Framework for automated comparison of machine learning based botnet detection approaches

Network security

Master Thesis

Group 1025

Aalborg University
Electronics and IT

AALBORG UNIVERSITY
STUDENT REPORT

Electronics and IT
Aalborg University
http://www.aau.dk

Title:
Framework for automated comparison of machine learning based botnet detection approaches

Theme:
Master thesis

Project Period:
Fall Semester 2016

Project Group:
1025

Participant(s):
Nikolaj Bové Højholt

Supervisor(s):
Jens Myrup Pedersen

Copies: 2

Page Numbers: 78

Date of Completion:
June 2, 2016

Abstract:

Malicious software is a security problem that has been around for many years. The topic that is currently, being extensively investigated is botnet detection. Botnets are global networks of compromised computer, that a botmaster can use to for example run Distributed Denial of Service attacks campaign. Time and time again, researcher have to create good data sets for training and testing the machine learning algorithms. The goal of this project is to create a framework of a system which is publicly available, and that enables easy comparison between various machine learning based botnet detection methods, where each detection method is tested with the same data sets for training and testing. This also requires extensive knowledge about best practices in capturing, labelling and merging data sets into training and evaluation sets.

Contents

Preface

This report has been written by a student on Network and Distributed systems as a 10th semester project. This project was prosed by Jens Myrup Pedersen. The purpose of this project is to create a framework for a system that enables automatic comparison of machine learning based botnet detection approaches, this system would have to be widely available, and a benchmark platform for easy comparison between different machine learning based botnet detection approaches. The system is developed using a Python webframework known as pyramid, a MySQL database on the back-end part of the system together with an application also written in Python.

Materials, such as the Python pyramid scripts, SQL queries and the back-end application together with an online version of the report for recreate ability is publicly available through the groups project website on `http://kom.aau.dk/group/16gr1025/`, this will be referred to a "🌍online repository" in the report.

The group would like to give a thanks to Jens Myrup Pedersen, Matija Stevanovic and Egon Kidmose for supervision and feedback on the report and on implementation suggestions.

Aalborg University, June 2, 2016

Nikolaj Bové Højholt
<nhajho11@student.aau.dk>

Glossary

Botnet	robot network. 2
CnC	Command & Control. 2
DDoS	Distributed Denial of Service. 3
DGA	Dynamically Generated Addresses. 5
DNS	Domain Name System. 4
DPI	Deep Packet Inspection. 5
FN	False Negative. 8, 10
FP	False Positive. 10
HTTP	Hypertext Transfer Protocol. 5
HTTPS	Hypertext Transfer Protocol Secure. 5
IDS	Intrution Detection System. 9
IRC	Internet Relay Chat. 5
ISP	Internet Service Provider. 8
LAN	Local Area Network. 8
malware	malicious software. 1
MCFP	Malware Capture Facility Project. 27
MLA	Machine Learning Algorithm. 7

OS	Operating System. 18
P2P	Peer-2-Peer. 6
SSH	Secure Shell. 22
TP	True Positive. 8

Chapter 1

Introduction

This chapter presents the problem with malicious software (malware) in this day and age. The focus is then be shifted from general malware and focus on botnet in particularly. A small introduction to how botnets function and why botnets are a huge problem are presented, before going into some of the work that has already been done, in the field of botnet but also trying to take some inspiration from the general work that is being done to fight malware. Finally, this chapter presents a problem statement, that will be worked with throughout the rest of this project.

1.1 Malicious Software

As computers become more integrated, into the everyday life of people, both at home and at work. With an ever increasing amount of users coming online every year [23], the number of potential targets for malware also increased. Malware describes any kind of software that has been created with malicious purpose. When looking at the history of malware, the motivation has for creating malware has clearly changed.

In the beginning malware was mostly composed of viruses as the Internet was not as accessible as today. In simple terms these viruses would mainly try to corrupt host files or boot sectors and thereby rendering the host computer unusable for most users. The primary reason behind the viruses in the early days of malware history was to establish a reputation for the creator [3].

As the Internet grew, the number of different malware threats also grew. The network worm, unlike viruses that used several days to months to spread across some geographical regions. The network worm could spread across the globe in a matter of seconds, by exploiting software vulnerabilities, thereby accessing other computers that had the software vulnerability that the worm could exploit. As time progressed and malware evolved from only causing havoc, a new threat in the form on trojans. Trojans unlike viruses and worms would not self-replicate and only spread by pretending to be a legit and benign application from either a website or e-mail attachment, but by executing would cause everything from irritation to the user (opening windows, changing screensaver etc.) to damaging the host (deleting files, keylogger, spreading more malware, opening backdoors etc.) [2]. From stealing information with keylogging features in trojans, the potential of acquiring banking information and thereby financial gain can easily be imagined.

The state-of-the art malware is, malware that has the ability of providing remote access to a compromised machine, giving the attacker, the creator of the malware, full access to the machine over the Internet. As a result the machines compromised with this new and sophisticated malware become a part of a robot network (Botnet), where each compromised machine effectively becomes a bot (also known as zombie machine) and if the attacker has control over several bots, it becomes a network of bots, or botnet. It is important to note that botnet in nature can be used with both good and bad intention. A "good" botnet [4] could be an organization that requires more processing power and thus uses a botnet where willing participants share some of their computational processing power. However, throughout this report the term botnet will be used in order to describe malicious botnet. In a botnet, the infected hosts (compromised machines) serve a botmaster (botherder). The botmaster controls each bot often through a Command & Control (CnC) server, which is the interface the botmaster uses in order to control the bots. The different methods a botmaster uses will be seen in section 1.2

1.2 Botnet

Throughout this section light will be cast on why botnet is a big priority malware threat and also how botnets work with CnC servers as previously explained.

1.3 Botnet Threat

One of the reasons that botnets are a big priority malware threat is partly due to the many different attack vectors a single bot can do, but more importantly the amount of financial damage it can cause. On a global level malware in it self is reported to have caused more than 1 trillion by 2011 [22]. As highlighted below, botnet attack vectors causes a significant sum of financial damage, as a botmaster can use the botnet available to initiate different attack vectors [4], also known as attack campaigns:

Distributed Denial of Service (DDoS) attack All bots in a botnet are instructed to connect to a specific site/server. Given some of the larger botnets controlling more than 1 million bots [3], many services are unable to handle this many simultaneous requests. Depending on the payload the target could use up all of the bandwidth for a services, thus further insuring that a service is unavailable. According to [22] the average DDoS attack in 2011 caused over $180000 in damages, making DDoS attacks the most costly attack that year.

