A MALWARE DETECTION SYSTEM USING DOMAIN NAME INFORMATION

PASCAL BOUWERS

Software Engineering and Distributed Systems
Faculty of Mathematics and Natural Sciences
University of Groningen

November 17, 2015 version 1.1

Pascal Bouwers: *A malware detection system using domain name information,* © November 17, 2015

SUPERVISORS:
Dr. Alexander Lazovik
Dr. Rein Smedinga
Ad Buckens

ABSTRACT

It is a continuous challenge for companies to detect malware infected clients in their network. Cyber-attacks are a constantly growing threat for companies, especially ones that have valuable and critical information that they need to keep confidential. This confidentiality can be breached by malware infected clients in their network, which can lead to both financial as well as reputational damage to the company.
In this work we present a network-based malware detection system that is able to detect malware infections inside a network by logging the DNS requests and responses that leave and enter the network. This DNS traffic is used to classify requested domain names as either legitimate or malicious. This allows for the detection of malware infections within a network by identifying hosts that create DNS requests for malicious domain names.
The presented system is able to correctly classify 93,66% of the domain names from a test set as either legitimate or malicious. This test set consists of 250.000 legitimate domain names and 250.000 malicious domain names.

ACKNOWLEDGMENTS

I would like to thank Ad Buckens and John Zuidweg for giving me the opportunity to join their information security team at EY as a thesis intern.
I thank my fellow colleagues at EY for the discussions that we have had and always being available to answer my questions.
Finally, I would like to express my gratitude towards my supervisors Dr. Alexander Lazovik and Dr. Rein Smedinga for their support and guidance.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ASCII	American Standard Code for Information Interchange
APT	Advanced Persistent Threat
AV	Antivirus
BYOD	Bring Your Own Device
CIA	Confidentiality Integrity Availability
DDoS	Distributed Denial-of-Service
DGA	Domain Generation Algorithm
DMZ	Demilitarized Zone
DNS	Domain Name System
EY	Ernst & Young
FN	False Negative
FNR	False Negative Rate
FP	False Positive
FPR	False Positive Rate
HIDS	Host-based Intrusion Detection System
HTTP	Hypertext Transfer Protocol
IDS	Intrusion Detection System
IP	Internet Protocol
IRC	Internet Relay Chat
LAN	Local Area Network
MRQ	Main Research Question
NIDS	Network-based Intrusion Detection System

OS	Operating System
PSL	Public Suffix List
P2P	Peer-to-Peer
RQ	Research Question
SVM	Support Vector Machine
TCP	Transmission Control Protocol
TLD	Top-Level Domain
TTL	Time To Live
USD	United States Dollar
VPS	Virtual Private Server
WAN	Wide Area Network

1 INTRODUCTION

The damage caused by cybercrime globally runs into the billions of USD annually. [4] Malware plays an important role in cybercrime. Because of this, companies spend large amounts of resources on security software and appliances every year. This includes web- and email filtering, firewalls, antivirus software (AV), intrusion detection systems, and other similar products. Most of a company' cyber security budget will be allocated to such products. These products are built into enterprise networks in order to protect the network from malware entering it. All these cyber security products are unfortunately not enough to prevent all malware attacks. According to AV Comparatives, the best antivirus software is able to detect 98.8% of all the known malware. [7] A different study on the effectiveness of antivirus software showed that the initial detection rate of a newly created malware is less than 5%. [31] In the current world, companies do get infected with malware. It is not a realistic goal to prevent 100% of all malware infections.

Malware (which stands for *malicious software*) is any software used to disrupt the normal operation of a computer system, to gather (confidential) information, or to gain access to a computer system. Software is considered to be malware by its intent to be malicious. Software that unintentionally harms a computer because of an error, flaw, failure, or bug is not considered to be malware.

The term malware is used for a large group of software, including viruses, worms, Trojan horses, rootkits, spyware, bots, adware, and other malicious software. The different types of malware will be discussed with more detail in Section 2.1.1. The goal of malware is always to compromise the key principles of a secure system, the CIA tried of Confidentiality, Integrity, and Availability. [22]

Recently a specific type of malware has emerged, which is a major threat to many organizations. This type of malware is more complex and long-lived, and usually has very specific goals such as stealing data rather than causing damage to an organization. This is often referred to as an Advanced Persistent Threat (APT). What is unique about APTs is that they specifically target business and government organizations. The name suggests that the malware uses advanced techniques to exploit systems and that the threat is persistent, extracting data from the victim and exchanging information with the attacker over a longer period of time.

For this project we would like to develop a network based malware detection system that is able to detect malware inside a network by logging the DNS requests and responses that leave and enter the network. We will use this DNS information to classify requested domain names from within the network as either malicious or legitimate. This allows us to find malware infections within a network by identifying hosts that generate DNS requests for malicious domain names.

1.1 PROBLEM CONTEXT

EY performs security audits for some of their clients. Since recently many of those clients request information on the likelihood that there are malware infections inside their corporate network. It is a continuous challenge for companies to detect malware and infected clients within their network. Cyber-attacks are a constantly growing threat for companies, especially ones that have valuable and critical information that they need to keep confidential. This confidentiality can be breached by malware infected clients inside their network, which can lead to both financial as well as reputational damage to the company.

Many big and complex software solutions exist that are able to detect and analyze malware. However, most of those are not easy to use and deploy, are intrusive, and are sometimes not suitable to provide malware detection on the level of a complete network. To solve this problem we want to develop a solution that is able to give an indication on the possibility of malware infections inside an enterprise network. Unlike most existing software, this solution can only be used in the short time frame that EY has physical access to a client' network when performing other tests, and should thus be able to provide results within a short period of time. The quick overall indication that this solution provides will be used to determine if it is necessary to conduct further research using more heavyweight and expensive antimalware software.

1.2 OBJECTIVES AND CONTRIBUTIONS

The goal of this project is to develop a classification system that is able to give an indication of possible malware infection on systems within an enterprise network. This is not a polar question, but the solution should be able to indicate that there might be malware on a system inside the network, in which case further analysis is necessary.

The key objectives of this project are:

1. The solution should require minimal effort to install and remove.
2. The solution must provide results in a short amount of time. Security auditing is usually done in a single workweek (five days), so preferably the system should deploy, collect data, classify, and be uninstalled within that time frame.
3. The system should be able to detect unknown malware.
4. The solution should be non-intrusive, able to perform its tasks while causing minimal or (preferably) no overhead and interference on the systems and the network itself.

The global contributions of this project are:

1. An overview of possible approaches to this problem, and the selection of the best approach for our goals and requirements.
2. An overview of existing software and research in the context of malware detection in corporate networks.
3. The software design of our solution.
4. A working proof of concept of our solution.
5. Tests and results of our solution.

1.3 RESEARCH FOCUS

The main objective will be to create a non-intrusive, lightweight system that is able to detect a malware infection. We define a system as non-intrusive, when it has no (or very minimal) interference and overhead on the existing systems and network. As was already explained in the introduction paragraph we want to achieve this goal using DNS domain name information. We will do a thorough background study on malware (detection), existing approaches to this problem and state of the art solutions in order to achieve our goals.
Our general main research question (MRQ) can be defined as:

MRQ1 *How can we design a non-intrusive, lightweight and practical (classification) system that is able to detect infections of unknown malware within a network?*

RQ1.1 *Is a host-based or network-based approach the best for our problem and objectives?*

RQ1.2 *Which classification features can we use to detect malware with our approach?*

RQ1.1 and RQ1.2 focus on sub-research questions (RQ) that ultimately lead to an answer to MRQ1. The other main research question focuses on the features of our classification. This question is concerned with the performance and detection rate of our system.

MRQ2 *How do the classification features we have selected for our classification system perform?*

- RQ2.1 *How do the individual features we have selected perform?*
- RQ2.2 *How do combinations of our selected features perform?*
- RQ2.3 *What results can different classification algorithms achieve with our features?*

RQ2.1, RQ2.1 and RQ2.3 are related to the tests of our system and give insight in the performance of our features and classification algorithms. The results of these research questions answer MRQ2 and allow us to find the features, feature combinations and classification algorithm that result in the best detection rate.

1.4 THE PROCESS

In order to answer the research questions that we have defined, we can divide this project into several phases.

INITIAL PHASE In this preparation phase we create the problem definition and context of the project. This phase results in the project proposal.

FIRST PHASE In this phase a literature and background study is performed. The main goals of this phase are to get a better understanding of malware (detection) and different types and approaches to malware detection. In essence, this phase provides the knowledge to be able to make an educated decision about the approach that we are going to use for our system. This will effectively answer RQ1.1.

SECOND PHASE In this phase we will research existing software and related work that use the same network-based approach as us. Based on this we will research features that we can use for our classification system, based on existing literature. The goal of this phase is to reduce the scope of the project and identify the information that we can use for our system. Because we already know that we are going to use DNS information, this phase will be the rationale behind why we will use DNS domain name information for our system. This will answer RQ1.1.

THIRD PHASE In this phase we will focus on the actual technical design including the selection of features and implementation of our system. The system will be designed using the approach selected in the first phase, using the information and scope selected in the second phase. The system will be designed and developed according to requirements that match the objectives of our project. The creation of the design will answer RQ1.2 and MRQ1.

FOURTH PHASE This phase focuses on creating a dataset, defining tests to measure the performance of our features, and presenting the results. The tests will be performed using the system developed in the third phase. The tests will be designed so that they will answer RQ2.1, RQ2.2 and RQ2.3.

FIFTH PHASE In this phase we will focus on future work and the final conclusion based on our test results from the fourth phase. The final conclusion will answer MRQ2.

Table 1 shows an overview of the output and deliverable for each phase.

PHASE #	OUTPUT
0	Project proposal
1	Select best approach for our project
2	Scope definition
3	Design and developed software
4	Results
5	Final document

Table 1: Overview of project output for each phase.

1.5 OVERVIEW

This document contains the following chapters (excluding the current introduction chapter).

BACKGROUND This chapter contains background information about malware and the detection of malware. The chapter will specifically focus on malware types, malware communication methods, malware detection techniques and technologies. This chapter will conclude with the selection of the best detection approach for our system.

STATE OF THE ART This chapter focuses on related work that uses the same approach as us. This chapter will conclude with the selection of the scope and data that we will use within our system.

THE SYSTEM This chapter describes the design of our system. We will specifically focus on the selection of features, design goals, requirements, decisions, and technical challenges.

RESULTS This chapter describes the tests we are going to perform, the dataset we have created and the results of the tests.

DISCUSSION & CONCLUSION This chapter presents the conclusion and discussion on our results, in addition to our thoughts on future work.

2 BACKGROUND

In this chapter we will provide more detailed background information on malware (communication) and malware detection methods and techniques.
Section 2.1 will provide an overview of different types of malware, their functionality, and how malware communicates with an attacker.
Section 2.2 will provide information on the most relevant approaches to malware detection by discussing different malware detection techniques and technologies. At the end of the section, an overview is given of typical (malware) protection layers that are used within an enterprise network.
This chapter will conclude with Section 2.3 where we select the best malware detection approach for our project. We will also provide the rationale behind this decision based on the research done that is described in this chapter.

2.1 MALWARE

The introduction chapter already gave a brief explanation of what malware is. In this section we will provide more detailed information on malware.

2.1.1 *Different types of malware*

There exist several different types of malware. This section will give an overview of the most popular types of malware, including a small overview of their behavior and their objectives. It is important to keep in mind that not all malware can be put in a single group; often malware will contain functionality from several different classes.

VIRUS A virus is a type of malware that can copy itself and infect additional computers. An extended definition by Microsoft is: *"A virus is a piece of code that attaches itself to other programs or files. When these files run, the code is invoked and begins replicating itself."* [21] In addition to infecting local files, viruses also try to infect remote computers. Viruses (and also worms) usually spread by infecting every vulnerable host they can find.

WORM Just like viruses, worms are also able to run by themselves. The main characteristic of a worm is its ability to self-replicate across network connections. Unlike viruses, worms do not attack themselves to other programs (although they might carry code that does). They have other ways to replicate themselves. Spafford defines a worm as *"a program that can run independently and can propagate a fully working version of itself to the other machine."* [26]

TROJAN HORSE Trojan horses are known for pretending to be useful, legitimate software, while simultaneously performing malicious actions without the user' knowledge. Often a Trojan horse will run the legitimate software while the malicious part will download additional malware in the background, infecting local files, or perform other malicious software.

ROOTKIT The main characteristic of a rootkit is its ability to hide itself and other malicious software from the user and other processes. A rootkit does not have to be malicious by definition, but its stealthy functionality is especially interesting for malware creators who want to hide their malicious software on a system. Because of this rootkits are often paired with other types of malware.

SPYWARE Spyware is a type of malware that collects (sensitive) information from a computer and usually transfers this information back to the attacker. Most spyware focuses on stealing information such as bank account credentials and credit card information.

BOT A bot is a type of malware that allows the attacker to remotely control the infected system. A computer infected with a bot is usually part of a big network of other bots (called a botnet). The attacker that controls this botnet is able to send an instruction to all computers in the botnet, making it especially useful for malicious purposes such as sending spam emails or performing DDoS attacks.

2.1.2 *Command-and-control communication*

Almost all modern malware use some networking capabilities to exchange information with the attacker. Specifically malware such as bots, APTs, and other long-lived attacks require information and instructions from the attacker in order to be successful. This long-lived control requires some form of a communication channel, which can be used to send commands and results. Such a communication channel is called a command-and-control (C&C) channel. As explained in the introduction, the goal of APTs is often to steal data from an organization. Because of this, sometimes additional channels exist to send stolen data from the victim to the attacker.

The C&C channel that the malware uses depends on the design of the malware. The malware designer has to determine how often he will need to communicate with the malware. Malware often infects systems that are inside a network with an unknown network structure to the attacker. Because of this, a lot of malware will need some instructions from the attacker. The attacker effectively has remote user level access to the system when it is able to communicate with the malware. The C&C channel allows the remote attacker to execute the same commands as a regular user of the system and also receive the results of those commands.

Figure 1 shows an overview of the usual communication phases of malware.

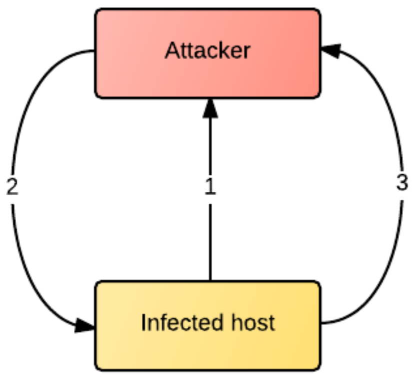


Figure 1: Typical malware communication phases.

C&C channels in malware usually follow this same traffic pattern:

1. Register a new infection to the attacker.
2. Send commands to the malware.
3. Send results from executed commands to the attacker.

PHASE 1 The moment when the APT will make its first call home to the attacker. The malware will usually send some additional information to the attacker, such as network and operating system information.

