University of Manchester
School of Computer Science

Malware Detection and Remote Quarantine in an Enterprise Environment

Third Year Project

Author: Corey Arthur, Bsc. Computer Science
Supervisor: Dr Ning Zhang, PhD

April, 2018

Abstract

Since the early invention of desktop computers, computer viruses have been developed and deployed around the globe. What may have started as a proof of concept exploration into the ability for computer programs to self-replicate is currently thought to cost billions of dollars' worth of damage each year. In 2017 alone, Panda Security reported more than 15 million different malicious portable executable files had been identified with a 50% increase in ransomware appearances from 2016. Of these malicious portable executables 99.10% were identified as unique [4].

This report will discuss the growing issue of malicious software and demonstrate the need for a new era of antivirus solutions incorporating dynamic network quarantining to help prevent the spread of network-propagating malware. Additionally, the report aims to highlight the advantages of a modular antivirus solution to enable malware detection to maintain pace with the ever-changing malware development ecosystem.

Acknowledgements

...

Contents

Introduction

While the first developed viruses may have only been designed as practical jokes to confuse, the incorporation of computer systems into business procedures and the popularization of the public internet provided the perfect opportunity for malware development to shift towards a more malicious payload. These payloads are designed to steal and/or destroy data held by the businesses but have been built upon over decades with new infection and spreading techniques and varied objectives, from creating a remote storage server to host stolen data to initiating compromised servers into a bot net which may be used to disable remote network infrastructure through synchronized carried out by hundreds of compromised devices.

Since the first malicious program was released, threat actors have reverse-engineered protocols, popular software and operating systems in hopes of finding a zero-day exploit. Such an exploit is one which has not yet been discovered and subsequently patched. These exploits would therefore allow attackers to infect many more machines than relying on targets with systems not containing the latest security patches. In the case of exploits targeting network protocols this creates a significant opportunity for threats to spread from a single infected computer to the rest of the network connected devices which utilise the protocol.

With the recent leak of the NSA's ETERNALBLUE zero-day exploit, targeted at the SMB protocol implementation available on Microsoft Windows devices and similar emerging network protocol exploits the need for an anti-virus solution which has the ability to remotely quarantine an infected device from a network is ever increasing.

Traditionally, antivirus solutions have focused on detecting viruses on a single machine through the use of basic detection techniques which may be carried out on the protected system without need for an analysis server. However, as malware complexity increases it is becoming less feasible for the analysis routines to take place on the system. Therefore, it is becoming more common for clients to perform basic analysis on their system before sending the results to a central server for more detailed analysis. In business environments where data stored may comprise of sensitive data, this approach cannot be used as it can never be ensured that the data sent for analysis does not compromise the sensitive data. It is for this reason that the availability of an antivirus solution which can be deployed by a business would be invaluable.

It is from these issues such that this project aims to investigate the feasibility of a self-hosted antivirus solution which can be expanded upon with the latest detection techniques. ¡ADD MORE HERE¿

1.1 Detection Techniques

In the early days of malware, the number of known malware samples was relatively small though . As a result, new methods are constantly being developed to be able to classify previously unseen families of malware using advanced techniques employing artificial intelligence.

1.1.1 Hash Based

As the number of malicious programs began to rise, it was clear that a solution was needed which would allow files to be classified. The simplest of these was to utilise a hashing algorithm to produce an identifier for a file's contents. As new samples were discovered, their identifiers (file hashes) were added to a database which could then be queried to determine a files classification. However, this required the file to have been previously encountered and was fully reliant on the files contents being identical. As a result, developers began to create many different variants of their malware, each containing a different token and therefore resulting in a different hash.

1.1.2 Byte Signature

Designed partially to combat the issues of hash-based techniques, byte signature instead stores common signatures which have been found to be present in malicious programs. These may be identifiers such as segments of a ransomware letter or anonymous bitcoin addresses for ransom payments to be sent to. Some tools allow multiple byte signatures to be combined with Boolean logic to create more robust rule sets to increase their accuracy.

1.1.3 Heuristics

By extracting features from files such as their metadata or the sections contained in the executable (.text, .code, etc) machine learning can be used to analyse the extracted features against those found within known malicious files to classify previously unseen samples. One heuristic which was often used by earlier antivirus systems was the "Entry Point" set by the executable which determined the offset at which the operating system began execution of the file. This was effective as malware which spreads to other files often appends the malicious code to the end of the legitimate executable and changes the entry point to that of the new malicious code. Alternatively, packers which encrypt malware to prevent static analysis and only decrypt the executable contents tend to have large .text sections consisting of encrypted code and short .code sections which comprise of the decryption routines.