Click fraud Instead of using botnets to takedown a service, the bots is redirected to a website controlled by the attacker and "click on ads". This generates revenue for the attacker, as an advertiser pays money per clicked ad. The ad revenue is often routed through an ad affiliation program, such as Google adSense or Yahoo!, thus effectively money laundering the ad revenue. According to [3] two different companies: Click Forensics (now part of Google) stated 18.6% of all clicks on monitored adverts where click fraud for second quarter of 2010. Anchor Intelligence stated for the same period 28.9% of all clicks on monitored adverts where click fraud.

Pay-per-install agent Pay-per-install business model is highly related to that of click fraud. Here the botmaster offers to install specific software on target machines for a customer. The software can be anything from benign to malicious software. According to a study presented in [24] the German Honeynet Project estimated that a botmaster can make about $ 430 a day from pay-per-install.

Spam relay Using a botnet to send spam has the potential of doing several things: Advertising phony or overprices products, scamming or phishing for financial gain by tricking users into giving up login credentials or credit card credentials, or by doing advance-fee schemes that tells users that they have a huge amount of money placed somewhere and are able to get hold of it, but not before donating money. In 2009 the estimated financial cost of spam emails was around $ 130 billion [3]. There are many more ways to use spam, and is only limited by the creativity of the spammers. According to [19] the 3rd quarter of 2015 the proportion of spam emails was 56.17 %.

Large-scale information harvesting When a bot has infected a computer it is easy for the botmaster to install a keylogger in order to record login and password information for different sites, search the content of the computer, alter the Domain Name System (DNS) configuration and thereby redirecting the victim to look-alike sites e.g. banking sites.

Randomware While not specifically botnet related, one of the newest and most potent randomware, cryptolocker can be argued to have bot functionalities. These funcatinalities are that, the cryptolocker often will try and connect to a server to get a RSA key in order to encrypt the harddrive. This connection could easilly be used for CnC commands aswell enableling all the other bot features. According to [6] the standard fee the creators of cryptolocker sets is $ 500 and increases the fee if not paid within a certain time frame.

1.4 Botnet Architecture

When a botmaster initiate communication with the botnet, it is often based on two different network architectures, commonly known centralised and decentralised. Both architectures have advantages and disadvantages, as well as different protocols that also have advantages and disadvantages. Throughout this section both concepts will be shed light upon together with what is known as a hybrid architecture where both the centralised and decentralised approach is combined.

1.4.1 Centralised

The centralised architecture is the simplest and most efficient botnet setup. All bots establish communication with one or more CnC servers as shown on Figure 1.1, as these CnC servers are under the control of the botmaster. The botmaster is therefore able to quickly and simultaneously communicate instructions to the CnC servers which then forwards the intructions to all the bots connected [3]. This architecture offers certain advantages for the botmaster, as the number of compromised machines, as before mentioned fast and simultaneously communication and easy to implement.

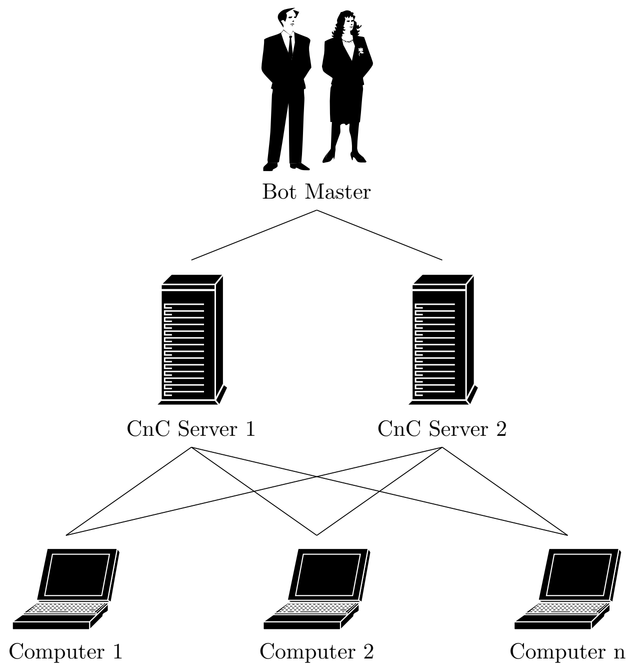


Figure 1.1: Botnet architecture known as centralised, here with two CnC servers

The first CnC servers where based on Internet Relay Chat (IRC) protocol, which provides an instantaneous one-to-many communication and can support a large number of clients being connected to one channel. Over the years Hypertext Transfer Protocol (HTTP) and Hypertext Transfer Protocol Secure (HTTPS) services has been gaining popularity in terms of usage [24]. As the IRC protocol often can standout, as the community that uses IRC has been declining steadily ever since 2003 [10] from just under 1 millions active users to about 400000 users in 2012.

Some of the disadvantages of the centralised approach is that it is centralised, as such, if all of the CnC servers are taken down the whole botnet is rendered useless. A way to provide redundancy Dynamically Generated Addresses (DGA) and/or fast-flux DNS is used. Another way is to use HTTPS as previously mentioned, as it add encryption and thus making Deep Packet Inspection (DPI) next to impossible.

1.4.2 Decentralised

In contrast to the centralised architecture, the decentralised does not have dedicated CnC servers, but purely rely Peer-2-Peer (P2P). The decentralised botnet is often referred to as P2P botnet. An illustration of P2P botnet can be seen on Figure 1.2. Each bot (peer in P2P terminology) has knowledge about some participating peers, thus information about the whole botnet is not known, thereby making it near impossible to obtain the full list of comprimised peers. When a new peer comes online [24], some studied P2P botnet has been known to try and connect to a number of predefined peers, thus taken those peers down would render the botnet unable to acquire new peers, but still rendering the botnet operational. Others have been known to start scanning for other peers by randomly searching IP ranges.