PHASE 2 Phase 2 is when the attacker sends instructions to the malware. There are usually three different types of instructions that the attacker sends to the malware:

EXECUTE This will execute a command on the system. This can be an operating system command, a script or a custom command from the malware.

DOWNLOAD This allows the attacker to instruct the malware to download additional files to the infected system. These files are usually tools that are used to perform additional attacks on the system or updates of the malware.

UPLOAD Used by the attacker to instruct the APT to upload a file from the infected system.

PHASE 3 During this phase the malware sends information is has gathered back to the attacker. Usually this is the result from the executed commands from the previous phase. When this phase is finished, the malware will go back to phase 2 where it will wait for new commands from the attacker.

The C&C channel is a very important aspect of the malware. The commands described above are able to give the attacker full control over an infected system.

2.1.2.1 *Methods*

From a high-level perspective malware communication can use two methods:

- pull
- push

The difference between these two methods is only in how they deliver commands to the infected systems, not in the commands and information that can be sent and delivered.

PUSH Malware that uses the push model will make a direct connection to the attacker. This allows the attacker to send commands directly to the malware. This connection is usually a TCP connection to a C&C server and is used to complete the three phases described in the previous section. The communication happens in real time, there is no extra delay between sending and receiving a command or result.
The network communication typically uses the IRC-protocol, where the malware joins a channel on an IRC server that is controlled by the attacker. This method is called the push method, because the attacker directly pushes commands to the malware.

PULL With the pull model the communication is initiated by the malware. When an attacker wants the malware to execute a command, it drops the command off on a hub. The location of the hub is determined in advance. The malware will query this hub to see if there are new commands that it needs to execute. If there are new commands, the malware will execute it and post the results to the hub. The attacker will periodically check the hub to retrieve the results.

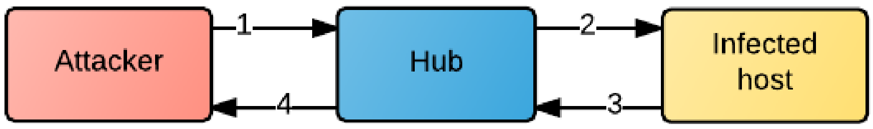


Figure 2: The pull communication model.

Figure 2 shows an overview of the pull model.

1. The attacker leaves a command at the hub.
2. The infected host periodically retrieves new commands from the hub.
3. The infected host leaves the result on the hub.
4. The attacker periodically retrieves new results from the hub.

Often HTTP is used with the pull model. The malware executes GET requests on a website that the attacker controls in order to collect new commands that the attacker would like the malware to execute. The malware will then execute those commands and use a POST request to submit the results of those commands back to the website.
Due to how this method works, the malware will need to periodically check for new commands to run, which means that there will always be a delay between receiving, executing and sending the results from a command. This method is called the pull method because the malware periodically polls a C&C server for new commands.

AUTONOMOUS MALWARE The most sophisticated and specifically targeted malware can be completely autonomous. This means that it does not need to communicate over the network with an attacker in order to function properly. This malware is usually used on a specific network that is not accessible from the internet. The Stuxnet worm is a very good example of such a malware. While it was able to communicate over HTTP, the main method of infection was with USB sticks, using a zero-day exploit. Because this is very sophisticated and highly targeted malware, we will not consider this type of malware. We will only focus on malware that uses some form of network communication, using either the pull or push method. Because this sophisticated malware is usually written to target a specific organization, it is not realistic to detect this type of malware using general malware detection systems.

THE C&C SERVER The C&C server architecture contains one or multiple hosts and plays an important role. These servers are controlled by the attacker and are used to coordinate the malware. Attackers will usually use multiple servers for redundancy purposes. These servers are often hosts which have been compromised by the attacker, allowing him to use it as C&C server. Some malware uses a P2P approach instead of centralized servers. P2P communication will be discussed with more detail in Section 2.1.2.3.

2.1.2.2 *DNS*

The most important protocol that most malware uses is the DNS protocol. Unless the IP addresses of its C&C servers are hardcoded into the malware, DNS is needed in order to find those IP addresses using their DNS address records. Malware almost never uses hardcoded IP addresses because they are easy to extract from the malware, making it trivial to block those IP addresses in a firewall, rendering the malware useless.
DNS gives the attacker flexibility. It allows the attacker to change the locations of the C&C servers without having to update or notify the malware. This is very useful for an attacker, because the location of C&C servers changes often because they get discovered and taken offline regularly.

FAST FLUX Fast flux is a technique where multiple IP addresses are associated with a single domain name. DNS records are changed very often so that the DNS record of a domain name directs to different IP addresses every certain amount of time. This makes it much harder to block the C&C traffic because the C&C servers are automatically changed so often. Using DNS queries the malware is notified of these changes. There is also a legitimate technique called round-robin DNS that works similar to fast flux with similar characteristics and is used by many popular websites as a form of load-balancing.

DGA In the past few years some malware began using a more advanced approach, where the malware will generate a domain name that resolves to the IP address of a C&C server. These domain names are generated using a Domain Generation Algorithms (DGA), also sometimes called domain fluxing. The attacker uses an algorithm to create a large list of domain names each day. One of these domain names will actually be registered, and used to point to the C&C server. The malware software that is installed on an infected machine will use the same algorithm with the same parameters to generate an identical list. The malware will connect to the domain names in the list until it finds the actual registered domain (the C&C server), or until a certain maximum has been reached. This is usually done on a daily basis, using for example the date as parameter for the algorithm so that each day different lists are generated. This technique allows the attacker to greatly reduce the amount of time they expose the C&C server via DNS, making it harder to apply countermeasures in time.

After the malware is able to locate the C&C server, the malware will attempt to contact it using one of the protocols that will be discussed in Section 2.1.2.3.

2.1.2.3 *Other protocols*

The protocols discussed in this section are often used by malware. In the previous section the DNS protocol was discussed. DNS is used to find the attacker; this section will focus on protocols that are used for the actual C&C communication.

IRC Originally designed as chat room software, the IRC protocol has been popular for malware C&C communication for a long time. IRC provides a simple interface for an attacker to control infected systems using a chat room style interface. IRC supports private communication between users, or group chat functionality within IRC channels. In addition, it provides administration functionality by allowing the owner to control who can access their channel.
IRC provides several mechanisms that make it very interesting for malware purposes. First, by using password protection on a channel, the attacker can restrict access for unauthorized users to the IRC channel that is used to control the infected machines. Second, IRC channels can send messages to everyone inside the channel, which allows the attacker to send the same messages to many different infected machines at once. Lastly, the private communication between users allows the attacker to send specific commands to specific infected hosts, and receive individual messages from specific hosts.
IRC is one of the easiest protocols for C&C communication. To create a connection with a C&C server, the infected system would have to be part of a network that does very little egress filtering. When the infected system is inside a network that blocks outbound traffic on default IRC ports, the malware needs to use other ports that are less likely to be blocked, such as ports 80 and 443 which are often used for outbound web traffic.

P2P P2P is a distributed architecture with hosts directly connected to other hosts. Data is transferred directly between these hosts, and the need for centralized coordination is almost non-existent. P2P is used to connect hosts together in order to create a private network that runs on top of an existing network, such as the internet.
Specifically bots have recently taken advantage of P2P networking over a more centralized approach. The biggest advantage is that no more C&C servers are needed, who can prove to be a single point of failure. A P2P network is much more robust and much harder to take offline.
Although this document focuses on enterprise network, malware that employs P2P architecture almost exclusively targets home networks.

The reason for this that P2P architecture relies on poor inbound and outbound filtering and using protocols such as UPnP to forward Internet exposed ports to infected systems. These circumstances almost only exist on home networks.
While current generations of P2P malware appear to only target home networks, this can very easily chance in the future. To adapt to enterprise networks, it is likely that P2P malware will switch from custom binary protocols to use some other communication protocol to bypass firewall and other cyber security products.

HTTP(S) HTTP is together with DNS the most interesting protocol in the context of enterprise networks. The previous protocols that we discussed (except DNS) are usually blocked in a tightly controlled environment such as an enterprise network. However, outbound web traffic is usually still allowed either directly or via a web proxy. When this is the case, HTTP can be used for C&C communication by malware. HTTP is the protocol most commonly used by web browsers, and is a simple text-based protocol that allows variable length for requests and responses.
When HTTP is used by malware, a GET request is usually used to retrieve commands from a C&C webserver, and a POST response is used to send the results to the webserver. Unlike the previously discussed protocols, HTTP uses the pull method for communication. The malware checks the C&C webserver periodically for new commands to execute.
HTTP provides flexibility for the attacker in where they store their commands and results. Obvious locations would be the GET and POST parameters, but there are some headers that can be used to carry data which might not attract immediate attention when analyzing network traffic. When wanting to use textual fields in an HTTP request or response, the malware will typically have to encode files or data into ASCII characters which are suitable for use with HTTP.
In order to prevent deep packet inspection, the malware can use HTTPS to encrypt all traffic between the malware and C&C server.

2.2 MALWARE DETECTION

A program that can detect malware Is able to analyze a set of programs, and can find whether is it is a clean normal program, or malware. Saeed et al. [25] created a more formal definition of a malware detection program, where the malware detection program D is a computational function. This program D works in the domain that contains a collection of programs P, and a collection of malware and clean programs. The malware detection program D analyses program p which is part of P, to find if it is a normal program or malware. They created the following definition:

$$D(p) = \begin{cases} \mathtt{malicious} & \text{if } p \text{ contains malicious code} \\ \mathtt{clean} & \text{otherwise} \end{cases}$$

This definition can be extended to account for false positives (FPs), false negatives (FNs), and undecided results depending on the function D. This can be rewritten as

$$D(p) = \begin{cases} \mathtt{malicious} & \text{if } p \text{ contains malicious code} \\ \mathtt{clean} & \text{if } p \text{ is a normal program} \\ \mathtt{undecided} & \text{if } D \text{ fails to determine } p \end{cases}$$

FP means that malware is detected even though the program is clean, FN means that actual malware is detected as a normal program, and undecided means that classification fails for unknown malware. Developers of malware detection systems track new programs, analyze them, and categorize them by putting them in a clean (white) or malicious (black) list. The undecided (gray) list contains programs that have not been classified, and are usually put in a controlled environment (such as a sandbox) for further analysis and classification.

2.2.1 *Techniques*

A malware detection system always uses a certain technique and a certain technology.

SIGNATURE-BASED Anti-malware products that use signature-based techniques identify malware by comparing a potential malware program with a database of known malware signatures. The main advantage of this technique is that it can be very effective and fast. The main disadvantage is malware will not be detected if it is unknown, or its signature is not in the database for some other reason. Because it can only detect malware that has actually been identified before, it has a very low false positive rate (FPR). However, because it can only detect malware that has been identified before it has a high false negative rate (FNR).

ANOMALY-BASED Anomaly-based systems work by detecting any kind of activity that is not part of the ordinary activity of a system. An anomaly-based detection program keeps track of a system' activities and classifies it either as normal or anomalous behavior. The difference of this technique compared to signature-based is that it detects malware based on classification instead of patterns. The main advantage of this technique is that it can detect unknown malware, because it does not rely on a database of already identified malware but rather on the detection of malicious behavior. This result in a lower FNR compared to signature-based detection. However, the biggest disadvantage is that it has a higher FPR.

2.2.2 *Technologies*

Several technologies exist to detect malware. Some of them reside on the host it protects, some of them outside of the host. All detection technologies use either a static, dynamic or hybrid methods based on signature or anomaly-based techniques. This paragraph will discuss these technologies.

INTRUSION DETECTION SYSTEM An intrusion detection system (IDS) monitors a network traffic or system activities, depending on the type of IDS. The IDS is usually a software application or dedicated device. There are several different types of IDSs, but they can be categorized as network-based and host-based detection systems. The goal of an IDS is not necessarily to stop an attack (although some do), but rather to detect and report an intrusion. In a passive system, the IDS will only detect, log and notify when an intrusion has been detected. In an active system, the IDS will actively try to prevent the intrusion. IDSs can use both signature-based detection as well as anomaly-based detection.

HOST-BASED IDS A host-based intrusion detection system (HIDS) monitors and analyzes the internal state and dynamic behavior of a computer system to detect if there are any internal or external malicious activities or processes that affect the computer system. Because this variant of an IDS is host-based, it operates on the same host that it monitors. An example of an HIDS would be an antivirus scanner.

NETWORK-BASED IDS A network-based intrusion detection system (NIDS) monitors the network nodes, analyzing network packets that pass them. Unlike host-based systems, NIDSs are only installed at specific points inside a network, such as systems that interface between the outside environment and the network segment that needs to be protected. Because this variant is network-based, it only operates outside of the hosts.

HYBRID IDS There is also a third type of IDS that uses a hybrid approach, meaning that both network-based as well as host-based capabilities are used. A hybrid IDS usually consists of nodes that collect network data, in addition to software installed on local machines. The data from all sources is then send and collected at a main system, after which it will be analyzed and classified. A hybrid IDS can be effective, because while host-based systems are aimed at protecting internal systems, they can be susceptible to external attacks. The opposite is true for network-based systems that protect against external attacks, but are not able to provide protection inside a local host.

2.2.3 *In an enterprise network*

The network of an organization is called an enterprise network. An enterprise is a computer network that is only used by organizations. An enterprise network usually includes a large diversity of computers and systems, such as desktop computers, servers, and mobile (BYOD) devices. The sizes of enterprises can vary greatly. An example of a very small enterprise network would be a small (home) office of 1 to 10 persons consisting of only a few desktop computers inside a LAN. At the other end of the spectrum would be an enterprise network consisting of multiple LANs (possibly over different locations) and connected by WANs as private (virtual) networks.
As opposed to enterprise networks, public networks are networks that provide connectivity to home networks and mobile devices. A home network usually consists of desktop computers, networked printers and televisions, tablets, laptops, and other mobile devices.
One of the largest differences between public and enterprise networks is that enterprise networks have systems in place that control the communication. Enterprise networks are separated from public networks by firewalls. Within an enterprise network, the systems are put into groups with different (communication) policies. For example, a webserver that needs to be publicly accessible from the internet would reside inside a DMZ.
Sometimes small networks have characteristics from both enterprise and home networks. When defining the problem we will use the basic definition of an enterprise network (in contrast with the public networks) that was discussed in this section.

2.2.3.1 *Malware protection in an enterprise network*

A typical enterprise network usually has multiple layers that protect the network and its systems from malware. In this section we give an overview of these protections in order to understand the bigger picture, and to see how it all fits together.
The communication model malware uses depend on the network architecture of the target host. How this architecture looks depends on the type, size and budget of the organization. If we take the small (home) office example again, it would likely have simple network architecture with a firewall and possibly an antivirus product on the desktop computers. These types of networks are usually designed with the goal of making sure that the user can do his work, valuing functionality over security.