1.1.4 Behavioural Analysis

In cases where obfuscation tools have been used to try to hide the inner workings of a program, or where programs have been 'packed' to encrypt the payload, behavioural analysis techniques can be used to inspect files and binaries in a dynamic environment. First of all, the file is sandboxed to allowing the program to run without modifying any files on the system. While in this sandbox, the program is allowed to operate as designed while the host system captures all interaction between the host and the malware. This information can then be analysed, possibly in combination with heuristics obtained through static analysis to classify the binary.

Design Considerations

As malicious actors constantly discover new techniques to spread and utilise malware, it is necessary that those defending against such threats also innovate upon detection techniques. As a result, in recent years anti-virus solutions have shifted towards new signature-less detection techniques, however these techniques are still constantly evolving.

2.1 Modular Scanner Engines

While this project will not be in long term or production use, it was decided that for the solution implemented to qualify as successful it must be easily extensible to allow custom file scanning and analysis techniques to be created and integrated into the system. Doing so would enable the solution to keep pace with the ever-changing malware ecosystem, futureproofing its ability to detect and classify malware.

2.2 Architecture

Since it was decided that a functional requirement of the system was the ability to interface with a gateway or firewall to allow remote quarantining of an infected device from the network, the architecture of the system had to be designed from a security perspective. Therefore, it was decided that a client-server architecture would be the best approach as this would mean only the server would have the credentials required to issue the instruction to quarantine a client from the network.

Alongside the security considerations, the use of a client-server architecture would open further avenues to explore such as the ability to monitor scan occurrences to identify clients which were not routinely scanning their system, and the ability to deploy schedules for clients to follow. Though these were not implemented but are discussed later in the report in section 5.1.1. ¡MORE INFO ON ARCHITECTURE?¿

2.3 Network Quarantine

Originally, the planned implementation of the network quarantine functionality focused on the use of an open source fork of the popular gateway utility PFSense which advertised the ability to interact with the system through the use of a high-level REST API. However, when looking further into the documentation, it was evident that the API advertised was not as extensive as was made out. The firewall functionality did not have any available API endpoints and therefore could not be configured remotely. Following this discovery, a number of alternative solutions were investigated.

Firstly, the use of a DHCP server allows clients configured with a dynamic IP address to request an available IP address from the server. By developing a basic DHCP server, a table of clients could be stored along with their associated IP and MAC addresses. To quarantine a system, a DCHPRELEASE instruction would be sent to the host who would then discard the current IP address and request a new address from the server but would be denied. Though this approach relies on all clients being configured with dynamic IP addresses and that they obey DHCPRELEASE instructions.

Alternatively, a DNS client converts domain names requested into the associated IP addresses and are often used within businesses to cache DNS entries to lower the overall number of queries required. A custom DNS client could be configured to respond to queries from quarantined clients with an alternative address, for example a webserver hosted by the company with information as to why the client had been quarantined.

However, on a local network malware usually spreads without the use of DNS queries so this approach would not succeed in our requirement of quarantining infected systems.

Finally, since many systems like PFSense use the IPTables utility in Linux, a custom client operating at the network gateway could be developed to dynamically add and remove IPTables rules to instruct the system to drop packets originating from or heading towards a quarantined client. This method was deemed the most suitable due to its simplicity, and flexibility. Additionally, connectivity to specific clients such as the antivirus server could be maintained on a "whitelist" basis. However, this approach requires the assumption that all traffic on the network must travel through the system operating as the firewall regardless of if the clients are currently on the same subnet.

Solution Development

With the functionality and architecture requirements decided during the design phase, the final implementation was achieved through a development process which took place over a number of iterations. As the solution also aims to maximise speed, the implementation utilises multithreading techniques allowing the concurrent execution of tasks. This is most evident during the scanning process, as each thread can operate on a separate file in parallel. This is possible due to the use of an Executor which operates as a task scheduler, storing the tasks which are waiting to be executed and distributing them to threads as they are available.