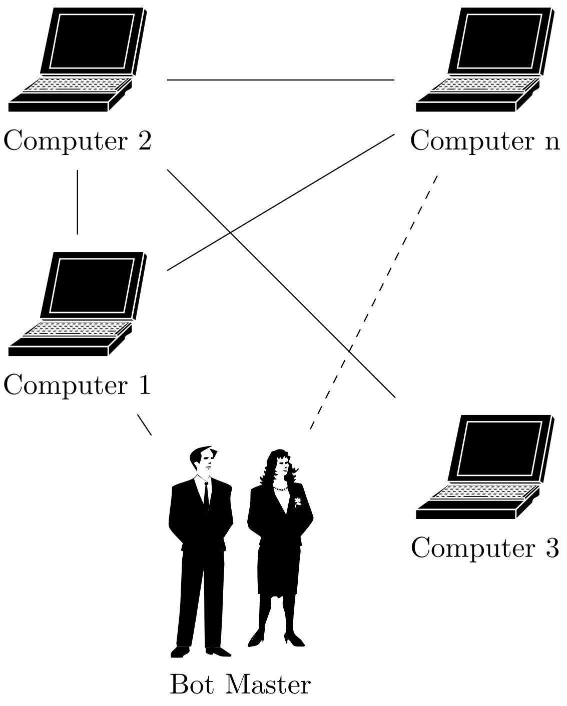


Figure 1.2: Botnet architecture known as decentralised

One of the major disadvantages is that the propergation time can be very long, in contrast to the centralised approach that has almost instantanious propergation time [3]. Another disadvantage compared to the centralised approach is that P2P traffic potentially will standout if compared to HTTP/HTTPS traffic as P2P traffic (in form of BitTorrent) accounted for 8% of the overall traffic in Europe 2015 [15] not including mobile traffic.

1.4.3 Hybrid

By combining the centralised and decentralised architecture creates a new type of topology for the botnet. Due to the similarities with centralised and decentralised, this type of architecture is simply known as hybrid. An example of a hybrid topology is that the botmaster communicates with some centralised servers. Meanwhile all the compromised peers are using the decentralised architecture to communicate with each other. Then only a few compromised peers communicate with the centralised servers. Thus creating an additional layer of protection to the botmaster, while having a higher level of control over the botnet then otherwise would be possible in a pure decentralised approach.

1.5 Related Work

In order to defend against bot malware, the first thing to do is to detect the malware it self, for this two approaches can be used, namely host based and network based detection. Host based detection is done by installing software on a machine e.g. anti-virus applications, where there are several well known and effective anti-virus companies that can deal with traditional malware, that all now offer free online scans, and sites such as `https://www.virustotal.com/` taking it a step further with the online scans as it uploads a given file to 56 different anti-virus companies (Appendix A), but as bot malware is communicating with a CnC server, updating bot malware, to be unrecognisable to anti-viruses, is fairly easy. Due to this, the preferred approach for detection bot malware is using network based detection.

As botnet rely on Internet traffic in order to communicate with the botmaster via the CnC servers to perform different attack campaigns. Network traffic detection is currently the pinnacle research topic for botnet detection. Additionally to network traffic analysis, many have utilised Machine Learning Algorithm (MLA) in order to identify botnet traffic.

A new study [18] has provided a comprehensive overview of existing scientific work on the use of MLA for botnet detection. However, the paper was only able to provide a paper comparison of the existing work. This is because of the fact that different detection methods were evaluated using different traffic data sets, that are often not publicly available.

Combined over the analysed papers from [18], a total of 15 different MLAs where investigated. From the most common supervised methods (Detection tree classifiers: C4.5, Random Forests, REPTree) and unsupervised methods (Hierarchical clustering and X-means). Common for all of the MLAs investigated is that the True Positive (TP) results, of all the bottom range of reported detection rate, where all above 90 %, while most of the algorithms had less than 1 % False Negative (FN) detections.

From [18] it is also clear that the number of different botnet families are often not disclosed (DNS based approaches) or kept at a very low number (≤ 3) with commonly $1 - 2$ different samples from the different botnet families. Only a few investigation used far more botnet families and samples from these families than the others like [21] which used 6 different bot malware families and a total of 188 different samples across the different families. The detection of all the different samples clearly showed that making an MLA that can detect everything is difficult, as they where able to detect been $49 - 100$ % of the family samples.

To further expand on the data sets that was used by the comprehensive survey shows that, of the 20 papers. 6 used data captured from a Local Area Network (LAN) environment, 8 used captured data from a University campus network and 9 where provided with data from Internet Service Provider (ISP) networks, all of which were used as background traffic. Some of the papers used more than a single capturing level interface.

This indicates the need for a future approach for reliable evaluation of the existing work, as there are many deviating factors, specially in the data sets, between the current approaches.

1.5.1 Evaluation Data

One of the main challenges of creating a good detection algorithm, can not be achieved without good data sets in order to train the MLA based detection methods. When creating good data sets, a substantial amount of data has be known before hand i.e. network traces labelled as malicious and non-malicious. If the labelling is off it could, especially for supervised MLA, produce a lot of FN compared to if the labelling is correct. Some of the most typical methods of used for creating a good evaluation set is as follows:

- Using an Intrution Detection System (IDS) (such as snort), signature-based botnet detection system or by relying on blacklisted IP addresses and domain names

- The use of HoneyPot machines that is used for generating botnet traffic, and merging the traffic with normal and benign data
- Similar fashion as using HoneyPots but by having the source code of the bots, and thus being able to control every aspect of the bots. Then merge the generated traffic with benign traffic
- Same as above, but where the source code is not available, but binaries are available. As real bot malware is being deployed, the bot can become part of an attack campaign, therefore are limits often placed on the Internet connection in order to protect third-parties

One thing to note is that capture of the evaluation data set is done at LAN, campus, ISP or core, everything needs to be captured at the same level. When looking at LAN, bots would often look to be working by it self, as compared to the ISP level, where there could be drawn correlation between bots using by one botmaster. The same can be said about LAN compared to campus network.

Using on a small fraction of the available botnet families/samples has the inherent risk of tailoring a detectiong algorithm to only detect a specific botnet family or samples. However, it should be noted that targeting specific types/families of botnet achieves better detection performances when detecting that specific type/-family of botnet. Using a high number of botnet families/samples will improve the detection method, so it would be able to generalise or not. If a MLA based detection approach is able to generalise, means that it can potentially be used to discover new unknown botnet families/samples, but also has the drawback not being as precise in discovering new mutations of a botnet family as one that has been trained for that specific botnet family. As for a general botnet detection [18] proposes that substantially more botnet families/samples are included into the network traces.

1.6 Problem Statement

In section 1.1, the current state of malware was briefly explained. With the conclusion that botnet is state-of-the-art malware, and has the potential of causing a number of problems and financial losses, which was further elaborated on in section 1.3. The botmaster has the possibility to cause these variety of different attack campaigns, that can target the local user of the infected machine (e.g. identity theft, cryptolocker), but also external computer (e.g. DDoS). In 2011 [22] estimated that $ 1 trillion was caused in damage as the direct result of malware in general, and a closer look at [22] shows that the majority is caused by botnet related attacks.

