University of Manchester
School of Computer Science

NetProtect:
Malware Detection and Remote Quarantine Solution
for 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 [6]. With the increasing trend of network propagation as a means to spread malware, the need for a combined antivirus and remote quarantine facility in a business environment is ever increasing.

This project covers the development of NetProtect, a next generation malware detection and remote quarantine suite. Focusing on preventing infected systems from spreading to the remaining infrastructure whilst providing a modular interface to enable augmentation with new detection techniques, future-proofing the facility against the ever changing malware development ecosystem.

This report will discuss the growing issue of malicious software and demonstrate the need for a new era of antivirus solutions which can be integrated with firewall implementations by highlighting the advantages of a modular antivirus solution to enable malware detection to maintain pace with the ever-changing malware development ecosystem. The results of the report show how this could be achieved by combining detection techniques to increase the accuracy of the NetProtect system.

Acknowledgements

Contents

Chapter 1

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. This may be as simple as creating a remote storage server to host stolen data to initiating compromised servers into a botnet, 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 [5], targeted at the SMB protocol implementation available on Microsoft Windows devices, and similar 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. For that reason 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. Therefore, 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.

1.1 Malware Detection Techniques: an Overview

In the early days of malware, the number of known malware samples was relatively small. Though due to the large attack surface available its use in cyber crime has spurred its development, rapidly increasing the number of samples year after year. 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 Based

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 Based

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 Based

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.

1.2 Existing Solutions

In the antivirus field, a number of existing solutions exist with the aim of detecting and removing malware.

1.2.1 Carbon Black

Carbon Black, developed as an endpoint security solution expands on traditional antivirus systems with the use of big-data analytics in the cloud. The system is comprised of a number of modules including threat response which provides systems administrators with a management console to create and view alerts from hosts equipped with their endpoint security software.

1.2.2 Sophos

What started as a traditional antivirus solution, Sophos moved to a light client model with a web console for the management of scan results, configuration and alerts. Though

this is designed for the simple management of a number of home computers, the implementation could be expanded quite simply to allow larger networks to be configured for an enterprise environment.

1.2.3 Windows Defender

Designed for standalone operation on a host computer, Windows Defender and many other traditional antivirus systems have incorporated cloud processing into their systems. This speeds up the scanning process by analysing unknown files offline in the cloud, and storing the results for faster processing in future scans.

Chapter 2

NetProtect Requirements

To achieve the goals set for the implementation of NetProtect, a number of requirements must be met to ensure that the system is functional, with additional goals set to allow the product to be easily extended to maintain a high standard of operation even as malware development and obscuration techniques evolve.

2.1   Requirement 1: Malware Classification

The most necessary requirement of the implementation is the ability to scan files and classify them as malicious or non-malicious to a given degree. This may be achieved through a range of techniques, each of which should be evaluated for accuracy and speed.

2.2   Requirement 2: Extensibility

The extensibility requirement of the system aims to ensure that malware researchers are able to design and implement custom scanning and analysis techniques to be used within the NetProtect system alongside the modules which have been developed as part of this project. Doing so, this requirement will give the product the ability to reinforce its ability to detect malware with the latest detection methods.

2.3   Requirement 3: Network Quarantine

To prevent the spread of malicious files over a networked environment, this requirement ensures that an infected device is properly prevented from accessing other clients on the network. Ideally, the infected device should be able to communicate with pre-set hosts

on a whitelist basis to allow infected hosts to maintain communication with the business technical support department.

Chapter 3

NetProtect Design

With traditional antivirus products focusing on the protection of individual devices, their use in a business context as a front-line solution lacks the benefits of a central management interface. This leaves the security of the systems down to the user, who may forget, or not care to routinely perform antivirus scans.

NetProtect is instead designed to allow clients to be managed remotely, taking the responsibility from the user and providing a platform for which client scans can be analysed off-device with the future possibility of remotely initiating scans on a routine basis, or automatically quarantining systems which have not been scanned for an extended period of time.

3.1   Architecture Overview

As NetProtect has the functionality to make major changes to the network environment, the design architecture had to be considered from a security perspective. Therefore its architecture has been designed as a thin-client for use on host machines, reducing the attack surface available should a malicious actor gain access to a system, with a central server used to analyse the files submitted by the client.

To meet the functional requirement of being able to remotely quarantine a device from the network, NetProtect must interface with a gateway or firewall to be able to issue quarantine instructions. As this has the potential to be abused to damage the network, this interface must only be available to a trusted entity in the network which will be issued credentials to communicate with the gateway. Based on this consideration, a client-server architecture was chosen to allow the server to play the role of the trusted entity.

Alongside the security considerations, the use of a client-server architecture would open further avenues to explore to enhance security. These include 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. These were not implemented but are discussed later in the report in section 6.1.3.

Figure 3.1 shows how each component is connected, from which it can be seen how the client is treated as an untrusted entity preventing rogue access to the database and firewall components.
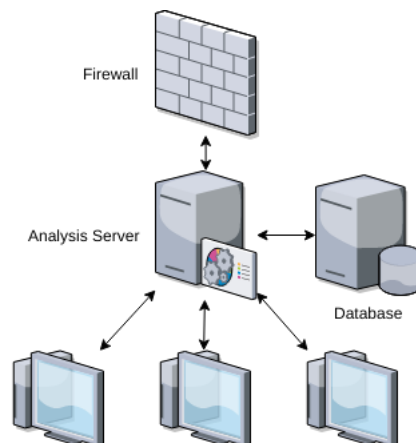
Figure 3.1: NetProtect solution architecture.

3.1.1 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.

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.

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.

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.

3.2 Architecture in Detail

3.2.1 Client Architecture

As the client is designed to simply extract the features required by the server for analysis, its architecture is simple. Figure 3.2 displays the structure of the underlying components, with its functionality revolving around the Scan Manager component. As clients initiate a scan via the interface, the Scan Manager generates the scan task from the chosen Scan Module and queues it for execution by the Task Scheduler. As the task begins generating reports to be analysed by the server, the Scan Manager stores the report reference before the Connection Manager sends the report for analysis.

Once reports have been sent for analysis, it is the job of the Scan Manager to periodically retrieve the results from the server.
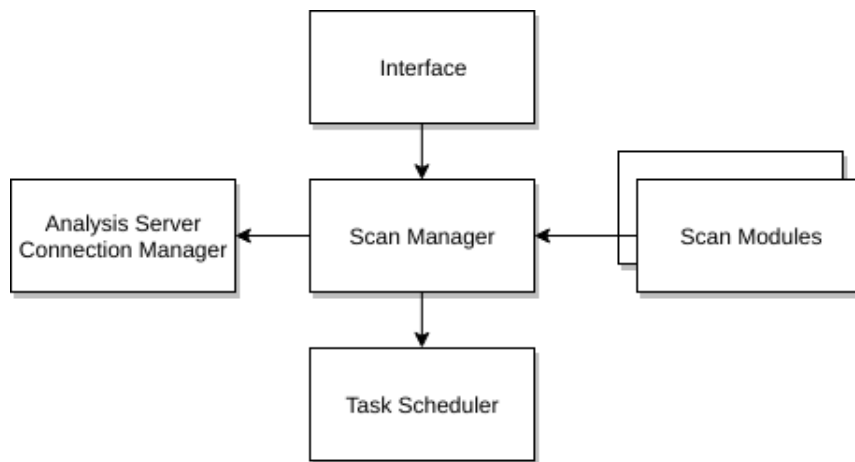
Figure 3.2: NetProtect client architecture.