3.1 Task Structure

With the aim of code modularity, the implementation uses a 'Task' class from which all tasks are derived, providing a number of functions to update other classes which 'listen' to updates to the task, for example progress bars on the overview panel.

The figure above shows the task class hierarchy with a small subset of the classes used in the implementation. For example, ScanTask implements FileSystemTask to provide the file system abstraction, allowing its implementation to be as simple as shown in the figure below.

3.1.1 Scan Task

The below code is the minimum template for a scan module which takes a single file and should then extract the features required for classification at the analysis stage. One

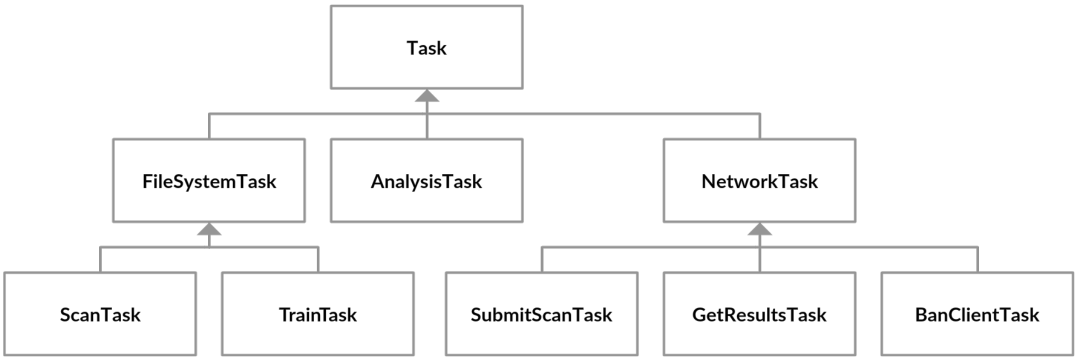

Figure 3.1: A subset of the Task class hierarchy.

implementation example may extract the metadata of the file or alternatively, run an external tool such as 7zip to extract compressed files and analyse their contents.

Listing 3.1: Template used to implement a new scan technique

```
public class ExampleScanTask extends ScanTask {
    protected ExampleScanTask(File files[], ScanType scanType){
        super("Example␣Scan", files, this, scanType);
    }

    @Override
    protected boolean processFile(File file){
        //Define extraction of features from single file.
    }
}
```

3.1.2 Analysis Task

Similar to the Scan Task, this code is the minimum template for an analysis module. The analysis module uses the features extracted by the corresponding scan module to determine a single files classification. This may be as simple as checking the files hash against a database of malicious files, or a complex analysis using artificial intelligence. Once the classification is complete, the result is added to the results report to be sent back to the client.

Listing 3.2: Template used to define analysis of file features.

```
public class ExampleAnalysisTask extends AnalysisTask {
    protected ExampleAnalysisTask(ScanReport scanReport){
        super("Example Analysis", scanReport);
    }

    @Override
    protected boolean processFile(FileFeatures features)
        throws SQLException {
        //Define classification procedure
        //for single files features.
    }
}
```

3.1.3 Network Task

As the primary method of communication between client and server or server and firewall, the network task aims to simplify such communication by abstracting the creation of the communication socket, handshake procedure, encryption and decryption routines. For a developer to create such a task, it is necessary only to define which messages to be sent to the server and the result of the task should an exception arise.

Since the software was designed with security considerations in mind, all communication between clients is encrypted. Firstly, clients must be configured with the analysis server's public key, this is used in the initial handshake procedure as clients generate an ephemeral symmetric key which is encrypted with the public key of the server. The use of a symmetric key allows large messages such as scan reports to be encrypted and decrypted much faster than the use of a public/private key pair. Once received, the server decrypts the key and responds with an encrypted nonce to which the client must decrypt and respond with the nonce-1 to prove the knowledge of the correct key. While this does little from a security perspective in this situation, it enables early detection of a communication error due to the invalid communication of the symmetric key.

3.2 C4.5 Algorithm

The ability to incorporate machine learning aspects into analysis modules is a key factor in being able to distinguish malicious files in samples which have not previously been encountered and analysed by malware researchers. This functionality allows a classification to be derived from a set of features extracted from a file by comparing with a dataset of features extracted from known malicious and known non-malicious files.

The requirements for the algorithm to be used in the project were simple, to support both continuous and discrete features, and to output a binary label to classify the result. The C4.5 algorithm was perfect for this purpose and creates a decision tree which can be easily traversed.

