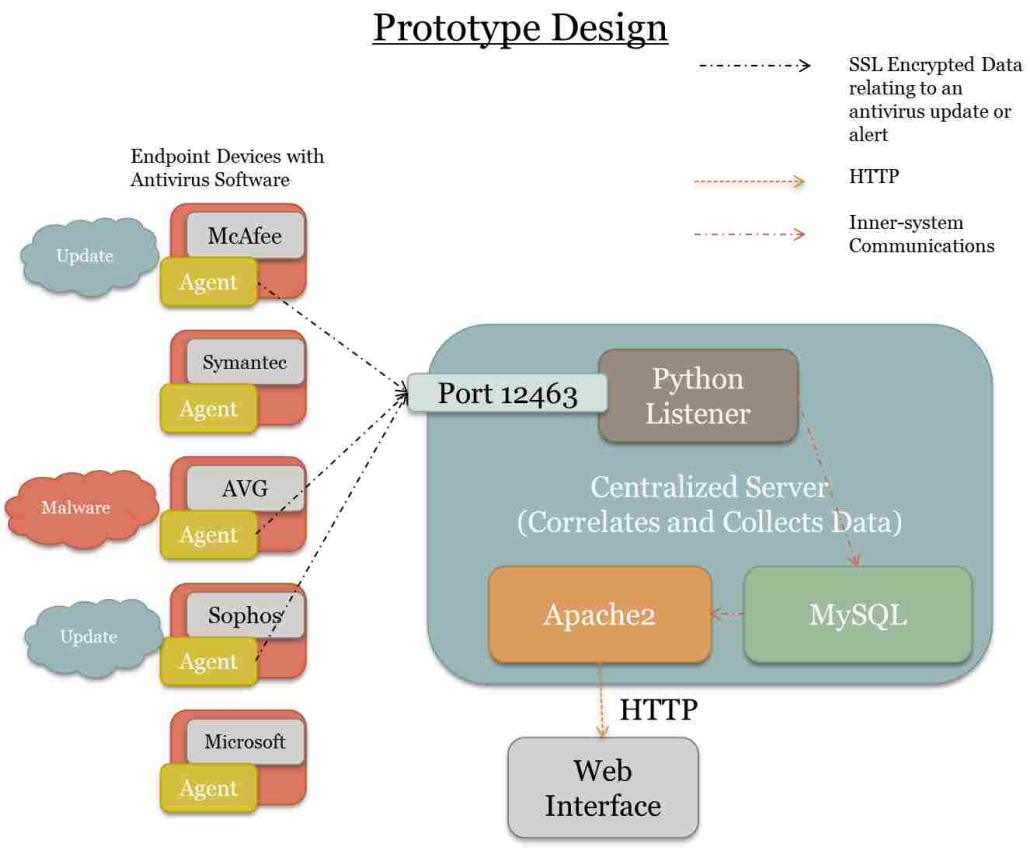

Figure 3: Proposed Distributed Agent Collection Design

3.2 **(R2) Malware Identifiers**

The agents will be used to gather data on malware strands detected by client antivirus software. This information will include the malware hash, name and any other type of identifying information the vendor might use in recognizing malware. The discovered data will be reported to the server and compared against other malware records in order to find what key characteristics identify the same piece of malware universally across all hosts.

3.3 **(R3) Antivirus Variations**

It has been found that different antivirus vendors use different naming conventions to describe the same piece of malware, though the hashes are the same. With this in mind, a hash of a malware file can be used to identify a threat across multiple vendors. The hash and name will be retrieved by the agent at the time of malware detection. These hashes along with the various corresponding vendor names will be formatted and displayed in a table on the server for easy comparison. If there is a correlation between vendors and antivirus naming conventions, the table will display the similarities. However, if the malware names are unrecognizable between vendors, the table will act as a universal connections resource for malware across antivirus software.

3.4 **(R4) Antivirus Malware Protection and Comparison**

The research will use data collected by the agents to compare antivirus vendors and find if there are quantifiable benefits that may be achieved by using multiple vendor products to detect malware.

3.4.1 Data Collection

The prototype will locate and strip the following pieces of data from an antivirus update event.

- Update timestamp
- Hash of virus definition
- Signature version

The prototype will additionally check for real time malware detections by the antivirus. The following data will be extracted from these events.

- Alert timestamp
- Malware name
- Hash of malware file
- Signature version
- Vendor
- Geo-location

These are selected characteristics of antivirus update and malware detection events that will help identify malware universally across multiple antivirus applications and are believed to be of interest to the research. The prototype will test whether the variables exist in each signature before communicating the data to the web server. The data will be formatted and displayed in the web interface for visual comparison of the products ability to release new updates and handle malware threats. Eventually, with real implementation of this framework over a large scale distributed agent network, the collected data can be used to evaluate antivirus resources in

combating new malware threats. This can be accomplished by relaying update notices to the latest malware alerts detected by the agents.

3.4.2 Antivirus Software and the Need for Multiple Installations

As explained in section 2.6, a single installation of an antivirus alone is not believed to be enough to protect a system against malware. This is due to:

1. Various product delays in responding to new threats and distributing antivirus signature updates to endpoint host machines
2. Unpredictability in the time it takes to release a new definition update

With multiple antivirus installations, the combined resources will decrease the time a device is vulnerable to a threat and improve the device's defense against malware. Instead of a host waiting for a single antivirus to respond to a threat, the host has multiple supporting vendors and has only to wait for the fastest.

The framework will prepare a structure for visualizing patterns between the timestamps, signature updates and alerts detected. Over prolonged use and data collection, this framework proposes the ability to aid users in discovering which definitions protect devices against certain specific threats and disclose response times. Malware samples will be tested against the antivirus software in the project scope. These samples will be downloaded randomly from the latest entries to easily accessible, free online malware dumps. This sample set's aim is to be unbiased in replicating a user's environment that potentially could be attacked by malware encountered arbitrarily. Appendix F contains a table of the malware in the sample set and its sources.

The agents will be able to detect which vendors have definitions that protect a device against these pieces or malware. It is believed that different vendors will have varying detections

and response times without consistency over an extended period of time. The percentage of malware detected will be graphed for each vendor. Following, combinations of software will then be graphed to see if there is additional protection provided by multiple installations of malware. This will determine whether this new model of security should be recommended for endpoint devices.

4 FRAMEWORK DEVELOPMENT

The framework consists of two primary components, the client side agents and the centralized data collection server. This chapter goes into detail on the construction of these framework mechanics.

4.1 Data Collection Server

The server is hosted on the Brigham Young University (BYU) Cyber Security Research Lab network. It includes a Secure Socket Layer (SSL) listener for incoming events from the agent clients, a MySQL database to store data and an Apache2 web server used to publicize and graphically display antivirus behavior for analytics.

4.1.1 Python Secure Agent Event Listener

The listener consists of a python script that specifically waits to receive SSL communications on port 12463 from the agents residing on client endpoints. SSL is used to encrypt and secure information delivery between server and client. Table 1 lists the libraries used by the script.

Table 1: Python Secure Agent Event Listener Libraries

Library Name	Description (The Python Standard Library)
socket	Creates a primitive networking interface
_mysql	Used to connect to a MySQL database and execute commands
re	Operator used to support regular expressions
thread	Allows for control of multiple threads
SSL from OpenSSL	Enables access to the Secure Socket Layer for encryption of peer authenticated communications
urlopen from urllib2	Module used to open and read URL requests

When a connection is accepted on the server, it immediately starts a new thread for the incoming stream. Threading enables the server to process multiple incoming requests at once. The more agents distributed among clients, the greater the need for the server to handle multiple incoming requests at various times. Within each new thread, information is read from the client and saved to a temporary variable. The server then verifies the reception by sending a received notification to the client before closing the connection.

4.1.2 MySQL Database

The temporarily saved data that is received from the agents is parsed and inserted into a MySQL database. Incoming messages are classified as either antivirus alerts or updates. A table in the database is dedicated to each type of communication. Table 2 shows the information fields received from each type of communication and tracked in the database. Values underlined must

be unique for each instance in order to prevent duplicate records. Due to the differences in antivirus structures, variable characters, or varchars, are used for each entity because of their ability to store numbers and characters of varying length.

Table 2: Alerts and Updates MySQL Table Structure

Updates Table	Alerts Table
Id (Primary Key)	Id (Primary Key)
Timestamp (varchar)	Timestamp (varchar)
Signature Version (varchar)	Malware Name (varchar)
Signature File/Files Hash (varchar)	Malware File Hash (varchar)
Antivirus Vendor (varchar)	Antivirus Vendor (varchar)
Software Version (varchar)	Software Version (varchar)
Operating System Distribution (varchar)	Signature Version (varchar)
IP Address (varchar)	Operating System Distribution (varchar)
	Action taken against the Malware (varchar)
	IP Address (varchar)

The Updates table tracks new antivirus updates. The timestamp, signature version, antivirus version, IP address and hash must be unique for each record instance. This is to ensure that each client IP only records one update for a new virus signature coordinating to a specific antivirus software version. In the instance that multiple agents share the same public IP address, the timestamp is also recorded. All hashes are of SHA256. This is because Symantec only uses

this hashing algorithm for their malware quarantines. In order to keep records universal across the agents, this particular hashing algorithm has been adopted. The Updates table's purpose is to show how often signatures are delivered to clients. In conjunction with data from the Alerts table, this information will help highlight the threat detection capabilities for each vendor in scope.

The Alerts table tracks all malware threats caught by the monitored antivirus products. The timestamp, malware name, hash and IP address must be unique per instance. All stored hashes in this table are again of SHA256. This hashing function was chosen because Symantec logs their malware hashes in this format. The other vendors do not record hashes in plaintext and require the agents to manually hash files. For consistency, all hashes were recorded in SHA256. With these limitations, each record represents a single attack on the client. Unique timestamps prevent the same attack from being caught and recorded multiple times. It is noted that a threat may attempt to attack the same host at different time intervals. For this study, if an attack occurs at different times, each instance is recorded and labeled as a separate attack.

In addition, if the agent is delivering a malware alert, the server writes an entry into the Markers table. This table records geo-location information to be used by the web server to graphically "mark" malware threats on a Google map. The location information is gathered from a request to http://ipaddress.is querying the client's public IP address. The latitude, longitude and malware name fields must be unique for each entry into the table. This allows multiple attacks to be recorded for a single location without duplicate attacks listed. One listing of an attack per location is sufficient to alert users of a threat in the area. Duplicate entries are believed to just take up storage space in the database.

Table 3: Geo-Location Variables Tracked in the Database

Markers Table
Id (Primary Key)
Latitude (varchar)
Longitude (varchar)
Malware Name (varchar)

Try and catch statements are implemented to catch any errors or issues and maintain server communications to the clients at all times.

4.1.3 Web Server

The installed Apache2 web server is used to display all agent gathered data. It consists of a single PHP page that dynamically writes out HTML content according to the data found in the MySQL database. The page is then formatted and stylized by CSS and JQUERY to be more user friendly. The address for the website is http://itsecurity.et.byu.edu:85/.

Content is divided up between six tabs on the website. The first tab, “Threats Detected,” discusses the objectives of the project along with displaying alert coordinates from the Markers table in a Google map.

Figure 4: Threats Detected Tab

The next tab, "Antivirus Update Notices," lists all of the updates obtained by the endpoint agents. Data is presented in an interactive JQUERY table that allows users to search, sort and change the amount of visible records. Data on this page notifies users of how often vendors update their signatures.

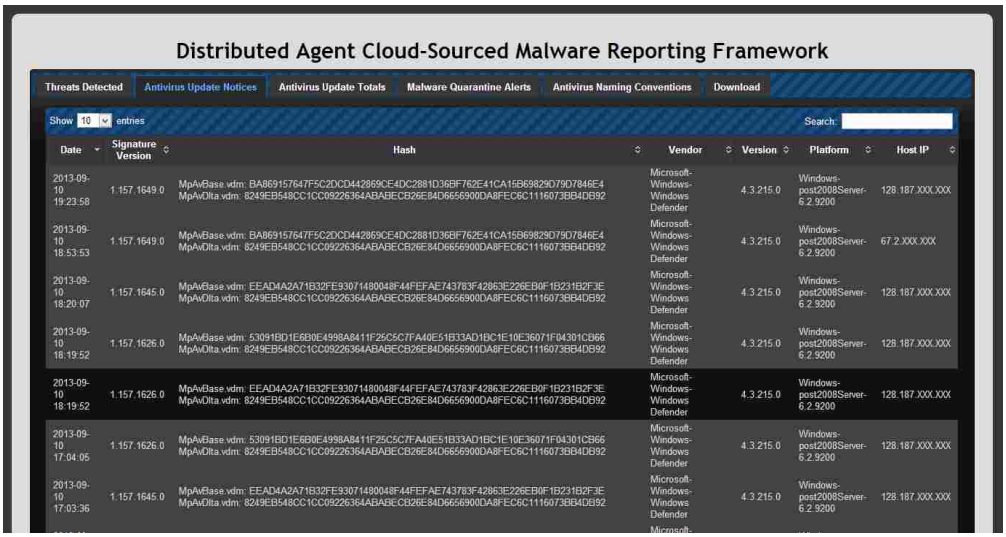

Figure 5: Antivirus Update Notices Tab

The following tab, “Antivirus Update Totals”, provides a listing of summary update statistics for each vendor. This provides detailed information on how often a vendor releases new signature updates to clients. However, the accuracy of the statisitcs depends on the clients availability. Most clients are not intended to be operational at all times and future work will look into creating an environment that will catch and report updates on an ongoing basis in order to report accurate times.

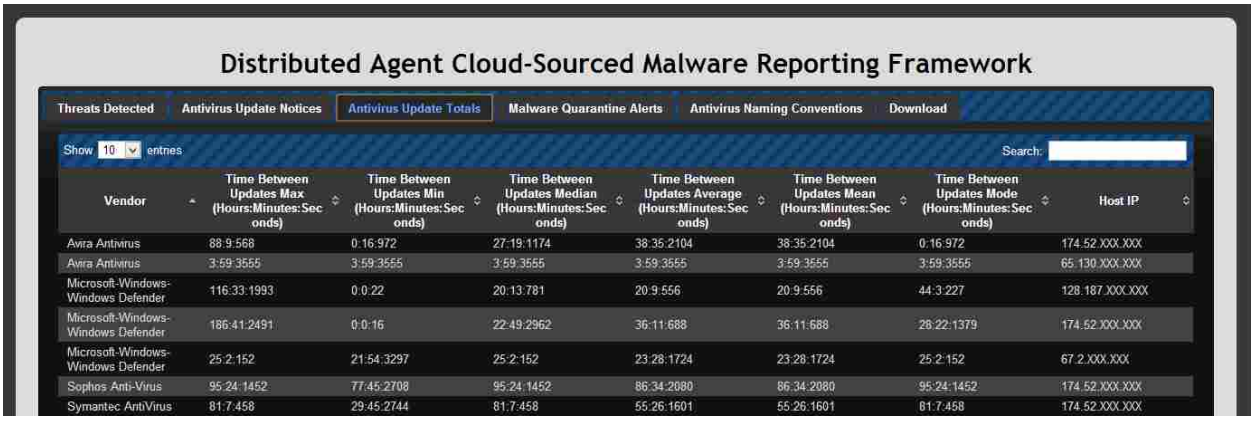

Figure 6: Antivirus Update Totals Tab

The “Malware Quaratine Alerts” tab is similar to the update notices tab except it displays malware alerts detected by the agents. This data shows users which signatures have detected specific malware strands. This framework intends to use this data to compare antivirus vendors and their capabilties in catching new malware strands.

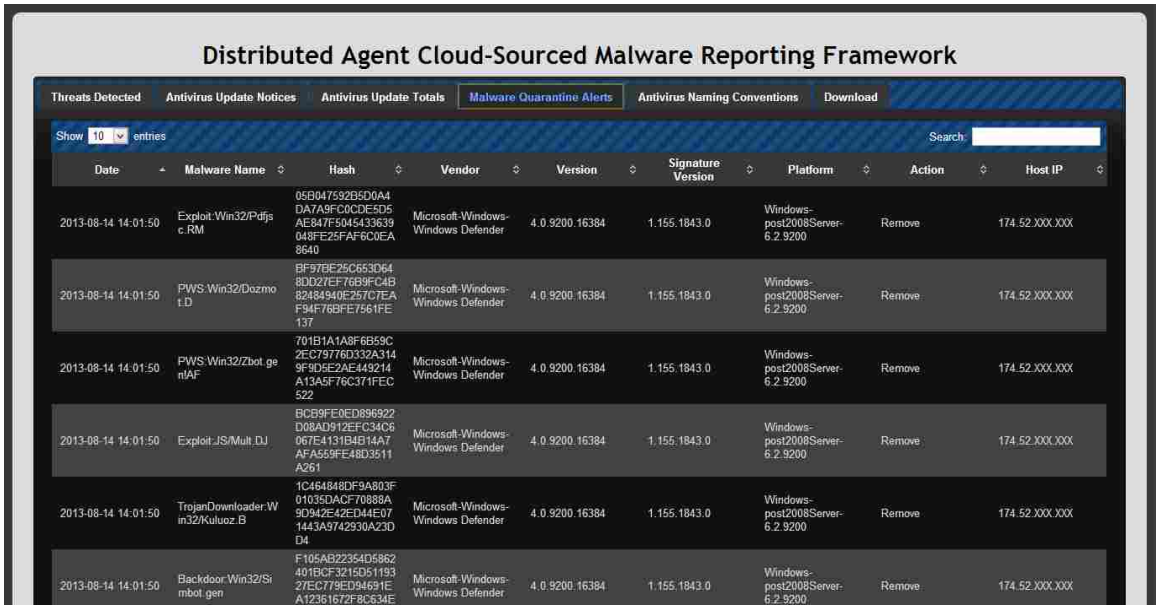

Figure 7: Malware Quarantine Alerts Tab

Continuing, the "Antivirus Naming Convention" tab has been found extremely useful in displaying direct relationships between antivirus vendors and a single piece of malware.