In contrast, large organizations are much more likely to have a specific budget for IT and a much more clear and defined strategy for security. In addition, they usually have networks separated by physical location, where functional components are separated from each other. Large organizations are also more likely to have multiple defensive layers, filter network traffic and perform network monitoring. Network architecture limit and control the type of network that can enter and leave the network. This of course affects the choice for a C&C channel.

FIRST LAYER Firewalls placed between the outside environment and the inner networks are usually the first layer of defense in most companies. They protect the networks of an organization from an outside attack. However, not all organizations filter outgoing traffic that leaves the network. This is especially true for smaller organizations that often prioritize user functionality.

SECOND LAYER Antivirus products are usually the second layer of defense in most organizations. A way to prevent the spread of a known malware inside an organization that is often used is to scan e-mail attachments before it gets delivered. However, many file types such as documents that are usually permitted can also contain malicious code. In addition, most antivirus products are only as effective as their signature databases.

THIRD LAYER The third layer of defense is usually a web proxy. Originally used to cache webpages, they are now often used to provide content filtering to control what content can be accessed by a user. In addition, web proxies often implement antivirus technology that allows the proxy to scan downloadable content for malicious code. This way, malicious downloads can be intercepted before they reach the user′ computer.

FOURTH LAYER The fourth layer of defense is on the desktop computer of a user. This layer is the configuration of the computer and any locally installed security software, usually antivirus products and other HIDSs. In a sense, this is the last layer of defense because its goal is to catch anything that made it to the actual host, making it past any of the previous layers.

FIFTH LAYER The final layer of defense is the NIDS. These systems use signatures to detect malicious content in network traffic flows. Just like most antivirus products, if the malware is unknown, there will be no signatures. Because of this, sometimes heuristic modeling is built into some IDSs, but this will always increase the FPR.

2.3 OUR APPROACH

In this section we will discuss the technique and technology we have selected to use for our malware detection system. We have decided to create a network-based system, using an anomaly-based detection technique. The following paragraphs will discuss the rationale behind this decision.

TECHNIQUE The only detection technique that is suitable for our goals and objectives is a malware detection system that uses an anomaly-based technique. An anomaly-based system allows us to detect unknown malware, unlike a signature-based system. Because detecting unknown malware is one of the main objectives of this project, we have no choice but to use this technique.

TECHNOLOGY The best malware detection technology depends on the goal and (network) environment. There are several reasons why a network-based approach is superior in our situation. The goal of this project is to give an indication of malware infection on one (or multiple) systems within an entire enterprise network. When using a host-based detection approach we would have to install detection software on every single system inside the network in order to find malware inside the network.

There are several reasons why this approach is not ideal for our goals and objectives:

1. It is impractical. We would have to install software on possibly every single system inside the network. After running the test we would have to uninstall everything again.
2. It is very time consuming. We have a five day time window to perform our tests, this includes installing and uninstalling anything we need in order to run the tests. For a large network this would take a lot of time if we would have to do this per single system.
3. It slows down the systems. It is very possible that a host-based detection system will be resource intensive and slow down the systems.
4. It invades privacy. Our solution would run on local systems and have access to any files on the system.

However, a network-based approach would suit our case much better. It would be able to solve all of the problems mentioned above and more:

1. It is not impractical. We could do the detection on a single point inside the network, most likely the point where the enterprise network connects to the internet. All C&C communication would pass that point making it the perfect place to do our detection.

2. It is easy to set up. We would only have to install and uninstall a device at one point in the network.

3. It allows us to scan different types of systems. With a host-based approach we would have to develop software for a specific platform. For example, if we would develop a host-based detection system for Windows we would be able to detect malware on all the Windows systems inside the enterprise network. However, in this situation we would not be able to detect malware on systems in the network that runs on a UNIX-based OS. This would conflict with our goal of giving an indication of malware infection of the complete network. Because a network-based detection approach is platform independent and analyses all traffic that leaves the network, we would be able to detect malware on any system, even smartphones that connect through the network.

4. It does not slow down the systems or network. Many routers implement functionality that allow third party developers to passively extract network traffic information from the router, without having any effect on the performance of the network. This allows us to potentially have zero impact on the performance of the network.

5. It is non-intrusive. There are multiple ways to do network traffic analysis. When we only work with network flow information and do not inspect the actual packets, we would not invade the privacy of the network users.

The reasons mentioned in this section make it very clear why a network-based detection system would be much better than a host-based approach. This effectively answers RQ1.1. In the next chapter where we will discuss research and existing software, we will only focus on related work that uses a network-based system and an anomaly-based detection technique. Other works are not relevant for us.

3 STATE OF THE ART

In this chapter we will discuss the state of the art approaches and research. The focus will be on research that – like us - uses a network-based approach using an anomaly-based detection method. Some of the research papers described in this chapter also have an implementation available as free and sometimes open source software. There are also many commercial software packages available, but we will not discuss those. The main reason for this is that it is very hard to find out how exactly some commercial software packages work, because most of the companies keep this a secret. In many cases it is only possible to get a general understanding of the techniques and technology used. In contrast, for the software that we discuss, the technique, technology, and inner workings are thoroughly discussed in papers.
We have tried to create a diverse selection of papers and ignore duplicate or very similar studies. This way we will get the best overview that shows different approaches to a network based system with anomaly based detection and gives a good understanding on the state of the art.
After discussing the state of the art, we will discuss the focus of our system, effectively narrowing down the scope for our design and project. This will be done in Section 3.2 of this chapter.

3.1 EXISTING WORK

In this section the existing work and solutions will be discussed. We will only focus on works that use a similar network-based approach as us.

3.1.1 *BotHunter [15]*

BotHunter detects C&C traffic using a monitoring strategy that focuses on the network traffic in the first phase (see Section 2.1.2) after a successful malware infection. Their approach uses an *evidence trail* as they call it that recognizes malware infections by the communication that occurs during the malware infection process. They call this the *infection dialog correlation strategy*. Malware infections are modeled as a set of loosely ordered communication flows that exist between the infected host inside a network, and one or more hosts on the outside that the attacker controls. They assume that all malware shares a common set of actions that occur during the infection phase: scanning

the target, downloading a binary file and running it, and establishing a C&C channel. They do not make the assumption that all malware will perform all these actions, or that their system will detect every single event, but rather that the system collects an evidence trail of all events that are related to infection. They keep track of this for every single host in the network, looking for a combination of sequences to see if it reaches a certain threshold. This threshold will determine if the system will recognize a host as infected.
The advantage of this approach is that it is able to detect unknown malware. The disadvantage is that the system is complex and needs to detect multiple infection related events before it can trigger an alert. When during a malware infection not all events are triggered or monitor the system will not detect the infection. This can be countered by loosening the model, but this will also increase the FPR.

3.1.2 *Detection of P2P C&C traffic [24]*

This approach focuses on detecting C&C communication that uses P2P architecture and tries to detect the malware before it attacks. They specifically focus on the C&C communication that occurs outside of the initial setup phase. They use a total of 17 different features that are taken from the network traffic, such as the source and destination IPs and ports of traffic flows.

Multiple machine learning classification techniques were used to investigate the possibility of detecting malicious traffic by analyzing network traffic behaviors only.

The machine learning techniques used in their work are the following:

- Nearest Neighbors classifier
- Linear Support Vector Machine
- Artificial Neural Network
- Gaussian Based classifier
- Naïve Bayes classifier

They have achieved a true detection rate above 90% for the Support Vector Machine, Artificial Neural Network and the Nearest Neighbors classifier with a total error rate of less than 7%. The true detection rate is above 88% for the Gaussian Based classifier and the Naïve Bayes classifier, but with a total error above 10%. These results indicate that it is possible to identify P2P C&C communication by only analyzing traffic behavior, thus without inspecting the content of packages.

The authors concluded that based on their comparison of five machine learning techniques, and the metrics we discussed, that the Support Vector Machine, Artificial Neural Network and Nearest Neighbor classifier are the top three machine learning techniques (among the five) that can be used to build a C&C detection framework.

3.1.3 *Detection of C&C traffic by temporal persistence [14]*

This approach focuses on detecting C&C traffic and the individual end-hosts that the malware connects to. They introduce something called *destination traffic atoms* which combines the locations outside of the network that are visited. They then compute the persistence, which is a measure of how often the destination atoms are visited. In short, they keep track how often a destination outside of the network is visited.
During the training phase, the destination atoms that are very persistent are added to a host' whitelist. They then track the persistence of new destination atoms that are not on the whitelist, to identify suspicious C&C traffic destinations. Their method does not require any prior information about destinations, ports, or protocols used for C&C communication, in addition to not needing to inspect the payload of the traffic.
The system is evaluated using user traffic collected from an enterprise network and overlaying the collected malware traces. They demonstrate that their method can correctly identify C&C traffic, even when it is very stealthy. In addition, filtering outgoing traffic using the whitelist constructed during the training phase will also greatly improve the performance of more traditional anomaly detectors. They also show that the C&C detection can be achieved with a very low FPR.

3.1.4 *DNS anomaly detection [30]*

This approach uses two DNS-based methods for identifying malware C&C servers based on DNS traffic. The first method looks at exceptionally high DNS query rates for specific domain names. High DNS query rates might indicate malware infection; because malware often uses a domain name that changes the location frequently, because attackers often change the location of their C&C servers. This technique used by malware called fast fluxing has been explained with more detail in Section 2.1.2.2. The second method of this approach consists in looking for a high number of These replies are sent from a DNS server when a DNS query can not be resolved because it does not match an existing domain name. These queries can correspond to malware trying to locate their C&C servers that have been taken down.

Their experiments show that the first method results in a very high FPR. The main reason for this is that many popular domain names (such as google.com) have a very low time to live (TTL) value. This means that the record for this domain in the DNS cache expires quickly and will have to be required at the DNS server again. This results in a spike of valid and legitimate DNS queries for popular domains, which by their method will be classified as malware. This explains the high FPR. However, the second method was very effective. Almost all of the detected domain names were also reported to be suspicious by others.

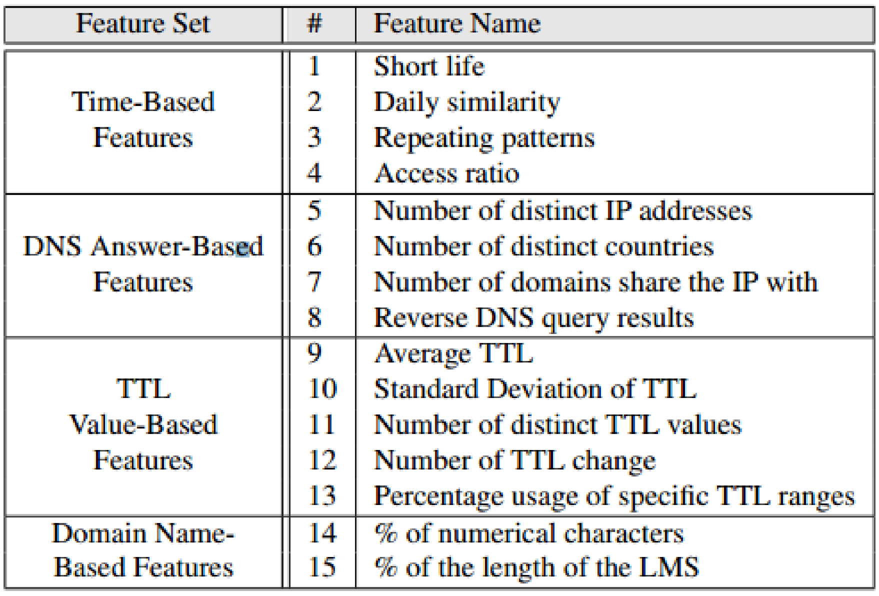
3.1.5 *C&C detection based on network behavior [28]*

This approach uses network flow characteristics such as bandwidth, packet timing, and burst duration to detect C&C communication. Traffic that is most likely not part of C&C activities is first removed. The remaining traffic is then classified into a group as potentially malicious. It then correlates the potentially malicious traffic to find common communication patterns that could suggest malware activity.
They used a total of 16 different features extracted from the network flows. They used several different machine learning techniques to classify the network into the two groups. They concluded that malware evidence can be extracted from a dataset that contains over 1.300.000 network flows.

3.1.6 *EXPOSURE [2]*

EXPOSURE is a system that uses DNS analysis techniques to detect domain names that are involved in malicious activity. They use a total of 15 different features - 9 of which were not previously used - that they extract from DNS traffic.

The features they present can be divided into four categories, as can be seen in Figure 3.



Feature Set	#	Feature Name
Time-Based Features	1	Short life
	2	Daily similarity
	3	Repeating patterns
	4	Access ratio
DNS Answer-Based Features	5	Number of distinct IP addresses
	6	Number of distinct countries
	7	Number of domains share the IP with
	8	Reverse DNS query results
TTL Value-Based Features	9	Average TTL
	10	Standard Deviation of TTL
	11	Number of distinct TTL values
	12	Number of TTL change
	13	Percentage usage of specific TTL ranges
Domain Name-Based Features	14	% of numerical characters
	15	% of the length of the LMS

Figure 3: Features used by EXPOSURE.

For their time-based features they used a very large dataset of actual DNS logs. These are general features that do not focus on a specific malicious DNS technique. The DNS answer-based features and TTL value-based features focus on detecting fast flux domains. Fast flux domains often have a very low TTL value in addition to many distinct IP addresses spread over several countries. The domain name-based features focus on detecting DGA.
They used a J48 decision tree for their classifier.

3.2 OUR WORK

Four our project we will focus on detecting malware using DNS information. As was explained in Section 2.1.2.2, almost all malware uses the DNS protocol, making it one of the most interesting protocols to focus on.
As shown in this chapter, most of the existing works that use a network-based approach focus on finding anomalies in network behavior. The largest downside of this is that in order to find those anomalies, a large dataset of existing malicious and benign network flows are needed. With regards to this project it makes more sense to focus on DNS information because datasets of malicious and legitimate domain names are much easier to find. In addition, almost all malware uses the DNS protocol. As was explained in Section 2.1.2.2, there are two malicious techniques that are used by malware in regards to DNS: fast flux and DGA. Especially the DGA technique is very often used at the moment, being used by very popular malware that has been released in the past two years. For this project we would like to focus on detecting these two techniques. We will create a classifier that

can detect if a domain name uses either fast flux or DGA. If one of those techniques is present in a domain name, it will be categorized as malicious. The features we will use for this system will be based on existing work that has been discussed in this chapter. The study by Villamarin et al. [30] focuses partly on detecting fast flux. EXPOSURE [2] focuses on detecting both fast flux and DGA. We will use some of the features that were used in those studies, in addition to variations of those features and some new features that we will introduce. This will be discussed in more detail in the next chapter.