However, since the implementation of the C4.5 algorithm used calculates the entropy of a discrete feature by the number of data elements which share the same value, each data element is expected to store a value for each feature. When hundreds of thousands of Boolean features are present this results in an extremely sparse matrix massively increasing the memory requirements to generate the final decision tree. To overcome this issue, the implementation was expanded to allow features to be specified with a Boolean type, which when calculating the entropy will instead determine the value of a feature through its existence in a HashSet of labels in a data element.

Due to the scale of the scenarios in which we will be using the algorithm, it was also preferable for the algorithm implementation to support multithreading. To enable the algorithm to utilise multithreading self-contained sections of code were located to be spread across the available threads. Through the use of the Java profiling tool JProfiler, it could be seen that the most frequently executed code was the calculation of entropy of a feature during the process in deciding the most profitable feature to split the data elements. By instead splitting the features into smaller groups, the best feature of each group could be identified, once all groups had completed execution the overall best feature could be found. Through this approach, the training time of one data set was reduced from thirteen hours to just below two, though this increase would also depend on the number of threads available to the computer at the time.

In the C4.5 algorithm, three base cases exist to the creation of a terminal node of the decision tree. One such base case is used to prevent overfitting of the dataset and creates a terminal node with the majority classification of the dataset should the number of data elements be below a specified threshold. From this it can be seen that any feature that occurs in data elements less than this threshold, will not massively influence the final tree as the resulting subsets of data elements will then be transformed into a terminal node due to the satisfaction of the base case.

3.2.1 Training Data Acquisition

In order to properly train the system in the detection of malware using the various methods planned, a sufficient number of samples of both clean and malicious files were required. As the detection methods implemented required real life samples to properly demonstrate the technique it was infeasible to automate the creation of malicious files with the degree of variation in operation as would be seen in the wild.

Instead the samples were acquired through VirusShare [5], an online malware repository aimed at malware researchers and security professionals. The site hosts compressed archives each containing thousands of samples of varying types. A number of these were downloaded resulting in 262144 samples, though some scan modules only perform on specific filetypes and to train the modules on all samples would have taken more time than available. As the files were named with only their md5 sum without a filetype, a short script was written utilising the linux file command to identify the file and sort accordingly. Table A.1 shows the types of malware identified after sorting.

Originally, a clean installation of windows 10 was used to train the system however, as with malicious samples, a large quantity of clean samples were required for proper training of the system. Without these, the system would likely over train on these files as they would not correctly represent an in-use system. To overcome this, the open-source package manager chocolatey [1] was used to automate the download and install of the most commonly downloaded windows programs until a sufficient sample was achieved before the hard disk was sorted using the earlier mentioned script. Table A.2 shows the types of clean files identified after sorting.

To speed up the training process, only the suitable files were then used to train a module. For example when training the function import module, only the portable executable files were used.

3.3 Implemented Scanner Engines

In order to evaluate the solution, a number of scanner engines were implemented. These display the antivirus client's ability to integrate existing detection techniques as well as the simplicity of developing and testing a new technique such as Function Import Analysis as below.

3.3.1 Hash Matching

The most basic method of identifying malware consists of the storage of file signatures of known malicious files and comparing the hashes of files scanned by the system to determine if they have been previously identified as malware.

3.3.2 Byte Signature / Rule Based Analysis

Actively developed since 2008, the open-source YARA project by VirusTotal [6] aims to allow malware researchers to easily attribute hand-crafted rules to malware samples to quickly identify malicious files or suspicious code sections. Rules are easily created by specifying byte signatures, strings and regular expressions to be matched to a file, along with a Boolean condition allowing advanced rules to be created by restricting matches to certain subsets of the identified signatures.

¡DISCUSS BYTE SIGNATURES¿

All files processed by YARA are done so in a platform independent manner, therefore providing the correct rules are present, the same output will be generated regardless of which machine the analysis is carried out on. This ensures that all clients will act in the same way when analysing a file, preventing inaccuracies caused by differing client platforms. Additionally, all files are analysed from a binary context, allowing rules to be generated for any file format ensuring that it is possible for the module to identify many forms of malware.

Due to its endorsement by the malware analysis community, it was decided that this implementation would be a welcomed addition to the system. Though since YARA was developed in C, an open source wrapper implementation providing Java Native Interface bindings was used to bridge the gap between the languages.