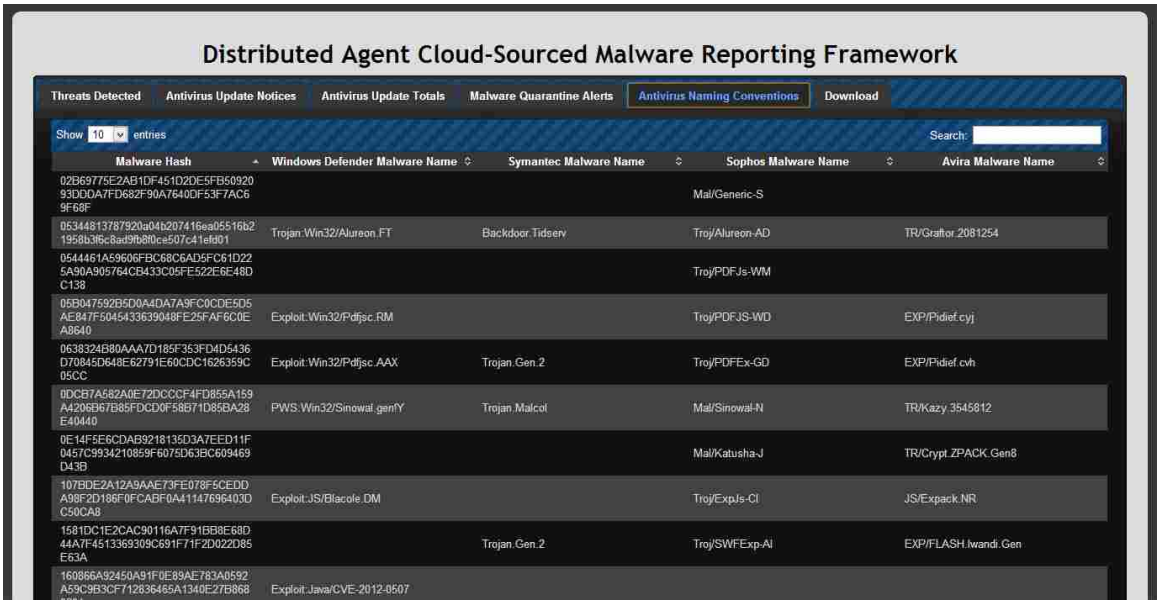

Figure 8: Antivirus Name Conventions Tab

The final tab provides download links to the different agent zipped packages. Users can choose to download the agent that coordinates with their specific antivirus. Each package includes everything necessary to operate the agent.

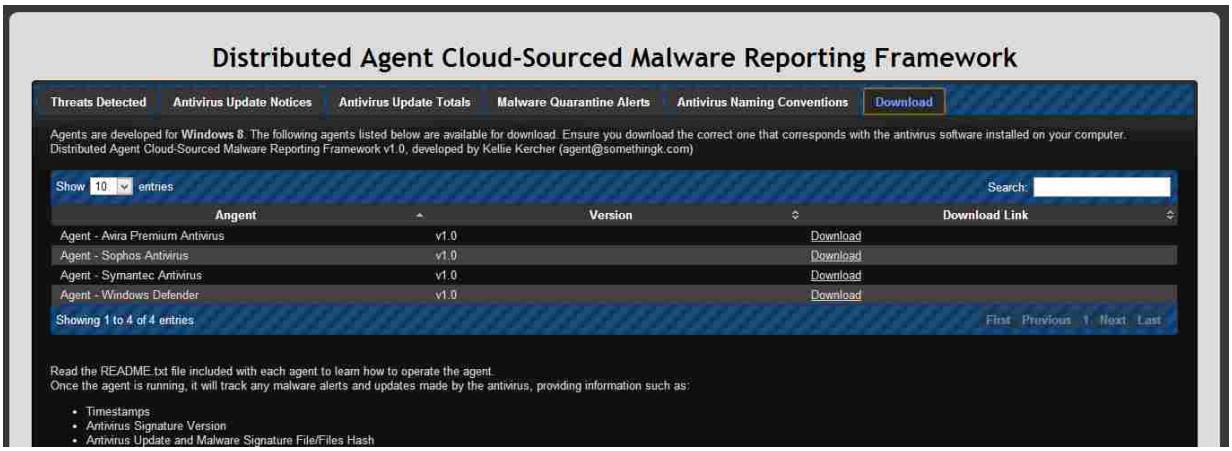

Figure 9: Download Tab

4.2 Client Side Agents

Initially it was believed that all antivirus software alerts and updates appeared in the Windows 8 event logs, enabling the development of a universal agent that could operate across multiple vendors. However, out of the proposed antiviruses for the project, Windows Defender and Symantec were the only ones to record all their events in the event log. The other vendors dealt with logging differently. This discovery led to each agent being customized for a specific vendor. It was also found that many of the antivirus companies encrypt their logs. The following vendors could not be analyzed for the project and were removed from scope:

- AVG
- Kaspersky
- McAfee

The remaining vendors compatible with the prototype included Windows Defender, Symantec and Sophos. Avira was later added to the list to increase the dataset. According to the OPSWAT "Market Share Report for Worldwide Antivirus Vendors" released December 2012, Avira had the 5th largest market share of 10.4% (OPSWAT), closely following Microsoft and Symantec. With this significant popularity, it was chosen as a good substitute for the encrypted antivirus solutions that would not work with the agent developments.

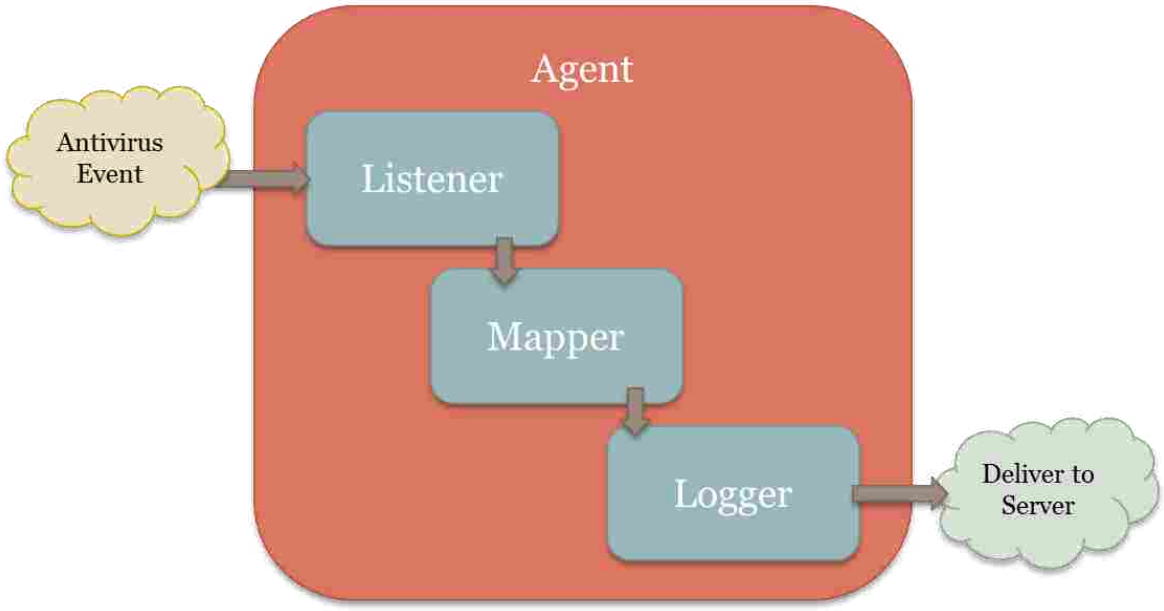

Figure 10: Agent Architecture

Each agent consists of three key components, which are the listener, mapper and logger functions. The listener watches the client environment for any antivirus related changes to

software or window log files. Once a relating event is detected, the listener triggers the mapper component. This function looks into the log file and gathers information relating to the event. Once all data is gathered, it is passed to the logger which formats the data and delivers it to the server. The logger function is programmed exactly the same in each agent, with the listener and mapper functions differing depending on the vendor. The following sections describe the listener and mapper functions in more detailed for each agent and coordinating vendor software.

In order for the agent to be more adaptable, each agent reads configuration settings from a marked file. Users can record and make changes to this file affecting where the agent addresses the server path along with system paths to the antivirus files. In future development, paths may change in new releases of the Windows operating system. As long as the antivirus logs events similarly, the agent will be able to operate with the new path entries without another development release.

The following sections go into detail on the individual actions and programing of each agent after they have read in configuration file details.

4.2.1 Windows Defender

Microsoft Windows Defender writes to its own Windows event log named Microsoft-Windows-Windows Defender/Operational. The logger component of the agent monitors this event log for changes. Python has a library for accessing the System, Application and Security event logs, however it does not include functions for accessing individual application event logs. With this in mind, it was found that Windows has a native command, wevtutil, which can be used to access any event log stored in the `C:\Windows\System32\winevt\Logs` directory. The ideal situation would have the agent trigger with a new event entry in the

Windows Defender log but the limitations on the querying ability of wevtutil prevent this action. Instead, the tool can be used to pull events created within a specified range of milliseconds. In the agent, an endless loop is created to run this command with the default delay time set to 3000 milliseconds. This variable time can be changed within the configuration file for faster or slower processors.

If event records are returned from the wevtutil and are not duplicates from previous requests, they are individual threaded and passed to the mapper. Within this component, the `getEventType` function is first called. This function uses regular expressions to determine the event id. Different events have different types of ids that describe the overall behavior of an event. For instance, in Windows Defender, an event id of 1116 describes a malware detection event or blocked file access. The `getEventType` function looks strictly for 1116 events and 2000 events otherwise known as antivirus updates.

With the detection of an 1116 event, the `getEventType` collects the hash with hashlib of the malware file detected by Windows Defender. This path is included in the event description. Following, the function calls `getAlert` which listens for an 1117 malware response event. Often there is a race case in collecting this file hash before the antivirus deletes or moves the file to quarantine. This is why 1116 events are captured. The agent has time to collect the filename and hashes before the antivirus has taken action against the malware and logs an 1117 event.

Even with these measures, the agent may still fail in getting a file hash. Hashes are necessary in correctly identifying malware across vendors. For researchers, it is proposed to turn off Windows Defender Real Time protection. Without this immediate scan, more time is presented to the agent to gather the file hash during a manual scan of a file or directory. Turning

this setting off is not recommended for the average computer user because it may leave their system very vulnerable. Windows 8 does not include the option to scan individual files natively in the context menu that appears when a user right-clicks on a folder. For the users' convenience, directions on how to set a registry field to include this option are provided in the agent package. Further details on the contents of the Agent package are included in the Usage and Project Distribution section.

The `getAlert` function uses the wevtutil command to pull events in search of the 1117 response to the earlier found 1116 event. Within the 1116 event's content is a malware id. This id is found with the use of regular expressions in `getEventType` and passed to `getAlert`. The function then searches all 1117 events for this malware id. The `getAlert` function loops through until this combination is found or until 25 minutes have passed. This timing prevents an infinite loop. In the case that multiple malware strands are detected at the same time, each detection is connected to a single thread that runs through this process.

When a coordinating 1117 event is found, regular expressions are used to search the event content for the timestamp of the event, antivirus signature version, malware name and action. The antivirus software version is found by using the python `GetFileVersionInfo` tool from the win32api library. This tool returns the version of a file provided, in this case the Windows Defender executable. The version of this file coordinates with the software version listed in the Windows Defender's main menu window. The platform python library is used to determine the operating system of the endpoint device running the agent. The client's public IP address is collected by the `getIP` function. This function sends a request to http://httpbin.org/ip. In response, this webpage delivers the public IP in plaintext JSON which the function collects and returns. Once all this reporting information is gathered, it is delivered to the server. Figure

11 displays an instance where an agent has delivered malware alert data found to the project server. The universal report format used by the logger for all agent alerts is as follows:

```
<MYSQL ENTRY TABLE NAME>*<TIMESTAMP>*<MALWARE
NAME>*<HASH>*<VENDOR>*<SOFTWARE VERSION>*<SIGNATURE
VERSION>*<PLATFORM>*<ACTION>*<PUBLIC IP>
```

The server will split the string by the '*' character and put the segments into the MySQL database.

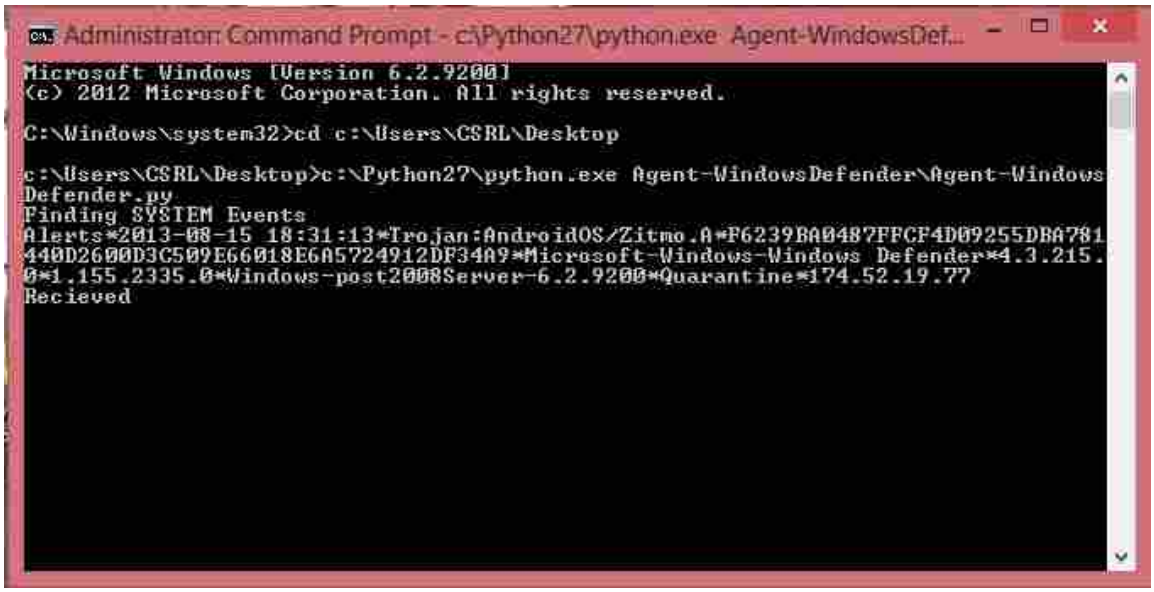

Figure 11: Agent Delivered Alert Report

Alternatively, if the agent mapper detects a 2000 event id within `getEventType`, it calls the `getUpdate` function and passes the event data. This function uses regular expression to pull the timestamp and the new signature version. The `GetFileVersionInfo` tool is again used to get the software version. A hash is taken of the Windows Defender `MpAvBase.vdm`

and `MpAvDlta.vdm` files. These are the signature files that identify the update. `MpAvBase.vdm` contains the base virus definition module. It is updated once a month with new virus definitions. `MpAvDlta.vdm` is updated multiple times a day. It contains all the changes that have occurred since the last base file was created. The platform python library is again used to locate the operating system of the client. The public IP address is collected by the `getIP` function described earlier. Once all this reporting information is found, the agent concludes by delivering the update information to the server. Figure 12 displays an instance where an agent has delivered update data to the project server.

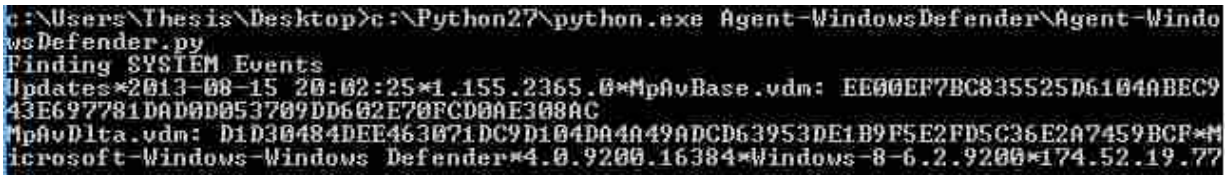

Figure 12: Agent Delivered Update Report

The universal report format for all agent logger updates is as follows:

```
<MYSQL ENTRY TABLE NAME>*<TIMESTAMP>*<SIGNATURE
VERSION>*<HASH>*<VENDOR>*<SOFTWARE VERSION>*<PLATFORM>*<PUBLIC
IP>
```

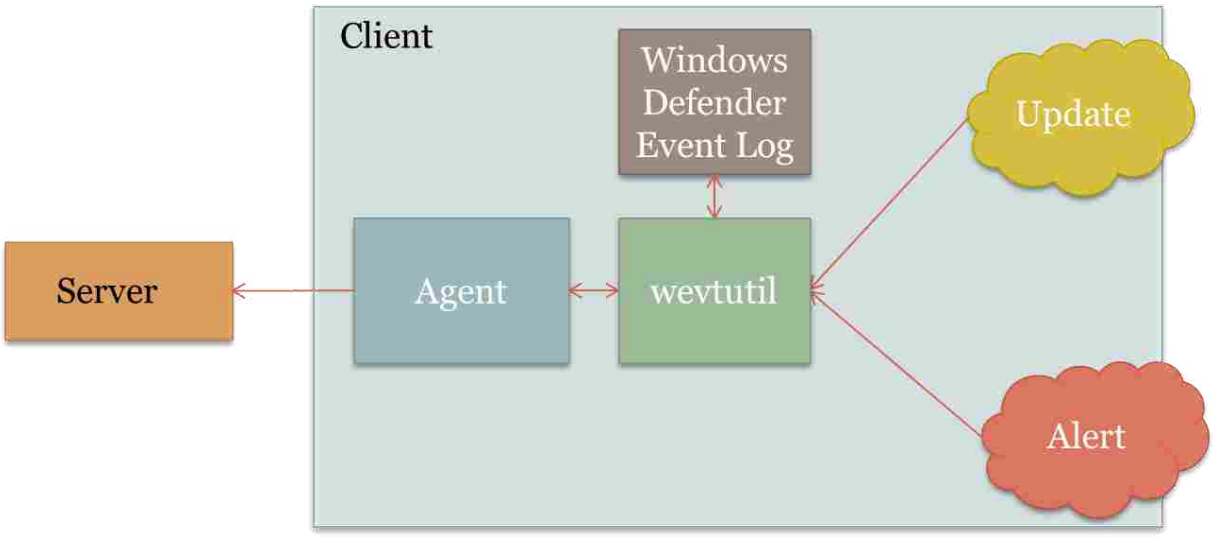

Figure 13: Windows Defender Agent Architecture

The libraries used by this agent are found in Table 4.

Table 4: Windows Defender Agent Libraries

Library Name	Description (The Python Standard Library)
socket	Creates a primitive networking interface
ssl	Creates an SSL wrapper for socket objects
GetFileVersionInfo, LOWORD, HIWORD from win32api	Tools that find and format the version of a file.
pprint	Prints formatted data
platform	Tool that returns the host's operating system platform
json	Decodes and encodes JSON strings

Table 4, Continued

Library Name	Description (The Python Standard Library)
re	Regular expression operator
os	Provides a toolset used to interact with the operating system interface
hashlib	Tool used to hash files
datetime	Returns formatted date and time objects
time	Allows access to time objects and conversion
sys	Returns system-specific
thread	Allows for control of multiple threads
urlopen from urllib2	Module used to open and read URL requests

4.2.2 Symantec

Symantec writes all updates and malware alerts to the Windows Application event log. This agent utilizes the win32evtlog library to access and read these event logs. The win32event library is used in conjunction with win32evtlog to create a listener that triggers a new thread operation whenever a new Application log event occurs. Each new thread operation calls `getEvent` and passes the function the event data retrieved by the listener.

The `getEvent` script, within the mapper component, analyzes the captured event data with regular expressions to determine if it is a Symantec event. The win32log library includes a

function to pull an event's source and event id. If it is found to be an event from Symantec, the agent gathers a few environmental variables. The platform python library is used to determine the operating system of the endpoint device running the agent. The function previously discussed in the Window Defender section, `getIP`, is used to get the public IP address of the client. `GetFileVersionInfo` is called on the Symantec main executable to retrieve the software version and the timestamp is taken from the event and formatted for consistency across all agent reports. Lastly, the current signature version is found in the `definfo.dat` file.