4 THE SYSTEM

As we have determined in Section 2.3, we will use a network-based detection system using an anomaly-based classification technique. In the previous Section 3.2 we have determined that we want to perform this detection using DNS domain name information. In this chapter we will present the design of our system. We will give a general overview of the system; discuss what input data we use and how we receive it, the features that the system will use for classification, how we process the data and try to detect if a domain name is malicious and provide the database of the system.
We will conclude on how the system will answer MRQ1, how it is in line with our objectives, what challenges we have encountered and how it compares to existing solutions.

The general idea of the system design is that it logs all the DNS traffic that leaves and enters the network, filtering for DNS-request for domain conversion. In practice, this means that we will log all the domains that are accessed from within a network. We will create a device that will be placed inside the network that will log this information. The software that we will create for this device is able to both log the DNS and determine if a domain is malicious or not. After logging a domain name query, the domain name is compared to a whitelist. If the domain name is whitelisted, it will be discarded. If it is unknown, meaning it is not on the whitelist, it will be further analyzed. The component that will make the decision will be a classifier using a decision tree. When a domain is unknown (i.e. not on the whitelist) we will look for several features and characteristics of the domain that could be indication for a malicious domain name that uses either fast flux or DGA. The extracted features will be used in the trained classifier to determine if it is malicious or not. The domain name will then be stored in a database and flagged as malicious or legitimate, based on the outcome of the classifier. The next sections of this chapter will explain the system in more detail.

4.1 OVERVIEW

In this section we will provide a general overview of the system and its components. The components itself will be discussed in more detail in the next sections, in this paragraph we are trying to give an overview that shows the general responsibilities of each component, how they are connected, and how they fit in the process and flow of the system.

According to Bow et al. [5] a classification system can be divided into three phases: data acquisition, data preprocessing, and decision classification. See Figure 4. In the data acquisition phase the data is converted into a format that is suitable for preprocessing. The data is then used for the second data preprocessing phase, and grouped into a characteristic set of features as output. The third phase is the actual classifier that classifies the set of features to a certain class.

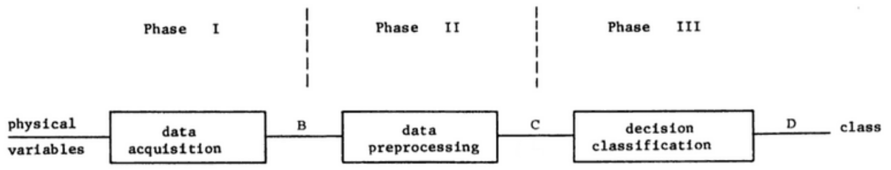


Figure 4: Phases of a classification system.

The classification part of our system follows the exact same phases.

PHASE I The first data acquisition phase is concerned with retrieving the full DNS information for a requested domain name. This data is stored in a log file.

PHASE II In the second data preprocessing stage the full DNS information is taken from the log file. The relevant data is then extracted from this log file. The domain name is then matched against a whitelist. A domain name that exists on the list is discarded. This phase concludes with extracting the features from the remaining domain names.

PHASE III The third decision classification phase uses the feature vectors from the previous phase to actually label the domain name as either malicious or legitimate.

The system consists of 8 components, as can be seen in Table 2. Figure 5 shows an overview of the full system.

ID.	NAME	PHASE
A	DNS Log Creator	I
B	DNS Log File	I
C	DNS Log Monitor	II
D	DNS Log Parser	II
E	Whitelist Matcher	II
F	Feature Extractor	II
G	Classifier	III
H	Database	-

Table 2: Overview of the components of our system.

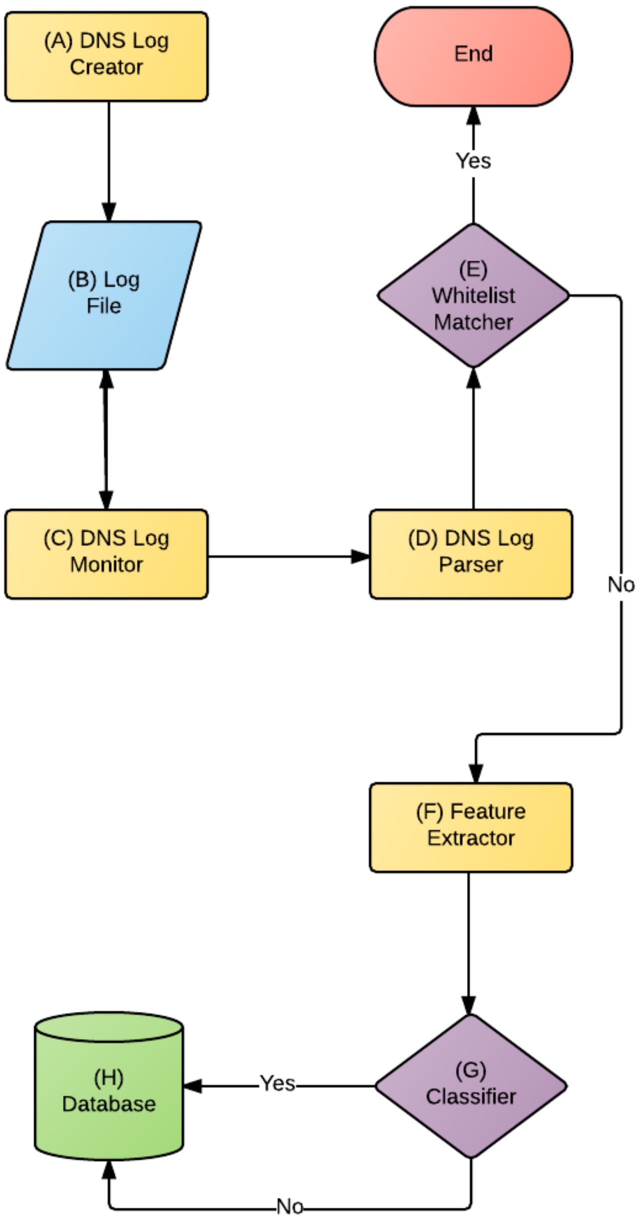


Figure 5: Overview of the system.

4.1.1 *Components*

This section discusses all the components in more detail.

(A) DNS LOG CREATOR Creates the logs file containing the domain name DNS information that we want to classify. Section 4.2.1 will go into detail on how we receive the queried domain names from the network and how they are stored in the *DNS Log File*.

(B) DNS LOG FILE The logs file that contains the full DNS queries and response data from domain names accessed from within the network. This file is continuously monitored by the *DNS Log Monitor* for new domain name records. This component will be discussed with more detail in Section 4.2.1.

(C) DNS LOG MONITOR Monitor for new DNS records in the log file. When a record is retrieved, it will be removed from the log file and passed to the *DNS Log Parser*. This component will be discussed with more detail in Section 4.2.2.

(D) DNS LOG PARSER This component parses the DNS record from the DNS Log Monitor. The parser extracts out all the information that we need for our classification and stores it in a custom DNS object. This component will be discussed with more detail in Section 4.2.2.

(E) WHITELIST MATCHER This component has access to the whitelist that we use. When it receives the DNS object from the parser it will compare the domain name from this object to the whitelist. When there is a match, the process stops and the DNS object will be discarded. When there is no match, meaning the domain name is unknown, it will be passed to the *Feature Extractor* component for further analysis. This component will be discussed with more detail in Section 4.3.2.

(F) FEATURE EXTRACTOR This component extracts and calculates all the features from the DNS object. In principle, it extracts a feature vector for our classifier from the DNS object. This data is then passed on to the *Classifier* component. This component will be discussed with more detail in Section 4.3.3.

(G) CLASSIFIER The trained classifier will use the feature vector from the extractor to classify the object as malicious or not. After classification the DNS object with the classification result is stored into the database. The classifier can also be used in training mode, which is not relevant when using the system in production mode. In training mode we can input a labeled dataset in order to train the classifier. This will be discussed in more detail in the next chapter in Section 4.3.4.

(H) DATABASE The database contains all the DNS objects that have passed through the classifier. This means that every requested domain name that is unknown (i.e. does not exist on the whitelist) will eventually end up in the database after being flagged by the classifier as malicious or legitimate. The database design will be discussed with more detail in Section 4.4.

4.2 DATA INPUT

In this section we will discuss the components that are relevant for the data acquisition. In our case this concerns how we detect and log the domain names (and their full DNS information) that are accessed from hosts within a network. We will discuss the following components in more detail:

- (A) DNS Log Creator
- (B) DNS Log File
- (C) DNS Log Monitor
- (D) DNS Log Parser

4.2.1 *DNS Log Creator & DNS Log File*

The first and most important component of this section is the *DNS Log Creator*. This component creates the log file containing the domain name DNS information that we want to classify. In order for this component to log this information, it needs to detect the domain names that are queried from within the network that the system analyses, including other relevant DNS information.

4.2.1.1 *Logging the DNS data from the network*

We have identified three possible ways to log the required DNS information from the network.

NETWORK SWITCH WITH PORT MIRROR This is the approach we will use in our proof-of-concept. We have placed a switch before the modem that connects to the internet. All traffic that goes to the modem now first passes the switch, after which it goes to the modem. Technically there does not really chance anything and the network flow is still intact and completely unmodified.
The network switch has a specific function called port mirroring. This feature allows us to specify port(s) on the router which should be

mirrored to a mirror port. In practice this means that we can copy the network traffic from the incoming port from the network, and the outgoing port that connects to the router, to the mirror port. We then connect our device that contains the DNS logging software to the mirror port. This allows us to receive all incoming and outgoing data from our network on our device that runs our system. We do not interact with the existing network stream at all, the only thing we do is copy it and analyze the copied stream for relevant DNS data. Figure 6 shows setup in action.

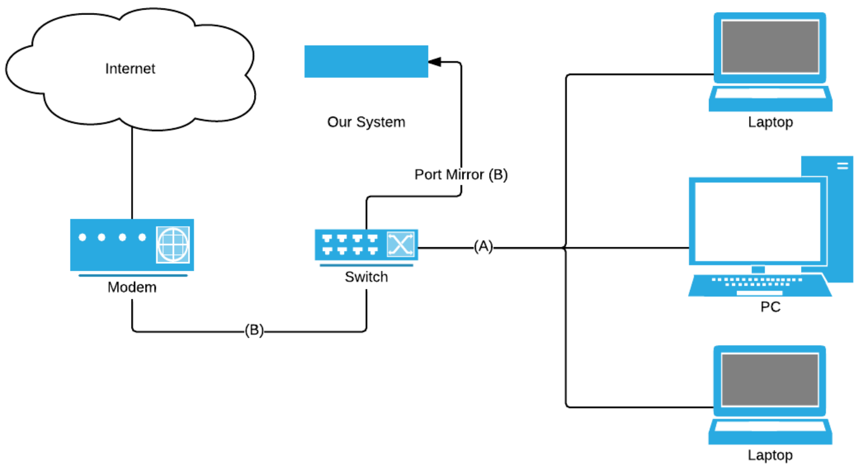


Figure 6: Our system inside a network.

NETWORK TAP This approach is very similar to network switch with port mirror approach. The only difference is that we do not use a network switch but a designated tap that splits the existing network cable into two, allowing us to tap the network stream from the newly created cable. This new cable effectively figures as a port mirror in the previous approach.

LOG DIRECTLY FROM DNS SERVER The final approach is dependent on the network that is being logged. If it is possible for the network administrator to force all DNS queries to go through a corporate DNS server, we can log the DNS queries straight at the destination. All DNS software is able to log the DNS requests into an external file. If we know that all DNS queries in a network will go through that DNS server we can simply turn on logging and parse the DNS log files of the server.

4.2.1.2 *Extracting the DNS data from the network stream*

Using the port mirroring setup we have access to all network traffic that enters and leave the network. The next step is to extract all the DNS requests and responses from this large stream of live network data.

Our initial approach was to use Wireshark [6], a free and open-source packet analyzer. Wireshark allows us to put network interfaces into promiscuous mode, allowing us to see all network traffic that is visible on that interface, even when the traffic is not addressed to the actual network interface. This was exactly what we needed in order to capture all the copied traffic that our system would receive from the port mirror switch.
However, it proved to be more work than anticipated to write a program that was able to simply log all DNS domain name requests and corresponding responses from a network stream. The main reason was that there were many duplicates that would slow down the whole process.

PASSIVEDNS We were able to find a small piece of software called PassiveDNS [3] that is free, open-source and does exactly what we need. According to the author of the software, PassiveDNS is *"a tool to collect DNS records passively to aid Incident handling, Network Security Monitoring (NSM) and general digital forensics."*
PassiveDNS analyses traffic from a network interface just like Wireshark and automatically outputs the DNS queries and responses to a log file. It is also able to cache and aggregate duplicate DNS responses in memory, limiting the amount of data and duplicates in the log file, without losing any of the actual DNS requests and responses.
Listing 1 shows a snippet with example output from the log file that PassiveDNS produces, showing how the log file looks like and what kind of information it stores.

Listing 1: Example line from a PassiveDNS log.

```
1437487607.235353||192.168.2.15||192.168.2.254||IN||rooster.rug.
    nl.||A||129.125.36.172||83637||0
```

As can be seen from the Listing 1, the log uses a delimiter between the different values. PassiveDNS stores 9 different values for a single DNS request and response.
Table 3 shows the definition of each value from the example log snippet that was taken from one of our log files. PassiveDNS stores these 9 values for every single domain name query.

CONCLUSION Using a network switch with port mirroring allows us to analyze all the network data that enters and leaves the network. Connecting a device running our system to the port mirror allows us access to this data. The PassiveDNS effectively is our *DNS Log Creator*. It filters out and extracts exactly the data that our system needs and stores it in a log file, which will act as our *DNS Log File*.

VALUE	DEFINITION
1437487607.235353	UNIX timestamp of the time that the DNS response was received. The timestamp shows the time microseconds.
192.168.2.15	The (local) IP address of the client that initiated the DNS query.
192.168.2.254	The IP address of the DNS server. In this case it is the local modem.
IN	Resource record class.
rooster.rug.nl	Queried domain name.
A	Record type. We are only interested in A records. A records translate queried domain names to a IP addresses.
129.125.36.172	The DNS answer. This is the IP address associated with the queried domain name.
83670	TTL value. Shows after how many seconds the local cache should be refreshed.
0	Is described in the documentation as count. We are not sure what this value does, in our own file with over 250000 DNS query logs we have never seen this value be anything else than 0.

Table 3: Data in a PassiveDNS record.

4.2.2 *DNS Log Monitor & DNS Log Parser*

The *DNS Log Monitor* keeps track of the *DNS Log File* that is created by the *DNS Log Creator*. As was explained in the previous section, the log file constantly changes when domain names are accessed from within the network and the DNS records are stored in the log file. The *DNS Log Monitor* keeps track of new changes to this log file, extracting new records from the log file, after which the records are sent to the *DNS Log Parser*, and the extracted records are removed from the log file. Because of this the *DNS Log File* will only contain new DNS records that have not been accessed by the monitor yet.