Interface

To enable the user to interact with the system in an efficient manner, a simple user interface was required providing the functions of initiating a scan, and informing the user of the state of existing in-progress scans.

Figure 3.3 displays the general user interface showing some system statistics, and the general status of the current task.

A more detailed view of the scan tasks can be seen in figure 3.4 displaying the files being scanned, their status and where files have already been scanned, their classification. Additionally, a panel containing information on the report submission process is displayed below.
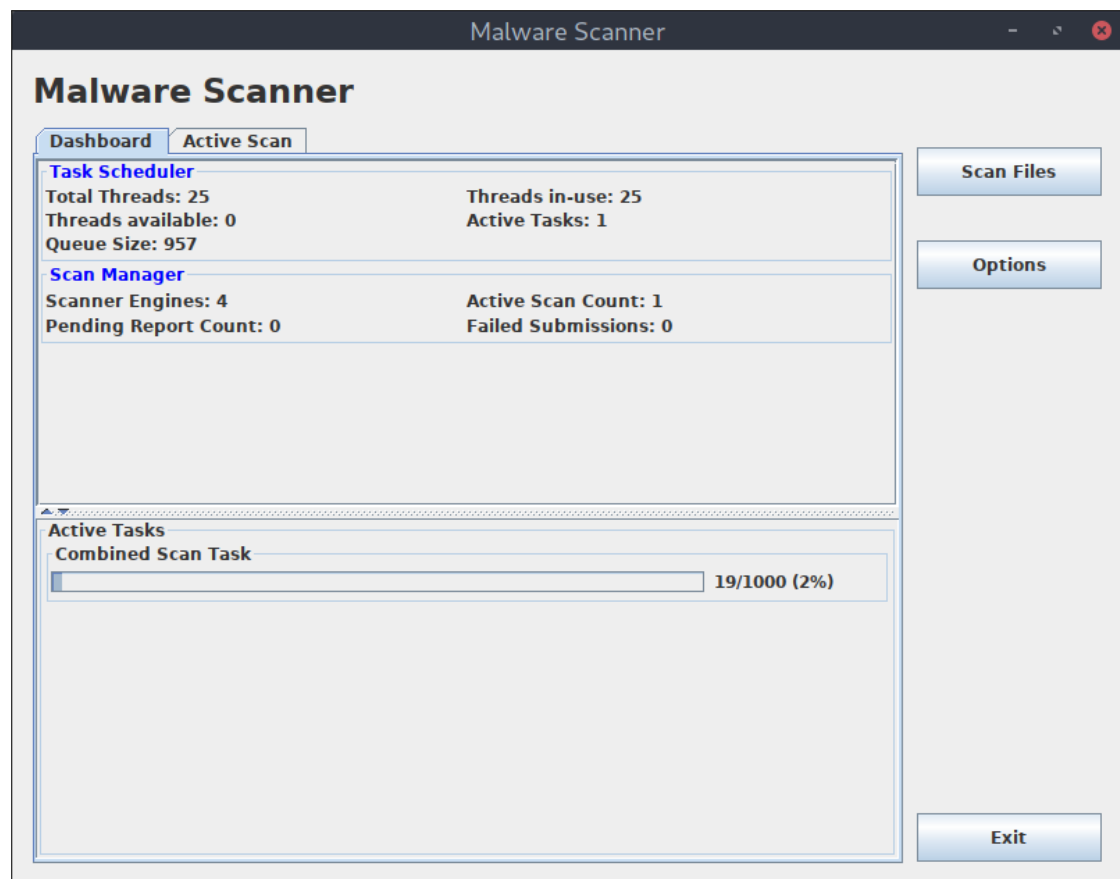
Figure 3.3: NetProtect client interface.
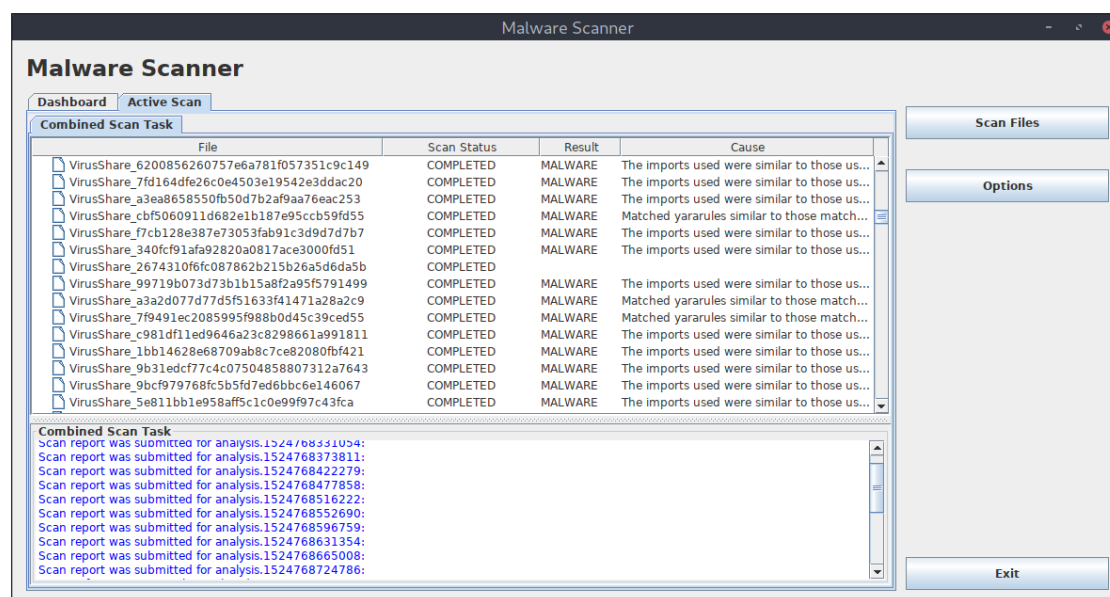
Figure 3.4: NetProtect client scan interface.

Scan Modules

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.

While NetProtect will not be in long term or production use, it was decided that a functional requirement was that 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.

Hash Based Scan Module

The most simple of the scan modules, the hash based scan determines the file classification by calculating the file hash, and submitting it to the server.

Byte Signature Based Scan Module

As with the technique described in section 1.1.2, comparing the file content to byte signatures provided by malware researchers allows simple observations to be made about

14

a files operation. This module extracts 'interesting' byte signatures, which are sent to the server for analysis.