Looking at the problems and potential future work proposed by [18]. It was easy to see that detecting botnet is not an easy task. Given the functionality of a bot it is easy for a botmaster to change part of the source code of the available bots, and thereby altering some of the behaviours of the bot, potentially making them harder to detect. That botnet are hard to detect is one of the reasons that botnet is still a hot topic. section 1.5 highlighted, some of the potential future work for research of botnet detection methods, with focus on the data sets.

- Data sets
  - General
  - Capture level
- Number of different botnet malware used
  - Families
  - Samples

Capturing data sets, both for training and evaluation, is an essential part of investigating botnet detection algorithms when based on MLA. A good data set for training a detection algorithm should have little to no False Positive (FP) and FN when exposing it for several different evaluation data sets. The problem is, that it is not a trivial thing to create good data sets, as stated previously is section 1.5, that every research paper, uses different data sets. There are no common way to verify different detection methods against each other, as the very foundation they are built upon, that being the data sets, are different. As a side note to the data sets, the capturing level of the data sets, are not equal. As some researchers might have access to ISP networks to gather data, others only having access to self produced data sets that have been created in a controlled environment.

As the capturing interface level can be different, so can the number of botnet malware precent in the data sets. It was shown in [18], that most approaches only have a single or a couple of different botnet malware present in the data sets. While others have upwards of several hundreds. Although, not to discredit any attempts to tailor a MLA to detect a single version of bot malware, as their results speaks for them self. It still poses the question: "*How will a tailored MLA work with higher presence of various bot malware?*".

Putting these points together, and a solution to this is having a publicly available system, that will allow comparison between existing detection methods based on the use of MLAs. The system will provide data sets for every MLA, that has been made available to the system. When testing the performance of the MLA, the data sets will be split up into training and evaluation data sets, here the same training data sets will be the same for all MLA. This will provide a fair comparison between the different MLA. This leading up to the problem statement for the project:

> "*In order get a better comparison between botnet detection using machine learning algorithms. How to design a system that is publicly available and provides the same data sets for every algorithm, enabling easy comparison?*"

From the problem statement, arises a question pertaining to the data sets, that also has be explored throughout this project, that requires investigation before designing the system mentioned in the problem statement:

> "*As data sets are a fundamental requirement and challenge of any machine learning algorithm botnet detection method. What is/are the best practice for generating data sets?*"

This question has two parts to it. The first part is to look into the data sets them self, how to capture the traffic properly, and what implications the different methods of traffic capture has. The second part is labelling the data correctly, as it was hinted at in subsection 1.5.1 small errors in the labelling can cause MLA to make FP or FN.

In chapter 2 the question of best practice for generating data sets is presented, this includes capturing data sets, how to label the data sets with respect to malicious and non-malicious traffic. Finally, the chapter discusses how to create training and evaluation data sets when merging data sets together before presenting the conclusion, that covers how data capturing and labelling will be done throughout the rest of the project. A high level system design is presented in chapter 3, introducing the systems use cases as well as flowcharts in the form of activity diagrams, this leading up to the system requirements, that is based on a variation of the MoSCoW model [25]. A more thorough system design is showcased in chapter 4, this system design is based on the implementation of the system, and includes a front-end and back-end system. In chapter 5 the final system is put to the test, from the system requirement, several tests are designed in order to verify every requirement of the system has been fulfilled. Lastly in chapter 6 the conclusion and discussion for the project will be presented.

Chapter 2

Data Set Challenge

One of the main prerequisites for guaranteeing any reliable performance evaluation of detecting botnet traffic, is in the network traces used for both testing and evaluation the MLAs. When using this data, whether the MLA is supervised or unsupervised, requires the substantial knowledge of the data i.e. what is non-malicious traffic and what is malicious traffic. This can be split up into three main topics, namely how to obtain network traffic in section 2.1, how to label the traffic correctly in section 2.2 and in section 2.3 how to merge the obtained traffic into a training and evaluation data set. This leading up to a couple of ways data sets and the corresponding labels can be used to MLAs, and with a conclusion of which method will be used for this project in section 2.4.

2.1 Datasets

As mentioned in subsection 1.5.1, when generating data sets, the network interface, at which the capturing takes place, should to stay the same, to keep the traffic consistent. For capturing traffic in a LAN environment, it would likely result in a single to a few botnet family/samples due to the small amount of online devices compared to e.g. an ISP. At a campus environment, upwards of several thousand devices are online doing the day, thus the number of different botnet families/samples is also likely to increase. Given the size of a campus environment, a small insight into the coordination between bots of the same family/sample could be present. At the ISP/core network even more communication is available, thus a larger insight into the coordination between botnet families/samples could be present. The amount of traffic does obviously also increase as the network interface shifts to larger networks.

When trying to generate botnet traffic, there are two possible ways of doing this. Capturing botnet traffic in a live setting or try to generate network traces synthetically via a controlled environment e.g. HoneyJar projects [5]. Each of these methods has they own inherent advantages and disadvantages.

2.1.1 Real Traffic

Capturing botnet network traffic in a normal LAN, Campus or ISP/Core network, also known as *in the wild*, where the data is generated by computers owned by normal users and where botnet traffic is being used for illicit purposes, might seem like the obvious choice. It is however never that straight forward, although this methods has advantages it also brings some disadvantages that one needs to consider.

- Advantages
  - Realistic traffic
- Disadvantages
  - Privacy
  - Labelling
  - Challenging to acquire high level network interface traffic (i.e. campus/ISP)
  - Guaranteed bot samples
  - Representative traffic

Given that the network trace is taken on normal operating networks, the trace can be used to represent both the malicious and the benign data sets. As the bot malware is connected to the network, CnC servers (centralised structure) have the possibility to push out updates to all connected bots, it can therefore be argued that capturing traffic in the wild, will reduce the number of bot variations within a botnet family. The captured data sets from real traffic, is the best possible data available for research, as botnet detection software should be able to find botnet traffic in the mists of real traffic.

All of this requires that bot traffic is present on the network, as this can not be guaranteed. Given that bot traffic is present, traffic captured for research purposes, has better evaluation performance the newer the data sets are, as newer data sets have more up to date botnet traffic and non-malicious traffic patterns.