The *DNS Log Parser* receives new DNS records from the *DNS Log Monitor*. The records it receives are in the same format as shown in Listing 1. The *DNS Log Parser* will parse the records and extract the information that we need for our features. It is important to note that sometimes a single DNS query can result in multiple responses. The example from the previous section showed a DNS query with a single response. When a domain name has multiple IP addresses attached to the name we will receive separate responses for each single IP address. An example from the *DNS Log File* showing a DNS query for a domain name with multiple IP addresses is shown in Listing 2.

Listing 2: Example lines from a PassiveDNS log.

```
1437317609.307733||192.168.2.15||192.168.2.254||IN||twitter.com
    .||A||199.16.156.102||26||0
1437317609.307733||192.168.2.15||192.168.2.254||IN||twitter.com
    .||A||199.16.156.70||26||0
1437317609.307733||192.168.2.15||192.168.2.254||IN||twitter.com
    .||A||199.16.156.38||26||0
1437317609.307733||192.168.2.15||192.168.2.254||IN||twitter.com
    .||A||199.16.156.6||26||0
```

As we can see, a DNS query for the twitter.com domain results in four responses, one for each associated IP address. However, we can easily see that they originate from the same DNS request because the timestamp (in microseconds) is identical. The *DNS Log Parser* uses this timestamp information so that it can group multiple DNS records from the log file to the same original DNS query. Because we are not interested in the IP addresses itself, but rather the number of IP addresses associated with a domain name, we will only keep track of the number of responses.
An overview of the information that the parser extracts can be seen in Table 4. The example values show the output from the DNS File Parser using the twitter.com query example from above.

DATA	EXAMPLE VALUE
Timestamp	1437317609.307733
IP address	192.168.2.15
Domain name	twitter.com
IPs	4
TTL	26

Table 4: Data that the parser extracts.

This information is stored in an object and sent to the next component, the Whitelist Matcher.

4.3 PROCESSING

In this section we will focus on the main components of the system that are responsible for the actual classification of a domain name. We will first discuss the different features that our system uses and the rationale behind choosing them. Then we will go into detail about the different components that are relevant for the processing stage:

- (E) Whitelist Matcher
- (F) Feature Extractor
- (G) Classifier

4.3.1 *Features overview*

Arguably the most important part of the system is the features that we are going to use for our classifier. As we have explained in Section 2.1.2.2, there are two different DNS-based malicious techniques that are interesting for us: fast flux and DGA. The features we have selected try to detect if a domain name uses those techniques.
All the features we use can be extracted from the data that is shown in Table 4. We will divide our features over two categories: fast flux features and DGA features. Table 5 shows an overview of our features and the technique that it is trying to detect.

IDENTIFIER	NAME	TECHNIQUE
1	Number of IP addresses	Fast flux
2	TTL	Fast flux
3	Ratio digits to letters	DGA
4	Domain name length	DGA
5	Suspicious TLD	DGA
6	Special characters	DGA
7	2-gram	DGA
8	3-gram	DGA

Table 5: Feature overview.

The features will be discussed in more detail, including how they work and why we think that they are relevant.

4.3.1.1 *Fast flux features*

We have selected two features for the detection of fast flux domain names. Part of the study by Villamarin et al. [30] also focused on detecting fast flux domain names. They also used the *Number of IP addresses* and *TTL* of a domain name to try to classify a domain name as using fast flux or not. They were not very successful. Their experiments showed that using both features resulted in a very high FPR. According to them, the main reason for this is that many popular domain names (such as youtube.com) also use a very low TTL value and also have many different IP addresses associated with their domain name as a form or load balancing. In essence, very popular domain names have the same characteristics as malicious fast flux domain

names.
We will try to combat the high FPR by using a whitelist. Our whitelist will be based on Alexa' [29] list of domains. This list contains the most popular domain names. Because we will whitelist the most popular domain names and thus filter them out before classifying, we expect to get much better results by eliminating the main reason that the authors pinpointed to be the cause of their high FPR.

(1) NUMBER OF IP ADDRESSES Because one of the characteristics of fast flux domains is the high number of associated IP addresses, it makes sense to use this as one of the features to detect fast flux domains.

(2) TTL The TTL is a mechanism that limits the lifetime of data, similar to an expiration date. TTLs are also used in DNS, where these values are set by an (authoritative) name server for a specific record. When a DNS record is queried, that record will be cached for the time specified by the TTL. Because the DNS records of domain names that use fast flux are updated very often, it will also have a very low TTL value, making it a very interesting feature. Although we explained that round-robin DNS load-balancing also has the same characteristics as fast flux domains (including a low TTL value), it is a technique mostly used by popular websites. However this will probably not be an issue as we use the top 250.000 from Alexa.com as whitelist, so the most popular websites will be filtered out and classified as legitimate before they reach the classifier.

4.3.1.2 *DGA features*

The features in this section are used to detect DGA domain names. We use a total of 6 different features for the detection of DGA domain names. All of these features can be extracted from the domain name itself.

(3) RATIO OF DIGITS TO LETTERS While looking through datasets containing malicious and legitimate domain names one of the first things we noticed was that malicious domain names are much more likely to consist of (many) numbers. The goal of domain names is to translate a name to an IP address, because IP addresses are much harder to remember. Therefore it makes sense that websites use easy to remember domain names that consist of existing words or a company name. Malware authors are not concerned with this because they do not have to be remembered by humans. This is especially true for domain names that are generated with DGAs. Because of this we have selected this feature for DGA detection.

(4) DOMAIN NAME LENGTH One of the other things we noticed was that malicious names are often much longer than legitimate ones. The same reasons as the number ratio apply to the argument for this feature. Longer names are harder to remember, which malware authors usually not care about. Because of this malicious domain names are much more likely to be long. Based on the length a certain number of points is added to the total malicious score.

(5) SUSPICIOUS TLD Some top-level domains (TLDs) are statistically much more likely to be malicious than others. For example, a domain with a *.gov* extension is much less likely to be malicious than a *.ru* domain, even if nothing else is known about the domain. Because of this it would be interesting to test this feature.

(6) SPECIAL CHARACTERS This feature checks if the domain name contains suspicious characters, such as Chinese or Russian ones. Russian and Chinese domain names have a relatively high likelihood to be used for malicious purposes. When a domain name contains Russian or Chinese characters which are not supported by most operating systems and keyboards in the western world, it is very likely that the connection was not initiated by a person.

(7) 2-GRAM In the Dutch and English languages there are certain combinations of characters that never occur. When a domain does contain such a sequence it is very likely that it is a domain used for malicious purposes, created with a DGA. It would be very interesting to create a list with these 2-character sequences and see if any – and how many - of those sequences exist in a domain name.

(8) 3-GRAM Works the same as the 2-gram feature, except now we use a 3-character sequence.

4.3.2 *Whitelist Matcher*

The *Whitelist Matcher* is responsible for maintaining a whitelist that is used before feature extraction. This component receives an object similar to the one displayed in Table 4. The *Whitelist Matcher* will extract the domain name from this object and match it against a whitelist. If there is a match, we assume that the domain name is legitimate and we discard the domain name and abort the classification process. If there is not a match in the whitelist then the domain name is unknown, meaning we need to send it to the *Feature Extractor* after which it will be classified and stored in the database.
The whitelist that we use is the top 250.000 domain names from Alexa. Alexa is a company that provides web traffic analysis and statistics. It also publishes a list with the top $1,0$ million most popular websites.

We make the assumption that the top 250.000 most popular domain names in the world from that are legitimate. Our whitelist is contains the 250.000 most popular domain names in the world.

4.3.3 *Feature Extractor*

This component takes the DNS records containing unknown domain names that have not been found on the whitelist. What the Feature Extractor does is extract the necessary data from the DNS object that it receives from the *Whitelist Matcher* and uses it to calculate each of the features that we have discussed in Section 4.3.1. After extracting all the required information, the *Feature Extractor* puts all the feature values in an object and passes it to the *Classifier* where it will be classified and labeled. For each feature we will give a short explanation on how we extract the feature value from the DNS data object, as can be seen in Table 4 and put in in a feature object for the classifier that contains all the features from Table 5. We will give an example DNS object that the *Feature Extractor* receives and describe how the Feature Extractor would extract the features from this object. The example that we will use is shown in Table 6 and is taken randomly from our log file.

DATA	EXAMPLE VALUE
Timestamp	1437480226.483242
IP address	192.168.2.15
Domain name	docs.google.com
IPs	6
TTL	0

Table 6: Example DNS object.

(1) NUMBER OF IP ADDRESSES For this feature we simply take the data straight from the DNS object that the *Feature Extractor* receives. As can be seen from Table 6, the object that the *Feature Extractor* receives contains a value called IPs, which is essentially the *Number of IP addresses* feature. In this case this feature would have the value 6.

(2) TTL For this feature we also take the data straight from the DNS object that the Feature Extractor receives. As can be seen from Table 6, the object that the Feature Extractor receives contains a value called TTL, which is essentially the TTL feature. In this case this feature would have the value 0.

(3) RATIO DIGITS TO LETTERS For this feature we start with the Domain Name value from the DNS object that is received from the *Feature Extractor*. In this case that value would be *docs.google.com*. As can be seen, this domain name can be split into three parts, the subdomain name, the domain name, and the domain extension.

- Subdomain: docs
- Domain: google
- TLD: com

Because we are only interested in the actual name of the domain, we take the domain name including any subdomain names and we remove the dots (.) and the TLD. In this case this would leave us with the value *docsgoogle*. This value is then used as the basis to count the digits and letters. If there are no digits, the ratio is set to 0. If there are no letters, the ratio is set to 1. In other cases, we use Equation 1.

$$ratio = \frac{numberofdigits}{numberofletters} \tag{1}$$

In this case the ratio would be 0. In our example case, the feature would have the value of 0.

(4) DOMAIN NAME LENGTH For this feature we will also use the Domain Name value from the DNS object. Similar as to feature (3), we will again identify the (sub) domain name(s) and domain extension (TLD) and remove the TLD from the domain name and any dots (.) when subdomains are used. In our example case this would give us the value *docsgoogle*. This value is then used to determine the length of the domain name; in this case that value would be 10. This value is then used as the feature.

(5) SUSPICIOUS TLD For this feature we will again use the Domain Name value from the DNS object. We will determine the TLD value of the domain name and compare this to a list of suspicious TLDs. We have identified the following TLDs as having an above average potential to be malicious: *.biz, .cn, .tw, .cc, .ws, .ru, .tt*.

This list is created based on a study by McAfee [20], which focused on identifying the most popular domain extensions used for malicious C&C communication.

The feature value will act as a Boolean value. It will be set to 1 if the domain extension is suspicious, and 0 if it is not suspicious. In the example case, the identified TLD is com, which is not on the suspicious list, resulting in the feature to have the value 0.

(6) SPECIAL CHARACTERS For this feature we will use the full Domain Name value from the DNS object. Because of the way the DNS system works it will be very simple to detect special characters, such as Chinese or Russian characters. Because the DNS system is only able to process alphanumerical ASCII-characters in addition to the - character, any non-alphanumerical domain name will be converted to an ASCII-domain name.
This conversion is done by an algorithm called ToASCII. When ToASCII detects that a domain name contains a non-alphanumerical character it will use the Nameprep [17] algorithm, which converts the domain name to lowercase and does some other normalizations, after which it will translate the result to ASCII using Punnycode [9]. When this is completed the final step is to prepend a four character string "xn–" to the converted domain name. This prefix is called the ASCII Compatible Encoding prefix, and is used to distinguish Punnycode encoded domain names from regular ASCII domain names.
Because of how this works in the DNS system, all we need to do is detect the "xn–" prefix in a domain name. If this prefix is present in the domain name, it means that the original domain name used non-alphabetical characters and has been converted in order to work with the DNS system. The feature is a Boolean value. When the prefix is present, it is set to 1; otherwise it is set to 0. Because in our example case the domain name has no prefix, it means that no special characters were used and thus the feature is set to 0.

(7) 2-GRAM For this feature we will also use the Domain Name value from the DNS object. The value we will use for our calculation is the domain name, including any subdomains. In our example case, the base value would be *docsgoogle*. The main goal of the feature is to detect the presence of 2-character combinations that do not exist in the English language. Because of this we will only focus on the domain name itself and discard the domain extension.
The 2-gram list that we have created with the non-existent character combinations can be found in Appendix A and contains 128 2-character combinations. For this feature we will use this list and determine how many of these combinations exist within our domain name, including subdomains. In our example case we will count how many of the 128 2-character combinations are present in the *docsgoogle* value.
A different way to look at it would be to split the domain name in 2-character combinations and see if they occur on the list. *docsgoogle* is made up of the following 2-grams: *do, oc, cs, sq, go, oo, og, gl, le.*

None of the above 2-grams appear on our list with non-existing character combinations from Appendix A. Because of this, the value of the feature would be 0.

(8) 3-GRAM The process for this feature would be exactly the same as for feature 7. The only difference is that this time we repeat the process using a 3-character combination. We use the same base value again and compare it to a list of non-existent 3-character combinations that we have created and can be found in Appendix B. This list has a total of 11.806 non-existent 3-character combinations. Using the example again, our base *docsgoogle* value does not contain any of the 3-gram combinations from Appendix B. This means that in our case the feature would have the value 0

The full feature object that we have created based on the example input from Table 6 is shown in Table 7.

FEATURE ID.	EXAMPLE VALUE
1	6
2	0
3	0
4	10
5	0
6	0
7	0
8	0

Table 7: Example DNS object.

4.3.4 *Classifier*

The classifier is the main component in the sense that it is responsible to make the actual decision on the maliciousness of a domain name. The *Classifier* component takes a feature object like seen in Table 7 and uses this in order to classify the domain name.
The classifier itself is supervised and uses a J48 decision tree that has been implemented by using the Weka library [16]. This will be discussed in more detail. Information about the dataset that we have used to train our supervised classifier can be found in the next chapter in Section 5.1.1. The main process of the classifier is taking the feature object from the *Feature Extractor*; convert it into a feature vector that can be used by the Weka library. This feature vector is then classified by the trained J48 decision tree, also implemented by the Weka library. After classification, all the domain name information, including the classification result is stored in the database. The design of this database will be discussed in Section 4.4.

WEKA LIBRARY Our classifier is implemented using the Weka library. Weka (Waikato Environment for Knowledge Analysis) is a popular suite of machine learning software written in Java. It is free software and available under the GNU license.
What makes Weka great for our case is that it has implemented a large collection of both supervised as well as unsupervised classification methods. Referring back to our research questions, we want to test features, combinations of features and how our features perform using several classification algorithms. Weka library allows us to create a dataset using a standard format that includes our data, features and labels and is then able to easily test this using multiple classification algorithms.

In the next chapter we will go into more detail about the dataset that we have created and the different classification algorithms that we will test.