Import Heuristic Based Scan Module

An experimental scan technique, the import heuristics scan module extracts the functions imported from a portable executable file to attempt to determine the executables purpose. The list of features extracted from portable executable files are sent to the server for classification. However if the file is not a portable executable file, or does not have an import section it will be skipped.

Combined Technique Scan Module

While the hash based detection technique is quickly becoming ineffective, its effectiveness at detecting known malware is unrivalled, therefore the combined technique uses each of the implemented modules in order, ensuring all known malware is detected, with a fallback to newer heuristic techniques.

3.2.2   Server Architecture
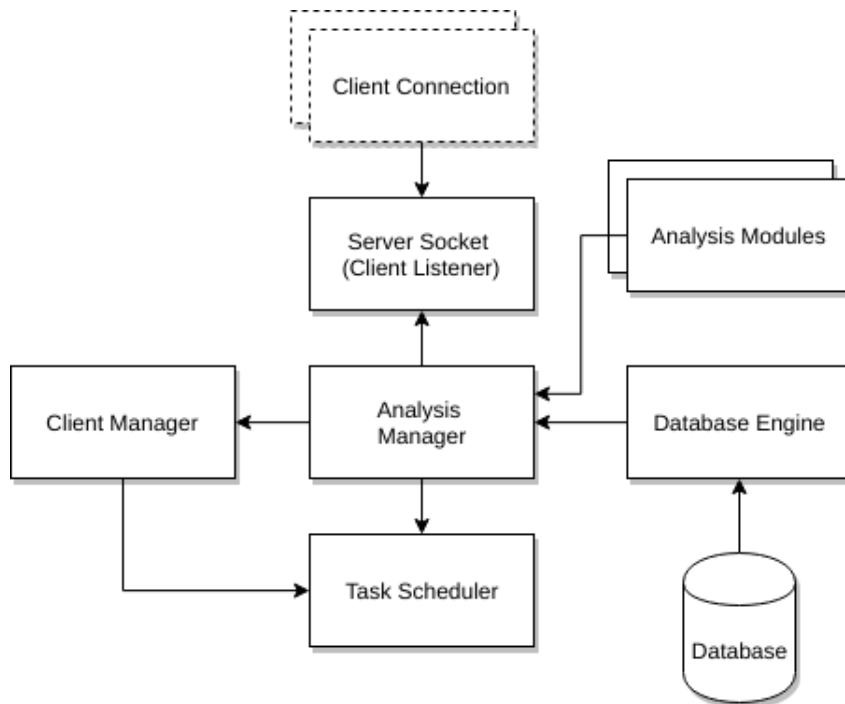
Figure 3.5: NetProtect server architecture.

As seen in figure 3.5, the server architecture is built around the central component, the analysis manager. Firstly, clients connect with the Server Socket component which processes the requests made and transmits them to the analysis manager. These requests are then recorded by the analysis manager and queued for execution within the task scheduler.

In the case malware is detected during analysis, the analysis manager will then notify the client manager, which will in turn generate a task to quarantine the client, which will then be queued for execution with the highest priority.

Interface

While the server architecture is primarily designed as an automated component, as this is the core component of the system the decision was made to provide system administrators with a simple interface to view the status of ongoing tasks, a statistics panel to give a quick overview of the usage, and a simple control interface to remove clients from quarantine once the threat has been controlled.
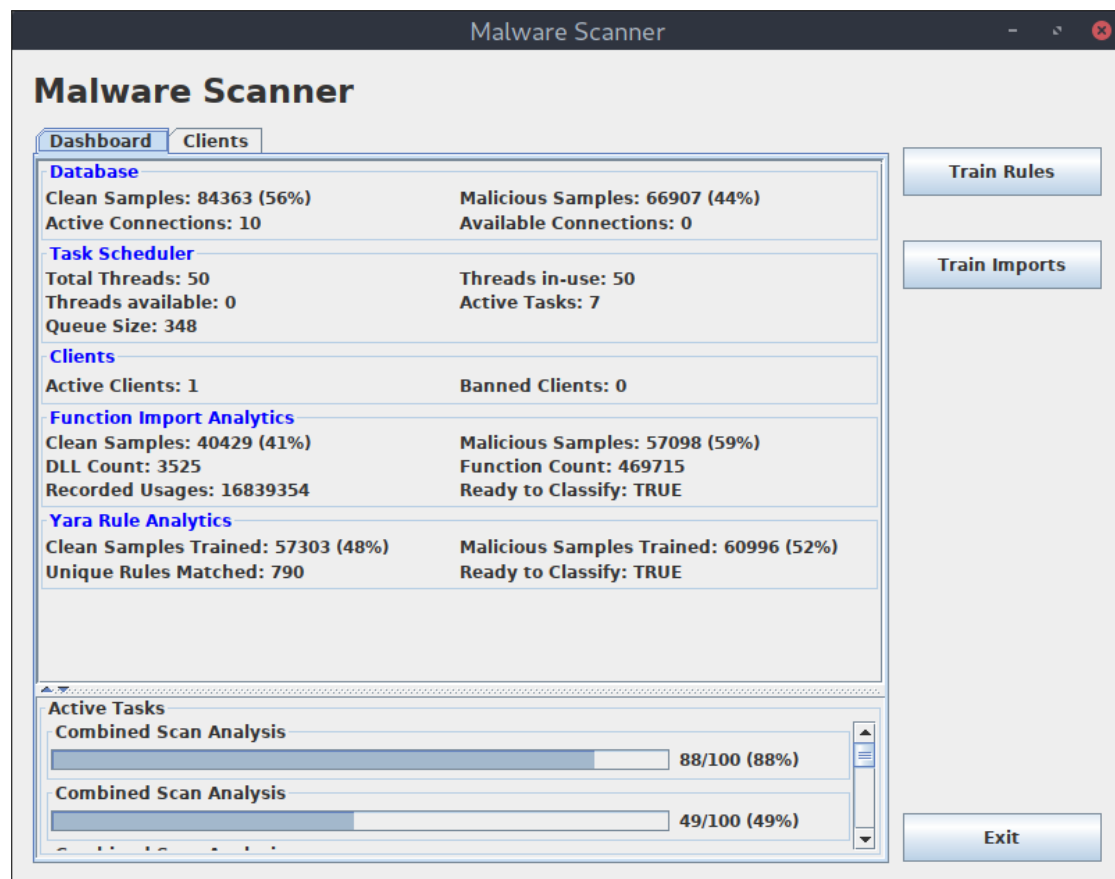
Figure 3.6: NetProtect server interface.

Figure 3.6 shows the server interface, with detailed statistics about the systems usage and analysis module state. A number of analysis tasks can be seen processing within the interface.

C4.5 Algorithm

Incorporating 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 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 suitable for this purpose and creates a decision tree which can be easily traversed.

The algorithm implementation is simple, for each feature in the dataset, the entropy

17

is calculated at the current level where the data be split using that feature. The best feature is then selected as the one which provides the highest entropy gain, and recorded as a node in the tree. To optimise the process, three base cases exist to generate a terminal node: all data elements have the same label, the number of data elements is below a threshold, or there exists no data elements in that portion of the tree.