After these variables are collected, the agent looks at the event's id. An id of 1090453511 indicates a virus definition update. With an update request, the agent collects the new signature version from the event content and takes a hash of the Symantec update virus `catalog.dat` file to uniquely identify the update's content. Symantec uses this `catalog.dat` file to build custom signature files depending on the host's platform. This information is all reported to the server.

A 400 event describes an instance where the antivirus has detected and blocked the user from attempting to download a piece of malware. If the agent comes upon this event, it filters out the malware name detected and the action taken against it. The operating system platform is again detected along with the public IP address by the `getIP` function. This data along with the previously collected environmental information is then reported to the server.

The last event the agent looks for is a 109045355 or malware detection incident on the device. Similar to the block event, the agent uses regular expressions to get the name and action performed against the malware. In addition, the agent then queries the quarantine for the malware. By default Symantec creates an entry in the quarantine for each detected piece of malware. Within this entry, the software records the hash and name of the malware in plaintext.

The agent uses the `getHash` function to locate the newest quarantine entry that contains that name of the discovered malware and attempts to collect the hash of the file. This information along with the detected operating system and the public IP address obtained by the `getIP` function is reported to the server by the logger.

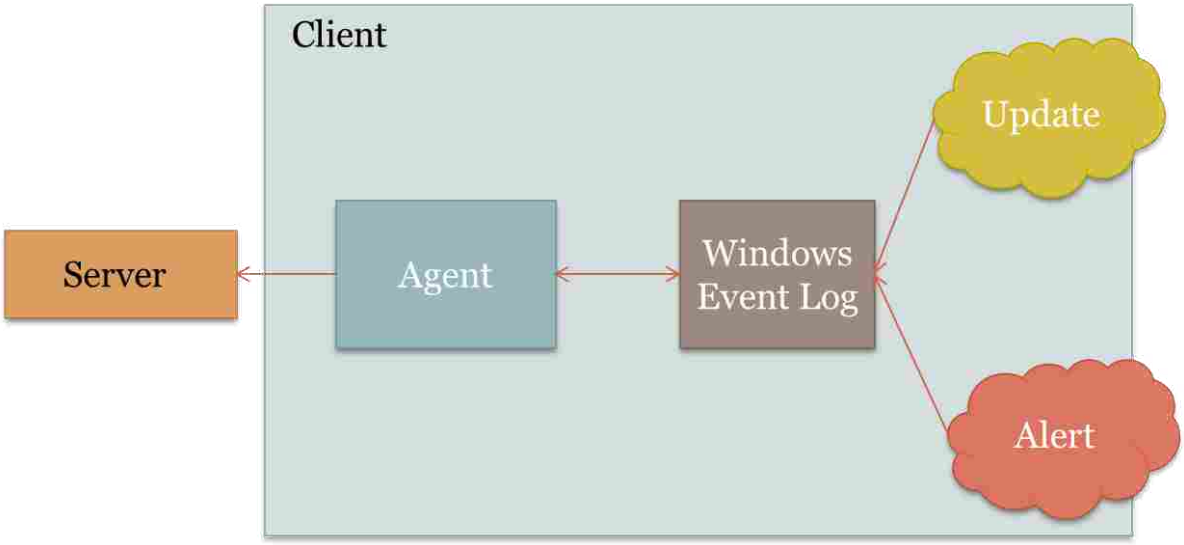

Figure 14: Symantec Agent Architecture

The libraries used by this agent are found in Table 5.

Table 5: Symantec Agent Libraries

Library Name	Description (The Python Standard Library)
win32evtlog	Provides an API for accessing the Windows event logs
win32event	Includes functions for interacting with the win32 event API
win32api	Encapsulates the Windows Win32 API into python calls
win32con	Contains tools for accessing the Windows registry files

Table 5, Continued

Library Name	Description (The Python Standard Library)
win32evtlogutil	Tool used to retrieve the actual content body of text for event
socket	Creates a primitive networking interface
ssl	Creates an SSL wrapper for socket objects
GetFileVersionInfo, LOWORD, HIWORD from win32api	Tools that find and format the version of a file.
pprint	Prints formatted data
platform	Tool that returns the host's operating system platform
json	Decodes and encodes JSON strings
re	Regular expression operator
os	Provides a toolset used to interact with the operating system interface
hashlib	Tool used to hash files
datetime	Returns formatted date and time objects

4.2.3 Sophos

Sophos will write malware detection notifications to the Windows Application event log but it does not write signature updates to this log. So in addition to using the win32log and win32event libraries as in the Symantec agent, the Sophos agent also has a listener watching the Sophos update log text file in the `C:\ProgramData\Sophos\AutoUpdate\Logs\` directory for changes. A change made to this log indicates an update to the Sophos signature file.

This listener works by constantly checking the modified time on the log file. If the modified times change then the agent knows an update has been made and it triggers.

If the win32 libraries detect a new event log, a new thread is created that passes the event to the `getAlert` function. This mapper function's first priority is to determine if the event's source is Sophos. The win32log library includes a function to pull an event's source along with other event details. If the event is from Sophos numerous environmental variables are then collected. The timestamp is collected from the event and formatted along with the Sophos software version by the `GetFileVersionInfo` tool. Following, the agent looks at the event id.

An event id of 542638091 or 539295776 indicates a webpage being blocked or a malware detected on the host. For the blocked event, the name and action is collected and delivered to the host along with the earlier collected environmental data. If the event deals with malware detection on the host, the malware name and file is collected by the agent through regular expressions on the event content. The file is then hashed with hashlib In some instances this step fails because the antivirus was quicker than the agent and has already removed the file. The hash is not required in order to create a report. However, in order to prevent this, Sophos on-access protection can be disabled. Without this setting the user is required to manually scan suspicious files providing more time for the agent to react. This setting is not recommended for inexperience computer users because it may render their system vulnerable to attacks. The IP address is collected by the `getIP` function described in the Windows Defender section along with the platform retained by the python platform tool. Once all data is collected, it is delivered to the server.

If an update was detected instead of a malware incident, the `getUpdate` function is called. This is a simple function that collects the same environmental variables as the `getAlert` function and in addition uses regular expressions to get the latest update information from Sophos's main log file, `SAV.txt`. This information includes the signature version and the number of viruses it can detect. Finally, the `hash` function is then called. This function looks for and hashes the latest IDEs or signature files within the last specified number of minutes. Users can set this number in the configuration file depending on how often Sophos is set to update. The data is then accumulated with the platform retained by the python platform tool and the public IP found with the `getIP` function. Everything is then reported to the project server by the logger.

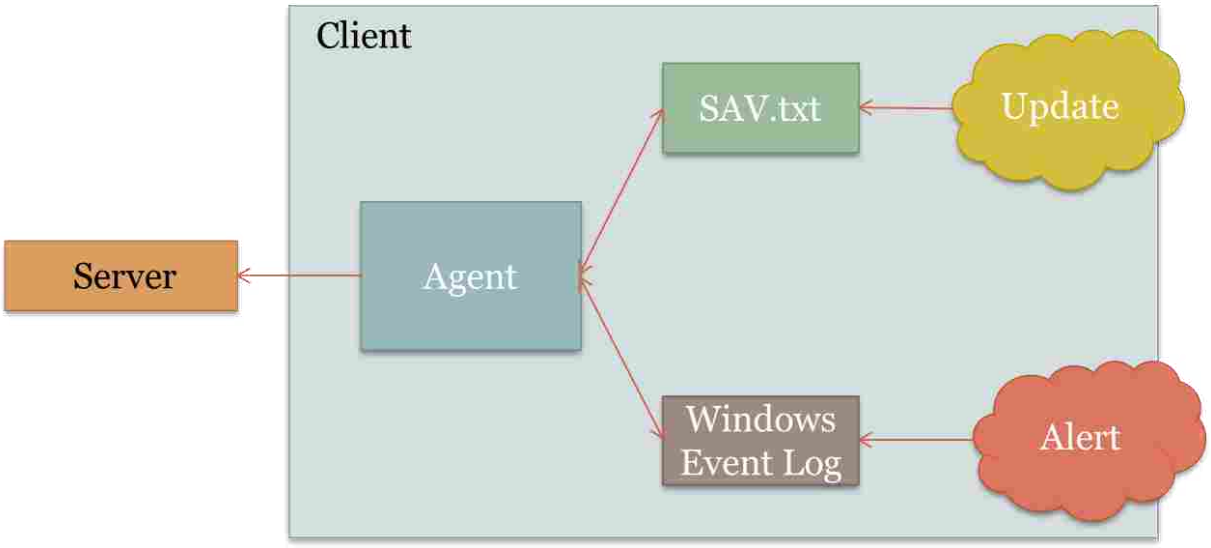

Figure 15: Sophos Agent Architecture

The libraries used by this agent are found in Table 6.

Table 6: Sophos Agent Libraries

Library Name	Description (The Python Standard Library)
win32evtlog	Provides an API for accessing the Windows event logs
win32event	Includes functions for interacting with the win32 event API
win32api	Encapsulates the Windows Win32 API into python calls
win32con	Contains tools for accessing the Windows registry files
win32evtlogutil	Tool used to retrieve the actual content body of text for event
socket	Creates a primitive networking interface
ssl	Creates an SSL wrapper for socket objects
GetFileVersionInfo, LOWORD, HIWORD from win32api	Tools that find and format the version of a file.
pprint	Prints formatted data
platform	Tool that returns the host's operating system platform
json	Decodes and encodes JSON strings
re	Regular expression operator
os	Provides a toolset used to interact with the operating system interface
hashlib	Tool used to hash files
datetime	Returns formatted date and time objects

Table 6, Continued

Library Name	Description (The Python Standard Library)
time	Allows access to time objects and conversion
sys	Returns system-specific
thread	Allows for control of multiple threads
codecs	Includes multiple file reading codecs and base classes
urlopen from urllib2	Module used to open and read URL requests

4.2.4 Avira

Similar to Sophos, Avira only records malware detection events into the Windows event log. In order for this agent to be able to catch the same alerts as the other agents, multiple listener triggers had to be developed. The first trigger the agent uses combines the win32event and win32evtlog libraries to detect incoming Windows Application events for Avira malware detection alerts. When a new event appears, a new thread is created and calls `getAlert`, passing the function the incoming event. This mapper function checks that the event's source is Avira and then looks at the event id. If the event has an id of -2147479353, it indicates malware detection caught by Avira. This function then uses regular expressions and the win32evtlog tools to gather information on the event. This data includes the malware name, file, action and timestamp of detection. The file is hashed using the hashlib library. If a hash is not found, it is noted in the reporting data sent to the server. To get the software version of Avira, the Avira build file is opened and queried for the current version. The current virus signature version is

found and read from the registry. This information is then bundled with the detected platform and public IP address and delivered to the server.

The agent also watches for modification on the `C:\ProgramData\Avira\AntiVir Desktop\BACKUP` directory. If there has been a change, it means Avira has backed up previous data in order to take in new virus signature files. When a change is made to this directory, the agent first calls `findFile` with the requested file type being an update log. This function then returns the latest update log file recorded in the Avira log directory. Avira makes a new log file for each operation it performs and the type of log is easily identified by the naming convention. Update logs include "Udp" in the name while scanning files include "AVSCAN" in the name followed by the date and time. Once the log file is returned, the function calls the `getEvent` function and passes it the detected log file along with the boolean update argument set to true.

The third trigger deals with the Avira webguard. Since the other agents detect malicious web content, it was a goal to have this agent also report suspicious web content. Every single blocked web address is recorded in the Avira webguard.log file. If a modification on this log file is detected the agent will call `getEvent`, with the boolean argument block set to true.

The `getEvent` mapper function will perform different functions depending on the arguments provided. For all event types, the function reads the Avira build file to get the current software version and reads the registry to get the current signature version. If update is set to true, the function will read through the update log provided and hash each of the new signature file paths listed in the log. The hashes are combined along with the signature, timestamp, software version, platform found from the python platform tool and public IP returned by the

`getIP` function described earlier. This information is formatted and finally delivered to the server.

For a blocked event, the getEvent function reads the webguard.log file and uses regular expressions to capture the last entry. Inside this entry is the malware name and action taken against it. For these types of events there is no need of hashes because there is no file, it has been blocked. Finally, this information along with the earlier collected data, the platform and public IP address is delivered to the server by the logger.

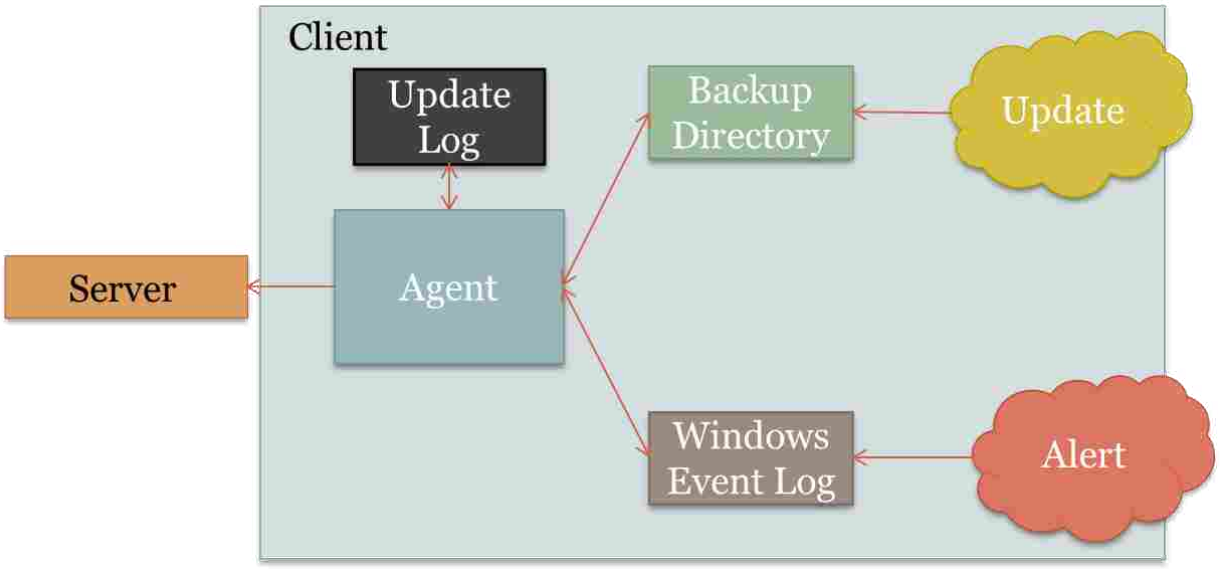

Figure 16: Avira Agent Architecture

The libraries used by this agent are included in Table 7.

Table 7: Avira Agent Libraries

Library Name	Description (The Python Standard Library)
win32evtlog	Provides an API for accessing the Windows event logs
win32event	Includes functions for interacting with the win32 event API
win32api	Encapsulates the Windows Win32 API into python calls
win32con	Contains tools for accessing the Windows registry files
win32evtlogutil	Tool used to retrieve the actual content body of text for event
socket	Creates a primitive networking interface
ssl	Creates an SSL wrapper for socket objects
re	Regular expression operator
os	Provides a toolset used to interact with the operating system interface
hashlib	Tool used to hash files
datetime	Returns formatted date and time objects
time	Allows access to time objects and conversion
sys	Returns system-specific
thread	Allows for control of multiple threads
codecs	Includes multiple file reading codecs and base classes
urlopen from urllib2	Module used to open and read URL requests

4.2.5 Problems Encountered

There were numerous problems encountered while developing these agents. The following sections describe the issues and solutions used to overcome the challenges.

Antivirus Encrypted Logs

The most difficult issues dealt with finding software that did not encrypt all its logs files. This was especially pertinent when the antivirus did not log events to the Windows event logs. Due to intellectual property laws, no attempts were made to decrypt logs or extract private data from any of the antivirus software. Some of the incompatible vendor possibilities explored included:

- Panda
- AVG
- Trend Micro
- Kaspersky
- Norton
- McAfee

A lot of time went into exploring the file structure and application data for each of these vendors. For future work, different tools such as Procmon could be used to identify antivirus behavior. The vendors could also be contacted in order to find if they have interest in working with the project. These efforts may lead to new antivirus agent support.

Logging Differences

Once an antivirus was discovered that recorded logging in plaintext, the next step was to attempt to figure out how to get update and alert data from the logs. Each development was a new learning experience. Virtual machines were utilized in the testing and development stages. A virtual machine is a virtualized environment that replicates the behaviors of a physical computer. With this setup, machines could be reverted back to previous snapshots making it easier to test if an antivirus correctly responded to an update or malware alert. However, it was still a slow process of trial and error experimenting to see which scripting logic would return desired results. Another issue involved race cases where the antivirus deleted a malware file before the agent could hash the file. It was discovered with Windows Defender and Sophos that when real time scanning was enabled, the agent had difficulties hashing a malware file before an antivirus discovered and removed it. With this setting turned off, the agent had plenty of time to hash the file. However, it is not recommend for all users to turn of this additional layer of security. If a malware hash fails, the server will still receive a report of the incident but the hash will not be included.

Endless Hashing Loop

In some instances where the agent could hash the file first, the antivirus would get caught in an endless loop responding to the file hashed as a new malware event. With the new event, the agent would again respond and attempt to hash the file, repeating the process infinitely until the agent was stopped. This was fixed by using the python `file` function instead of `open`. The open function from the python standard library creates a file object while the file function creates

a constructor or type. The object created by `open` may have been interrupted as a new malware instance by the antivirus causing the infinite loop.