The YARA rules used in the system were taken from an open-source project [7] aimed at maintaining a collection of rules available under many different categories from rules matching specific malware families to utility rules matching general suspicious strings such as bitcoin addresses and code segments such as Anti-VM or Anti-Debug routines in a binary aimed at preventing malware analysts from analysing the malicious files.

Since the YARA project only outputs the rules as to which a file matches it is not suitable for classification alone, therefore the aforementioned C4.5 algorithm was used enabling the system to learn how to classify files based on the matched YARA rules. A subsection of the resulting decision tree can be seen in Appendix B.1.

3.3.3 Function Import Analysis

While performing static analysis on a malicious Portable Executable (PE32/PE64) binary, a common technique used by analysts is to enumerate the reusable code libraries and the functions imported from each. By inspecting these functions, it is possible to gather a basic understanding of the underlying purpose of an executable file. For example, if it is seen that the WinHttpOpen function from WINHTTP.dll has been imported we can expect that the executable most likely interacts with a http website. While this may be innocuous by itself, combined with URLDownloadToFile and WinExec it could suggest that the program opens a http connection before downloading and executing a file. This technique is used by so called 'Dropper' malware, which isn't malicious by itself but instead downloads a remote payload before saving the file and executing it at a later time.

Since the antivirus solution being developed also focuses on the ability to easily trial new malware detection techniques, this scan module was added as an experiment into the feasibility of combining the static analysis technique with machine learning to automate the detection of portable executable files using imported functions for malicious purpose.

PortEx [3], developed by Karsten Hahn as part of his master's thesis while at HTWK

Leipzig, is a Java library focusing on static analysis of Portable Executable binaries and the identification of malformed binaries as a result of malicious infection [2]. The tool provides a number of utilities for identifying anomalies, extracting strings and visualizing the format of the binary however in this case only the utility for extracting the imported functions will be used, though in future the module may be expanded to incorporate the anomaly detection functionality.

Similar to the rule based analysis module, this module makes use of the C4.5 machine learning algorithm to determine the classification of a file based on the imported functions extracted from the binary. A subsection of the resulting decision tree can be seen in Appendix B.2.

3.3.4 Combined Module

The combined module implemented simply carries out the feature extraction procedures of the hash module, rules analysis module and the function import analysis module. The analysis of the file then executes the analysis routines of each of those modules in turn. This approach increases the scan time but allows more opportunities to detect the malicious files however suffers from an increased chance of false positives since only one module is required to vote malicious no matter the number who vote for the classification of clean.

A better approach may involve weighting each analysis module based on its accuracy and having each vote on the classification of the file, 0 for clean and 1 for malware. The final classification would then be made through the use of the weighted sum of these votes being above, or below a given threshold.

3.4 Scanning Procedure

When a user initiates a scan and selects the folders to be scanned, the client system creates a new scan task using the scan module selected by the client. This allows a range of scans to be performed by the client should they wish to execute a specific scan. For example, a user may simply wish to perform a hash scan on a large file or a multi-technique scan on the entire computer.

3.4.1 Scan Process

Once the task has been created, the scan manager sends it to the task scheduler to be executed once a thread becomes available. The system will then traverse the chosen directories creating a new subtask for each file and queueing them into the task scheduler.

As a file is processed by the system, the features extracted are stored in a report which is sent to the server at a suitable interval. By default, this is set to occur once the scan report contains around one hundred items. This aims to minimise the time wasted initiating the session without the server having to hold a socket open for a client when others may potentially be ready to submit, without delaying the scan process by waiting for all files to be processed before submission.

3.4.2 Report Submission

To submit a scan report, the scan manager creates a network task to begin communication with the server. This task performs the necessary session initiation with the server and sends an analysis command along with the scan report encoded in a JSON representation. Once the submission is confirmed by the server the connection is closed, the scan manager will record the identifier of the report and periodically query the analysis server for the results.

3.4.3 Analysis

When an analysis request is received by the server, an analysis task is created for the appropriate module and sent to the task scheduler. The implementation of the analysis task will then consider the features extracted for a file to determine how it should be classified. As with the scan module, this process may use external tools such as the submission to a cloud analysis platform or other techniques. In the case a malicious file is detected, the analysis manager will be notified by the task which may then request the quarantine of the device from the network.