Hash Based Analysis Module

As with the associated scan module, the hash based analysis module is simple. Each file hash submitted by clients is checked for existence in a database of known malicious files, files are classified as malicious only if the signature is present.

Byte Signature Based Analysis Module

Using the byte signatures matched by the clients during the scanning process, the server utilises the C4.5 algorithm described in section 3.2.2 to classify files using the decision tree generated during the algorithms training phase. The tree is navigated using the extracted matching byte signatures until a terminal node is reached, classifying the file.

Import Heuristic Based Analysis Module

As with the byte signature based analysis module, the algorithm uses the C4.5 algorithm to classify files. The decision tree is navigated with the path taken chosen as a result of the imported functions within the portable executable file. The terminal node at which the tree arrives is the resulting classification of the file.

Combined Technique Analysis Module

To overcome the shortfalls of the other techniques, the combined analysis module performs hash based, byte signature based, and import heuristic based analysis in order. Should any of the stages identify the sample as malware, the analysis is completed ignoring the result of the subsequent stages.

3.2.3 Firewall Architecture

The architecture of the firewall element of the NetProtect system displayed in figure 3.7 shows the simple composition of the two components required. Firstly, the server

Figure 3.7: NetProtect firewall architecture.

socket allows clients to send commands which are processed to instruct the client manager quarantine, or remove clients from the quarantine by interacting with the iptables utility.

Quarantine Component

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 required. 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. The DHCP protocol implementation can be seen as defined in RFC2131 [2].

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.

However, the chosen method was the use of the iptables utility in Linux, a custom client operating at the network gateway was developed to dynamically add and remove rules instructing 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. Though one issue with 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.

Chapter 4

NetProtect Implementation

With the functionality and architecture requirements decided, 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.

4.1 Design Patterns

4.1.1 Template Method Pattern

With the aim of code modularity, the implementation uses a 'Task' class from which all tasks are derived. Within NetProtect, a task is defined as a runnable segment of code which can be queued, and subsequently executed by the task scheduler. These are used throughout the application as a number of subclasses which can be extended to implement additional functionality as specified in the requirements section 2.2. To achieve this, the template method pattern was used along with Java inheritance allowing tasks to be created for various purposes, with only implementation specific procedures needing to be written.

Figure 4.1 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.
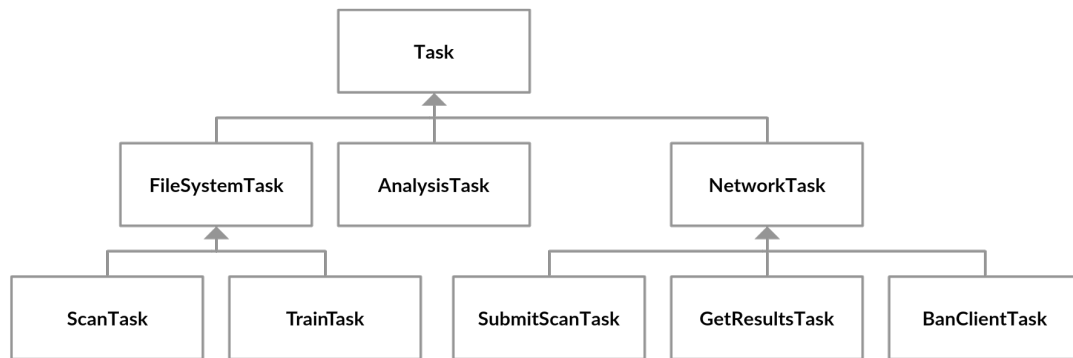
Figure 4.1: A subset of the Task class hierarchy.

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 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 4.1: Template used to implement a new scan technique

```java
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.
    }
}
```

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 4.2: Template used to define analysis of file features.

```java
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.
    }
}
```

Network Task

As the primary method of communication between client and server or server and fire-wall, 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.

4.1.2 Observer Pattern

To integrate the interface of the client and server components with the state of the various 'tasks' used within NetProtect, the observer pattern was implemented, allowing interface components to subscribe to state updates from other components. For example, the 'Task Scheduler' component notifies listeners of task creation and completion and is used by the active tasks panel to dynamically create and destroy progress panels for the various tasks.

23

4.2   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 [7], 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.

4.3   Libraries Utilised by NetProtect

4.3.1   YARA - Pattern Matching Utility

Actively developed since 2008, the open-source YARA project by VirusTotal [8] 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.

Listing 4.3: YARA rule to match anti-debug procedures used by malware.

```
rule anti_dbg {
        meta:
                author = "x0r"
                description = "Checks if being debugged"
                version = "0.2"
        strings:
                $d1 = "Kernel32.dll" nocase
                $c1 = "CheckRemoteDebuggerPresent"
                $c2 = "IsDebuggerPresent"
                $c3 = "OutputDebugString"
                $c4 = "ContinueDebugEvent"
                $c5 = "DebugActiveProcess"
        condition:
                $d1 and 1 of ($c*)
}
```

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.

The YARA rules used in the system were taken from an open-source project [9] 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 researchers analysing the malicious files. Listing 4.3 is an example of such a rule, matching strings which check for the presence of a debugger.

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 library is used within the NetProtect client as part of the byte signature based scanning module described in section 3.2.1.

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.

4.3.2 PortEx - Portable Executable Analysis

PortEx [4], 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 [3]. 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.

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.

This utility is used within the client import heuristic based scanning module described in section 3.2.1. 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.

4.4 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 6.1.4.
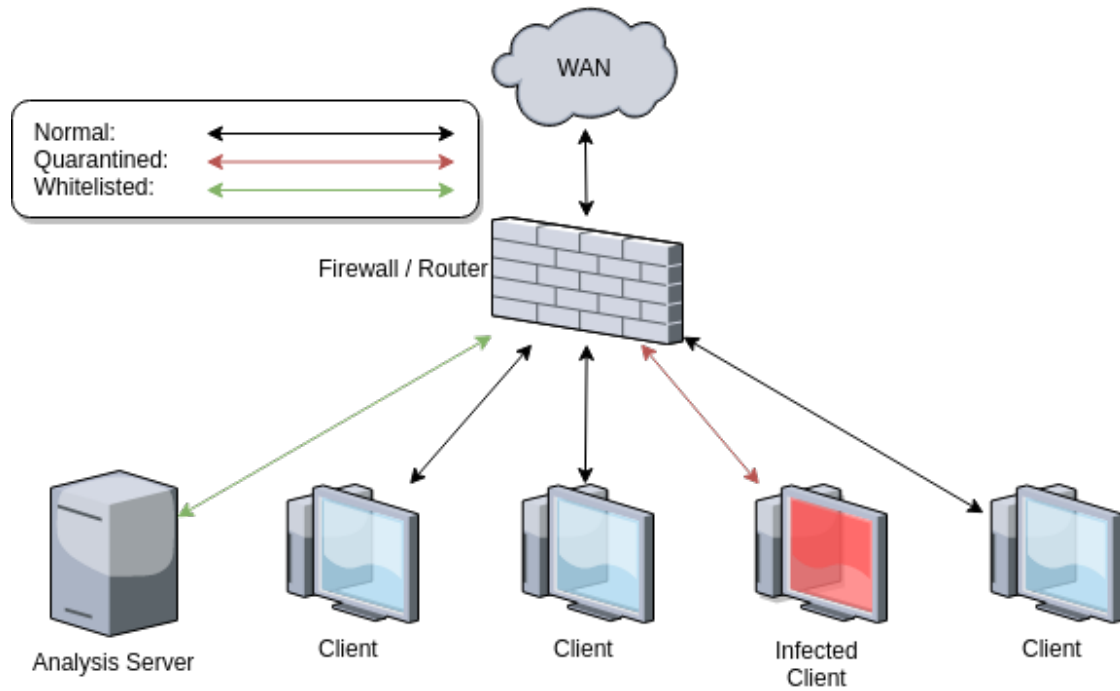