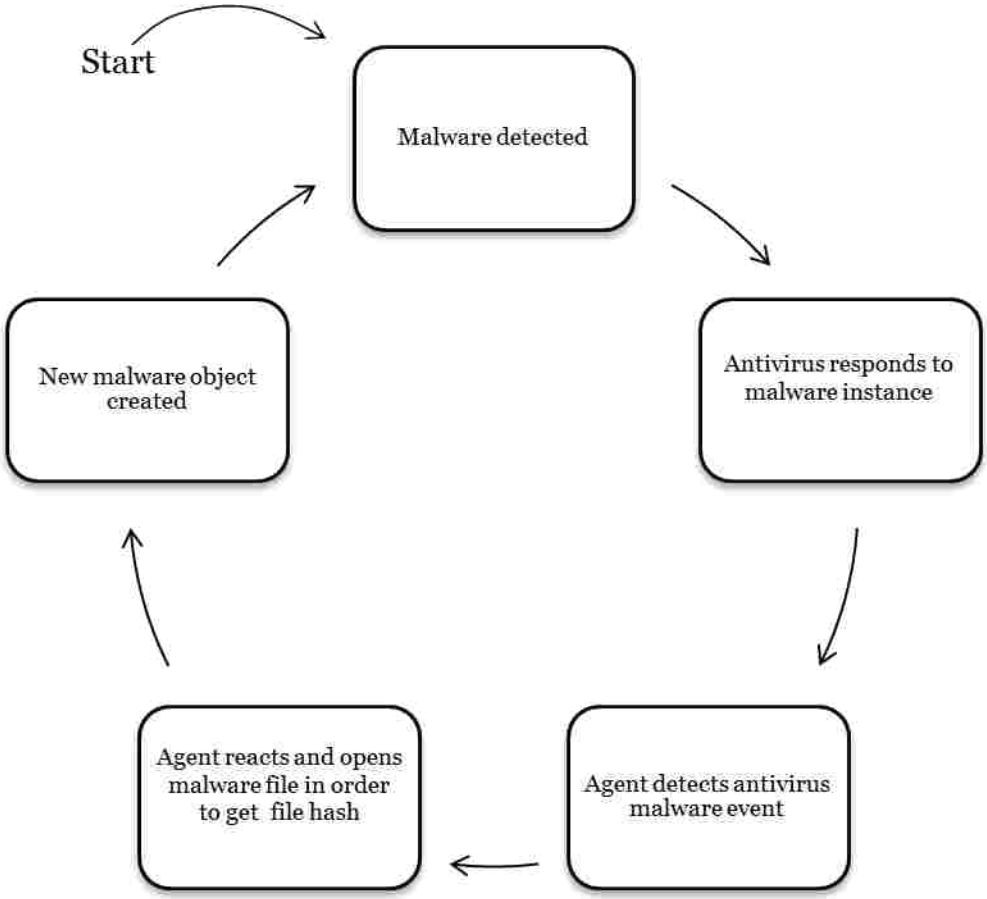

Figure 17: Endless Hashing Loop

Resource Consumption

There was also a concern for the processor consumption of the agent. If agents were too heavy and slowed down the computer processes, they most likely would not be used on a regular basis. The whole point of the agent is to run quietly in the background catching the random

malware alerts and updates received by the antivirus. This wasn't a problem for most of the developed agents except for the Avira agent. Avira scatter writes data to numerous log files.

In the beginning the agent monitored four directories for changes. When one of the directories was edited the agent would look to find the new file entry made in that specific directory. Once the file was found the agent would read through it as the file itself was being written. The agent would also have to read multiple other files to get all the environmental variables. This severely loaded up the processor and slowed down all operations on the host. The agent had to be completely redeveloped. Instead of watching numerous directories it was discovered that the agent does communicate some alerts to the Windows event logs. Even though these events were not as detailed as the entries into the text log, it was found more important to have a lighter agent that does not disrupt the user experience. Also the Avira agent is set to watch a log file modified time. Avira will not open and read a file until it has been unmodified for at least five minutes. It is assumed that after five with no changes that the file has been completely written out and is safe to read. It was found that reading a file while it is being written slows the processer considerably and contributed to Avira's consumption.

There were many times when agent development was believed to have been completed but then multiple small errors became apparent. Many checks and exception cases were put into the agents to prevent endless loops. There may be a time when an antivirus operation may abruptly stop before finishing a log entry causing the agent to stall. If this happens, the agent is set to stop querying for a specific log entry after 25 minutes.

Stress Testing Overload

When introducing numerous malware samples into an environment at once, some of the agents were at first unable to handle the massive amount of alerts. Some events would even be skipped. After further development, the agent behavior was corrected. Multi-threading allowed the agents to responded to multiple events at once without pausing the event listening triggers.

Duplicate Entries

Despite its benefits, multi-threading events often caused duplicate entries. To prevent these doubles, there are checks scripted on the server and client. The server's MySQL database requires certain field combinations to be unique before being inserted into a table. The clients manually check to assure that an event that previously triggered an agent does not match the last triggering event. These checks severely cut down the number of duplicate entries in the database. Often a client's antivirus text or event logs may record the same malware incident twice but with different timestamps. These types of report duplicates are unpredictable and not blocked by any of the checks. These events are not blocked because it cannot reliably be determined if the malware attacked the system twice within a close proximity or if this is the same malware attack recorded multiple times in the logs.

User Configuration

One objective to development was to create an agent that can immediately run out of the box in its target environment without any necessary changes from the user. This was more a convenience to users rather than a necessity. With this in mind, some of the previously mentioned antivirus vendors could not be used. Some software products have configuration

settings available that allow the user to set the antivirus to write to the Windows event logs. However, this requires the user to understand what the event logs are and how to access them. The project framework aims to get as many user clients as possible running agents. Some users may be deterred by the need to make additional configuration settings. The agent is more likely to attract users if little configuration is required.

4.3 **Testing**

Testing was used throughout the entire developmental stages of the framework to assure the agent exhibited the correct functionality. Final testing stressed the agent's abilities to handle a massive attack of numerous malware samples downloaded from various sources and also its processor consumption.

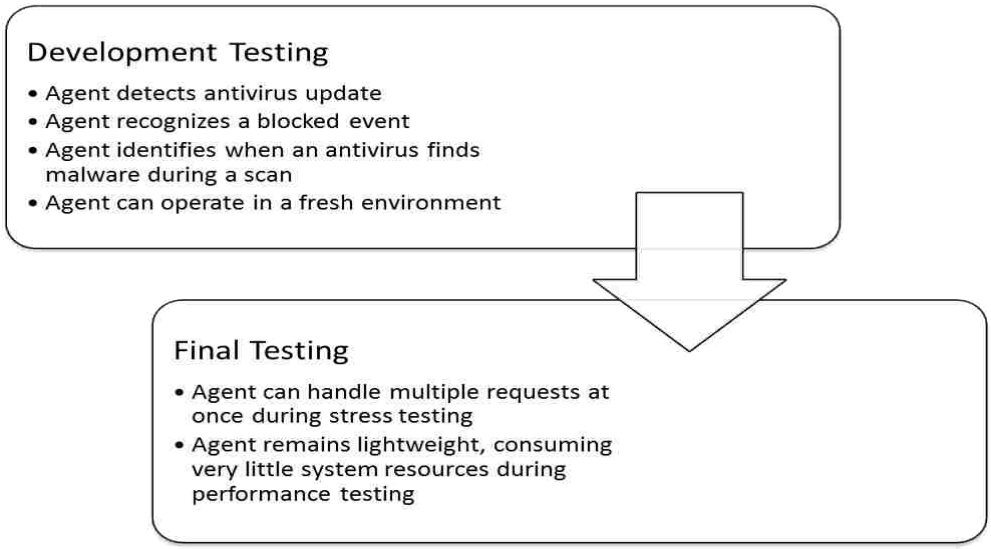

Figure 18: Testing Model

4.3.1 Development Testing

The testing environment consisted of five virtual machines. A virtual machine is a virtualized environment that replicates the behaviors of a physical computer. Four of the virtual machines were each installed with a different antivirus. An agent development was copied into each environment. Each agent was specifically designed and tested to work with the coordinating antivirus installed on the virtual machine. The fifth virtual machine was used to test each antivirus and agent in a fresh environment in order to ensure the agent could work on another machine and was not limited to the development environment.

There were four main tests an agent had to pass in order to move onto final testing:

- Agent detects antivirus update
- Agent recognizes a blocked event
- Agent identifies when an antivirus finds malware during a scan
- Agent can operate in a fresh environment

Agent Detects Antivirus Update

The first test required updating the antivirus and ensuring the update was detected by the agent. If an agent delivered the correct update variables to the server following a client update, than the agent passed the test.

Agent Recognizes a Blocked Event

The second test required the agent to correctly respond to a blocked webpage or file event. During this test, a browser was opened and directed to the Eicar Test Virus download,

http://www.eicar.org/download/eicar.com.txt. The agent passed the test by correctly identifying the malware alert and delivering a report notice to the server.

Agent Identifies when an Antivirus Finds Malware During a Scan

The third test required the agent to identify when and antivirus captures malware during an antivirus scan. For this test, a single piece of malware was downloaded from http://contagiodump.blogspot.com in a zip file and extracted onto the client's desktop. The test was successful if the agent correctly detected the malware sample and delivered the coordinating reporting data back to the project server.

Agent can Operate in a Fresh Environment

The final test ensured that the agent could be copied into a new environment and by fully functional. For this test, the fifth virtual machine was used. An antivirus was installed on the machine followed by the agent. If the agent retained full functionality and passed the prior tests mentioned, the agent was successful. After each agent test, the machine was reverted back to the state it was in before the antivirus was installed. This prepared the virtual machine for the next agent and antivirus installations.

4.3.2 Final Testing

There were two primary tests involved in the final stages of testing:

- Agent can handle multiple requests at once during stress testing
- Agent remains lightweight, consuming very little system resources during performance testing

Agents can Handle Multiple Requests at Once During Stress Testing

40 malware samples were used to test each agent's capabilities in handling malware in real time. A full list of the malware samples can be found in Appendix F. The malware samples were individually zipped and copied onto the desktop of each virtual machine. The malware was unzipped and scanned simultaneously to stress the agent and its report functionality. The purpose of this test was to overwhelm the agent with an unrealistic scenario so as to ensure the agent could handle a typical user environment where malware is encountered a few times a month. If agents failed this test, they were put back into development and tested until it could successfully handle the load.

Agent Remains Lightweight

The final test looked into the performance impact of an installed agent. Each virtual machine in the testing environment consisted of the same less than average specs. This included 2 GB of memory, 1 processor, single core and 20 GB of hard disk space. The hosting machine includes 16 GB of memory, 1 processor, i7 quad-core with 2.40GHz CPUs and 500 GB of hard disk space. At the time of the tests, one virtual machine was powered on at a time running default windows operations, their antivirus and the coordinating agent. This setup was used to provide a free environment in order to accurately view the agent's maximum consumption without any type of bottleneck constrictions. The agent was set to a service in each environment and ran in the background as each of the first three development tests described were performed multiple times. For the malware detection tests, different samples of malware were tested one at a time in order to obtain a representative average. Consumption was monitored on the Window Task Manager. The results of the test are seen in Table 8.

Table 8: Agent Consumption Monitor

	Average Listening Consumption	Average Update Consumption	Average Blocked Consumption	Average Malware Detection Consumption
Windows Defender	0% CPU 8.9 MB Memory 0 MB/s Disk 0 Mbps Network	20.3% CPU 8.9 MB Memory 0.1 MB/s Disk 0.1 Mbps Network	3.1% CPU 8.9 MB Memory 0 MB/s Disk 0.1 Mbps Network	1.7% CPU 9.0 MB Memory 0 MB/s Disk 0.1 Mbps Network
Symantec	0% CPU 5.9 MB Memory 0 MB/s Disk 0 Mbps Network	1.4% CPU 5.9 MB Memory 0 MB/s Disk 0 Mbps Network	2.5% CPU 5.8 MB Memory 0 MB/s Disk 0.1 Mbps Network	10.9% CPU 5.8 MB Memory 0.1 MB/s Disk 0.1 Mbps Network
Sophos	0% CPU 4.9 MB Memory 0 MB/s Disk 0 Mbps Network	3.6% CPU 6.4 MB Memory 0.1 MB/s Disk 0.1 Mbps Network	0.9% CPU 5.5 MB Memory 0 MB/s Disk 0.1 Mbps Network	2.7% CPU 5.7 MB Memory 0 MB/s Disk 0.1 Mbps Network
Avira	0% CPU 5.7 MB Memory 0 MB/s Disk 0 Mbps Network	6.0% CPU 4.9 MB Memory 0.1 MB/s Disk 0.1 Mbps Network	0.8% CPU 5.9 MB Memory 0 MB/s Disk 0.1 Mbps Network	2.1% CPU 5.8 MB Memory 0 MB/s Disk 0.1 Mbps Network

These tests revealed that the agent had little impact on the computer environment performance and a user's experience, except during times when Windows Defender ran an

update. However, the spikes displayed in the table each lasted less than a second. From these tests it is believed the user's activities will be unaffected by the agent.

4.4 **Security**

The security of the client side agents and server was considered throughout the entire development of the framework prototype. If this product is going to be distributed publically it needs to be secure for the protection of user privacy. Steps were taken on the server and client side to secure the project.

The server looks specifically for communications coming in on a designated high numbered port. It is not an obvious or common port for services. This will not prevent the port from being discovered but attackers just looking for low level common ports will not detect it. Communications between client and server are also encrypted in SSL. This discourages others from listening in on communications and using man in the middle attacks between the agent and server. This type of attack is where an attacker intercepts messages between devices and manipulates data packets for malicious intentions.

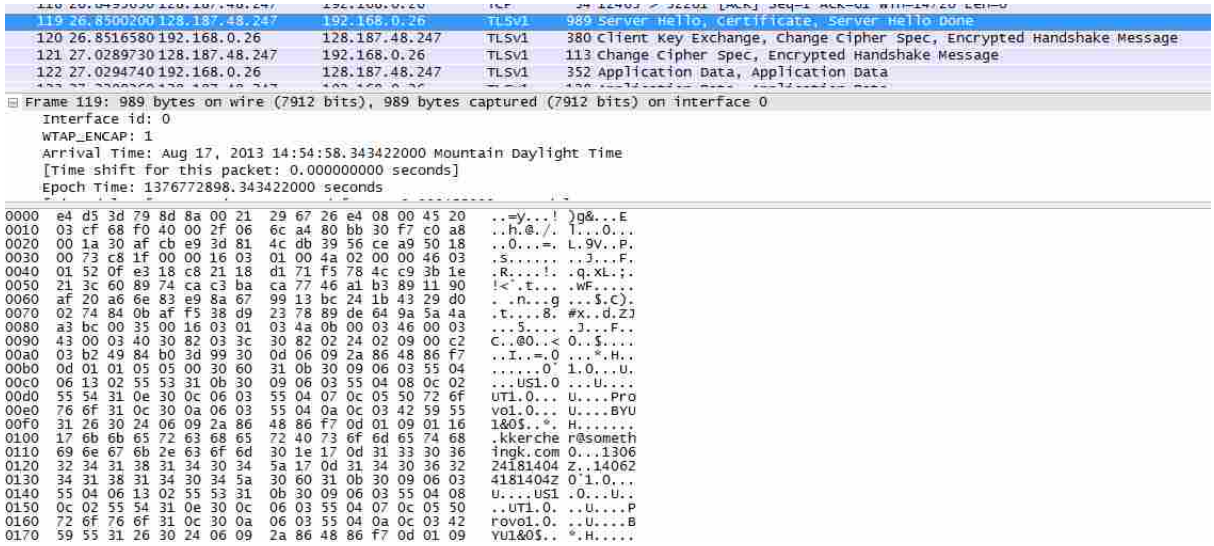

Figure 19: Wireshark Capture of Agent SSL Communications

Within the server, different user accounts are issued ownership over the python listener and web services. This enacts access controls on the services and what files they can and can't access, preventing traversal attacks where hackers attempt to access protected files. The Apache2 server is also set to forbid requests for directory listings. Users will not be able to see the contents or file structure of the web server.

There are no form submissions or user interactions that deal directly with the MySQL database through the web interface. This prevents attackers from performing SQL injections. There is a search box in the website controlled by JavaScript. It searches the current content already displayed on the page. The page does reload and it does not relay the query anywhere on the page for cross-side scripting attacks.

Lastly, there is a concern for the user and the information disclosed by the agent. The website completely enumerates their antivirus version information and operating system platform. This information can be passed to an attacker and used against a client. However,

information that could disclose the identity of the machine and user is not collected, except for the public IP address of the network containing the client. The private IP address of the machine is not collected. Despite the full public address being collected and stored in the server database, the website does not disclose the full IP, only the first half of the address is visible. This is to help ensure records displayed on the web interface cannot be traced back to a client. Administrators alone have access to the full IP addresses in the database. The measures described help maintain user privacy and secure data stored on the server.

In conclusion the following measures were taken to reduce the risk compromise:

- High server listening port
- SSL encryption
- Server access controls
- Limited access to the MySQL database
- Forbidden directory access through the browser
- Cross-side scripting filters
- Limited information disclosure

4.5 **Usage and Project Distribution**

Each agent comes completely packaged with an executable created by the py2exe module. Python is not required to be installed on a client in order to run the agent. Everything needed to run the agent has been packaged and is included in the agent zip folders distributed on the web server. The source code is not intended for public release until it's confirmed whether or not the agents protect the intellectual property of the antivirus vendors, however the code will be available on an internally hosted GIT server dedicated to the agent framework project. This GIT

server is managed by the BYU Cyber Security Research Lab and is only available to participating faculty members and student researchers.

There are two ways the agent can be operated. The client can choose to directly run the executable with administrator privileges, this method will open up a command prompt window. The second method is for those who want the agent to run silently in the background, the agent can be set as a service. Instructions on how to create a service are included in each agent's package. Once the agent is running, the user can forget about it. On its own it will collect data on antivirus updates and malware alerts. This data will then be relayed to the server without disrupting the user's experience.

The user can choose to turn off real time protection in order to help contribute more hashes to the study but this is not required and not recommended for the everyday user. Users who turn this feature off will have to remember to run manual scans on a regular basis. The Windows Defender agent package comes additionally with a registry script. This script enables the right-click scan menu item which may be found useful for users who have chosen to turn off real time protection. By default this option is not included in Windows 8.

Figure 20: Windows Defender Right-Click Scan Menu Item

The agent framework prototype is freely available to anyone who wants to download and use the agents. Each new client will contribute an increase in information received by the server and will help build upon the dataset. This material can then be used to compare vendor performance and better answer the question of if there is a need to have multiple antivirus installations on a client.

5 FRAMEWORK ANALYSIS

This section analyzes the final framework prototype and answers the proposed hypothesis questions:

- (R1) What are the challenges and techniques utilized in creating an agent malware reporting framework architecture?
- (R2) What are the key characteristics that are suitable for universally identifying malware strands?
- (R3) Is there a correlation between antivirus malware naming conventions?
- (R4) What quantifiable benefits may be achieved by using multiple vendor products to detect malware?

5.1 (R1) Framework Design Challenges and Techniques

As predicted, it is possible to develop a cloud sourced malware reporting framework that uses distributed agents to assess the performance of antivirus software based on malware signatures. This was evident by the successful development of a proof of concept prototype that analyzed the update and malware alert events of an antivirus. The key components of the prototype are the individual client side agents and the listening server. This framework would not be able to meet its intended purpose without these two features. The prototype proof of concept

was completely custom developed. The reason for this effort was to ensure control over the prototype behavior and guarantee expected performance.

The agents individually detect and respond to antivirus events occurring on an endpoint device and are partially customized for a client's particular vendor. At the time of development, a universal agent could not be created because each antivirus analyzed logged events uniquely. An agent consists of three components, the listener, mapper and logger. Each agent contains the same logger function however the other two components vary. For future work, a base universal agent may be considered with plugin options that will correctly handle setting up the agent listener and mapper functions depending on the antivirus vendor.

With each discovered malware alert or signature update, the agent queries and extracts predefined variables describing the event. The agents are necessary for obtaining a dataset concerning antivirus resources and abilities. Without this data, the framework would not provide any insight into vendor performance. Information retrieved by the clients is delivered to a single location. The server acts as a centralized data collection destination for all agents. It stores and presents records in a formatted table. Accurate conclusions can only be determined when a complete referencing dataset is presented in a readable presentation. The server is a necessary component because it provides a graphical interface for reading the data reported by the agents and completes the framework design.