Even though using real traffic provides the best possible data, there are also some severe drawbacks of using it. One of the main drawbacks, is that the network trace is not labelled, and has to be done manually or by using an automated process (e.g. Snort) that can label the data, which can not guarantee 100% correct labels. The requirements of using an automated process is high, as wrong labels, can make a MLA make wrong predictions, thus making either FP or FN. The features of these labels will be expanded upon in section 2.2.

When capturing data sets, considerations should also be put into, what kind of setting the traffic should be in, asking the question "*What traffic is representative?*". Traffic that is captured from a home network of a couple in their 20's are different than the traffic captured from a couple in their 60's. Capturing on a campus or within a company is also significantly different, both to each other but also compared to a home network. A campus should likely have more research related traffic while the traffic at a company could be different based on which type of company it is, as a cleaning company would have different traffic than the employees at a Internet Service Provider.

Another drawback is that capturing botnet traffic packets "in the wild" is legal/ethical aspect, as capturing packets in the wild can compromise the privacy of users. Although some of the major websites (e.g. Reddit and Facebook) provide encrypted connections, so that the payload can not be inspected, not every site offers this, and would therefore require the users consent, which is only feasible for smaller networks. Even if the connection is encrypted, the meta-data is not, and this can still be used to compromise the privacy of users.

One of the more practical issues with using real data is the availability of the data, at least on higher level network interfaces, where the most diverse traffic appears and the best possible way to observe the coordination between bot samples in a botnet family. Obtaining this data goes back to the privacy aspect of the users. While there is also the aspect that a network with the size of a campus, ISP or core might not be willing to simply forward network traces.

2.1.2 Controlled Traffic

In contrast to using real traffic, where everything is caught in the wild, is to create network traces in a controlled environment. The controlled environment, often reffered to as a Honeypot setup, is one of the perferred ways of gathering data sets [14] [26], although both papers do not, them self, make a controlled setup, they both use data sets which originated from another controlled environment such as [7].

A Honeypot setup is often small controlled network environment, and can be done it two different ways. The first setup will require that the network is open for the outside where the machines will be left vulnerable for exploits, simply waiting to be infected with malware, although this is a potential "waste of time" if there has been no attempts to infect the machines, but this method will provide some insight into how malware infects vulnerable machines. The second setup is where the machines are purposely infected with for example bot binaries, still method is sure to generate some bot related traffic.

Traffic generated in a controlled has advantages and disadvantage that will be listed here. Throughout the rest of this section, synthetic traffic will be used referring to traffic generated in a controlled environment.

- Advantages
  - Privacy
  - Labelling
  - Controlled environment
- Disadvantages
  - Realistic traffic
  - Scope
  - Network interface level
  - Legal/Ethical aspects
- Potential
  - Fully control the activity

A huge advantage of creating synthetic network traces, is that it is being done in a controlled environment. This can be done with a HoneyJar setup as previously mentioned. In the HoneyJar setup are HoneyPots, these are referred to as the vulnerable computers that are to be intentionally infected with bot malware. The malware can be with both binaries i.e. the executable, or with the source code the malware.

With software compiled into binaries, it is not possible to change anything regarding the functionalities of the software. Available malware in binaries are therefore as intended by the creator of the malware intended it to be. This ensures optimal realism of the traffic produced by the malware, since the configuration can not be changed with binaries. Older malware might not function as initially intended, this could be due to CnC servers that has been taken down, either "shutting" down the bot as no instructions has been received. Other bots (e.g. a variation of Agobot) has been shown to start attacking certain sites if the CnC server is non responsive [17], it was however not found whether or not the bot attacked due to the missing CnC server, but the CnC server had been taken down. Although bot binaries ensures best possible realism in terms of the behaviour of the bot, an Internet connection is required, this makes it possible for the bot to potentially make a DoS attack, be part of a DDoS attack, SPAM campaigns etc. To ensure that the bot does minimal damage to any outside systems by for example a DoS attack, the amount of bandwidth available is minimised, therefore if the bot is making an attack, the amount of packets that has been emitted by is relatively low if the upload was not limited.

When access to the source code is given, this effectively enables modification to the bot malware. By changing the source code, the original CnC server could be changed to a fully controlled CnC server, thereby ensuring that the victim computer is not sending vulnerable information to a botmaster, but is kept on the local CnC server. Changing the CnC server in the source code should not have any affect on the behaviour on the bot it self when executed. One of the drawbacks of using this method is the availability of bot source code, compared to binaries. Second is the authenticity of the source code, as for example small parts of the code has been modified or the source code is build upon reverse-engineering a bot binary etc. Factors that can play a role in the behaviour of the bot.

The last advantage is privacy concern, that comes with gathering network traces in a controlled environment. When doing this in the wild, as mentioned before, if the content being send is encrypted, it is possible to see sensitive data, although not what a user is communicating to certain service or person. When creating synthetic botnet traces, there are no real users. As the network is set up, with the intent to deploy bot malware, a user would not be present unless asked and given consent of what the purpose of the environment is for.

By using a controlled environment, where the traffic is known (i.e. the traffic not caused by the bot). This provides an easy setting for labelling the traffic, as normal background traffic can easily be eliminated in this setting, where the only traffic contribution is from the bot malware it self. When trying to compare the synthetic trace to the real traffic trace, the former clearly lack realism in the traffic. This is how ever unavoidable when creating synthetic traces. Creating a small environment also has the disadvantage that the network scope, potentially is much smaller than when caught *in the wild*, as there is no "easy" way to acquire higher network level traces.

In the case that the source code for the bot malware is not available, there are some legal and ethical concerns that comes from deploying bot binaries malware on machines. When a bot binary is executed there is a possibility that the bot will immediately start contributing to an attack campaign, and this can have some legal implications, but also some ethical implications. When trying to obtain malicious network traces from the malware directly, researchers often try and limit the Internet connection as much as possible, as gaining real insight into how a bot behaves is important, but researchers are still being held accountable for any potential damage caused by the malware they deployed.

2.1.3 Benign Traffic

Assumptions have to be made in order to only capture benign traffic. If the computer has been had a clean install of a fresh Operating System (OS), is can be assumed clean, and only produce benign traffic. Some of the advantages and disadvantages of capturing benign traffic:

- Advantages
  - Labelling
  - Realistic
  - Data
- Disadvantages
  - Privacy

  - Clean hosts
  - Representative data

Given the assumption that the devices used in a network is clean, the labelling is easily done as all traffic would be benign. But as malware propagate through different vectors such as drive-by-download or vulnerability exploitation, a host can be infected within within a small time frame or years of connectivity. It all depends on the kind of applications installed, website visited or bad (lack) of security.