Figure 4.2: A simple network example.

Figure 4.2 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.

```
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
```

Figure 4.3: Sample iptables commands generated by the firewall.

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 4.3.

4.5   Optimisations to the C4.5 Algorithm

Though the C4.5 algorithm described in section 3.2.2 is designed to be fast, a number of optimisations were required to speed up its operation further, and to reduce memory requirements for the generation of large trees.

Firstly, since the implementation of the 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.

Also, 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.

Chapter 5

Evaluation

5.1 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 1000 clean, and 1000 malicious files were scanned by the system. All files used were Portable Executable files sampled at random from the VirusShare [7] database.

In the following sections, "Sample Visibility" is defined as the percentage of commercial antivirus products which are able to detect a given sample. i.e. a visibility of 100% shows that all tested antivirus products detect a given sample, and 0% shows that no antivirus product detected the sample as malware.

5.1.1 Import Heuristic Based

As can be seen in table 5.1, the import analysis module has an overall accuracy of 82.3%. While this is relatively good for an experimental technique, the false negative rate of 32.4% would not be satisfactory for a standalone classifier and may cause unnecessary quarantining of systems in a production environment. Though by considering figures 5.1 and 5.2, it can be seen that the majority of these inaccuracies occur in samples which are only detected by 12% of commercial antivirus products, and that beyond this point the detection technique's accuracy rapidly improves, with an accuracy of over 90% for samples which have a visibility greater than 30%.

	Prediction		
	Clean	Malware	Total
Actual Classification Clean	970	30	1000
Malware	324	676	1000
Total	1294	706	

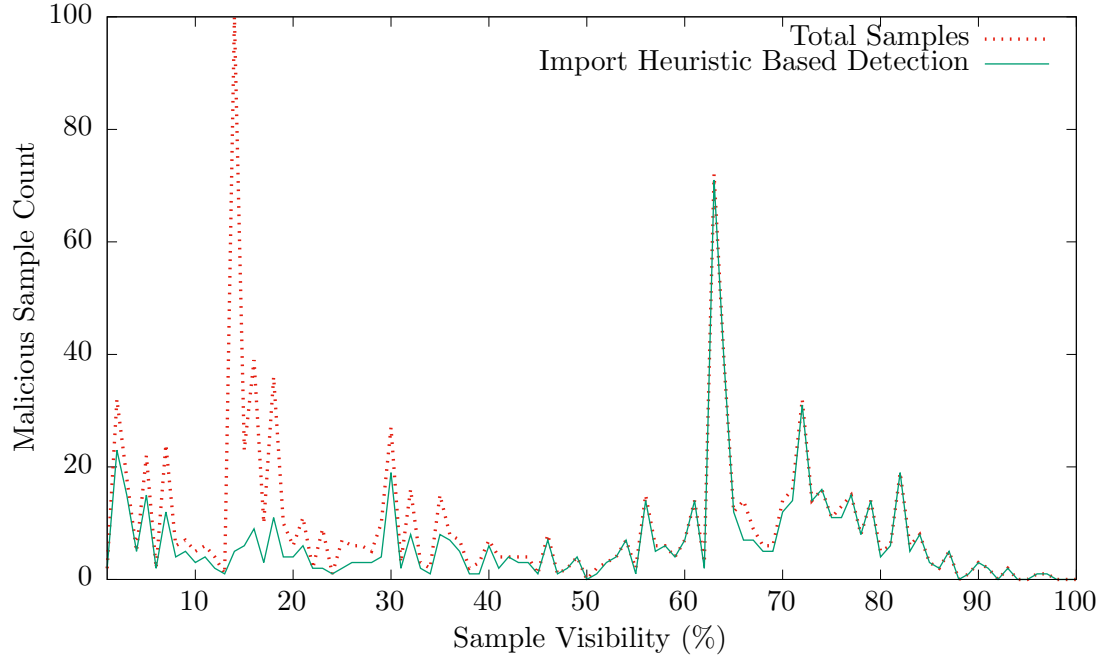Table 5.1: Confusion matrix of samples using the Import Heuristic Based detection technique.

Figure 5.1: Detection counts for Sample Visibility between 0% and 100%
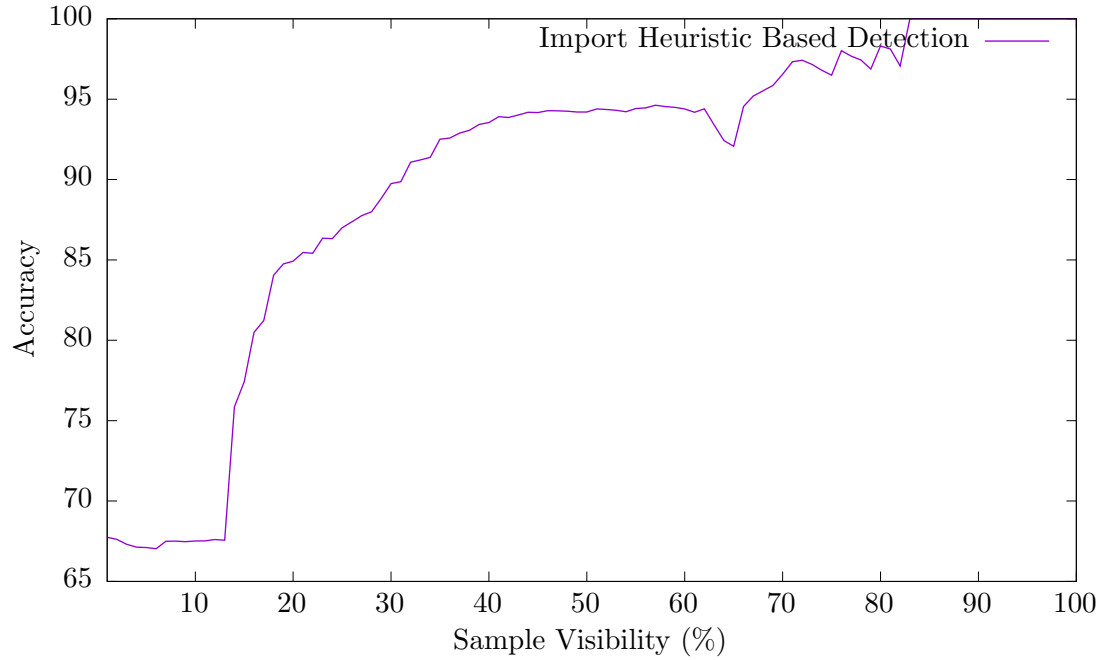
31