5.2 **(R2) Universal Malware Identifiers**

It was found that the malware hash is the only suitable universal identifier for individual malware strands. This became apparent when analyzing antivirus naming conventions. Malware names are localized to a specific vendor's definition. For this reason, names cannot be used to

identify the same piece of malware across vendors. In order to compare antivirus vendors and their resources to combat a specific threat, the malware in question's hash is necessary. Since a name is not universal it cannot be compared across vendors. A hash needs to be used in order to correctly query and compare different antivirus definitions and resources.

There is an issue with the same piece of malware having different variants. These deviations would not have the same hash. To further improve upon the agent development, fuzzy hashing could be implemented. A fuzzy hash is a hash taken of a file that can be compared against others hashes in order to determine a percentage match. Instead of a one to one ratio between hash comparisons, one fuzzy hash could match multiple malware strands. With received data from the agents, the server can use fuzzy hashes to match one hash to all deviations of a single malware threat. This would drastically change how malware is identified and coordinated with the different vendor naming conventions.

5.3 **(R3) Vendor Naming Conventions**

From the tested malware samples, there are no visible patterns or consistent similarities in the naming conventions between all of the antivirus software. Table 9 shows a comparison between vendor names gathered by the agents in the prototype.

Table 9: Naming Convention Comparison

Identifying Malware Hash	Malware Names by Vendor			
	Windows Defender	**Symantec**	**Sophos**	**Avira**
05344813787920a04b207416ea05516b21958b3f6c8ad9fb8f0ce50741efd01	Trojan:Win32/Alureon.FT	Backdoor.Tidserv	Troj/Alureon-AD	TR/Graftor.2081254
0638324B80AAA7D185F353FD4D5436D70845D648E62791E60CDC1626359C05CC	Exploit:Win32/Pdfjsc.AAX	Trojan.Gen.2	Troj/PDFEx-GD	EXP/Pidief.cvh
0DCB7A582A0E72DCCCF4FD855A159A4206B67B85FDCD0F58B71D85BA28E40440	PWS:Win32/Sinowal.gen!Y	Trojan.Malcol	Mal/Sinowal-N	TR/Kazy.3545812
1C464848DF9A803F01035DACF70888A9D942E42ED44E071443A9742930A23DD4	TrojanDownloader:Win32/Kuluoz.B	Trojan.Gen	Troj/Agent-WGO	TR/Rogue.kdv.637381
3407BF876E208F2DCE3B43CCF5361C5E009ED3DAF87571BA5107D10A05DC7BC4	Trojan:Win32/R2d2.A!rootkit	Backdoor.R2D2	Troj/BckR2D2-A	TR/GruenFink.2

Occasionally, some names may be similar across a few vendors, but very rarely are the names universally consistent. Because of this behavior, there are no noticeably detectable naming convention patterns between antivirus products. Typically the first responder to a piece of malware gets the rights to name the malware. The name often comes from a string found

within the malware or the author's name. However, the longer a piece of malware is live the more likely it has developed numerous strands. In some circumstances a vendor identifies a strand as a new piece of malware and utilizes its own naming convention to identify the threat as evident from the data collected in the table. This research provides real-time correlation of naming conventions based on the sample's hash allowing easy identification of a strand even if a vendor has applied a different name.

5.4 (R4) Benefits of Multiple Vendor Installations

The graph below illustrates the percentage of malware threats detected by the antivirus software out of the complete sample set. The entire malware sample set is listed in Appendix F. The data was collected by the client side agents and delivered to the server for centralized analytics.

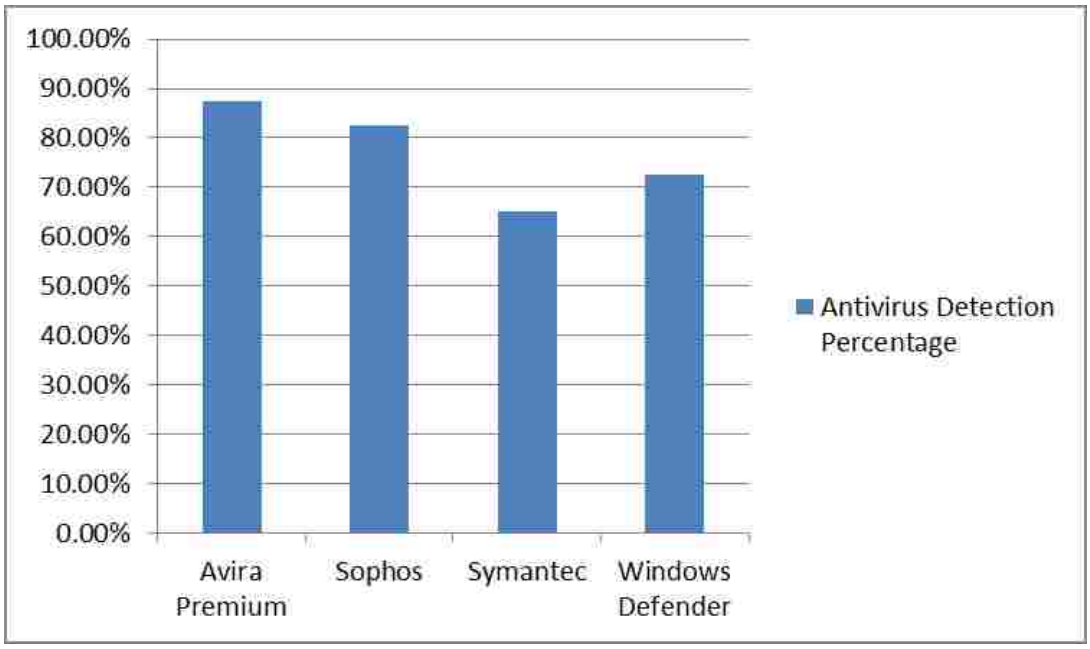

Figure 21: Percentage of Antivirus Detections out of 40 Samples of Malware

From the data collected by the agents, it is apparent that not every piece of malware from the sampling set was detected by each vendor. On average, the antivirus software protected against 76.875% of the malware threats in the test environment. With this in mind, by increasing the number of installations on a device, resources are combined and there is a greater chance of detecting malware. This is seen in Figure 22.

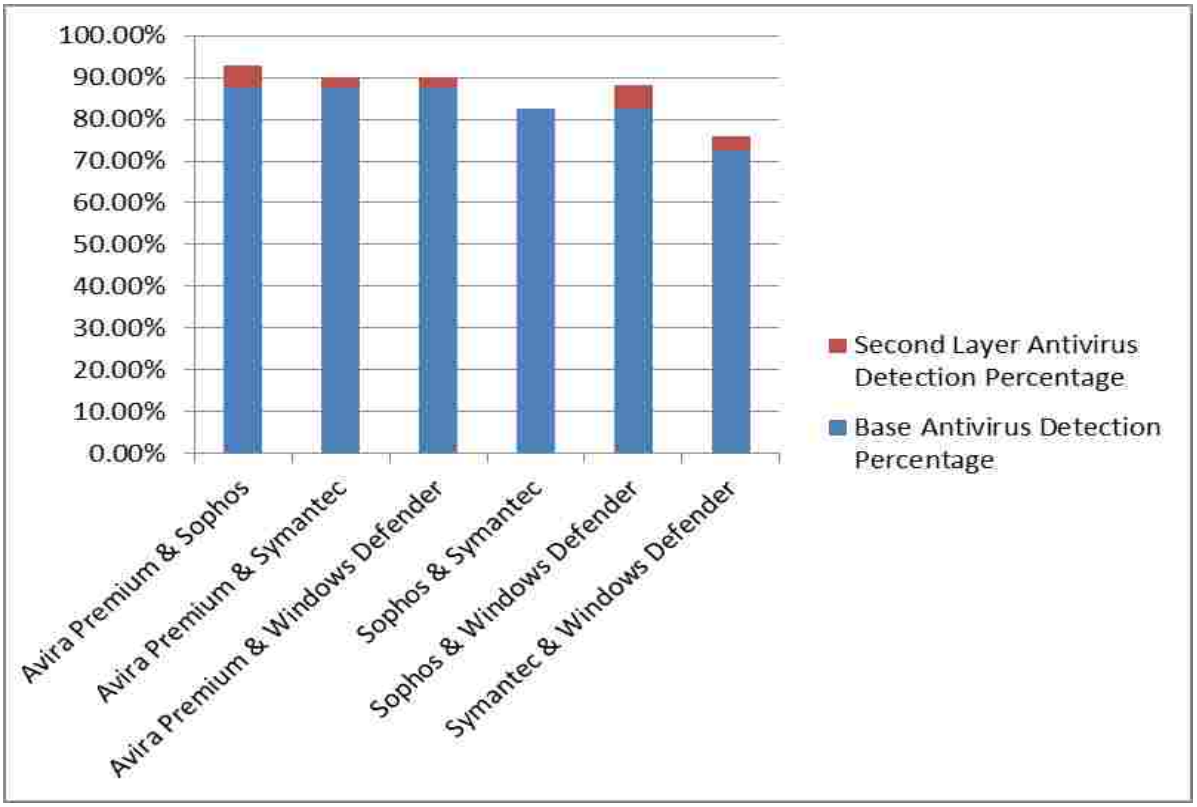

Figure 22: Percentage of Two Antivirus Installation Detections out of 40 Samples of Malware

On average, 86.667% of the threats were detected by two installments of antivirus software. With each client averaging an increase of 4.75% in malware samples detected. The graph shows the antivirus with the greatest detection rate as the base antivirus. The second layer

shows the additional support provided by a second provider. Even though it may not appear to be an exceptional increase in protection from the base provider, it is evident that not all vendors have succeeded in detecting every malware sample. It is not always known which provider is the most proficient. However, with another installation, a device has a greater chance of being protected from more threats. With three installations, even greater protection percentages are achieved.

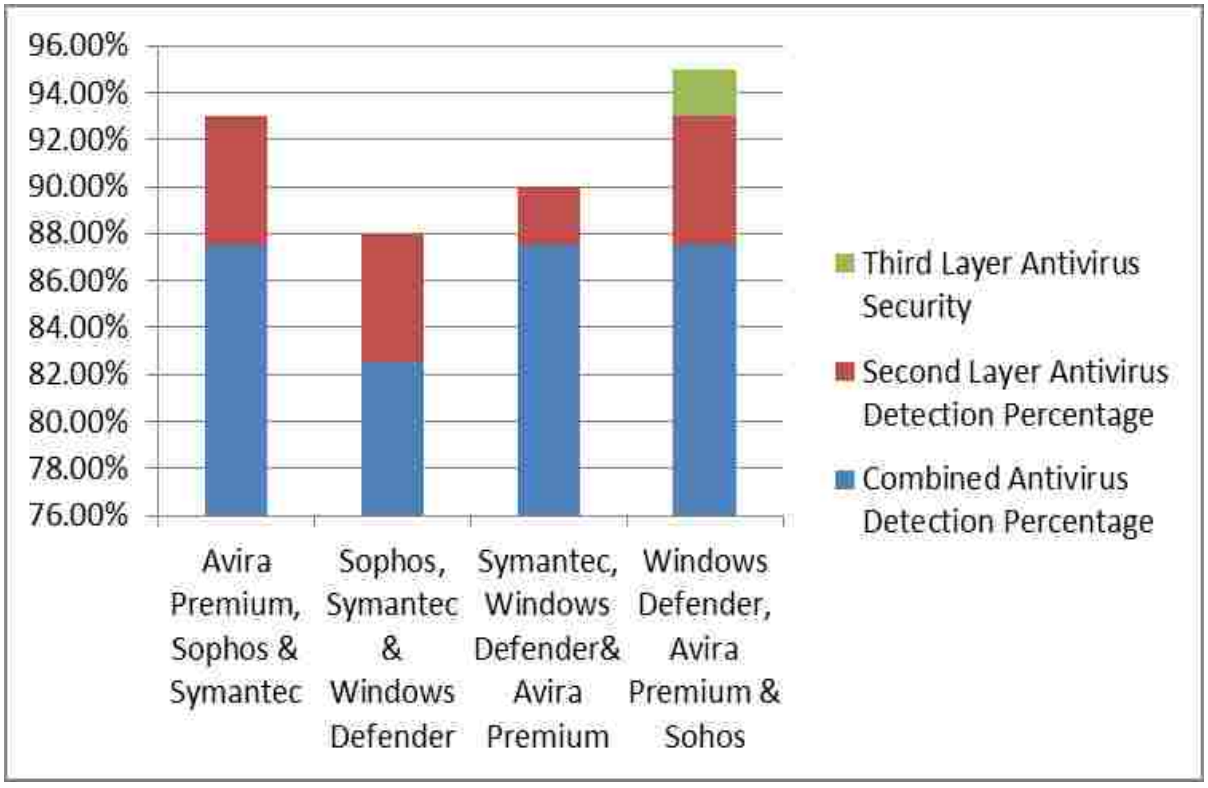

Figure 23: Percentage of Three Antivirus Installation Detections out of 40 Samples of Malware

Three installments on average provided 91.5% protection against the malware samples tested. Figure 24 illustrates the maximum benefit of multiple installations of antivirus software found in the study.

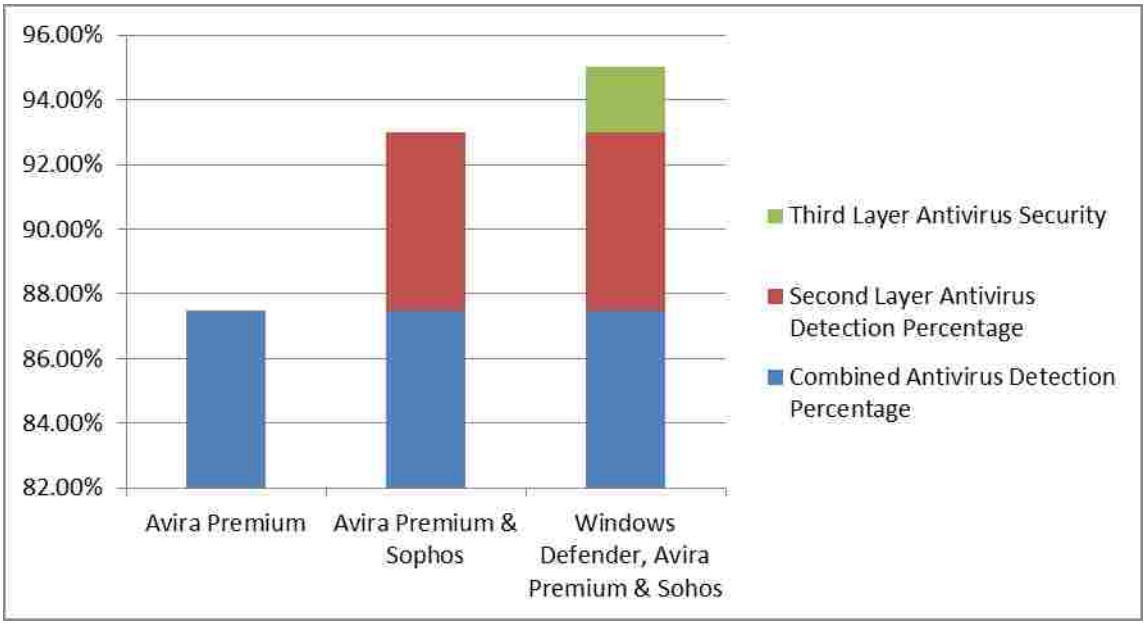

Figure 24: Comparison between One and Three Antivirus Installations

From the data, it is recommended that users considered at least two different vendor installations of antivirus software for their client machine in order to increase their system's security. Future work should focuses on increasing the malware sample size to further support this claim.

6 CONCLUSION AND FUTURE WORK

The purpose of this research was to assess the viability of a cloud sourced malware reporting framework that utilizes distributed agents to evaluate the performance of antivirus software based on malware signatures. From the research, it was found that an agent based malware reporting framework may be used to collect details on the performance of antivirus software. A prototype proof of concept was built to demonstrate the feasibility of such a framework. It consisted of two key components, the centralized collection server and the individual client side agents. Detected malware alerts and antivirus updates were reported from the agents to the server.

6.1 Future Research

The following section discusses opportunities for future research and project improvements comprising:

- Use of fuzzy hashes
- Agent uptime and accuracy
- Improving the antivirus agent support
- Universal Agent
- Increasing the number of agent clients and malware samples

6.1.1 Fuzzy Hashes

From the data collected by the agents, it was discovered that the only way to identify malware universally across antivirus software was by a hash. The naming conventions of the malware vendors contain no recognizable universal patterns. For this reason, the prototype provides a reference for the monitored antivirus software and their naming conventions for a specific hash. However, it should be noted that SHA256 may be ineffective against identifying malware strands within a family and should be addressed in future research. A fuzzy hash can be implemented to further identify malware strands and similar deviations across multiple vendors. This will eliminate the need for each deviation of a specific malware file hash to be recorded in the data. It will instead require one hash that can identify the majority of deviations.

6.1.2 Agent Accuracy

One problem with the prototype design is that it relies on the client machines being powered on at all times in order to get accurate readings of antivirus performance. However, many clients are powered off after use. One solution would be to create a permanent environment where a grouping of virtual machines remains operational at all times. Each virtual machine would then include a unique antivirus installation and coordinating agent in order to provide more accurate antivirus update notifications from all supported vendors. Future research should further look into this resolution and others in order to address the situation.

6.1.3 Agent Support

Further research should also look at increasing the antivirus vendor agent support along with improving development code for client side performance. Many vendors were found

incompatible with the prototype because their log information in encrypted. Attempting to decrypt or read this private data is illegal due to intellectual property laws. However, there may be an opportunity to work together in the future with different antivirus companies and with their permission create an agent compatible with these encrypted records. Also it may be beneficial to look into tools such as Procmon to increase the number of antivirus software the agent supports. If API functions are available, an agent may be able to utilize these tools to analyze and gather event data from new processes coordinating with an antivirus.

6.1.4 Universal Agent

After reviewing the prototype development created as a proof of concept for the framework, it is evident that a universal agent construction may be possible. Each agent consists of the same three components. These parts include an event listener, a data mapper and report logger. The agent logger remains the same across vendor installations while the other two functions differ depending on how an antivirus logs events. A universal agent can consist of the logger component along with plugin options that setup the other two components for the client's antivirus software improving upon the current design. This new implementation will lead to simplified agent developments. Only additional plugins will need to be programmed for new vendors while the agent remains the same.

6.1.5 Expanding the Study

Continuing, the prototype can also be improved upon by increasing the number of endpoint devices with installed agents. The BYU Information Technology's security lab will implement the agent on each desktop in order to increase the number of distributed agents and

detected malware events. Also, BYU's Office of Information has been contacted about the development will discuss installing agents in some of the campus labs as soon as the operating systems are updated to Windows 8. This information will contribute to the current prototype's dataset and further identify antivirus trends and performance. With an increase in clients, more malware samples are expected to be reported. This data will also contribute to the research and further identify trends in multiple antivirus installations.

6.2 **Project Contribution**

Malware is a malicious, rapidly growing threat targeting endpoint devices. Numerous vendors supply antivirus software that can help protect a user against these threats. Each vendor has their own set of virus definitions varying in resources and capabilities in recognizing new strands of malware. Users can benefit from a system that can evaluate antivirus performance in order to be better informed about their security options, in addition to becoming aware of prevalent threats occurring in their network.