The benign traffic is as realistic as can be, as it is with botnet traffic caught in the wild (subsection 2.1.1), where the only real difference is the assumption of clean hosts. As stated upon in real traffic, to define what is representative traffic has to be decided upon, as traffic patterns changes depending on the location e.g. at home, at the office etc. And traffic that might be representative for a certain user group in one country might not be representative for a similar user group in another country.

As it was with capturing malicious traffic in the wild, privacy/ethical concerns are also present when capturing benign traffic on public networks. This can however be solved by creating a small environment, based on a Honeypot project, without infecting the clients. This approach will create bias, as the users will not be able make a representative part of the population traffic patterns. This approach also helps in the labelling, as it will be easier to make sure all computers within the network are clean.

2.2 Labeling Data

The labelling of data is used to identify whether a packet, flow or host is malicious or non-malicious. Depending on the labelling resolution (packet, flow or host) of the data sets, created based on section 2.1.

Given a MLA labels based on malicious and non-malicious packets, and the ground truth labelling of the data sets are based on flows. The the MLA is not able to get credit for all correctly/falsly labeled packets, but will only get credit for malicious flows, if a single packet within that flow is labelled as malicious. This would require that the MLA would also output which flow id the packet where assosiated with. Had the data set been labelled according to hosts, the MLA would have to include the IP of a malicious or non-malicious packet.

The labels for the data sets are seen as the ground truth. A post comparison of the labels can only produce label resolution based on the bottlenecks label level, this is outlined in Table 2.1, where it is clearly seen that if an algorithm produces labels according to malicious and non-malicious hosts, then the outcome of the comparison can only be treated as host based labels.

Data set / Algorithm	Packet	Flow	Host
Packet	Packet	Flow	Host
Flow	Flow	Flow	Host
Host	Host	Host	Host

Table 2.1: If the data set label level is host, and the algorithm labels according to flows. The resulting compared labelling will only be able to tell if the algorithm found the malicious hosts, as the data set labelling is not able to provide more information.

Each of these methods of labelling, has its own advantages and disadvantages and will be outlined based on data set labelling. The algorithms will be treated as unknowns to the system (see section 3.2 for how this is implemented).

2.2.1 Packet label

Labels on packet level, will provide the best insight into which packets are directly malicious, but labelling specific packets as malicious is not a simple task. In this section some of the advantages and disadvantages will be presented:

- Advantages
  - Accuracy
  - Target malicious packets
- Disadvantages
  - Labelling
  - Definition of malicious packets
  - Unnecessary information

When using packet labelling, all malicious and benign packets has to be known and correct. Given a MLA that produces packet level labelling, the performance of the algorithm is tested to the fullest extent, as every packet is labelled, and TP, TN, FP and FN are given to every packet. Given this, it can easily be seen which algorithm is better at finding packets are specific bot families/samples, and also which it treats as non-malicious, this can provide a solid insight into improving MLAs for detecting some bots more accurately or more diverse bots. If an MLA is tailored towards a single bot family/sample, this approach can also provide some insight into the algorithms performance towards other bot families/samples. Intuitively a tailored MLA will produce more FP and FN compared to generic botnet detection MLAs, as the tailored approach should not be able to identify as many bot families/samples as a generic method.

In [18] 13 out of the 20 papers investigated reported that the MLA analysed traffic based on flows, thus also states if the flow is of malicious or benign origins. If terms of comparing packet level labelling from the data sets to flow level labelling from the MLAs, where the "bottleneck" is the MLA, and therefore if a single packet in a flow is malicious, then the whole flow is to be considered malicious, the same is valid for packet to host comparison. Given that a host is being attacked from the outside, and the packets from this attack is labelled as malicious, the as the host did not initiate the connection, it is not safe to assume that the host is malicious. The resulting labelling will be less accurate than had the MLA used packet level labelling, as it was presented in [18], the approach used by the majority is the flow based method. The data set labels needs to created with respect to this, thereby adding more information to the truth about the labelling than otherwise. In the case of pure packet labelling the labels could look like: <packet number>, <B/M>. Where the packet number is provided from capturing packets using e.g. Wireshark, and B/M being labels for either Benign or Malicious. Adding compatibility to a flow based labelling algorithm, an obvious information to add to the data set labelling is <stream id>.

Comparing data sets with packet level labelling to MLA that produce host based labelling, overall accuracy is lost, as the only output that can be produced from this setup is finding malicious and non-malicious hosts. When looking at host level labelling, the data set labels are required to contain the IPs within each packet label. As there is otherwise no way to combine a labels from packet level to labels from host level.

One of the main drawbacks of using packet to packet labelling, is the labelling. As only the packets contributing to malicious activity can be considered malicious, while the other packets in the same flow are not considered malicious, which also brings up the question "how to define malicious packets?". Depending on the definition a packet in a flow with malicious content can be defined as malicious, as the connection is to server that hosts malicious content. For this project, the definition malicious packets are defined by the payload. An example of this is a flow that sends information to a CnC server using http(s). The three way handshake can not be considered malicious in a packet level labelling, as a three way handshake by nature is not malicious. When the flow start transferring a payload containing for example keyloggings, these packets are to be considered malicious. While the last RST packet, like the three way handshake, can not be malicious. Due to this the labelling would have to be precise. As any mislabelled packet could "confuse" the MLA, creating more FP and FN as a result.

One could argue that used packet level labelling on data sets with MLA that labels according to packet, might not be the best approach. As the FP and FN might be artificially high, simply because of the large amount of packets that needs to be labels individually, which almost per definition can not be labelled with respect to statistics, as this would imply flow analysis effectively making an MLA that labels according to flow. Looking at the implementation, of an MLA that identifies packets being malicious or benign, into a corporate environment, where a detection algorithm eventually could help the company. The approach might be to costly, in computational power as well as time, compared to a flow or host based.

2.2.2 Flow label

The flow based labelling is commonly used for research purposes as presented in [18]. This could stem from, research botnet detection was the state-of-the-art MLA detection, in [8] flow based labelling where used to try and identify different applications, [8] also states that traditionally classifications methods used packet labelling, primarily using port numbers to classify the traffic. They follow this up with "An application may use ports other than its well-known ports to avoid operating system access control restrictions". Some services that can use a well-known port, such as Secure Shell (SSH) often recommend changing the default port from 22 to a random higher port number. This is an advantage flow based labelling has over packet based labelling. There are some more advantages to flow based labelling as well as disadvantages:

- Advantages
  - Default research label level
  - Potential identification of bot

    - Usable on host label level

- Disadvantages
    - Labelling
    - Can not benefit packet level labelling algorithms
    - Definition of a flow

Looking at flow labels for the data set and MLA labels being packet labels, has the same limitations to the end result as previously mentioned with data sets being labels according to packet level and the MLA labels with respect to the individual flows. The same goes for MLA that labels according to hosts, as it is not possible to get better "resolution" than host based labels, as this would be the bottleneck in the labelling process.