Figure 5.2: Accuracy of Import Heuristics Based Module

5.1.2 Byte Signature Based

Table 5.2 shows that of the 2000 files scanned, the byte signature based technique had an accuracy of 97.85% with only 30 false negatives, and 13 false positives. From this it is clear that the use of byte signatures is critical in an antivirus solution to detect signs of malicious routines through the use of well crafted signature rules. While figure 5.3 shows that the accuracy of the system is almost constant amongst both high profile samples and less obvious samples, it is not possible to derive a single technique for detecting all malicious samples. As shown by figure 5.4 even beyond the 80% visibility mark, a number of samples still managed to evade detection.

		Prediction		
		Clean	Malware	Total
Actual Classification	Clean	987	13	1000
	Malware	30	970	1000
	Total	1017	983	

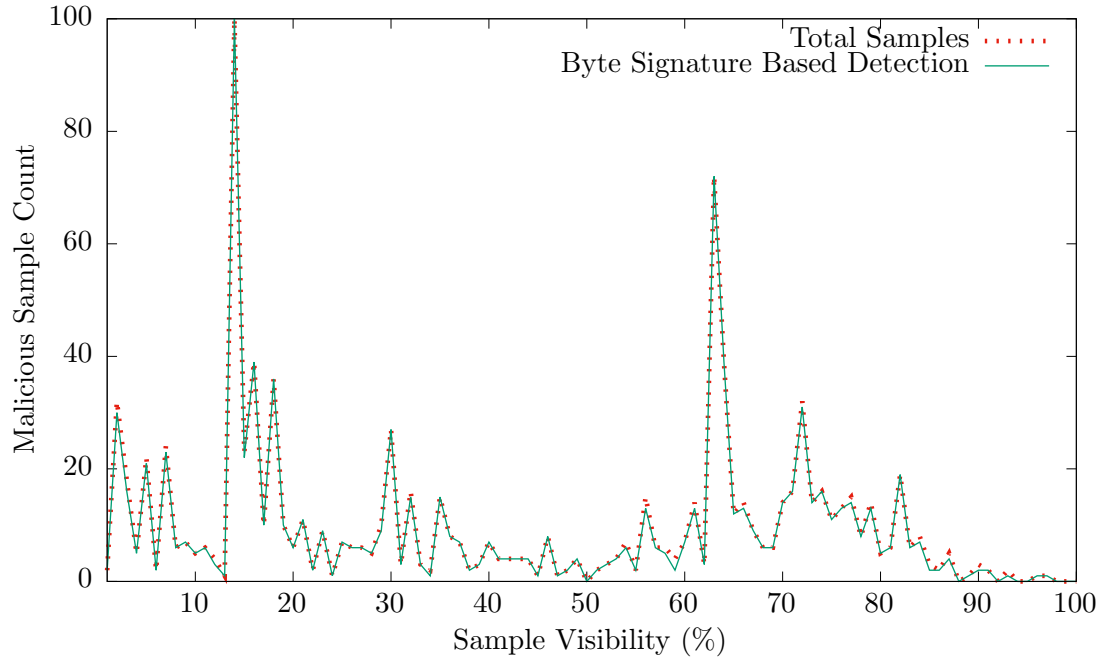Table 5.2: Confusion matrix of samples using the Byte Signature Based detection technique.

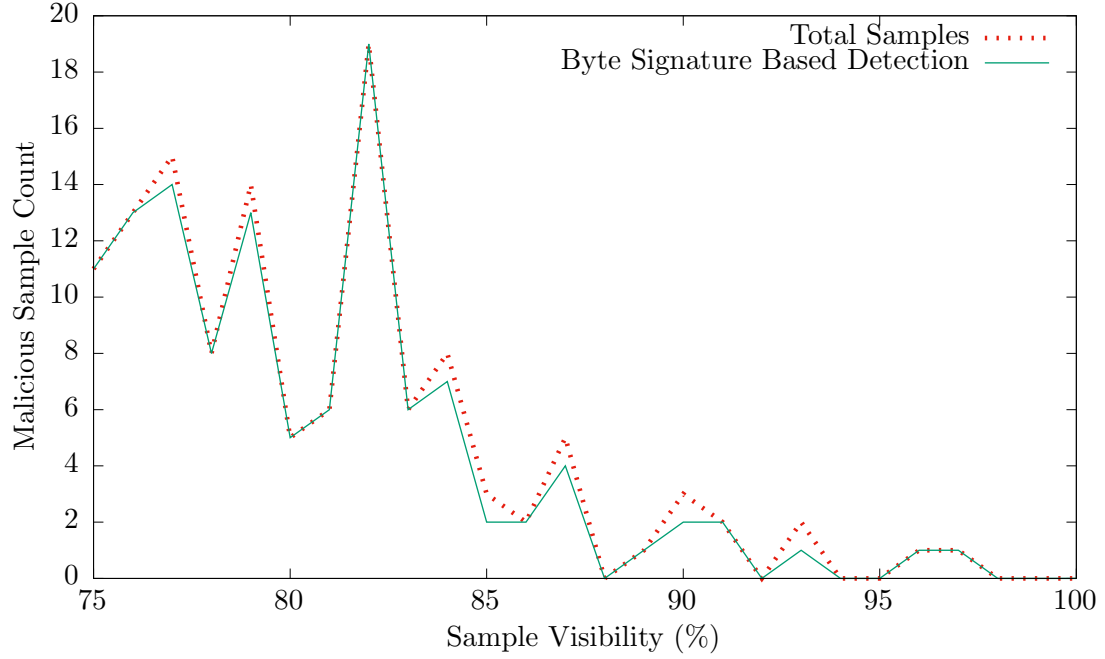Figure 5.3: Detection counts for Sample Visibility between 0% and 100%

Figure 5.4: Detection counts for Sample Visibility between 75% and 100%

5.1.3 Combined Technique

When using the combined detection technique, table 5.3 shows that by combining the three basic detection techniques, overall accuracy slightly decreases to 97.45% as opposed to 97.85% achieved through the use of byte signatures alone however, the false negative rate drops from 3% to 0.9%. This is advantageous in ensuring as many malicious files are detected as possible however comes at a trade-off for an increased false positive rate, up to 4.2% from 1.3% achieved from byte signature based detection.

| | | Prediction | | |
		Clean	Malware	Total
Actual Classification	Clean	958	42	1000
	Malware	9	991	1000
	Total	967	1033	