The framework introduced in this research utilizes a cloud sourced malware reporting system to benefit users and provide real time information in order to educate and assist in security decisions. It localizes threats by geo-location along with informing clients on how active vendors are in updating their definitions with new signature files. This reporting system benefits the user by exposing current malware activities and the abilities antivirus technologies have to combat these threats. In summary, the research contribution includes in real time:

- A system centric view of malware detection
- Correlation of malware identifiers

- Antivirus performance evaluations
- Support for a new security model with multiple vendor installations

Concluding, malware threats are increasing at a pace that vendors cannot match. A single vendor does not have the resources to combat every attack. Research has shown that there is a need to change the security model for endpoint devices (Lee 2013). The data discovered by the agent prototype further confirmed that antivirus software does not protect against all threats and that there is in fact a need for change. From the project findings it is concluded that by combining antivirus resources with multiple vendor installations, a client will increase their device's security. This new model will improve host defenses against malware.

REFERENCES

"2013 Trustwave Global Security Report." Accessed April 13, 2013. https://www2.trustwave.com/2013GSR-TY.html?aliId=1417176.

"About VirusTotal." VirusTotal. https://www.virustotal.com/en/about/ (accessed September 9, 2013).

"Antivirus Market Analysis: December 2012 | OPSWAT | Software management and security solutions." OPSWAT. http://www.opswat.com/about/media/reports/antivirus-december-2012 (accessed August 15, 2013).

Aycock, J. *Computer Viruses and Malware*. Springer, 2006.

"AV-TEST - The Independent IT-Security Institute: Test Procedures." http://www.av-test.org/en/test-procedures/ (accessed April 13, 2013).

Cambridge, R. D. "Method and System for Bi-directional Updating of Antivirus Database," July 18, 2006. http://www.google.com/patents?id=OaB6AAAAEBAJ (accessed February 8, 2013).

"Comparatives||tests - Reviews - Reports." Accessed April 13, 2013. http://av-comparatives.org/comparativesreviews (accessed April 13, 2013).

"Download Python." Python Programming Language – Official Website. http://www.python.org/getit/ (accessed August 21, 2013).

Gashi, I., V. Stankovic, C. Leita, and O. Thonnard. "An Experimental Study of Diversity with Off-the-Shelf AntiVirus Engines." In *Eighth IEEE International Symposium on Network Computing and Applications, 2009. NCA 2009*, 4 –11, 2009.

Hodges, V. and S O'Donnell. "Method and System for Providing Automated Updating and Upgrading of ...," March 7, 2000. http://www.google.com/patents?id=TGEDAAAAEBAJ (accessed February 8, 2013).

"Internet Security Threat Report." http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_2011_21239364.en-us.pdf (accessed April 13, 2013).

Jennings, N. R., and M. J. Wooldridge. *Agent Technology: Foundations, Applications, and Markets*. Springer, 1998.

Jennings, N. R., K. Sycara, and M. Wooldridge. "A Roadmap of Agent Research and Development." *Autonomous Agents and Multi-Agent Systems* 1, no. 1 (January 1998): 7–38. doi:10.1023/A:1010090405266.

Lee, R. "Is Anti-Virus Really Dead? A Real-World Simulation Created for Forensic Data Yields Surprising Results" Computer Forensics and Incident Response. *Blog*, April 9, 2012. http://computer-forensics.sans.org/blog/2012/04/09/is-anti-virus-really-dead-a-real-world-simulation-created-for-forensic-data-yields-surprising-results (accessed February 8, 2013).

Maggi, F., A. Bellini, G. Salvaneschi, and S. Zanero. "Finding Non-trivial Malware Naming Inconsistencies." In *Information Systems Security*, 144–159. Edited by Sushil Jajodia and Chandan Mazumdar. Lecture Notes in Computer Science 7093. Springer Berlin Heidelberg, 2011. http://link.springer.com/chapter/10.1007/978-3-642-25560-1_10.

Nwana, H. S., and D. T. Ndumu. "A Brief Introduction to Software Agent Technology." In *Agent Technology*, edited by Nicholas R. Jennings and Michael J. Wooldridge, 29–47. Springer Berlin Heidelberg, 1998. http://link.springer.com/chapter/10.1007/978-3-662-03678-5_2.

Posey, B. "Microsoft Exchange Server Security Dos and Don'ts" TechTarget. *SearchExchange*. http://searchexchange.techtarget.com/feature/Microsoft-Exchange-Server-security-dos-and-donts (accessed March 18, 2013).

"Python v2.7.5 documentation." The Python Standard Library. http://docs.python.org/2/library/ (accessed August 16, 2013).

Sanok, Jr, D. J. "An Analysis of How Antivirus Methodologies Are Utilized in Protecting Computers from Malicious Code." In *Proceedings of the 2nd Annual Conference on Information Security Curriculum Development*, 142–144. InfoSecCD '05. New York, NY, USA: ACM, 2005.

Sycara, K., A. Pannu, M. Willamson, D. Zeng, and K. Decker. "Distributed Intelligent Agents." *IEEE Expert* 11, no. 6 (December 1996): 36 –46. http://ieeexplore.ieee.org/ielx3/64/11937/00546581.pdf?tp=&arnumber=546581&isnumber=11937 (accessed January 19, 2013).

Sukwong, O., H. S. Kim, and J. C. Hoe. “Despite the Widespread Use of Antivirus Software, Malware Remains Pervasive. A New Study Compares the Effectiveness of Six Commercial AV Products.” http://theone.ece.cmu.edu/papers/94.commercial.2011.compmag.pdf (accessed April 13, 2013).

Tian, R.. *An Integrated Malware Detection and Classification System*. Deakin University. (2011). Accessed April 24, 2013. http://dro.deakin.edu.au/view/DU:30043244.

“Why one virus engine is not enough.” http://www.gfi.com/whitepapers/why-one-virus-engine-is-not-enough.pdf (accessed April 13, 2013).

APPENDICES

APPENDIX A. AGENT – AVIRA PREMIUM ANTIVIRUS

A.1 Agent-Avira.py

```
"""Avira Distributed Agent Cloud-Sourced Malware Reporting
Framework v1.0 - Copyright 2013
   Kellie Kercher - agent@somethingk.com
   http://www.somethingk.com

   The code is available to anyone interesting in progressing
the research in agent based malware analysis. Please contact me
for
   suggestions, questions or improvements. If you are utilizing
this code please give credit to the project.
"""

import win32evtlog, win32event, win32api, win32con
from win32api import GetFileVersionInfo, LOWORD, HIWORD
import win32evtlogutil
import socket, ssl, pprint, platform, re, os, hashlib, codecs,
time, sys, thread
import datetime as dt
import json
from urllib2 import urlopen

#http://nullege.com/codes/show/src@w@i@WinSys-3.x-
0.5.2@winsys@event_logs.py

def getIP():
     ip = json.load(urlopen('http://httpbin.org/ip'))['origin']
     return ip

             def findFile(path, type): #Find the coordinating log
     now=dt.datetime.now()
     ago=now-dt.timedelta(minutes=.1)
     mtime = lambda f: os.stat(os.path.join(path, f)).st_mtime
     latest = list(reversed(sorted(os.listdir(path),
```

```
key=mtime))) #Look at the latest log files
     for name in latest: #Loop through to find the latest update
or alert log
          st=os.stat(path+"\\"+name)
          mt=dt.datetime.fromtimestamp(st.st_mtime)
          if mt>ago:#Ensure the log was modified at the time of
the trigger
               if name.find(type) > -1: #begining of alert
filename
                    return (path+"\\"+name)
     return

def getAlert(event, status):
     global registryPath
     global registryKey
     global build
     global logs
     global last
     try:
          if event.SourceName == "Avira Antivirus":
               eventID = event.EventID
               if eventID == -2147479535:
                    msg =
str(win32evtlogutil.SafeFormatMessage(event, logtype))
                    hash = "Malware file unavailable or deleted
before agent could hash."
                    hashFile = re.search('in the
file\n?(.+(\n.+)?)', msg)
                    if hashFile:
                         if
os.path.isfile((hashFile.group(1)).strip()):
                              hash =
(str(hashlib.sha256(file(str((hashFile.group(1)).strip()),
'rb').read()).hexdigest())).upper()
                    sourceName = str(event.SourceName) #Setup
default values for variables
                    buildFile = open(build, 'r')
                    version = 'Not Found'
                    name = 'Not Found'
                    action = 'No Action'
                    timeGen = str(event.TimeGenerated)
                    newdate = dt.datetime.strptime(timeGen,
'%m/%d/%y %H:%M:%S')
                    timeGen = newdate.strftime('%Y-%m-%d
%H:%M:%S')
                    hKey = win32api.RegOpenKey
```

```
(win32con.HKEY_LOCAL_MACHINE, str(registryPath)) #read the
registry for definition version
                    value, type = win32api.RegQueryValueEx
(hKey, registryKey)
                    sig_version = value
                    if buildFile: #read build file for product
version information
                         file_read = buildFile.read()
                         reg = re.search("ProductVersion=(.+)",
file_read)
                         if reg:
                              version = (reg.group(1)).strip()
                         buildFile.close()
                    name = re.search("AntiVir has detected
'(.+)'", msg).group(1)
                    duplicate =
str(name)+"*"+hash+"*"+str(sourceName)+"*"+str(version)+"*"+str(
sig_version)+"*"+str(platform.platform())+"*"+str(action)+"*"+st
r(getIP())
                    if last != duplicate:
                         last = duplicate
                         report =
"Alerts*"+str(timeGen)+"*"+str(name)+"*"+hash+"*"+str(sourceName
)+"*"+str(version)+"*"+str(sig_version)+"*"+str(platform.platfor
m())+"*Moved to Quarantine*"+str(getIP())
                         return sendEvent(report)
          return
     except Exception:
          pass
     return

def getEvent(update, log, block=False): #read the latest event
data and report to the server different results determine by the
antivirus event type
     global registryPath
     global registryKey
     global build
     global blocked
     global signatureDirectory
     try:
          timeGen = dt.datetime.now().strftime('%Y-%m-%d
%H:%M:%S')
          sourceName = "Avira Antivirus" #Setup default values
for variables
          buildFile = open(build, 'r')
          version = 'Not Found'
```

```
          name = 'Not Found'
          action = 'No Action'
          if buildFile: #read build file for product version
information
               file_read = buildFile.read()
               reg = re.search("ProductVersion=(.+)", file_read)
               if reg:
                    version = (reg.group(1)).strip()
               buildFile.close()
          hKey = win32api.RegOpenKey
(win32con.HKEY_LOCAL_MACHINE, str(registryPath)) #read the
registry for definition version
          value, type = win32api.RegQueryValueEx (hKey,
registryKey)
          sig_version = value
          if update: #if event is an update
               if log:
                    print log
                    f = ""
                    h = ""
                    while True: #loop through flagged files and
get the hash before file is deleted
                         mt=os.path.getmtime(log.strip())
                         now=time.time()
                         if (now-mt)>=300:
                              f = codecs.open(log.strip(), 'r',
encoding='utf16')
                              break
                         else:
                              time.sleep(300)
                    print "5 up"
                    if f:
                         lines = f.read()
                         f.close()
                         hashSearch = re.findall("was copied to
'(.+)'\.", str(lines)) #Find signature files for hash
                         for hFind in hashSearch:
                              if hFind.endswith('.vdf'):
                                   hash_file =
open(str(hFind.strip()), 'rb').read()
                                   check =
hashlib.sha256(hash_file).hexdigest()
                                   name = re.search("vbase\d+",
hFind)
                                   h =
h+str(name.group(0))+".vdf: "+str(check)+"\n"
```

```
                    if h:
                         report =
"Updates*"+str(timeGen)+"*"+str(sig_version)+"*"+str(h)+"*"+sour
ceName+"*"+version+"*"+str(platform.platform())+"*"+str(getIP())
                         return sendEvent(report)
          elif block: #if blocked access file
               f = codecs.open(blocked, "r", encoding="utf16")
               lines = f.readlines()
               for i in range(0, len(lines)):
                    line = lines[i]
                    n = re.search("Contains code of the (.+)",
line) #read through log and pull out data
                    if n:
                         name = n.group(1)
                         i = i+1
                         line = lines[i]
                         a = re.search("Executed action: (.+)",
line)
                         if a:
                              action = a.group(1)
               f.close()
               report =
"Alerts*"+str(timeGen)+"*"+name.strip()+"*No
File*"+sourceName+"*"+str(version)+"*"+str(sig_version)+"*"+str(
platform.platform())+"*"+str(action.strip())+"*"+str(getIP())
               return sendEvent(report)
     except Exception, e:
          print sys.exc_traceback.tb_lineno, str(e)
          pass
     return

def sendEvent(result): #send report results to server
     global callHomeServer
     global callHomePort
     try:
          print result
          s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
          ssl_sock = ssl.wrap_socket(s)
          ssl_sock.connect((callHomeServer, int(callHomePort)))
          ssl_sock.sendall(result)
          data = ssl_sock.read()
          print data
          del ssl_sock
          s.close()
     except Exception:
          print "Event not delivered to server"
```

```
          pass

#Read in Config Variables
try:
     config = open(os.path.join(os.path.dirname(sys.argv[0]),
'config.txt'), 'r').read()
     callHomeServer = re.search("callHomeServer = '(.+)'",
config).group(1)
     callHomePort = re.search("callHomePort = '(.+)'",
config).group(1)
     update = re.search("update = '(.+)'", config).group(1)
     logs = re.search("logs = '(.+)'", config).group(1)
     blocked = re.search("blocked = '(.+)'", config).group(1)
     registryPath = re.search("registryPath = '(.+)'",
config).group(1)
     registryKey = re.search("registryKey = '(.+)'",
config).group(1)
     build = re.search("build = '(.+)'", config).group(1)
except Exception:
    print 'Configuaration file (config.txt) is unavailable or
formatted incorrectly, unable to start agent.'
    sys.exit()

#Ensure Trigger files exist
while True:
     if os.path.isdir(logs) and os.path.isfile(blocked) and
os.path.isdir(update): #make sure directories exisits, if not
wait till it does
          break

#Setup environment
watcher1 = os.stat(blocked)
watcher2 = os.stat(update)
this_modified1 = last_modified1 = watcher1.st_mtime
this_modified2 = last_modified2 = watcher2.st_mtime
last = ""
lastlog = ""

server = 'localhost' # name of the target computer to get event
logs
logtype = 'Application'
handler = win32evtlog.OpenEventLog(server,logtype)
handlerEvent = win32event.CreateEvent (None, 1, 0, None)
flags =
win32evtlog.EVENTLOG_BACKWARDS_READ|win32evtlog.EVENTLOG_SEQUENT
IAL_READ
```

```
print "Finding SYSTEM Events"
win32evtlog.NotifyChangeEventLog(handler, handlerEvent)

while True:
     try:
          if this_modified2 > last_modified2: #Watch for changes
on the update file modified time, if there is a change in
modified time, trigger for updates
               report = findFile(logs, 'Upd-')
               if report:
                    if report != lastlog:
                         lastlog = report
                         event =
thread.start_new_thread(getEvent, (True, report))
               last_modified2 = (os.stat(update)).st_mtime
          elif this_modified1 > last_modified1: #Watch for
changes on the webguard file modified time, if there is a change
in modified time, trigger for blocked access files
               last_modified1 = this_modified1
               event = getEvent(False, None, block=True)
          elif win32event.WaitForSingleObject (handlerEvent,
500) != win32event.WAIT_TIMEOUT: #Watch application event log
for updates, if new event appears, trigger
               events =
win32evtlog.ReadEventLog(win32evtlog.OpenEventLog(server,logtype
), flags,0)
               for event in events:
                    alert = thread.start_new_thread(getAlert,
(event, "go"))
          watcher1 = os.stat(blocked)
          watcher2 = os.stat(update)
          this_modified1 = watcher1.st_mtime
          this_modified2 = watcher2.st_mtime
     except Exception:
          pass
```

A.2 config.txt

```
/* EDIT THE BELOW IP TO MATCH THAT OF THE TRACKING SERVER AND
ENSURE THE ANTIVIRUS DIRECTORY PATHS ARE CORRECT */
/* v1.0 */

/* If any of these variable are changed, please restart the
service. */
```

```
callHomeServer = 'itsecurity.et.byu.edu'
callHomePort = '12463'
update = 'C:\ProgramData\Avira\AntiVir Desktop\BACKUP'
logs = 'C:\ProgramData\Avira\AntiVir Desktop\LOGFILES'
blocked = 'C:\ProgramData\Avira\AntiVir
Desktop\LOGFILES\webguard.log'
registryPath = 'SOFTWARE\Wow6432Node\Avira\AntiVir Desktop'
registryKey = 'VdfVersion'
build = 'C:\Program Files (x86)\Avira\AntiVir Desktop\\build.dat'
```