4.4 DATA STORAGE

For our data storage we use a simple MySQL database that runs on the same device as our system. The only component that stores information in the database is the Classifier that – after classifying – stores the domain name, including its features, other relevant DNS information, and the classification result into the database.
Our database contains the following tables.

CLIENT This table stores the unique clients within a network. A DNS query always originates from a certain client (local IP address). That information is stored in this table.

DOMAIN NAME This table stores the unique full domain names, including subdomains and the TLD.

DOMAIN REQUEST This table stores the actual individual DNS queries, extracted features and classification result. When the client that creates the request already exists, it will be linked to the existing client; otherwise it will be created in the client table. If the domain name has already been logged before, it will be linked to the domain name record from the domain name table. If the domain name has not been queried before, a new domain name record will be created.

The full information for the tables, including the different fields, references and data types can be found in Appendix C.

4.5 CONCLUSION

We have answered RQ1.1 about the best approach in Chapter 2 and Chapter 3. In Section 4.3.1 of this chapter we have discussed our features and the rationale behind them, effectively answering RQ1.2.
The overarching question was MRQ1, designing a non-intrusive, lightweight and practical system that is able to detect unknown malware within a network. This chapter has answered this question by presenting the design of our system.

NON-INTRUSIVE Our system is non-intrusive. We do not access, modify or interfere with any of the network streams that leave and enter the system. All we do is create a copy of the network stream, send it to our system, and use the copy of this network stream for analysis.

LIGHTWEIGHT The system only uses minimal DNS information. The actual classification only uses features that can be calculated and extracted from three values: the number of IP addresses, the TTL value, and the actual full domain name. The system works fast and is able to detect, classify and find malicious domain requests in real time.

PRACTICAL The system is practical in the sense that it can run on a simple hardware device. It is very easy to set up and does not need any additional resources. All that needs to be done is place a switch at the point in the network where the internal network is connected to the outside network (the internet). When this is placed, the switch can start copying the network stream to our device right away and start detecting malware in real time.

4.5.1 *Challenges*

We have encountered two issues when designing and developing our system. We will briefly discuss the problems.

4.5.1.1 *Double domain extensions*

The first issue we encountered was with our *Feature Extractor*. As we have discussed in Section 4.3.3, this component uses the full domain name and for some features needs to remove or extract only the TLD from this string. The issue lies in the fact that extinguishing the domain extension from the rest of the domain is not always as straight forward as it seems when working with TLDs that consists of two parts with a dot (.) delimiter.

A simple case:
When the *Feature Extractor* receives the domain name *docs.google.com* and we need to remove the TLD, it is very easy. We split the string using the dot (.) as delimiter and we take out the last substring. This will result in *docs.google*, which is indeed our full domain name with the TLD removed.

A more troublesome case:
When our *Feature Extractor* receives the domain name *google.co.uk* and we use the same approach as the above example, it would result in *google.co*. This is not the full domain name with the TLD removed, but the full domain name with half of the TLD removed. If we used the same approach then it would look like co is the main domain name and google is the subdomain name, which is not the case.

We solved this issue by using the Mozilla Public Suffix List [12]. This PSL is an attempt by Mozilla to build a database of TLDs. By using this list we can now simply compare the end of a full domain name with the PSL in order to extract the correct domain extension.

4.5.1.2 *Hosting providers in whitelist*

Another problem that we have encountered is with our *Whitelist Matcher*. As explained we make the assumption that the 250.000 most popular domain names in the world (from the Alexa list) are all legitimate.
The problem lies in the fact that very popular hosting providers also occur on this whitelist, even though their subdomains are used by their hosting customers. There is of course absolutely no guarantee that the subdomains used by their customers are used for legitimate purposes, because those customers could be anyone.
Take for example the domain for Amazon cloud service *amazonaws.com*. It is a very popular domain name; it is actually #152 on the list of most popular domain names. When taking a random *amazonaws.com* example from the log file, such as *www-i-1628877250.us-east-1.elb.amazonaws.com* we can see that this domain name references to a VPS from one of Amazon' customers and could potentially be used for malicious purposes. Because the main domain name, *amazonaws.com* is part of the whitelist our system would discard this DNS query immediately.
A possible solution to this problem has been proposed in Section 6.2.1.

5 RESULTS

In this chapter we will present the tests that we have performed, including their results. The tests that we will do try to answer our main research question MRQ2, and its sub-research questions RQ2.1, RQ2.2 and RQ2.3. We want to get insight in how well our system performs and how well our features perform.
Our tests have been designed with the goal of answering these questions:

- How well do our individual features perform?
- How well do certain combinations of features perform?
- How well do our features perform using different classification algorithms?

We have split up our tests in two parts. For the first part we will test our features individually using different classification algorithms. In the second part we will test some combinations of features using different classification algorithms. The results or these tests should enable us to answer the related research questions. The results of our tests are described in Section 5.2.

The dataset that we have created contains a total of 500.000 labeled examples. 250.000 samples (50%) of this dataset contain legitimate labeled domain names; the other 50% contains labeled malicious domain names that have been generated with reverse-engineered DGAs from actual existing malware. Unfortunately we were not able to obtain a dataset of live fast flux domain names. Because of this we were only able to test the features that were related to detecting the DGA technique. The dataset will be discussed with more detail in Section 5.1.1.

As explained, we will test each of our features individually in addition to some combination of features. We will do each test three times, each for a different classification algorithm. The three classification algorithms that we will use are: a J48 decision tree that is an open source implementation of the C4.5 algorithm [23], a Naïve Bayes classifier [19] and a Support Vector Machine [8] (SVM).

5.1 TEST PREPARATION

In this section we will discuss the test preparation. We will first discuss the dataset in more detail and conclude with the software setup we used in order to perform our tests.

5.1.1 *The dataset*

As explained in the chapter introduction, we have a dataset containing 500.000 labeled samples. Half of these samples – 250.000 – are labeled as legitimate and are taken from the top 250.000 most popular domain names from Alexa. This is the same list as we use for our whitelist in the *Whitelist Matcher* component from section Section 4.3.2. The other half contains 250.000 labeled malicious samples.

Unfortunately we were not able to find a substantially large dataset for fast flux domain names. The features that we use to detect fast flux domains: Number of IP addresses and TTL are extracted from DNS query responses and thus can only be obtained from live and active domain names that are actually in use. After a thorough search for live fast flux domains, we were only able to find a small collection of active fast flux domains. We have contacted several organizations that specialize in malware, such as The Shadowserver Foundation [13], and they all confirmed that at the moment there are not many active malware threats that use the fast flux technique. On the other hand, they all confirmed that the DGA technique is very popular at the moment and that it would be much more beneficial to focus on this technique.

Obtaining domain names that use the DGA technique was much easier to achieve. The DGA for many popular pieces of malware that have been released in the past few years have been reverse-engineered. When we have the reverse-engineered DGA from a specific malware we can use this algorithm to generate the exact same malicious domain names as the malware itself would. This allows us to generate as many malicious domain names for our dataset as needed, containing domain names that the actual malware would generate itself. We have identified five pieces of malware for which we were able to find the reverse-engineered DGA and used these to create the 250.000 malicious samples for our dataset.

We have used the following malware:

- Conficker [11]
- Cryptolocker [18]
- Torpig [27]

- Necurs [10]
- Ranbyus [1]

For each of these pieces of malware we have used an implementation of the reverse-engineered DGA that each malware piece uses. For each piece of malware we have generated 50.000 samples, which make for the total of 250.000 maliciously labeled samples in our dataset. Because the malware pieces we have selected are from the past few years and they are different types of malware, we assume that the combination of all 250.000 samples is a good representation of domain names that are generated with a DGA. Figure 7 shows an overview of our dataset.

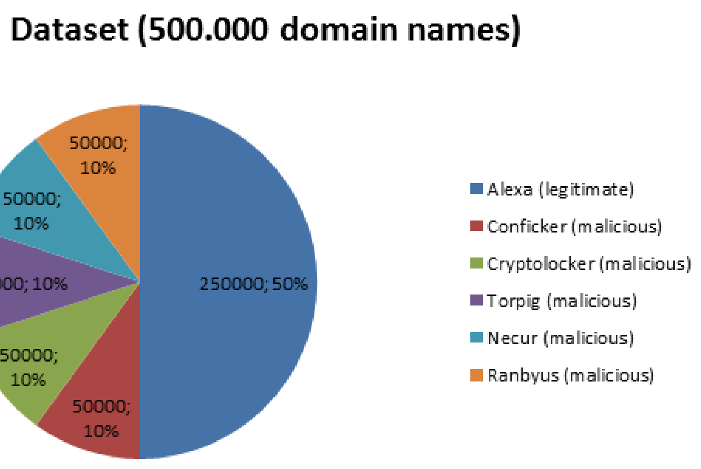


Figure 7: The dataset.

This dataset will be used to test our 6 features that have been selected for DGA detection.

5.1.2 *The setup*

As explained we have used implemented reverse-engineered DGAs from existing malware to generate a total of 250.000 malicious samples, and used the Alexa top to obtain 250.000 legitimate domain names.
We have created a setup where we import the full dataset into our software, and then the *Feature Extractor* component from our detection system is used to extract the features from the data samples. This information is then converted and stored into a data format that can be used by the Weka library which is used for classification. We then use this library to perform our tests.

5.2 THE TESTS & RESULTS

Section 5.2.1 will discuss the tests that we have done with regards to the individual features. The tests where we have tested combination of features will be discussed in Section 5.2.2

For each single test we will use a 10-fold cross validation.
Using cross validation allows us to use our complete dataset for both training and testing purposes. For the 10-fold cross validation we will randomly divide our original 500.000 dataset into ten equally sized subsets, each containing 50.000 samples. Of the ten subsets, one is used as validation data for testing; the other nine subsets are used as training data. This process is repeated 10 times, where a different subset is used each time as validation data.

After each single test we will calculate the following metrics.

TRUE POSITIVE (TP) The number of samples that are classified as malicious that are actually malicious domain names.

TRUE NEGATIVE (TN) The number of samples that are classified as legitimate and that are actually legitimate domain names.

FALSE POSITIVE (FP) The number of samples that are classified as malicious, but are actually legitimate domain names.

FALSE NEGATIVE (FN) The number of samples that are classified as legitimate, but are actually malicious domain names.

TRUE POSITIVE RATE (TPR) The TPR from Equation 2 is the proportion of samples which were classified malicious, among all domain names which truly are malicious.

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{2}$$

TRUE NEGATIVE RATE (TPR) The TNR from Equation 3 is the proportion of samples which were classified legitimate, among all domain names which truly are legitimate.

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}} \tag{3}$$

FALSE POSITIVE RATE (FPR) The FPR from Equation 4 is the proportion of samples which were classified as malicious, but are actually legitimate, among all samples which are legitimate domain names.

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} \tag{4}$$

FALSE NEGATIVE RATE (FNR) The FNR from Equation 5 the proportion of samples which were classified as legitimate, but are actually malicious, among all samples which are malicious domain names.

$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{TP} + \mathrm{FN}} \tag{5}$$

ACCURACY The accuracy from Equation 6 is the proportion of correctly classified samples (both malicious and legitimate) among all samples in the full dataset.

$$accuracy = \frac{(\mathrm{TP} + \mathrm{TN})}{(\mathrm{TP} + \mathrm{FP} + \mathrm{FN} + \mathrm{TN})} \tag{6}$$

We will use the accuracy to determine the best features of our system.

5.2.1 *Individual features*

In this section we will discuss the results of the tests that we have done for each individual feature. For each feature we will use the 10-fold cross validation and calculate the metrics discussed in the previous section. Each feature will be tested three times, once for each classification algorithm.

5.2.1.1 *Ratio digits to letters*

The first feature we have tested was the *ratio digits to letters*. The results are shown in Table 8.

The results show that the J48 decision tree performed by far the worst. The Naïve Bayes and SVM algorithm performed much better in comparison, although still not very well – just marginally better than a coin flip. With an accuracy of $53,93\%$, the SVM performed slightly better than the Naïve Bayes classifier.
What is interesting to see is that almost all the samples in the dataset were classified as malicious. 100% of malicious domain names were correctly labeled as malicious, but also 230.353 of the 250.000 $(92,14\%)$ of the legitimate domain names were classified as malicious. Only 19.647 $(7,86\%)$ legitimate domain names were correctly classified as

METRIC	J48 (%)	NAÏVE BAYES (%)	SVM (%)
FPR	92,14	92,14	92,14
FNR	100,00	0,00	0,00
TPR	0,00	100,00	100,00
TNR	7,86	7,61	7,86
Accuracy	3,93	53,80	**53,93**

Table 8: Ratio to digits results.

non-malicious.
The rationale behind this feature was that we expected malicious domain names to have a higher possibility of containing digits, but after inspecting the dataset we found that in the case of our dataset none of the malicious domain names contained a digit.

5.2.1.2 *Domain name length*

The results of the *domain name length* feature can be seen in Table 9.

METRIC	J48 (%)	NAÏVE BAYES (%)	SVM (%)
FPR	36,97	64,02	38,65
FNR	65,14	51,40	40,35
TPR	35,87	48,59	59,65
TNR	63,03	35,98	61,35
Accuracy	49,44	42,29	**60,50**

Table 9: Domain name length results.

The results show that the Naïve Bayes classifier performs the worst and the SVM performs the best with an accuracy of $60,05\%$.
The feature shows a high FPR and FNR of respectively $38,65\%$ and $40,35\%$.
We expected there to be a difference in length between legitimate and malicious domain names. The results show that this feature alone is not good enough, but it might be useful in combination with other features. We will explore this in Section 5.2.2.

5.2.1.3 *Suspicious TLD*

The results of the *suspicious TLD* feature can be seen in Table 10.

The results show that both the J48 decision tree as well as the Naïve Bayes classifier perform bad with this feature, with an accuracy of $8,62\%$. The SVM performed much better in comparison with an accuracy of $55,27\%$.

METRIC	J48 (%)	NAÏVE BAYES (%)	SVM (%)
FPR	100,00	100,00	6,71
FNR	82,75	82,75	82,75
TPR	17,25	17,25	17,25
TNR	0,00	0,00	93,29
Accuracy	8,62	8,62	**55,27**

Table 10: Suspicious TLD results.

The feature classifies $93,29\%$ of the legitimate domain names correctly, with a FPR of $6,71\%$. The main reason for the low accuracy is that the feature is only able to detect a small part of the malicious domain, with a TPR of $17,25\%$.
The feature could still be useful, because the FPR is relatively low but it will only be able to detect a small portion of the malicious domain names as malware.

5.2.1.4 *Special characters*

The results of the *special characters* feature can be seen in Table 11.

METRIC	J48 (%)	NAÏVE BAYES (%)	SVM (%)
FPR	99,89	99,89	99,89
FNR	100,00	100,00	0,00
TPR	0,00	0,00	100,00
TNR	0,11	0,11	0,11
Accuracy	0,05	0,05	**50,05**

Table 11: Special characters results.