3.5 Network Quarantine Implementation

To ensure true quarantine for network devices infected with malware, the implementation was required to have the functionality to block all network communication between devices from a remote system. This prevents infected systems being able to reconnect to the network through the changing of the host network settings, such as changing to a static ip address in a scenario where DHCP was used to quarantine a system.

For this reason, the unix iptables utility was chosen to enable rules to be added forcing packets sent by a quarantined device to be dropped, or rerouted. The use of rerouting is advantageous in cases where it is preferable that infected devices are instead directed to limited functionality mirrors of the requested services. For example, an infected device may have web traffic rerouted to the business IT Helpdesk portal, or a transparent network proxy which may employ enhanced blocking rules during the quarantine period.

However as mentioned previously, this implementation requires that all local traffic be routed via the gateway device designated as the quarantine firewall. This has the disadvantage of putting a higher load on the gateway device, though this can be alleviated through the use of multiple analysis servers and firewalls as described in section 5.1.2.

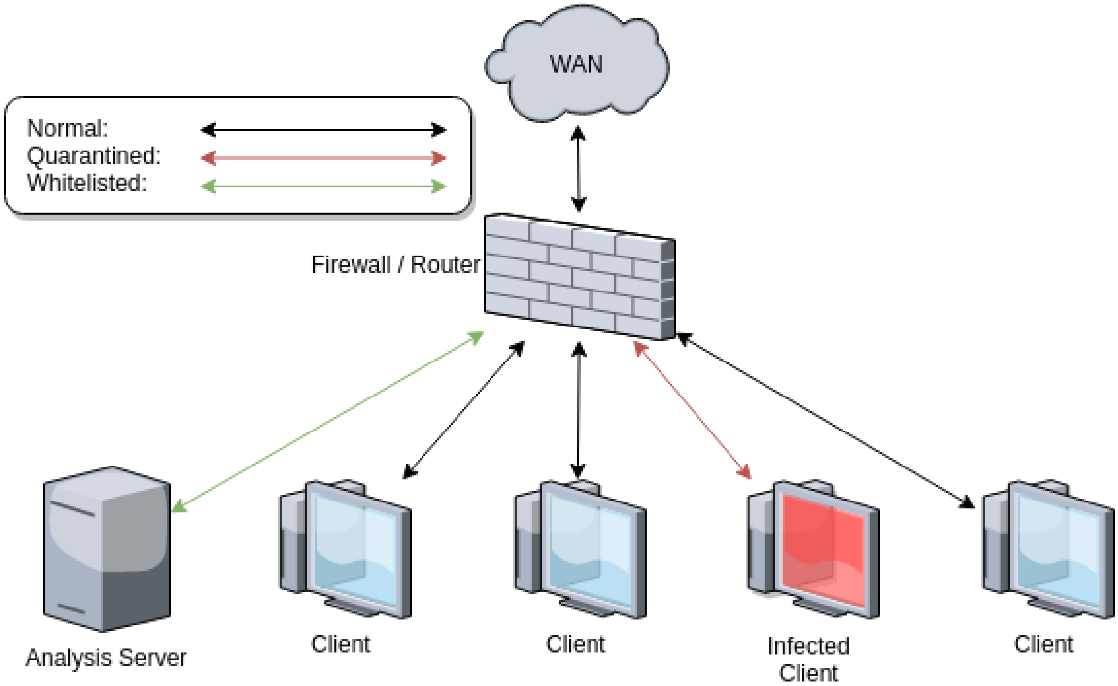

Figure 3.2: A simple network example.

Figure 3.2 shows a simple network example, where all clients share a single subnet. In this scenario, one client has performed a scan which the analysis server has determined to contain malicious files and has subsequently quarantined the client. The network traffic rules have been displayed in the figure as color coded arrows.

The 'clean' clients may communicate along the black and green connections allowing them to access the analysis server, the internet and other 'clean' clients though any transmission to the infected client will be dropped by the firewall.

The infected client may only communicate with the router and any client connected via a green connection. Therefore it may not access the internet, or any other non-whitelisted client on the network.