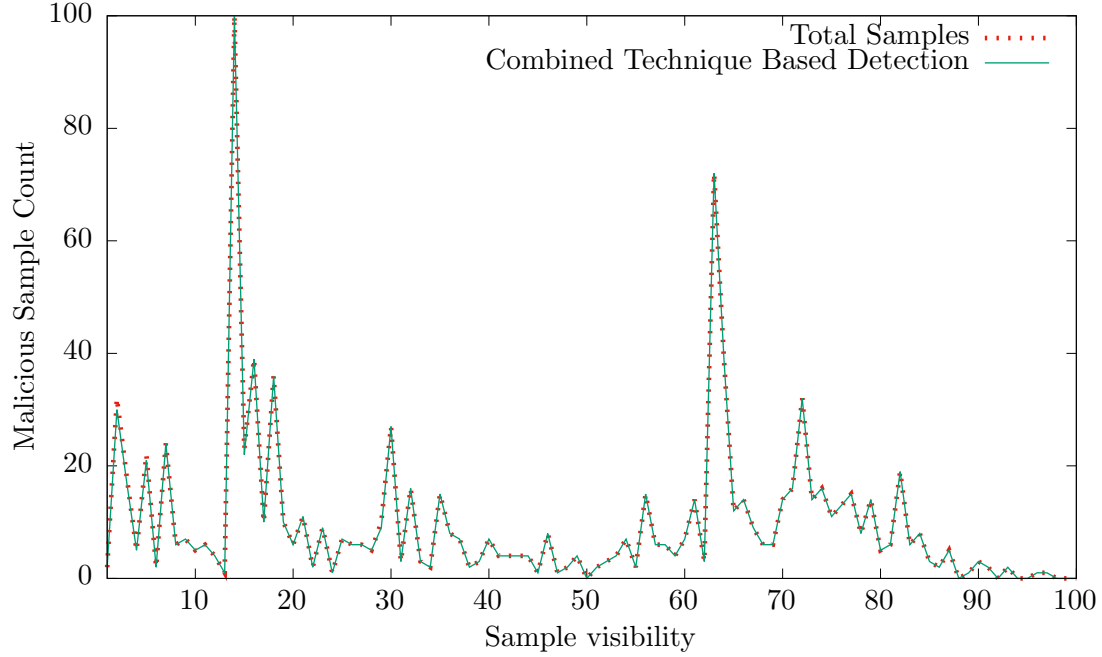Table 5.3: Confusion matrix of samples using the combined detection technique.

Figure 5.5: Detection counts for Sample Visibility between 0% and 100%

5.1.4 Conclusions

Overall, the results of the accuracy analysis show that it is infeasible to create a single detection technique to classify previously unseen samples with high accuracy due to the massive variation of techniques and payloads utilised by malware. Since programs exist for a massive range of usages, there is also a high amount of intersection between the functionalities used by malicious software and non-malicious software which use similar techniques for a legitimate purpose. Therefore, it is much more reliable to identify potentially dangerous segments of code through the use of byte signatures.

The use of a combined technique as implemented in NetProtect provides more opportunities for malicious files to be identified though comes at a cost to the false positive rates of the system. Though a more robust combination technique is briefly described in section 6.1.3 which instead uses a weighted approach for each detection module used in the combination.

Due to limitations beyond control, the samples selected as test data are taken from a discovery date within a three month period. This may effect the results of the analysis as it is possible a new malware family was identified during this period causing an above-average number of samples to be detected.

5.2 Speed

Using the same selection of samples as used to test the accuracy of the detection techniques, the time taken to scan 1000 files was recorded for each of the detection techniques. For comparison, the total and average file sizes can be seen in table A.3.

From table 5.4 it is clear that the speed of commercial antivirus products is much faster than the modules used in NetProtect. However, these products all utilise cloud scanning, which submit the file hashes of files to be scanned to a remote server which retrieves the classification of the file in cases where the file has previously been scanned, preventing the need to scan common files multiple times saving time in the overall scan.

By scanning the same selection of files on a device not connected to the internet, the scanning speeds show a visible difference highlighting the benefits of the cloud scanning technique.

Product	Sample Count	Total Time (s)	Time / GB
NetProtect: Import Heuristic Based	2000	108.19	49.86
NetProtect: Byte signature Based	2000	516.61	238.07
NetProtect: Combined Technique	2000	1242.81	572.72
Norton	2000	60.04	27.67
Kaspersky	2000	65.36	30.12
Windows Defender	2000	116.95	53.89
Norton w/o Cloud Scanning	2000	99.36	45.79
Kaspersky w/o Cloud Scanning	2000	89.47	41.23
Windows Defender w/o Cloud Scanning	2000	132.76	61.18

Table 5.4: Scan speed comparison of various products.

5.3 Quarantine Speed

To test the speed of quarantine, a single malicious file was scanned using Hash Based Detection with the client recording the time the scan started, and the firewall component recording the time the quarantine rules were put into effect. However, this test was performed on a virtual network without additional traffic. On a production network, minor delays would be present.

As seen in table 5.5, the time taken from detection to quarantine is minimal at only 185ms, however the time taken to detect the malicious file was 2267ms. For a single file this may be considered to be unsuitable however much of this time could be attributed to the connection process between client and server which is mitigated through the use

of batch submissions of file features. However, files scanned early may be delayed in submission until more files have completed the scanning process. An additional submission technique is described in section 6.1.1 which may improve the submission times.

Process	Time Taken (ms)
Scan Initiated	0
Scan Report submission	1294
Malware Detection	973
Quarantine Effective	185
Total	2452

Table 5.5: Time taken from scan initiation to network quarantine.

Chapter 6

Conclusion

The development of NetProtect was completed as an investigation into the use of various malware detection techniques in combination with a firewall product to automatically quarantine devices infected with malware.

From the evaluation of the final product developed, it is clear that to perform effectively within an enterprise environment without disruption to the network, a combined firewall and antivirus solution must maintain a high detection accuracy to prevent unnecessary quarantining of hosts which have falsely been identified as infected.

The analysis of the implemented detection techniques also prove that an antivirus system should combine multiple detection techniques to produce a robust classifier able to detect different malware families.

However, if a suitable detection accuracy could be achieved, the use of a combined solution using conditional packet routing, such as the iptables utility as used in NetProtect, to prevent malware propagation over a network would prove invaluable.

6.1 Future Improvements

6.1.1 NetProtect Client

Byte Signature Based Analysis

The use of byte signatures is beneficial due to its format independent nature allowing files of any type to be classified. However, due to time constraints the implementation is trained only on portable executable files. To achieve the best results, the implementation may also include the mime-type of each file as a feature to be considered during the C4.5 algorithms training phase, allowing training files to be partitioned by filetype. This would

in effect create a sub tree within the generated decision tree for each filetype trained by the system.

6.1.2 Scan Report Submission

As seen during analysis, the submission of scan reports may cause files scanned early in the scanning process to have their submission delayed until additional files have been scanned. An alternative approach may be to allow the client to open a connection with the server and submit the features of each file in a stream as they are scanned, this would prevent files being delayed but could create additional delays where the number of clients connected to the server reaches a maximum causing future connections to be dropped.