The J48 decision tree and Naïve Bayes perform very badly. In comparison, the SVM has a much higher accuracy of $50,05\%$, but this is still only marginally better than a coin flip.
The feature was able to correctly identify all 250.000 malicious domain names as malicious. However, the feature only detected 263 legitimate domain names as legitimate. This resulted in a FPR of $99,89\%$.
After inspecting the dataset, only a very small amount of domain names contained special characters ($< 0,1\%$).

5.2.1.5 *2-gram*

The results of the *2-gram* feature can be seen in Table 12.

METRIC	J48 (%)	NAÏVE BAYES (%)	SVM (%)
FPR	10,53	10,53	10,53
FNR	19,22	19,22	19,22
TPR	80,78	80,78	80,78
TNR	89,47	89,47	89,47
Accuracy	**85,12**	**85,12**	**85,12**

Table 12: 2-gram results.

All three classification algorithms showed exactly the same result, all having an accuracy of $85,12\%$.
This feature was able to correctly identify $80,78\%$ of the malicious domain names as malicious. In addition, $89,47\%$ of the legitimate domain names were recognized as such.
This feature seems to work very well. It is able to detect $80,78\%$ of the malicious domain names with a FPR of $10,53\%$. The FPR is higher with $19,22\%$.

5.2.1.6 *3-gram*

The results of the *3-gram* feature can be seen in Table 13.

METRIC	J48 (%)	NAÏVE BAYES (%)	SVM (%)
FPR	3,49	3,49	3,49
FNR	11,88	11,88	11,88
TPR	88,12	88,12	88,12
TNR	91,51	91,51	91,51
Accuracy	**92,31**	**92,31**	**92,31**

Table 13: 3-gram results.

All three classification algorithms showed exactly the same result, all having an accuracy of $92,31\%$. This is significantly better than the *2-gram* feature which already did very well.
This feature was able to correctly identify $88,12\%$ of the malicious domain names as malicious. In addition, $96,51\%$ of the legitimate domain names were recognized as such.
This feature does very well and is by far our best performing feature. It is able to detect almost 90% of the malicious domain names with a FPR below $3,5\%$. The FNR is higher with $11,88\%$.

5.2.1.7 *Overview*

Table 14 shows an overview of all the individual features with the classification algorithms that achieved the highest accuracy.

FEATURE	ALGORITHM	ACCURACY (%)
Ratio digits to letters	SVM	53,93
Domain name length	SVM	60,05
Suspicious TLD	SVM	55,27
Special characters	SVM	50,05
2-gram	SVM	85,12
3-gram	SVM	**92,31**

Table 14: Overview best individual feature results.

5.2.2 *Feature combinations*

In this section we will show the results of the tests we performed by combining features. We will test a combination containing all six of our features. In addition, we will test the combinations of our best performing feature, the *3-gram*, with every other feature.

5.2.2.1 *All features*

The results of the combination of all features can be seen in Table 15.

METRIC	J48 (%)	NAÏVE BAYES (%)	SVM (%)
FPR	5,98	19,63	5,48
FNR	19,50	6,87	8,74
TPR	8,50	93,13	91,26
TNR	94,02	80,37	94,52
Accuracy	87,26	86,75	**92,89**

Table 15: Results of all features combined.

The SVM performed best with an accuracy of 92, 89%.
The combination of all features was able to detect 91, 26% of all the malicious domain names, and correctly identified 94, 51%. Although the Naïve Bayes classifier had a slightly higher TPR, this comes at the cost of a much worse TNR. The FPR is 5, 48% and the FNR is 8, 74%.

5.2.2.2 *All features without special characters*

The results of the combination of all features without the special characters feature can be seen in Table 16.

METRIC	J48 (%)	NAÏVE BAYES (%)	SVM (%)
FPR	7,60	17,57	6,15
FNR	5,08	3,85	8,12
TPR	92,40	96,15	91,88
TNR	94,92	82,43	93,85
Accuracy	**93,66**	89,29	92,87

Table 16: Results of all features combined without the special characters feature.

The J48 decision tree performed best with an accuracy of $93,66\%$. The SVM does slightly worse. The SVM FPR is slightly lower; however the J48 decision tree performs better on the FNR, TPR and TNR metric, resulting in a higher accuracy.

5.2.2.3 *All features without special characters and ratio digits to letters*

Because combining all features without our worst performing feature - *special characters* - seemed to perform better than any of our other individual features and combinations, we decided to go further and take out both out worst and second worst performing feature - both *special characters* as well as *ratio digits to letters*. The results are shown in Table 17.

METRIC	J48 (%)	NAÏVE BAYES (%)	SVM (%)
FPR	5,07	5,27	5,81
FNR	7,84	10,36	8,53
TPR	92,15	89,64	91,47
TNR	94,93	94,73	91,19
Accuracy	**93,54**	91,19	92,83

Table 17: Results of all features combined without the special characters and ratio digits to letters feature.

The J48 decision tree performed best with an accuracy of $93,54\%$. Compared to the other algorithms, the J48 decision tree scored better on all of our metrics. However, leaving both the features *special character* and *ratio digits to letters* out resulted in a worse performance than only leaving special charters out. Because of this we will not explore any further by removing the third worst feature, and so on.

5.2.2.4 *3-gram combinations*

We have also tested our best performing feature - *the 3-gram* - with every other feature. The test results can be found in Appendix D. In all cases the SVM algorithm performed the best.
Combining the *3-gram* feature with any of the other features does not seem to do much for any of the metrics, although the accuracy of any of the *3-gram* combinations is (slightly) better than the individual *3-gram* accuracy. The best increase in accuracy compared to the individual *3-gram* feature is less than 0, 3% absolute value. The FPR, FNR, TNR and TPR are all very similar, which is confirmed by the small change in accuracy.

5.2.2.5 *Overview*

Table 18 an overview of all the feature combinations and the classification algorithms that achieved the highest accuracy.

FEATURE	ALGORITHM	ACCURACY (%)
All	SVM	92,89
All without special characters	J48	**93,66**
All without special characters and ratio digits to letters	J48	93,54
3-gram and ratio digits to letters	SVM	92,39
3-gram and domain name length	SVM	92,57
3-gram and suspicious TLD	SVM	92,38
3-gram and special characters	SVM	92,34
3-gram and 2-gram	SVM	92,46

Table 18: Overview feature combination performances.

5.3 OVERVIEW

The combination where we use all features without the *special characters* feature results in the best performance with an accuracy of 93, 66%. This is only slightly better than the individual *3-gram* feature which achieves an accuracy of 92, 31%. This shows us that the *3-gram* feature is by far our most important feature.
The SVM algorithm performed best for most of our tests. For the

individual feature tests, it was always the (shared) best performing algorithm, sometimes significantly better than the other two.
For all features combined and all the two feature combinations with the *3-gram* it also performed the best. However, we achieved the highest accuracy with the J48 decision tree when we combined all features without the special character feature.

6 DISCUSSION & CONCLUSION

In this final chapter we will discuss our system, dataset and the features. In addition, we will also discuss possible future work for our system. We will finish this chapter with a conclusion, where we will discuss our research questions and answers.

6.1 DISCUSSION

In this section we will briefly discuss our system, our dataset and the performances of our features.

6.1.1 *The system*

The system that we have designed and implemented works according to objectives and goals we have set in the introduction chapter. Based on this fact alone, the system is a success.
Most of the previous network-based systems use features based on the time that a C&C connection is established, or other similar features that require massive amounts of data in order to find meaningful patterns. Most of the systems that we described in Section 3.1 are able to log information at the ISP level in order to gain enough data to train their classifiers.
Our system works different by only considering the individual request, although the analysis is done using datasets with existing domain names. The benefit of only considering the individual request is that our system is able to detect a C&C connection as malicious the first time it encounters it after our system has been installed.
Other existing systems work based on blacklists. Our system is able to detect domain names from malware that it has not seen before and is not dependent on an external source that provides blacklists.

6.1.2 *The dataset*

The dataset that we used contained 500.000 labeled samples. The 250.000 samples labeled as legitimate and 250.000 labeled as malicious.

LEGITIMATE SAMPLES The legitimate were all taken from the Alexa top 250.000 most popular domain names in the world. Because the dataset was so large, it was not realistic to confirm that every single domain name that we used for our legitimate dataset was in fact non-

malicious. Using the Alexa top 250.000 was our best option, because the more popular a domain name, the less likely it is to be malicious. Because the whitelist contained domain names from all over the world, it had the additional benefit of containing domain names with all kinds of domain extensions, which is relevant for our *suspicious TLD feature*. In addition, a good portion of the domain names had subdomain names, which is relevant for our *domain name length* feature.
The only issue with the whitelist is the problem that has been described in Section 4.5.1.2 regarding hosting providers. We have proposed a possible solution to this problem in Section 6.2.1.
We think that the Alexa top 250.000 dataset we have used is a good representation of legitimate domain names.

MALICIOUS SAMPLES As discussed in Section 5.1.1 we were unfortunately unable to include any meaningful number of fast flux domain names in our malicious dataset. For this reason we have only included domain names created by a DGA.
In order to create a diverse DGA malicious dataset we have used the reverse-engineered DGAs of five different malware pieces. Each DGA contributed 50.000 samples to our malicious dataset. The benefit of this is that the malicious dataset contains domain names that would actually be used by real, popular and relatively new malware. We have made sure that our selections of five different types of malicious DGAs are a good representation of the DGA technique as a whole. This is important, because the system should also be able to detect domain names generated with algorithms that are not used in our dataset. We have made sure that the DGAs that we have selected all have some different characteristics, some generate a domain name with a fixed length, and some use a fixed TLD, while others also generate sub domain names. We think that our malicious dataset is a good representation of domain names that are generated by a DGA.

6.1.3 *Features*

Because our malicious dataset only contained DGA domain names we were unable to test our two features that we have selected to detect fast flux domain names.

6.1.3.1 *Ratio digits to letters*

This feature was able to correctly classify 53,93% of the domain names. We expected that malicious domain names were relatively more likely to contain digits than legitimate domain names.
The dataset that we used contained 19.647 legitimate domain names that contained at least one digit. The malicious domain names from our dataset contained not a single domain name that used a digit. Although we have tried to create our malicious dataset with as diverse

as possible DGAs, we have not found any DGA that would generate domain names containing numbers. That merely means that there is no popular malware with a reverse-engineered DGA that generates domain names containing. We conclude that existing popular malware that uses a DGA is very unlikely to generate domain names containing digits.

6.1.3.2 *Domain name length*

This feature was able to correctly identify 60, 50% of the domain names. Although not a great result, it does show that the domain name length can be used as a feature, possibly in combination with other features.
Our expectation was that malicious domain names were more likely to be longer than legitimate domain names.
The legitimate domain names in our dataset had an average length of 10 characters, while the malicious domain names had an average length of 12 characters. This confirms our expectation that malicious domain names are longer.
We conclude that this feature could be helpful in combination with other features.

6.1.3.3 *Suspicious TLD*

This feature was able to correctly identify 55, 27% of the domain names. Not particularly good, but it does show some potential for the feature.
We expected that some domain extensions that we have labeled as suspicious were much more likely to be used for malicious domain names than legitimate ones.
There were 16.787 domain names in our legitimate set that were classified as having a suspicious TLD. 43.114 of our malicious domain names had domain extensions that were classified as suspicious. This confirms that malicious domains are more likely to have the TLDs that we have labeled as suspicious.
We conclude that there is potential for this feature, but that we probably have to modify our list of suspicious TLDs in order to achieve better results.

6.1.3.4 *Special characters*

This feature correctly identified 50, 05% of the domain names. This is not much more than a coin flip. This means that either the feature is bad or that the dataset does not have the right balance.
Our expectation was that from the Western world, a connection made to a domain name using Chinese, Russian or other non-ASCII characters was more likely to be malicious.

In our dataset 263 legitimate domain names contained special characters and 0 malicious domain names contained special characters.
We conclude with that we believe that our observation that requesting domain names using special characters from a computer in the Western world is still very likely to be malicious, but that the occurrence of this is extremely low and thus very difficult to verify.

6.1.3.5 *2-gram and 3-gram*

The *2-gram* feature was able to correctly identify 85,12% of the domain names. The *3-gram* feature performed even better, identifying 92,31% of the domain names correctly. Both these features are by far our best.
We expected that (generated) domain names are much more likely to contain 2- and 3-character combinations that do not exist in the English language, because they are created by random characters and are not based on actual words like legitimate domain names. The results confirm this.
The legitimate domain names contained an average of 0,49 3-grams from the list we have created. The malicious domain names contained an average of 6,03 3-grams from our list.
We conclude with the remark that these results are very good and perform consistently well on our full dataset that is made up of DGAs from different types of malware. However, if DGAs start generating domain names using combinations of existing words than this feature would probably become irrelevant.

6.2 FUTURE WORK

For our future work we will discuss a few additions and improvements that can be made to the system that we have created.

6.2.1 *Filter hosts*

A small problem we had with our whitelist was that it also contained hosting providers. When a popular hosting provider on our whitelist uses its subdomains for customers, our system would whitelist that domain name, even though the subdomain would redirect to the server of a customer who could potentially host a malicious C&C server.
For a future version of our system it would be beneficial to keep track of popular hosting providers around the world such as Amazon, CloudFlare, Akamai, and others. The system should automatically remove these hosting providers from the list to ensure the integrity of the whitelist.

6.2.2 *Integration with existing IDSs*

Our system is passive in the sense that it only detects requests for malicious domain names and stores this information. It does not actively block the malicious connection.
One of the advantages of our system is that it is able to detect a malicious connection before the actual connection is made. Before the actual connection to a server is made, the IP address is requested via the DNS domain request. Because we are able to detect the DNS request and response for a malicious domain name, we already know that a malicious application is going to create a connection to the associated IP address before it is made.
It would be very beneficial to export this data to an already in place IDS (such as a firewall) so that it can automatically block the IP address that has been flagged as malicious by our system. This way our system can greatly complement other existing systems.

6.2.3 *Additional features*

Some additional features would possibly interesting to test and use in a future version of our system.

DOMAIN AGE Because the domain names used for malicious purposes are usually only used for a very short amount of time before they are blocked, it would be very interesting to use information about the registration date of the domain for classification purposes. Because a dataset containing live and active domains is needed for this, we were unable to test this feature.

3-GRAM WITH DUTCH Because *3-gram* was by far our best performing feature, it would be interesting to use it with 3-grams created from different language. For example, when we are classifying a Dutch *.nl* domain name we would use a 3-gram list that was created using a Dutch dictionary.

RATIO OF DUPLICATE CHARACTERS It would be interesting to use a feature where we will use the ratio of duplicate characters in a domain name as feature. Because, as our 3-gram feature confirms, legitimate domain names are much more likely to be made up of words from a dictionary, whereas malicious domain names often contain random characters. Because actual words are much more likely to use certain characters more often, we can expect legitimate domain names to have a higher ratio of duplicate characters in a domain name then malicious domain names.