Figure 3.3: Sample iptables commands generated by the firewall.

```
iptables -F Forward
iptables -I Forward -d AnalysisServerIP -j ACCEPT
iptables -I Forward -d WhitelistedServiceIP -j ACCEPT
iptables -I Forward 3 -s InfectedClientIP -j REJECT
```

As the firewall system loads, it first clears the existing rules to ensure previous entries will not interfere with its operation. Since iptables operates using the first matching rule, the whitelisted ip addresses are then added to ensure all clients may access these services before any clients which were quarantined before the system was shutdown are added with the reject action to prevent the traffic originating from these from being forwarded on to their destination. An example of the commands executed on the system to achieve this operation are displayed in figure 3.3.

One improvement which may be made to the system is to replace the use of infected host ip addresses with a MAC address filter instead. However, it is not trivial for the server to obtain this information from the connected client, though one method may be to retrieve the MAC address from the server's ARP table.

Evaluation

4.1 Accuracy

As the system will quarantine clients which are seen to be infected with malware, the modules should aim to be as accurate as possible. A false positive could delay the workflow of a user while a false negative could allow malware to go undetected potentially damaging the business infrastructure and its data.

To measure the accuracy of the modules 5000 clean, and 5000 malicious files were scanned by the system. All files used were Portable Executable files sampled at random from the VirusShare [5] database.

4.1.1 Import Analysis

As can be seen in table 4.1, the import analysis module has an accuracy of 95.61%. While this accuracy is relatively high for an experimental technique, in a production environment where thousands of files are scanned routinely, the false positive rate of 5.7% could cause unnecessary quarantining.

		Prediction		
		Clean	Malware	Total
Actual Classification	Clean	4715	285	5000
	Malware	154	4846	5000
	Total	4869	5131	

Table 4.1: Confusion matrix of samples using the Function Import Analysis module.

		Prediction		
		Clean	Malware	Total
Actual Classification	Clean	4863	137	5000
	Malware	45	4955	5000
	Total	4908	5092	

Table 4.2: Confusion matrix of samples using the Yara Rules Analysis module.

4.1.2 Yara Rules

Table 4.2 shows a confusion matrix of the results of scanning the 10000 testing files using the Yara Rules / Byte Matching module. As shown by the results, the module is 98.18% accurate with only a 0.91% false negative rate.

4.2 Speed

4.3 Antivirus Comparison

Conclusion

5.1 Future Improvements

5.1.1 Analysis Server

5.1.2 Network Quarantine

Appendix A

A.1 Sample Counts

Format	Count
HTML document	176331
PE32 executable	58070
gzip compressed data	11846
ASCII text	2834
ELF	2793
data	2319
Zip archive data	2291
RAR archive data	553
XML 1.0 document	551
Composite Document File V2 Document	550
Macromedia Flash data (compressed)	538
Other	3286

Table A.1: Malicious sample counts for various filetypes.

Format	Count
ASCII text	100677
data	47289
XML 1.0 document	43137
PE32 executable	42899
Python script	32809
python 3.6 byte-compiled	27370
PNG image data	27211
PE32+ executable	23053
Ruby script	21015
C source	20037
HTML document	16206
C++ source	9731
UTF-8 Unicode text	8579
Perl5 module source	6566
Java archive data (JAR)	6347
current ar archive	5795
python 2.7 byte-compiled	5549
UTF-8 Unicode (with BOM) text	5087
Other	68315

Table A.2: Clean sample counts for various filetypes.