6.1.3 NetProtect Analysis Server

Combined Technique Analysis Module

The combined module implemented simply carries out the feature extraction procedures of each of the implemented scanning 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.

Client Management

To enhance the system security further client management features could be implemented in future developments. To ensure clients are routinely scanning their system, the addition of a server-designated scanning schedule would allow system administrators to force the client to carry out a scan.

Additionally, clients which have been disconnected for an extended period of time could be quarantined until the system has been scanned. This in-effect creates a 'air lock' to ensure systems which have been migrated from other network locations are deemed safe before being accepted into the network.

6.1.4   NetProtect Firewall Server

The current implementation of the network quarantine function relies on the quarantining of devices from the network by their assigned ip address. This system is flawed in that an infected client could simply request a new ip address from the DHCP server. This can be remedied with static DHCP binding to ensure clients always receive the same ip address though it would be more beneficial for the firewall server to replace the use of infected host ip addresses in iptables rules 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 to be considered may be to retrieve the MAC address from the server's ARP table.

To reduce the load on a single system, another future improvement would be to allow a central analysis server to be designated, which may then distribute the analysis tasks among several 'worker' servers, also having the benefit of increasing analysis speeds. Another improvement to reduce the load is to allow each subnet within a network to contain an analysis server with each of these configured to communicate with the subnet gateway in a tiered architecture.

Appendix A

Appendix A

A.1   Malicious 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.

A.2 Non-Malicious Sample Counts

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.

A.3 Testing Data File Size

Classification	Count	Total Size	Average Size
Malicious	1000	1745MB	1745KB
Non-Malicious	1000	425MB	42KB

Table A.3: Total and average file size of the testing sample.

Appendix B

Appendix B
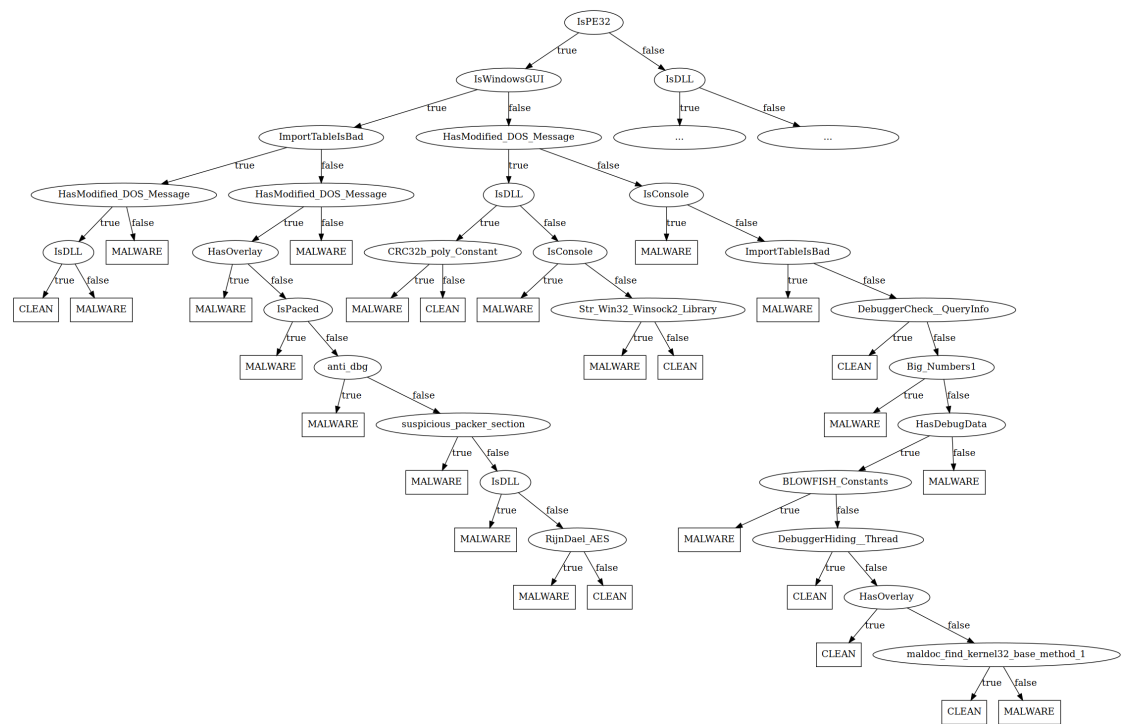
B.1   C4.5 Tree - Yara Rules

Figure B.1: A subset of the decision tree generated using C4.5 on extracted YARA rules.
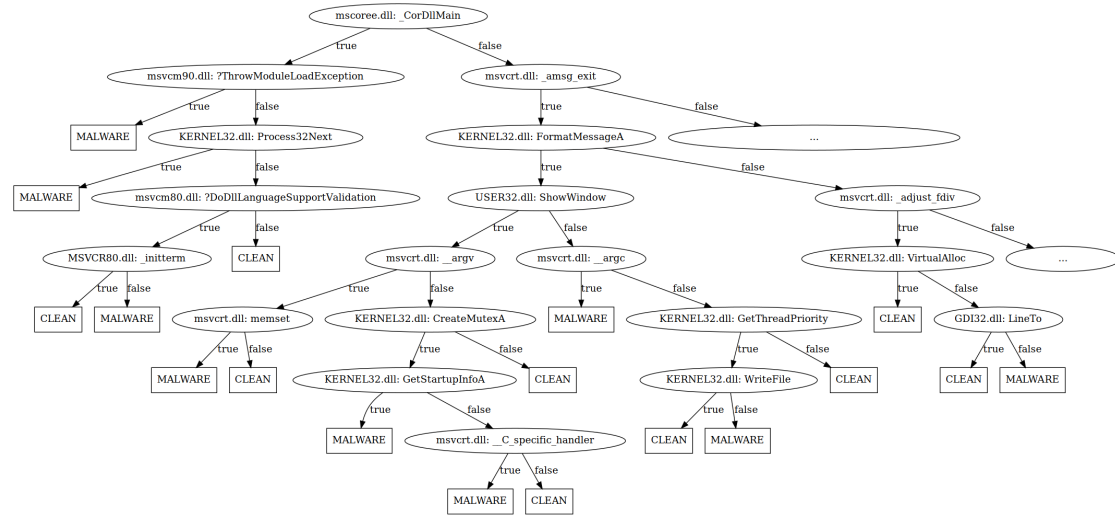
B.2   C4.5 Tree - Function Import Heuristics

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] *Dynamic Host Configuration Protocol: Request for Comments 2131.* URL: `https://www.ietf.org/rfc/rfc2131.txt`.

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

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

[5] *Rapid7 Vulnerability & Exploit Database: CVE-2017-0144.* URL: `https://www.rapid7.com/db/vulnerabilities/msft-cve-2017-0144`.

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

[7] *VirusShare.* URL: `https://virusshare.com/`.

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

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