Given that the data set labelling and the MLA labelling are both flow level labelling, is the optimal case for flow based performance detection. In [8] shows that using flows, and thereby flow level labelling, it is possible to use MLA to identify specify applications, this could also be applied for botnet detection. This could allow for quick classification of which bot a machine is infected with, from there finding counter measures to deal with a specific bot malware.

Flow based detection, and thereby flow level labelling, seems to be the method of choise for most research investigations, many of these having good results, some of which was presented in [18]. This clearly indicates that the flow based labelling has huge interest in the research community. Given that flow based labelling is the default choise for many research purposes, this also provides some challenges for provided a system that performs benchmarks of botnet detection MLAs. An exmaple of this is the very definition of a flow, while the definition of a flow might seem easy, there are several definition of a flow. As such [12] defines packets in a flow as "a set of IP packets passing an observation point in a network during a certain time interval". Effectively stating that every packet within with the same header information, destination IP:port, is defined as the same flow. In [13] defines a flow as "a sequence of packets sent from a particular source to a particular unicast, anycast, or multicast destination", later also stating that a flow could consist of all packets that are sent from a source to a destination using a single connection. But also emphasises that this can not be mapped 1:1. Another definition of a flow could be a connection starting with e.g. three-way handshake or the first packet to a new host:port and all packets flowing in the connection until a RST packet closes down the connection. This can potentially cause some confusion to the labelling process, as the data sets are labelled using one definition and an MLA uses another definition of a flow.

2.2.3 Host label

Throughout the previous two section, packet and flow based labelling has a lot of assumptions and clear definitions that needs to be defined before they can be used. While this is also true for host based labelling, it does not care about the definition of a flow, as long as a flow is malicious, the host is labelled as malicious, this is also true for packet labelling to host labelling. While there still is the nuance of a host being attacked from the outside, the host would actively have to be initiating malicious traffic before it can be labelled as malicious.

- Advantages
  - Labelling
  - Potential for enterprise usage
  - No definition problems
- Disadvantages
  - Accuracy
  - Can not benefit packet or flow labelling algorithms

A huge advantage of the host based labelling, is the labelling it self. Given a controlled setup subsection 2.1.2 and a non-malicious controlled setup subsection 2.1.3, each pcaps from the different setups are already defined as malicious and non-malicious. No further labelling is necessary, as all hosts from the controlled setup is considered malicious, as bot malware has been running on the hosts in that network. With the non-malicious controlled setup, every host has been cleaned, and can initially be assumed clean. This statement only holds true, as long as the environment in which the data sets where gathered are controlled, as labelling malicious hosts in data sets captured "*in the wild*" are still a difficult process.

It goes without saying, that MLAs that labels on packet or flow level, will not be able to benefit from the data sets being host level labelling. Given for example a corporation, that might not care which flows are containing botnet traffic, but only interested in finding any potential malicious traffic within their local network and clean an infected host. With the assumption that corporations only care about find hosts that are being used in a botnet, the fastest and easiest way is to use the host level labelling.

With host level labelling, a host can only be malicious or benign. There is no need to define flow for the data set labels, and there is no need to define which packets in a malicious flow is malicious. A host is considered bot infected and thereby malicious if a single packet from a host can be labelled as malicious with the approach in subsection 2.2.1, this of course scales up to flow based labelling, as a single flow, with any flow definition applied, that is considered malicious will treat the host as malicious.

2.3 Merging Data sets

A common way to use the captured data sets is to merge them together into a single big pcap file [26] [14], both papers proposed the usage of TcpReplay. TcpReplay is a Unix application that enables replay of traffic data into the network, using this with several data sets allows the merging of a new single data set. After which the data set will be split into a training and evaluation data sets as illustrated on Figure 2.1.

Upon merging network traces together, there are some pre-merging process that needs to be done before hand.

- IP address will have to be changed, ensuring all hosts (malicious and benign) are "on the same" subnet.
- MAC address will have to be changed, ensuring all unique IP has unique MAC addresses.
- All malicious should share IP and MAC with benign hosts.
  - Ensuring all malicious hosts also have benign traffic.
- Changing the packets capturing timestamps.
  - Merging pcaps together will most likely have different timestamps.

Changing the IP and MAC addresses is a fairly simply process and can be done using Wiresharks tcprewrite functionality, the code for this is available on the online

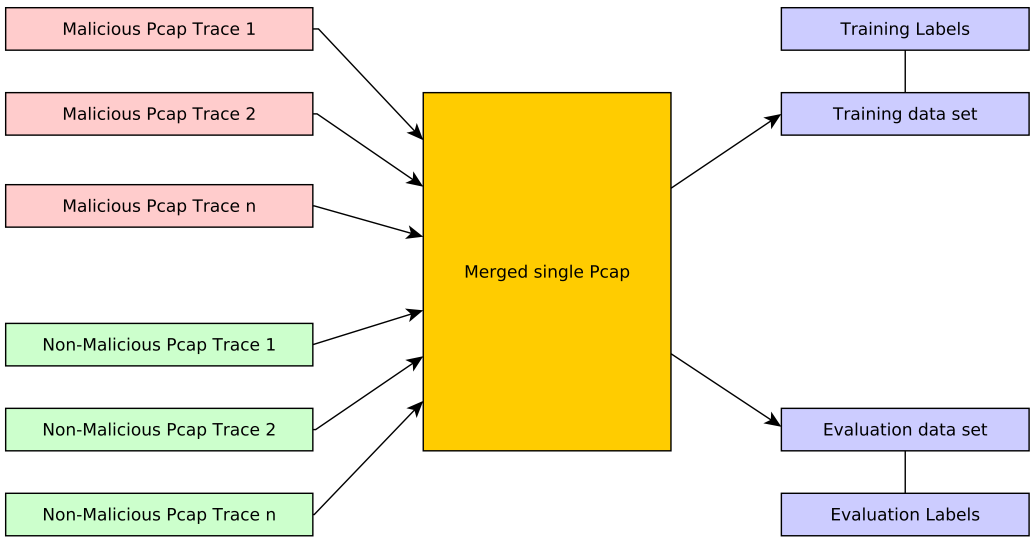


Figure 2.1: Process of obtaining malicious and benign traffic sets, merging them and splitting them into training and evaluation data sets needed for training and evaluating MLA based detection methods.

repository. Changing all the IP and MAC addresses ensure that a detection approach can not "simply" assign malicious traffic to a single range of IPs. Upon changing the MAC address ensures that no single address is assigned to two different IPs, as it could cause some potential confusion when applying detection methods.