APPENDIX B. AGENT – SOPHOS ANTIVIRUS

B.1 Agent-Sophos.py

```
"""Sophos Distributed Agent Cloud-Sourced Malware Reporting
Framework v1.0 - Copyright 2013
   Kellie Kercher - agent@somethingk.com
   http://www.somethingk.com

   The code is available to anyone interesting in progressing
the reseach in agent based malware analysis. Please contact me
for
   suggestions, questions or improvements. If you are utilizing
this code please give credit to the project.

     In order for the agent to work properly, windows event
logging must be enabled.
     For complete malware hashes, on-access scanning needs to be
turned off.
     This setting for on access scanning is not recommended for
the typical user.
"""

import win32evtlog, win32event, win32api, win32con
import win32evtlogutil
from win32api import GetFileVersionInfo, LOWORD, HIWORD
import socket, ssl, pprint, platform, re, os, hashlib, codecs,
time, sys, thread
import datetime as dt
import json
from urllib2 import urlopen

#http://nullege.com/codes/show/src@w@i@WinSys-3.x-
0.5.2@winsys@event_logs.py

def getIP():
     ip = json.load(urlopen('http://httpbin.org/ip'))['origin']
```

```
     return ip

def hash(path): #hash the signature files found, these will be
the files changed or added by the update
     global updateInterval
     now=dt.datetime.now()
     ago=now-dt.timedelta(minutes=int(updateInterval))
     hash=""
     for root, dirs, files in os.walk(path):
          for name in files:
               p = os.path.join(root,name)
               st=os.stat(p)
               mtime=dt.datetime.fromtimestamp(st.st_mtime)
               if mtime>ago:
                    if name.endswith(".ide"):
                         h = hashlib.sha256(file(str(p),
'rb').read()).hexdigest()
                         hash = hash+name+":
"+(str(h)).upper()+"\n"
     return hash

def getAlert(event, status): #parse the latest event for malware
alerts
     global sig_version
     global antivirusExe
     global last
     try:
          if event.SourceName == "Sophos Anti-Virus": #Verify it
is a sophos event
               timeGen = str(event.TimeGenerated)
               newdate = dt.datetime.strptime(timeGen, '%m/%d/%y
%H:%M:%S')
               timeGen = newdate.strftime('%Y-%m-%d %H:%M:%S')
               sourceName = str(event.SourceName) #Pull data
from the event using the win32evtlog library
               eventID = event.EventID
               msg =
str(win32evtlogutil.SafeFormatMessage(event, logtype))
               get_version = GetFileVersionInfo(antivirusExe,
"\\")
               version =
str(HIWORD(get_version['FileVersionMS']))+"."+str(LOWORD(get_ver
sion['FileVersionMS']))+"."+str(HIWORD(get_version['FileVersionL
S']))+"."+str(LOWORD(get_version['FileVersionLS']))
               if eventID == 11 or eventID == 542638091:
#Specific Event ID for blocked alert
```

```
                    name = re.search("Virus/spyware '(.+)'",
msg).group(1)
                    action = re.search('detected at "(.+)"',
msg).group(1)
                    report =
"Alerts*"+timeGen+"*"+str(name)+"*No
File*"+sourceName+"*"+str(version)+"*"+str(sig_version)+"*"+str(
platform.platform())+"*Access to "+str(action.strip())+" was
blocked.*"+str(getIP())
                    return sendEvent(report)
               elif eventID == 32 or eventID == 539295776:
#Specific Event ID for action alert
                    name = re.search("belongs to virus/spyware
'(.+)'.", msg).group(1)
                    file_num = re.search('File "(.+)"',
msg).group(1)
                    if last != file_num:
                         last = file_num
                         try:
                              h =
(str(hashlib.sha256(file(file_num.strip(),
'rb').read()).hexdigest())).upper()
                         except Exception:
                              h = "Malware file unavailable or
deleted before agent could hash."
                         report =
"Alerts*"+timeGen+"*"+str(name)+"*"+str(h)+"*"+sourceName+"*"+st
r(version)+"*"+str(sig_version)+"*"+str(platform.platform())+"*Q
uaratined*"+str(getIP())
                         return sendEvent(report)
     except Exception:
          pass
     return

def getUpdate(timeGen, status): #get antivirus update
information
     global antivirusExe
     global sig_version
     global signatureDirectory
     global main_log
     global lastUpdate
     time.sleep(10) #wait for update vaurables to be changed
accordingly
     try:
          sourceName = "Sophos Anti-Virus"
          get_version = GetFileVersionInfo(antivirusExe, "\\")
```

```
          version =
str(HIWORD(get_version['FileVersionMS']))+"."+str(LOWORD(get_ver
sion['FileVersionMS']))+"."+str(HIWORD(get_version['FileVersionL
S']))+"."+str(LOWORD(get_version['FileVersionLS']))
          f = codecs.open(main_log, "r", encoding="utf16")
          sig_version = "Not Found"
          line = f.read()
          v1 = re.findall("(\d+) (\d+)  Using detection data
version (.+) \(", line) #check for update regular expression
          if v1: #if there is an update
               ver1 = v1[-1][2]
               v2 = re.findall("This version can detect (.+)
items.", line) #get version information
               if v2:
                    ver2 = v2[-1]
                    sig_version = str(ver1) + " (Total viruses
with IDEs " + str(ver2) + ")"
                    h = hash(signatureDirectory) #Get hash for
latest updated definition files
                    if h:
                         temp =
str(sig_version)+"*"+str(h)+"*"+sourceName+"*"+version
                         if temp != lastUpdate:
                              lastUpdate = temp
                              report =
"Updates*"+str(timeGen)+"*"+str(sig_version)+"*"+str(h)+"*"+sour
ceName+"*"+version+"*"+str(platform.platform())+"*"+str(getIP())
                              return sendEvent(report)
          return
     except Exception:
          pass
     return

def sendEvent(result): #send report results to server
     global callHomeServer
     global callHomePort
     try:
          print result
          s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
          ssl_sock = ssl.wrap_socket(s)
          ssl_sock.connect((callHomeServer, int(callHomePort)))
          ssl_sock.write(result)
          data = ssl_sock.read()
          print data
          del ssl_sock
          s.close()
```

```
     except Exception:
          print "Event not delivered to server"
          pass

#Read in Config Variables
try:
     config = open(os.path.join(os.path.dirname(sys.argv[0]),
'config.txt'), 'r').read()
     callHomeServer = re.search("callHomeServer = '(.+)'",
config).group(1)
     callHomePort = re.search("callHomePort = '(.+)'",
config).group(1)
     main_log = re.search("mainLog = '(.+)'", config).group(1)
     update_log = re.search("updateLog = '(.+)'",
config).group(1)
     signatureDirectory = re.search("signatureDirectory =
'(.+)'", config).group(1)
     antivirusExe = re.search("antivirusExe = '(.+)'",
config).group(1)
     updateInterval = re.search("updateInterval = '(.+)'",
config).group(1)
except Exception:
    print 'Configuaration file (config.txt) is unavailable or
formatted incorrectly, unable to start agent.'
    sys.exit()

#Setup environment
while True:
     if os.path.isfile(update_log) and os.path.isfile(main_log):
#make sure log exisits, if not wait till it does
          break

f = codecs.open(main_log, "r", encoding="utf16")
sig_version = "Not Found"
line = f.read()
v1 = re.findall("(\d+) (\d+)  Using detection data version (.+)
\(", line) #check for update regular expression
if v1: #if there is an update
     ver1 = v1[-1][2]
     v2 = re.findall("This version can detect (.+) items.",
line) #get version information
     if v2:
          ver2 = v2[-1]
          sig_version = str(ver1) + " (Total viruses with IDEs "
+ str(ver2) + ")"
f.close()
```

```
watcher = os.stat(update_log)
this_modified = last_modified = watcher.st_mtime
server = 'localhost' # name of the target computer to get event
logs
logtype = 'Application'
handler = win32evtlog.OpenEventLog(server,logtype)
handlerEvent = win32event.CreateEvent (None, 1, 0, None)
flags =
win32evtlog.EVENTLOG_BACKWARDS_READ|win32evtlog.EVENTLOG_SEQUENT
IAL_READ

print "Finding SYSTEM Events"
win32evtlog.NotifyChangeEventLog(handler, handlerEvent)
last = ""
lastUpdate = ""

while True:
     try:
          if this_modified > last_modified: #Watch for changes
on the log modified time, if there is an update change in
modified time, trigger
               last_modified = os.stat(update_log).st_mtime
               timeGen = dt.datetime.now().strftime('%Y-%m-%d
%H:%M:%S')
               result = thread.start_new_thread(getUpdate,
(timeGen, "go"))
          if win32event.WaitForSingleObject (handlerEvent, 500)
!= win32event.WAIT_TIMEOUT: #Watch application event log for
alerts, if new event appears, trigger
               events =
win32evtlog.ReadEventLog(win32evtlog.OpenEventLog(server,logtype
), flags,0)
               for event in events:
                    result = thread.start_new_thread(getAlert,
(event, "go"))
          watcher = os.stat(update_log)
          this_modified = watcher.st_mtime
     except Exception:
          pass
```

B.2 config.txt

```
/* EDIT THE BELOW IP TO MATCH THAT OF THE TRACKING SERVER AND
ENSURE THE ANTIVIRUS DIRECTORY PATHS ARE CORRECT */
```

```
/* v1.0 */

callHomeServer = 'itsecurity.et.byu.edu'
callHomePort = '12463'
mainLog = 'C:\ProgramData\Sophos\Sophos Anti-Virus\logs\SAV.txt'
updateLog = 'C:\ProgramData\Sophos\AutoUpdate\Logs\alc.log'
signatureDirectory = 'C:\Program Files (x86)\Sophos\Sophos Anti-
Virus'
antivirusExe = 'C:\Program Files (x86)\Sophos\Sophos Anti-
Virus\SavMain.exe'

/* How often you update in minutes */
updateInterval = '1440'
```

APPENDIX C. AGENT – SYMANTEC ANTIVIRUS

C.1 Agent-Symantec.py

```
"""Symantec Distributed Agent Cloud-Sourced Malware Reporting
Framework v1.0 - Copyright 2013
   Kellie Kercher - agent@somethingk.com
   http://www.somethingk.com

   The code is available to anyone interesting in progressing
the reseach in agent based malware analysis. Please contact me
for
   suggestions, questions or improvements. If you are utilizing
this code please give credit to the project.
"""

import win32evtlog, win32event, win32api, win32con
from win32api import GetFileVersionInfo, LOWORD, HIWORD
import win32evtlogutil
import socket, ssl, pprint, platform, re, os, hashlib, datetime,
sys, thread
import json
from urllib2 import urlopen

#http://nullege.com/codes/show/src@w@i@WinSys-3.x-
0.5.2@winsys@event_logs.py

def getIP():
     ip = json.load(urlopen('http://httpbin.org/ip'))['origin']
     return ip

sig_version = "None"
parsed_sig_version = "None"

def find(name, path): #find a file in a provided path
    for root, dirs, files in os.walk(path):
        if name in files:
```

```
            return os.path.join(root, name)

def getHash(path, name): #get the hash through regular
expression in the provided quaratine path
     now=datetime.datetime.now()
     ago=now-datetime.timedelta(minutes=1)
     hash=""
     files = [f for f in os.listdir(path) if
os.path.isfile(path+'\\'+f)]
     for f in files:
          p = os.path.join(path,f)
          st=os.stat(p)
          mtime=datetime.datetime.fromtimestamp(st.st_mtime)
          if mtime>ago:
               q_file = output = open(p,'r').read()
               if re.search(name, q_file):
                    temp = re.search("[0-9a-fA-F]{64}", q_file)
                    if temp:
                         hash = temp.group(0)
     return hash

def getEvent(event, status): #parse the latest event for updates
or malware alerts
     global definfo, catalog, antivirusExe, quaratine1,
quaratine2, registryPath, registryKey, sig_version,
parsed_sig_version
     try:
          if event.SourceName == "Symantec AntiVirus" or
event.SourceName == "Symantec Network Protection": #Verify it is
a symantec event
               timeGen = str(event.TimeGenerated)
               newdate = datetime.datetime.strptime(timeGen,
'%m/%d/%y %H:%M:%S')
               timeGen = newdate.strftime('%Y-%m-%d %H:%M:%S')
               sourceName = str(event.SourceName) #Pull data
from the event using the win32evtlog library
               eventID = event.EventID
               msg =
str(win32evtlogutil.SafeFormatMessage(event, logtype))
               get_version = GetFileVersionInfo(antivirusExe,
"\\") #Find the antivirus executable and get file version
               version =
str(HIWORD(get_version['FileVersionMS']))+"."+str(LOWORD(get_ver
sion['FileVersionMS']))+"."+str(HIWORD(get_version['FileVersionL
S']))+"."+str(LOWORD(get_version['FileVersionLS']))
               hKey = win32api.RegOpenKey
```

```
(win32con.HKEY_LOCAL_MACHINE, str(registryPath)) #read the
registry for the definition file
                value, type = win32api.RegQueryValueEx (hKey,
registryKey)
                defs = str(value)+definfo
                f = open(defs, 'r')
                content = f.read() #Read file for current
definition data
                if content:
                     parsed_sig_version =
re.search("CurDefs=(.+)", content).group(1)
                     sig_version =
''.join(parsed_sig_version.split('.'))
                     sig_version =
sig_version.split(datetime.datetime.now().strftime("%Y"))
                     sig_version =
str(datetime.datetime.now().strftime("%y"))+sig_version[1]
#format defintion version
                f.close()
                if eventID == 7 or eventID == 1090453511:
#Specific Event ID for update
                     sig_version = re.search("Version: (.+)\.",
msg).group(1)
                     root_dir = value+"\\"+parsed_sig_version
                     path = find(catalog, root_dir)
                     if path:
                          hash =
(str(hashlib.sha256(file(str(path),
'rb').read()).hexdigest())).upper() #get hash of update
                     else:
                          hash = "No file"
                     report =
"Updates*"+timeGen+"*"+str(sig_version)+"*"+hash+"*"+sourceName+
"*"+version+"*"+str(platform.platform())+"*"+str(getIP())
                     sendEvent(report)
                elif eventID == 400: #Specific Event ID for
blocked alert
                     name = re.search("Attack: (.+)\ attack
blocked.", msg).group(1)
                     action = re.search("attack blocked. (.+)",
msg).group(1)
                     report =
"Alerts*"+timeGen+"*"+str(name)+"*No
File*"+sourceName+"*"+str(version)+"*"+str(sig_version)+"*"+str(
platform.platform())+"*"+str(action.strip())+"*"+str(getIP())
```

```
                    sendEvent(report)
               elif eventID == 51 or eventID == 1090453555:
#Specific Event ID for action alert
                    name = re.search("Security Risk Found\!(.+)\
in File", msg).group(1)
                    action = re.search("Action Description:
(.+)", msg).group(1)
                    file_num = re.search("([0-9]{1,4}\.){4}[0-
9]{1,4}", value).group(0)
                    path = quaratine1+file_num+quaratine2
#Create the path for the quaratine file
                    h = getHash(str(path), name)
                    if h == "":
                         h = "Malware file unavailable or
deleted before agent could hash."
                    report =
"Alerts*"+timeGen+"*"+str(name)+"*"+str(h)+"*"+sourceName+"*"+st
r(version)+"*"+str(sig_version)+"*"+str(platform.platform())+"*"
+str(action.strip())+"*"+str(getIP())
                    sendEvent(report)
     except Exception:
          pass
     return

def sendEvent(result): #send report results to server
     global callHomeServer
     global callHomePort
     try:
          print result
          s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
          ssl_sock = ssl.wrap_socket(s)
          ssl_sock.connect((callHomeServer, int(callHomePort)))
          ssl_sock.write(result)
          data = ssl_sock.read()
          print data
          del ssl_sock
          s.close()
     except Exception:
          print "Event not delivered to server"
          pass


#Read in Config Variables
try:
     config = open(os.path.join(os.path.dirname(sys.argv[0]),
'config.txt'), 'r').read()
```

```
     callHomeServer = re.search("callHomeServer = '(.+)'",
config).group(1)
     callHomePort = re.search("callHomePort = '(.+)'",
config).group(1)
     antivirusExe = re.search("antivirusExe = '(.+)'",
config).group(1)
     quaratine1 = re.search("quaratine1 = '(.+)'",
config).group(1)
     quaratine2 = re.search("quaratine2 = '(.+)'",
config).group(1)
     registryPath = re.search("registryPath = '(.+)'",
config).group(1)
     registryKey = re.search("registryKey = '(.+)'",
config).group(1)
     definfo = re.search("definfo = '(.+)'", config).group(1)
     catalog = re.search("catalog = '(.+)'", config).group(1)
except Exception:
    print 'Configuaration file (config.txt) is unavailable or
formatted incorrectly, unable to start agent.'
    sys.exit()

#Setup environment
server = 'localhost' # name of the target computer to get event
logs
logtype = 'Application'
handler = win32evtlog.OpenEventLog(server,logtype)
handlerEvent = win32event.CreateEvent (None, 1, 0, None)
flags =
win32evtlog.EVENTLOG_BACKWARDS_READ|win32evtlog.EVENTLOG_SEQUENT
IAL_READ

print "Finding SYSTEM Events"
win32evtlog.NotifyChangeEventLog(handler, handlerEvent)

while True:
     try:
          if win32event.WaitForSingleObject (handlerEvent, 500)
!= win32event.WAIT_TIMEOUT: #Watch application event log for
updates, if new event appears, trigger
               events =
win32evtlog.ReadEventLog(win32evtlog.OpenEventLog(server,logtype
), flags,0)
               for event in events:
                    result = thread.start_new_thread(getEvent,
(event, "go"))
     except Exception:
```

```
        pass
```

C.2 config.txt

```
/* EDIT THE BELOW IP TO MATCH THAT OF THE TRACKING SERVER AND
ENSURE THE ANTIVIRUS DIRECTORY PATHS ARE CORRECT */
/* v1.0 */

/* If any of these variable are changed, please restart the
service. */

callHomeServer = '192.168.178.144'
callHomePort = '12463'
antivirusExe = 'C:\Program Files (x86)\Symantec\Symantec
Endpoint Protection\Smc.exe'
quaratine1 = 'C:\\ProgramData\\Symantec\\Symantec Endpoint
Protection\\' First half of directory
quaratine2 = '\\Data\\Quarantine' Second half of directory found
within the version file
registryPath = 'SOFTWARE\Wow6432Node\Symantec\Symantec Endpoint
Protection\CurrentVersion\Content'
registryKey = 'VirusDefs'
definfo = '\\definfo.dat'
catalog = 'catalog.dat'
```