Appendix B

B.1 C4.5 Tree - Yara Rules

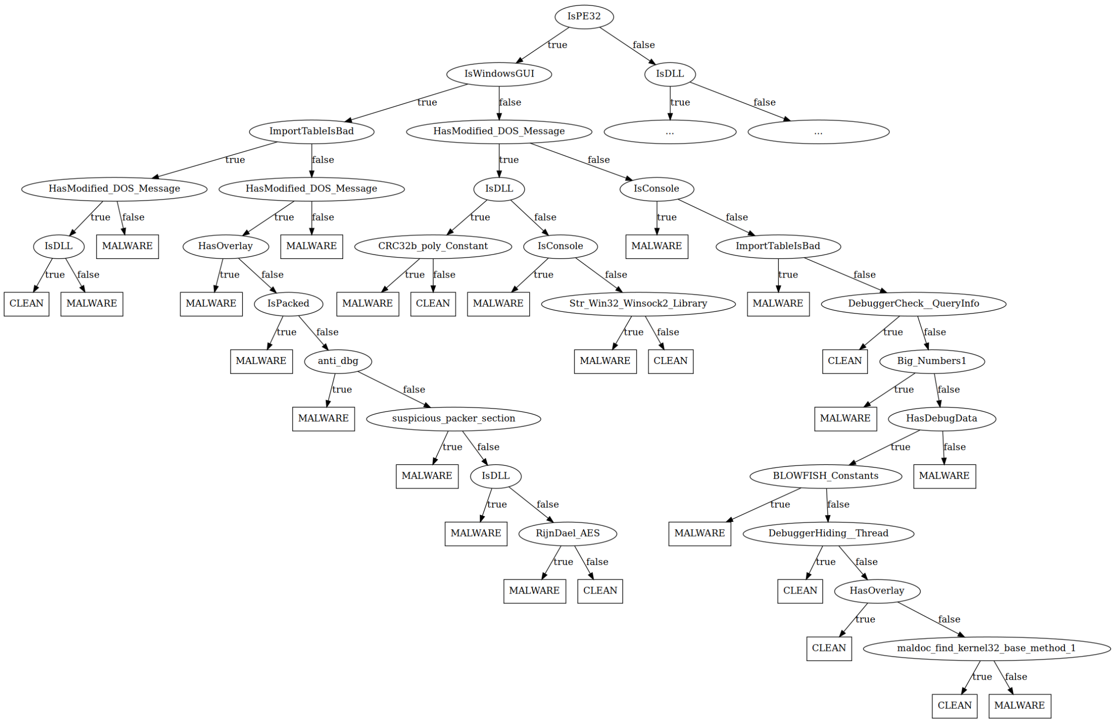

Figure B.1: A subset of the decision tree generated using C4.5 on extracted YARA rules.

B.2 C4.5 Tree - Function Import Heuristics

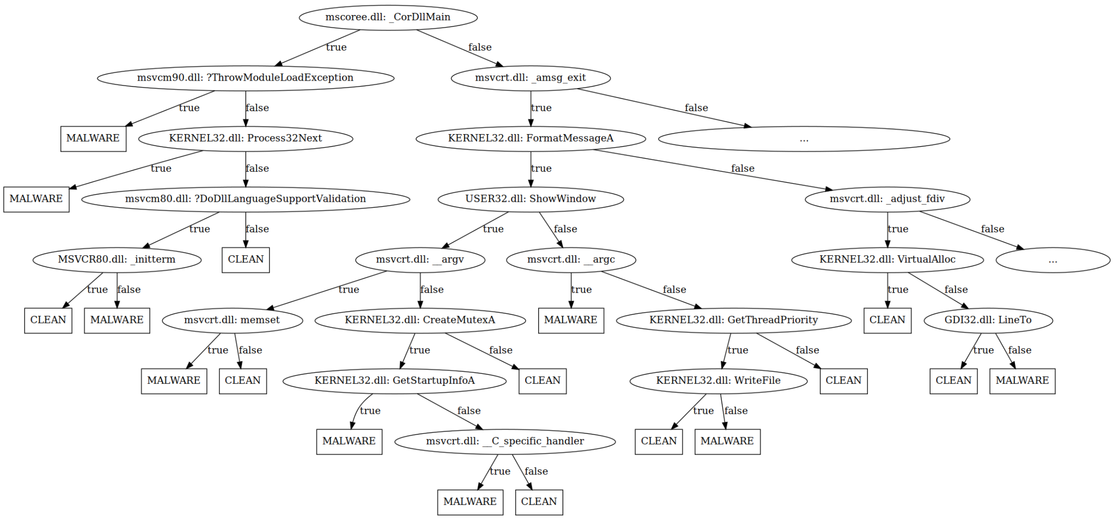

Figure B.2: A subset of the decision tree generated using C4.5 on function imports.

References

[1] *Chocolatey - like yum or apt-get, but for Windows*. URL: https://chocolatey.org/.

[2] Karsten Hahn. "Robust Static Analysis of Portable Executable Malware". type. HTWK Leipzig.

[3] *PortEx*. 2014. URL: https://github.com/katjahahn/PortEx.

[4] Panda Security. *PandaLabs Annual Report 2017*. report. institution.

[5] *VirusShare*. URL: https://virusshare.com/.

[6] *YARA, The pattern matching swiss knife for malware researchers*. 2013. URL: https://virustotal.github.io/yara/.

[7] *YaraRules Project*. 2015. URL: https://yararules.com.