6.2.4 *User interface*

Initially we planned to create a web based user interface that displays the detected malicious domain names and the source of the possible infection. Because this was a nice to have rather than an important requirement of our project we have not been able to finish the user interface.
The user interface should be able to:

- Show an overview of all the hosts that have performed DNS queries.
- Filter on hosts that have requested a domain name that our system has classified as malicious.
- Show an overview of all classified domain names, with filters such as the classification result, domain extension, and other characteristics.

6.3 CONCLUSION

In this document we have presented the steps we have taken to be able to create the malware detection system that we have presented in this thesis.
At the start of this project, we only knew that we were going to create a malware detection system and the goals and objectives that we had for our system.
Because we had no background in information security or any knowledge of malware, except basic knowledge that can be expected from a Computing Science student, the first part where we performed the background study was very important and informative, and also challenging. After the background study we were able to decide on an approach for our detection system; we decided to create a network-based malware detection system. This has been described in Chapter 2.
The next step was doing research into existing systems, solutions and state of the art that used a network-based approach. We have decided that using DNS domain name information was the best approach taking into consideration our resources, time, goals and objectives. This has been discussed in Chapter 3.
Next, we designed our system and developed our system. This has been described in Chapter 4.
The step after that was designing tests, performing those tests and presenting the results in Chapter 5.
The current final chapter described our last step, where we interpret the results, discuss possible future work and conclude with the answers to all of our research questions.

RQ1.1 A network-based approach that has been described in Section 2.3.

RQ1.2 Our classification is based on features that can be extracted from DNS queries and responses. The features are described in Section 4.3.1.

MRQ1 The final design has been presented in Chapter 4.

RQ2.1 The tests and results of the individual features have been described in Section 5.2.1.

RQ2.2 The tests of the combination of features and their results have been described in Section 5.2.2.

RQ2.3 All the tests that we have done on individual and combination of features were done in threefold, each time for a different classification algorithm. The SVM performed the best for each individual feature test. The SVM was also the best performing algorithm for the combination of feature tests. However, the best accuracy that we have achieved was with a combination feature test using a J48 decision tree.

MRQ2 We have discussed each feature including their performance results in Section 6.1.3.

We consider this project a success. We have designed a system that matches all of our goals and objectives. The classifier is able to correctly identify 93, 66% of all the domain names. In addition, there are enough possibilities proposed for future work to make this system even better and more useful.

APPENDIX

A

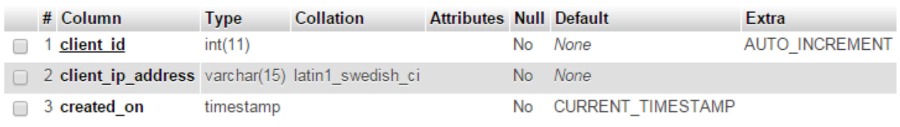
FORBIDDEN 2-GRAMS

- bk
- bq
- bx
- cb
- cf
- cg
- cj
- cp
- cv
- cw
- cx
- dx
- fk
- fq
- fv
- fx
- fz
- gq
- gv
- gx
- hk
- hv
- hx
- hz
- iy
- jb
- jc
- jd
- jf
- jg
- jh
- jj
- jk
- jl
- jm
- jn
- jp
- jq
- jr
- js
- jt
- jv
- jw
- jx
- jy
- jz
- kq
- kv
- kx
- kz
- lq
- lx
- mg
- mj
- mq
- mx
- mz
- pq
- pv
- px
- qb
- qc
- qd
- qe
- qf
- qg
- qh
- qj
- qk
- ql
- qm
- qn
- qo
- qp
- qq
- qr
- qs
- qt
- qv
- qw
- qx
- qy
- qz
- sx
- sz
- tq
- tx
- vb
- vc
- vd
- vf
- vg
- vh
- vj
- vk
- vm
- vn
- vp
- vq
- vt
- vw
- vx
- vz
- wq
- wv
- wx
- wz
- xb
- xg
- xj
- xk
- xv
- xx
- xz
- yq
- yv
- yy
- yz
- zb
- zc
- zg
- zh
- zj
- zn
- zq
- zr
- zs
- zx

B FORBIDDEN 3-GRAMS (SNIPPET)

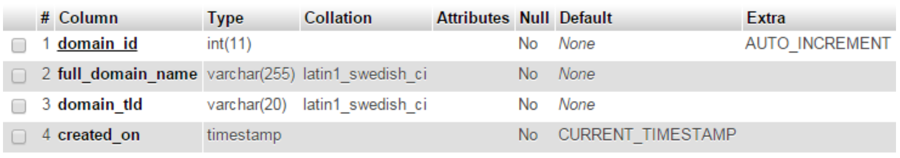
- aaa
- aab
- aac
- aad
- aae
- aaf
- aah
- aaj
- aak
- aao
- aap
- aaq
- aas
- aat
- aau
- aav
- aaw
- aax
- aay
- aaz
- abc
- abf
- abg
- abk
- abp
- abq
- abt
- abv
- abx
- abz
- acb
- acd
- acf
- acg
- acj
- acp
- acv
- acw
- acx
- acz
- adk
- adx
- aeb
- aee
- aef
- aeh
- aei
- aej
- aek
- aep
- aeq
- aet
- aeu
- aew
- aex
- aey
- aez
- afb
- afc
- afd
- afh
- afj
- afk
- afm
- afp
- afq
- afu
- afv
- afw
- afx
- afz
- agc
- agj
- agk
- agq
- agv
- agx
- agz
- ahb
- ahc
- ahd
- ahf
- ahg
- ahh
- ahj
- ahk
- ahp
- ahq
- aht
- ahv
- ahx
- ahy
- ahz
- aib
- aih
- aio
- aiq
- aiu
- aix
- aiy
- ajb
- ajc
- ajd
- ajf
- ajg
- ajh
- aji
- ajj
- ajk
- ajl
- ajm
- ajn
- ajp
- ajq
- ajr
- ajs
- ajt
- ajv
- ajw
- ajx
- ajy
- ajz
- akb
- akc
- akg
- ...
- zzx
- zzz

C DATABASE TABLES



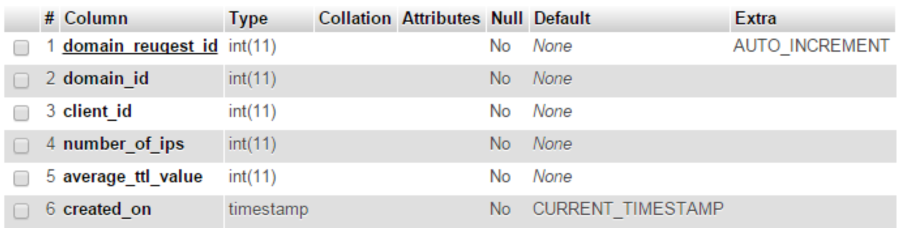
	#	Column	Type	Collation	Attributes	Null	Default	Extra
☐	1	client_id	int(11)			No	*None*	AUTO_INCREMENT
☐	2	client_ip_address	varchar(15)	latin1_swedish_ci		No	*None*	
☐	3	created_on	timestamp			No	CURRENT_TIMESTAMP	

Figure 8: Overview client table.



	#	Column	Type	Collation	Attributes	Null	Default	Extra
☐	1	domain_id	int(11)			No	*None*	AUTO_INCREMENT
☐	2	full_domain_name	varchar(255)	latin1_swedish_ci		No	*None*	
☐	3	domain_tld	varchar(20)	latin1_swedish_ci		No	*None*	
☐	4	created_on	timestamp			No	CURRENT_TIMESTAMP	

Figure 9: Overview domain table.



	#	Column	Type	Collation	Attributes	Null	Default	Extra
☐	1	domain_reuqest_id	int(11)			No	*None*	AUTO_INCREMENT
☐	2	domain_id	int(11)			No	*None*	
☐	3	client_id	int(11)			No	*None*	
☐	4	number_of_ips	int(11)			No	*None*	
☐	5	average_ttl_value	int(11)			No	*None*	
☐	6	created_on	timestamp			No	CURRENT_TIMESTAMP	

Figure 10: Overview domain request table.

D 3-GRAM COMBINATIONS RESULTS

METRIC	J48 (%)	NAIVE BAYES (%)	SVM (%)
FPR	6,98	92,14	3,34
FNR	11,89	1,33	11,89
TPR	88,12	98,67	88,12
TNR	93,02	7,86	96,67
Accuracy	90,57	53,26	**92,39**

Table 19: Combination of 3-gram and ratio digits to letters results.

METRIC	J48 (%)	NAIVE BAYES (%)	SVM (%)
FPR	6,59	6,58	5,95
FNR	19,67	14,03	8,91
TPR	80,33	85,97	91,09
TNR	93,41	91,42	94,05
Accuracy	86,87	89,69	**92,57**

Table 20: Combination of 3-gram and domain name length results.

METRIC	J48 (%)	NAIVE BAYES (%)	SVM (%)
FPR	4,19	4,19	4,19
FNR	11,74	11,60	11,05
TPR	88,26	88,40	88,95
TNR	95,81	95,81	95,81
Accuracy	92,04	92,10	**92,38**

Table 21: Combination of 3-gram and suspicious TLD results.

METRIC	J48 (%)	NAIVE BAYES (%)	SVM (%)
FPR	3,44	3,46	3,44
FNR	11,88	11,88	11,88
TPR	88,12	88,12	88,12
TNR	96,56	96,54	96,56
Accuracy	**92,34**	92,33	**92,34**

Table 22: Combination of 3-gram and special character results.

METRIC	J48 (%)	NAIVE BAYES (%)	SVM (%)
FPR	6,94	6,49	6,94
FNR	11,81	10,91	8,15
TPR	88,19	89,09	91,85
TNR	93,06	93,51	93,06
Accuracy	90,62	91,30	**92,46**

Table 23: Combination of 3-gram and 2-gram results.

BIBLIOGRAPHY

[1] Johannes Bader. *The DGA of Ranbyus*. URL: `http://www.johannesbader.ch/2015/05/the-dga-of-ranbyus/` (visited on 09/20/2015).

[2] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. "EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis." In: *NDSS*. 2011.

[3] Edward Bjarte. *PassiveDNS*. URL: `https://github.com/gamelinux/passivedns` (visited on 09/05/2015).

[4] Rainer Böhme. *The economics of information security and privacy*. Springer, 2013.

[5] Sing T Bow. *Pattern recognition and image preprocessing*. CRC Press, 2002.

[6] Gerald Combs et al. "Wireshark-network protocol analyzer." In: *Version 0.99* 5 (2008).

[7] AV Comperatives. *Anti-Virus Comperative Report*. 2014. URL: `http://www.av-comparatives.org/wp-content/uploads/2015/01/avc_sum_201412_en.pdf` (visited on 06/18/2015).

[8] Corinna Cortes and Vladimir Vapnik. "Support-vector networks." In: *Machine learning* 20.3 (1995), pp. 273–297.

[9] Adam M Costello. "Punycode: A bootstring encoding of Unicode for internationalized domain names in applications (IDNA)." In: (2003).

[10] P Ferrie. "The curse of Necurs, part 1." In: *Virus Bulletin* (2014), p. 4.

[11] Niall Fitzgibbon and Mike Wood. "Conficker. C: A technical analysis." In: *SophosLabs, Sophon Inc* (2009).

[12] Mozilla Foundation. *Mozilla Public Suffix List*. URL: `https://publicsuffix.org/list/public_suffix_list.dat` (visited on 09/12/2015).

[13] The Shadowserver Foundation. *Shadowserver Foundation*. URL: `http://shadowserver.org` (visited on 09/12/2015).

[14] Frederic Giroire, Jaideep Chandrashekar, Nina Taft, Eve Schooler, and Dina Papagiannaki. "Exploiting temporal persistence to detect covert botnet channels." In: *Recent Advances in Intrusion Detection*. Springer. 2009, pp. 326–345.

[15] Guofei Gu, Phillip A Porras, Vinod Yegneswaran, Martin W Fong, and Wenke Lee. "BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation." In: *Usenix Security*. Vol. 7. 2007, pp. 1–16.

[16] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. "The WEKA data mining software: an update." In: *ACM SIGKDD explorations newsletter* 11.1 (2009), pp. 10–18.

[17] Paul Hoffman and Marc Blanchet. *Nameprep: A Stringprep Profile for Internationalized Domain Names (IDN)*. Tech. rep. 2003.

[18] Keith Jarvis. "CryptoLocker Ransomware." In: *Viitattu* 20 (2013), p. 2014.

[19] George H John and Pat Langley. "Estimating continuous distributions in Bayesian classifiers." In: *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc. 1995, pp. 338–345.

[20] Shane Keats. "Mapping the Mal Web, Revisited." In: *McAfee SiteAdvisor* 4 (2008).

[21] Microsoft. *Computer Viruses*. URL: `https://msdn.microsoft.com/en-us/library/cc505929.aspx` (visited on 06/18/2015).

[22] No NSTISSI. "4009." In: *National INFOSEC Glossary* (2006).

[23] Salvatore Ruggieri. "Efficient C4. 5 [classification algorithm]." In: *Knowledge and Data Engineering, IEEE Transactions on* 14.2 (2002), pp. 438–444.

[24] Sherif Saad, Issa Traore, Ali Ghorbani, Bassam Sayed, David Zhao, Wei Lu, John Felix, and Payman Hakimian. "Detecting P2P botnets through network behavior analysis and machine learning." In: *Privacy, Security and Trust (PST), 2011 Ninth Annual International Conference on*. IEEE. 2011, pp. 174–180.

[25] Imtithal A Saeed, Ali Selamat, Ali MA Abuagoub, and Salman Bin Abdulaziz. "A Survey on Malware and Malware Detection Systems." In: *analysis* 3.10 (2013), pp. 13–17.

[26] Eugene H Spafford. "The Internet worm program: An analysis." In: *ACM SIGCOMM Computer Communication Review* 19.1 (1989), pp. 17–57.

[27] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. "Your botnet is my botnet: analysis of a botnet takeover." In: *Proceedings of the 16th ACM conference on Computer and communications security*. ACM. 2009, pp. 635–647.

[28] W Timothy Strayer, David Lapsely, Robert Walsh, and Carl Livadas. "Botnet detection based on network behavior." In: *Botnet Detection*. Springer, 2008, pp. 1–24.

[29] Alexa Top. *1,000,000 Sites*.

[30] Ricardo Villamarín-Salomón and José Carlos Brustoloni. "Identifying botnets using anomaly detection techniques applied to DNS traffic." In: *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*. IEEE. 2008, pp. 476–481.

[31] iMPERVA. *Assessing the Effectiveness of Antivirus Solutions*. 2012. URL: `http://www.imperva.com/docs/HII_Assessing_the_Effectiveness_of_Antivirus_Solutions.pdf` (visited on 06/18/2015).