APPENDIX D. AGENT – WINDOWS DEFENDER

D.1 Agent-WindowsDefender.py

```
"""Windows Defender Distributed Agent Cloud-Sourced Malware
Reporting Framework v1.0 - Copyright 2013
   Kellie Kercher - agent_research@somethingk.com
   http://www.somethingk.com

   The code is available to anyone interesting in progressing
the reseach in agent based malware analysis. Please contact me
for
   suggestions, questions or improvements. If you are utilizing
this code please give credit to the project.

   For complete malware hashes, real time protection needs to be
turned off.
   This setting is not recommended for the typical user.
"""

from win32api import GetFileVersionInfo, LOWORD, HIWORD
import socket, ssl, pprint, platform, re, os, hashlib, datetime,
time, sys, thread
import json
from urllib2 import urlopen

#http://nullege.com/codes/show/src@w@i@WinSys-3.x-
0.5.2@winsys@event_logs.py

def getIP():
     ip = json.load(urlopen('http://httpbin.org/ip'))['origin']
     return ip

def find(name, path): #find a file in a provided path
    for root, dirs, files in os.walk(path):
        if name in files:
            if root.find('Backup') < 1:
```

```
                return os.path.join(root, name)

def getEventType(record, status):
     try:
          eventID = re.search('Event ID: (.+)', record).group(1)
          file = None
          hash = None
          if eventID == str(1116):
               file = re.search("file:_(.+)", record) #pull out
the malware file path
               if file:
                    file = file.group(1)
                    list = file.split(';')
                    try:
                         f = open(list[0], 'rb').read()
                         hash =
(str(hashlib.sha256(f).hexdigest())).upper() #get file hash
                    except Exception:
                         pass
               malwareID = re.search('  ID: (.+)',
record).group(1)
               return getAlert(malwareID, hash)
          if eventID == str(2000):
               return getUpdate(record)
          return
     except Exception:
          pass

def getAlert(malwareID, hash):
     global antivirusExe
     global delay
     now=datetime.datetime.now()
     ago=now+datetime.timedelta(minutes=25)
     then = datetime.datetime.now()
     while True: #Loop through events untill a 1117 alert result
event appears.
          records = os.popen('wevtutil qe "Microsoft-Windows-
Windows Defender/Operational" /f:text
"/q:*[System[TimeCreated[timediff(@SystemTime) <=
\''+delay+'\']]]"').read() #Pulls the latest defender event from
the windows event log
          then = datetime.datetime.now()
          try:
               parsed = records.split('\n\n')
               for record in parsed:
                    if record != "":
```

```
                         eventID = re.search('Event ID: (.+)',
record).group(1)
                         if eventID == str(1117):
                              id = re.search("    ID: (.+)",
record).group(1)
                              if int(id) == int(malwareID):
                                   timeOccurred =
re.search('Date: (.+)', record).group(1)
                                   newdate =
datetime.datetime.strptime(timeOccurred, '%Y-%m-%dT%H:%M:%S.%f')
                                   timeOccurred =
newdate.strftime('%Y-%m-%d %H:%M:%S')
                                   sourceName =
re.search('Source: (.+)', record).group(1) #use regular
expressions to pull data from the eventlog
                                   get_version =
GetFileVersionInfo(antivirusExe, "\\") #Find the antivirus
executable and get file version
                                   version =
str(HIWORD(get_version['FileVersionMS']))+"."+str(LOWORD(get_ver
sion['FileVersionMS']))+"."+str(HIWORD(get_version['FileVersionL
S']))+"."+str(LOWORD(get_version['FileVersionLS']))
                                   sig_version =
re.search("Signature Version: AV: (.+), AS:", record).group(1)
                                   name = re.search("[^ ]Name:
(.+)", record).group(1)
                                   action = re.search("Action:
(.+)", record).group(1)
                                   if hash == None:
                                        hash = "Malware file
unavailable or deleted before agent could hash."
                                   report =
"Alerts*"+str(timeOccurred)+"*"+str(name)+"*"+hash+"*"+str(sourc
eName)+"*"+str(version)+"*"+str(sig_version)+"*"+str(platform.pl
atform())+"*"+str(action)+"*"+str(getIP())
                                   return sendEvent(report)
               time.sleep(1)
               if now>ago: #After 25 minutes return with no
results, this prevents endless loop
                    return
          except Exception:
               break
     return

def getUpdate(record):
     global antivirusExe
```

```
     global signatureDirectory
     global vdm
     global vdm2
     try:
          type = re.search("Signature Type: AntiVirus", record)
          if type:
               timeOccurred = re.search('Date: (.+)',
record).group(1)
               newdate =
datetime.datetime.strptime(timeOccurred, '%Y-%m-%dT%H:%M:%S.%f')
               timeOccurred = newdate.strftime('%Y-%m-%d
%H:%M:%S')
               sourceName = re.search('Source: (.+)',
record).group(1) #use regular expressions to pull data from the
eventlog
               get_version = GetFileVersionInfo(antivirusExe,
"\\") #Find the antivirus executable and get file version
               version =
str(HIWORD(get_version['FileVersionMS']))+"."+str(LOWORD(get_ver
sion['FileVersionMS']))+"."+str(HIWORD(get_version['FileVersionL
S']))+"."+str(LOWORD(get_version['FileVersionLS']))
               sig_version = re.search("Current Signature
Version: (.+)", record).group(1)
               path = find(vdm, signatureDirectory) #Find
signature file
               path2 = find(vdm2, signatureDirectory) #Find
signature file
               h = ""
               h2 = ""
               if path:
                    h = "MpAvBase.vdm:
"+(str(hashlib.sha256(file(str(path),
'rb').read()).hexdigest())).upper() #hash signature file
               if path2:
                    h2 = "\nMpAvDlta.vdm: "
+(str(hashlib.sha256(file(str(path2),
'rb').read()).hexdigest())).upper() #hash signature file
               h_full = str(h)+str(h2)
               if h_full == "":
                    h_full = "No File"
               report =
"Updates*"+str(timeOccurred)+"*"+str(sig_version)+"*"+h_full+"*"
+str(sourceName)+"*"+str(version)+"*"+str(platform.platform())+"
*"+str(getIP())
               return sendEvent(report)
          return
```

```
     except Exception:
          pass

def sendEvent(result): #send report results to server
     global callHomeServer
     global callHomePort
     try:
          print result
          s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
          ssl_sock = ssl.wrap_socket(s)
          ssl_sock.connect((callHomeServer, int(callHomePort)))
          ssl_sock.write(result)
          data = ssl_sock.read()
          print data
          del ssl_sock
          s.close()
     except Exception:
          print "Event not delivered to server"
          pass

#Read in Config Variables
try:
     config = open(os.path.join(os.path.dirname(sys.argv[0]),
'config.txt'), 'r').read()
     callHomeServer = re.search("callHomeServer = '(.+)'",
config).group(1)
     callHomePort = re.search("callHomePort = '(.+)'",
config).group(1)
     delay = re.search("delay = '(.+)'", config).group(1)
     antivirusExe = re.search("antivirusExe = '(.+)'",
config).group(1)
     signatureDirectory = re.search("signatureDirectory =
'(.+)'", config).group(1)
     vdm = re.search("vdm = '(.+)'", config).group(1)
     vdm2 = re.search("vdm2 = '(.+)'", config).group(1)
except Exception:
    print 'Configuaration file (config.txt) is unavailable or
formatted incorrectly, unable to start agent.'
    sys.exit()

print "Finding SYSTEM Events"
last = ""
full = ""

while True: #Loop through events untill a 1116 alert event
appears.
```

```
     try:
          records = os.popen('wevtutil qe "Microsoft-Windows-
Windows Defender/Operational" /f:text
"/q:*[System[TimeCreated[timediff(@SystemTime) <=
\''+delay+'\']]]"').read() #Pulls the latest defender event from
the windows event log
          if full != records:
               parsed = records.split('\n\n')
               for record in parsed:
                    if last != record and record != "":
                         result =
thread.start_new_thread(getEventType, (record, "Go"))
                         last = record
               full = records
          time.sleep(1)
     except Exception:
          pass
#http://bobthegnome.blogspot.com/2007/08/making-ssl-connection-
in-python.html
```

D.2 config.txt

```
/* EDIT THE BELOW IP TO MATCH THAT OF THE TRACKING SERVER AND
ENSURE THE ANTIVIRUS DIRECTORY PATHS ARE CORRECT */
/* v1.0 */

/* If any of these variable are changed, please restart the
service. */

callHomeServer = '192.168.178.144'
callHomePort = '12463'
/* If your computer is slow or your not noticing any
updates/alerts being delivered you may want to increase the
millisecond delay */
delay = '3000'
antivirusExe = 'C:\Program Files\Windows Defender\MSASCui.exe'
signatureDirectory = 'C:\ProgramData\Microsoft\Windows
Defender\Definition Updates'
vdm = 'mpavdlta.vdm'
vdm2 = 'mpavbase.vdm'
```

APPENDIX E. SERVER LISTENER

```
#!/usr/bin/python
import socket
import _mysql, re, thread
from OpenSSL import SSL
from urllib2 import urlopen

def DMStoDEC(geo):
     dec = geo.replace(r'&deg;',' ').replace('\'','
').replace('"',' ')
     final = dec.split(' ')
     direction = {'N':1, 'S':-1, 'E': 1, 'W':-1}
     return
(int(final[0])+int(final[1])/60.0+int(final[2])/3600.0)*directio
n[final[4]]

def readIncoming(connection, address):
     print 'Connection made on', address
     data = connection.recv(10000)
     connection.send('Recieved')
     connection.close()
     if data:
          print data
          con = None
          try:
               con = _mysql.connect('localhost', 'XXXXXXX',
'XXXXXXX', 'XXXXXXX')
               parsed = data.split('*')
               if parsed[0] == "Updates":
                    con.query("INSERT INTO Updates (time,
signature_version, hash, vendor, version, platform, host_ip)
VALUES ('"+parsed[1]+"', '"+parsed[2]+"', '"+parsed[3]+"',
'"+parsed[4]+"', '"+parsed[5]+"', '"+parsed[6]+"',
'"+parsed[7]+"')")
               else:
                    con.query("INSERT INTO Alerts (time,
```

```
malware_name, hash, vendor, version, signature_version,
platform, action, host_ip) VALUES ('"+parsed[1]+"',
'"+parsed[2]+"', '"+parsed[3]+"', '"+parsed[4]+"',
'"+parsed[5]+"', '"+parsed[6]+"', '"+parsed[7]+"',
'"+parsed[8]+"', '"+parsed[9]+"')")
                    try:
                         geo =
urlopen("http://ipaddress.is/"+str(parsed[9])).read()
                         lat =
re.search("Latitude</td><td>(.+)</td>", geo).group(1)
                         lng =
re.search("Longitude</td><td>(.+)</td>", geo).group(1)
                         lat = DMStoDEC(lat)
                         lng = DMStoDEC(lng)
                         con.query("INSERT INTO markers (name,
lat, lng) VALUES ('"+parsed[2]+"', '"+str(lat)+"',
'"+str(lng)+"')")
                    except Exception, e:
                         print e
                         pass
          except _mysql.Error, e:
               print "Error %d: %s" % (e.args[0], e.args[1])
               pass
          finally:
               if con:
                    con.close()

context = SSL.Context(SSL.SSLv23_METHOD)
context.use_privatekey_file('server.key')
context.use_certificate_file('server.crt')
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s = SSL.Connection(context, s)
s.bind(('',12463))
s.listen(5)

while True:
     (connection, address) = s.accept()
     thread.start_new_thread(readIncoming, (connection,
address))
s.close()
```

APPENDIX F. MALWARE SAMPLES

Table 10: Downloaded Malware Samples

Malware Name	SHA256 Hash	Source	Detecting Antivirus Software
Bredolab	CADC5E5DE727049C9EFBBE262F6483F404818B6EA784EA66D155A9B229BC085C	http://contagiodump.blogspot.com/	Avira Sophos Symantec Windows Defender
Bundestrojan	3407BF876E208F2DCE3B43CCF5361C5E009ED3DAF87571BA5107D10A05DC7BC4 BE36CE1E79BA6F97038A6F9198057ABECF84B38F0EBB7AAA897FD5CF3	http://contagiodump.blogspot.com/	Avira Sophos Symantec Windows Defender

	85D702F		
APT-Taidoor	F105AB22354D586 2401BCF3215D511 9327EC779ED9469 1EA12361672F8C6 34EA	http://contagiodump.blogspot.com/	Avira Sophos Symantec Windows Defender
Gamarue.F or Yakes	E142453F29ACD89 446BA13FD4AB1B 77B923FDD8F00B EC44EE86DA33A7 671FC76	http://contagiodump.blogspot.com/	Avira Sophos Symantec Windows Defender
Blackhole CVE-2010-0840	EE1FC2EC13E0678 24DBC950064115B 6D08705955C3F72 51F360183FACA51 93DA	http://contagiodump.blogspot.com/	Avira Sophos Symantec Windows Defender
Blackhole CVE-2011-3544	C13839854D0D950 319CA97538F1CC E6E050C5596D212 51BB6E925647BF3 E13D6	http://contagiodump.blogspot.com/	Avira Sophos Symantec Windows Defender

Blackhole CVE-2011-0611	1581DC1E2CAC90116A7F91BB8E68D44A7F4513369309C691F71F2D022D85E63A	http://contagiodump.blogspot.com/	Avira Sophos Symantec Windows Defender
Blackhole payload FakeAV	D2444EB298BCBCECC31C548B6F2554424304672E727FBF7497B3CC3DF2E36E24	http://contagiodump.blogspot.com/	Avira Sophos Symantec Windows Defender
TDL/Alureon	D7623DB7E16C1D5B9D20A263576AFC289E7F974CC9CF15F2032F441B8F87C73C	http://contagiodump.blogspot.com/	Avira Sophos Symantec Windows Defender
GameOver Zeus	701B1A1A8F6B59C2EC79776D332A3149F9D5E2AE449214A13A5F76C371FEC522	http://contagiodump.blogspot.com/	Avira Sophos Symantec Windows Defender
CVE-2010-0188	0544461A59606FBC68C6AD5FC61D2	http://contagiodump.blogspot.com/	Avira Sophos

	25A90A905764CB4 33C05FE522E6E48 DC138		Symantec Windows Defender
ZeroAccess.D	9ED60D93D43FC9 A8A670E4EAB9C0 DDDA65B59567B AD2FFE17F4518D 1AD368415	http://contagiodump.blogspot.com/	Avira Sophos Symantec Windows Defender
Kelihos.B	78CCEE8E07EBBC 84D9BA4F5D4952 CCC6BF516213559 B3317A915FD2566 C22FE1	http://contagiodump.blogspot.com/	Avira Sophos Symantec Windows Defender
Sinowal Mebroot Torpig	20FF8BE4C486709 951104EDFDB72F4 30DA479166833D8 544E961AE367B89 8A02 428CFFECA860E2 8ABCD97C24500C 83AD559A0618D6 9EA802803D3196D	http://contagiodump.blogspot.com/	Avira Sophos Symantec Windows Defender

	154AF1C B7A19966529CD48 4004C6403E28BD7 4548E20119421FF0 049A37543D948C4 5C1 0DCB7A582A0E72 DCCCF4FD855A15 9A4206B67B85FD CD0F58B71D85BA 28E40440		
Koutodoor.F	1765AC579AA3307 BD087B7DA60181 41A4FA7529DFBD 0C5A14AA7816B1 5745AC8	http://contagiodump.blogspot.com/	Avira Windows Defender
SCKeyLog.O	553BDD506F30C07 86FD9D02551388B FB3C4E6CC819343 E360FAA46CC100 3B7C7	http://contagiodump.blogspot.com/	Avira Sophos Symantec Windows Defender
Dozmot.D	BF97BE25C653D6	http://contagiodump.bl	Avira

	48DD27EF76B9FC 4B82484940E257C 7EAF94F76BFE756 1FE137	ogspot.com/	Sophos Symantec Windows Defender
“Microsoft Update” phish	05B047592B5D0A4 DA7A9FC0CDE5D 5AE847F50454336 39048FE25FAF6C0 EA8640 2A1D3DE21CB83A 8A2A16A3E5C61A 9214D1BBF3793E CFC0748ACFA15E 774BDE1B 0E14F5E6CDAB92 18135D3A7EED11 F0457C9934210859 F6075D63BC60946 9D43B C48DF0394939FCC B9A3AC0853D0AE 696D04E7C5230D3	http://contagiodump.bl ogspot.com/	Avira Sophos Symantec Windows Defender

	A6468EBCE257A0 BE4CCC B9D5F59CE63AA7 0E3F0398012DC59 DC8519947C0D413 4ECDCAA917A22 DECFF26 107BDE2A12A9A AE73FE078F5CED DA98F2D186F0FC ABF0A4114769640 3DC50CA8		
APT Speech.doc	6A70E797617BB89 58BFBE94A423744 47E3859C6B4EF1E 108D43A30B5DB7 4480B	http://contagiodump.blogspot.com/	Avira Sophos Symantec Windows Defender
Ramnit	F52BFAC9637AEA 189EC918D05113C 36F5BCF580F3C0 DE8A934FE343810 7D3F0C	http://contagiodump.blogspot.com/	Avira Sophos Symantec Windows Defender

TDL	05344813787920A0 4B207416EA05516 B21958B3F6C8AD 9FB8F0CE507C41E FD01	http://contagiodump.bl ogspot.com/	Avira Sophos Symantec Windows Defender
Bakcorox.A	05B047592B5D0A4 DA7A9FC0CDE5D 5AE847F50454336 39048FE25FAF6C0 EA8640 2A1D3DE21CB83A 8A2A16A3E5C61A 9214D1BBF3793E CFC0748ACFA15E 774BDE1B 0E14F5E6CDAB92 18135D3A7EED11 F0457C9934210859 F6075D63BC60946 9D43B C48DF0394939FCC B9A3AC0853D0AE	http://contagiodump.bl ogspot.com/	Avira Sophos Symantec Windows Defender

	696D04E7C5230D3 A6468EBCE257A0 BE4CCC B9D5F59CE63AA7 0E3F0398012DC59 DC8519947C0D413 4ECDCAA917A22 DECFF26 107BDE2A12A9A AE73FE078F5CED DA98F2D186F0FC ABF0A4114769640 3DC50CA8		
Downloader	A3253B1732A5014 6038A68B3B46260 F80BEC6C1C	http://contagiodump.bl ogspot.com/	Avira Sophos Symantec Windows Defender
OSX.RSPlug.A	2BDCDAB0A5D41 F4B6AA48E2AB55 177552C8419C3F8 CE140C4850A0616 D7A2F3E	http://contagiodump.bl ogspot.com/	Avira Sophos Symantec

	CBCF96C780F2D9C0482C7B26154CAB3C4E760AD78D4B742FD4D63E4C08760020		
Zitmo Android Edition	F6239BA0487FFCF4D09255DBA781440D2600D3C509E66018E6A5724912DF34A9	http://contagiodump.blogspot.com/	Avira Sophos Symantec Windows Defender
Eicar Test Virus	275A021BBFB6489E54D471899F7DB9D1663FC695EC2FE2A2C4538AABF651FD0F	http://www.eicar.org/	Avira Sophos Symantec Windows Defender
Unclassified Trojan	81610CDEBCCF492D65140034076CD9FB92FC9171A9C2E1088D18F581B3A6AB11	http://www.malwareblacklist.com/	Windows Defender
Unclassified Trojan	3FC22160DB5F2B307CC9679EA7E9E	http://www.malwareblacklist.com/	

	3BB9E5764227010 1A90121F35762E2 E619E		
Videos09u842937y4y3	373E4F41DF753D10436EE68CC5826147958C9650DB94F8795501E573B2E79DEE	http://www.malwareblacklist.com/	Sophos
Dldr.Delphi.Gen	97596D80FC8E49BD20C58A68C3A65FA9274BF546904EC380F16B0E14B5158AB0	http://www.malwareblacklist.com/	Avira Sophos Symantec
Win32.Generic	FACA641D2ACA636C670CAC681FCE30D72907EDDDB58010BEEDE61FB52CAFCE6F	http://www.malwareblacklist.com/	Avira Sophos Windows Defender
Rouge.9435288	86EDEEBD0A953646DA5D4C8F7F49686CFDC9A1180F46E1BD3C28425D	http://www.malwareblacklist.com/	Avira

	B23B6B25		
Dropper.Gen7	831EE2F602AD941AC9B776400F3CD553960C2756D4B7CA8DC9A88F9005D53940	http://www.malwareblacklist.com/	Avira Sophos Windows Defender
Rouge.1154657	DCBC781A1F3889B35AE3F55C862F2A2309C4E1BBC0088677374A445C08662E4C	http://www.malwareblacklist.com/	Avira
ATRAPS.Gen	72C090D6A4A7639313E8E2FC64ABD425C6A8F8CE1CF5125B5BC79AA5E222EBEC	http://www.malwareblacklist.com/	Avira Sophos
Artemis	F8917B602A887AC5A09255A7673DF6775C772F329C8F6D32D477208FCD526E8C	http://www.malwareblacklist.com/	Sophos
Lamar.skw.44	02B69775E2AB1D	http://www.malwarebl	Avira

	F451D2DE5FB5092 093DDDA7FD682F 90A7640DF53F7A C69F68F	acklist.com/	Sophos
Framer.R.1	E5A2CF61957340D 4E0F991A6DF9819 636110D687856EA E56C54D88EC6B2 1B86D	http://www.malwarebl acklist.com/	Avira
Variant.Graftor	9BB80BCAF6522C 2D4B02193C4764D 985EC2284BA819E C27D9D81ECDD04 DC31BB	http://www.malwarebl acklist.com/	
JS.Expack.FY	57CC9BE38355C8 D991CFC39A8478 AA44134B416FE1 EA908EEA396604 CE78C820	http://www.malwarebl acklist.com/	Avira Sophos Windows Defender