Making sure that merging malicious IP and MACs into a benign host, will help with the realism issue. Hosts in a network that has been infected with bot malware, will produce some malicious activity, but as the user of the host machine is unaware of the bot malware, a lot of normal benign traffic is also present. The ratio between benign and malicious traffic is normally far from each other. By merging malicious data sets with benign traffic sets, while still keeping many host clean, the ratio between the two can be kept.

Another thing to be wary about is the packets timestamps, as merging for example two data sets together, a non-malicious data set might have been captured around noon, while a malicious data set have been captured at midnight. If the data sets are merged without taking this into consideration, a MLA based detection approach could make the distinction, that packets captured around midnight are all labelled as malicious. An approach could be to align the first packet across all data sets, and adjust the timestamps in the final data set. However, this could potentially create some bias in the data set. This would be equivalent of either using Wiresharks merge function, that can merge several data sets together into a single one, and then correcting the timestamps, or using TcpRelay to simultaneously replay several data sets at the time same, which would eliminate the need for correcting the timestamps. In [26], [14], [20] and [9] the features the MLA based detection methods did not use timestamps, suggesting that it is not a common feature to use. However, it is something to keep in mind, as some detection methods might look at the actual capturing timestamps.

2.4 Analysis conclusion

This section will conclude the challenge that comes when looking at data sets. The result of which will be used in the benchmark system in the upcomming chapter 3

In order to keep the most realistic data possible, the best method would be to capture it *in the wild*, this was presented in subsection 2.1.1. Due to the drawbacks that is innate with capturing network traffic from a public network interface (all non home networks) and the labelling process required, it is deemed infeasible to pursue this approach any further.

Thus traffic generated from a controlled environment, releasing malware would be the obvious choise, but due to the setup required for generating malicious traffic, such as a HoneyJar network will not be pusuded. Projects such as the Malware Capture Facility Project (MCFP) [11] will be used instead. Every data set from the MCFP comes with labels of the traffic (see section 2.2) with more than 150 different traffic captures, each containing malicious traffic.

For benign traffic, the same approach, as the malicious traffic, will be used. The data sets here will be made from previous projects, that created a controlled environment with no bot activity present. Although both the malicious and the non-malicious traffic capturing approaches are less the optimal, they still provide a solid foundation for the data sets.

In section 2.3 the idea of merging data sets together into a single data set, for thereafter splitting it up into training and evaluation data sets with appropriate labels, will not be pursued any further. When the data sets are split apart, the system will become more modular, as it will be easier to "play" with the ratio of training data vs. evaluation data. It also has the advantage that adding more data to the pool of data sets is relatively easy. A drawback of this method is that algorithms authors has to make sure that their MLA can take in several pcap files from a folder location. As the data sets is not being mixed also creates another problem, as the pcap folder locations will have to be labelled as evaluation data set malicious or evaluation data set non-malicious and so on, this can potentially cause authors to try and cheat the system, as they from the very beginning knows about the evaluation labels.

As stated throughout section 2.2, using packet level labelling or flow level labelling, has a lot of assumptions that needs to be clearly addressed. Clear definition, on malicious packets or even how to define a flow. Using both of these could potentially cause confusion between the data sets provided by the system, and the creates of the MLA based detection methods that are uploaded. Although both packet level labelling and flow based labelling increases the overall benchmark of the MLAs, there are also fall groups, such as the labelling it self. As stated throughout the section, the labelling has to be perfect, as it can otherwise affect the MLAs labelling process negatively.

Running MLAs that labels according to flows, or packets. Looking at the current state-of-the-art MLA based detection methods as presented in [18] are flow based detection. However as it was presented in subsection 2.2.3, in a live environment, where a botnet detection MLA is running and operational, detection which flows are malicious in nature, might be considered to complex, as the only information needed in an corporate environment is which hosts are potentially malicious. Therefore, the remainder of the project the labelling will only be on host level labelling.

To summarise the analysis conclusion, moving forward in the report the following will be used.

1. Malicious pcaps collected from sites such as MCFP[11]
2. Non-malicious pcaps collected from previous projects [16] [1]
3. The pcaps will not be merged into a single large file or two separate files for malicious and non-malicious
4. Pcaps will be placed in folder according to:
    (a) Malicious training
    (b) Non-malicious training
    (c) Malicious evaluation

   (d) Non-malicious evaluation

5. Host based labelling

Chapter 3

System Design

Throughout this chapter, the design aspects of the system is presented. A high level overview in the form of use case in section 3.2, this gives a good indication of how the system functions. This is followed up by activity diagrams in section 3.3. Finally, leading up to the system requirements in section 3.4

3.1 System Description

In this section, a short introduction to the design of a system, that enables easy comparison between different botnet detection algorithms, be it MLAs or non-MLAs will be introduced.

It was stated in section 1.6, that the system should be publicly available system. For this a webserice will be used for the front-end system, that allows users to upload botnet detection algorithms. The uploaded algorithms, are all tested with the same training and evaluation data sets, which will follow the conclusion of section 2.4. When the evaluation of an algorithm has been performed, the results will be presented to the user, together with the other tested algorithms. The front-end system will provide access to the system, upon upload of a detection method, the back-end system performs a benchmark.

As the front-end system allows users to upload MLAs, small modifications would be needed in order to allow users to upload new data sets with corresponding labelling data. This could greatly increase the amount of data sets available to the system. When uploading new data sets with labelling data, they should not be put into "play" immediately, as the labelling data would have to be verified. As stated in section 2.2, bad labelling could course confusion for MLAs. Since this is not directly stated in the problem statement, as a requirements for the system, the ability to upload data sets will not be pursued, unless time allows it.

3.2 System Use Case

This section is presenting a high level use case diagram and description of the system. The use case diagram of the system is presented in Figure 3.1. A user interacts with the system, from here the user can do one of two things, look at algorithm results, or upload new algorithms or data sets.

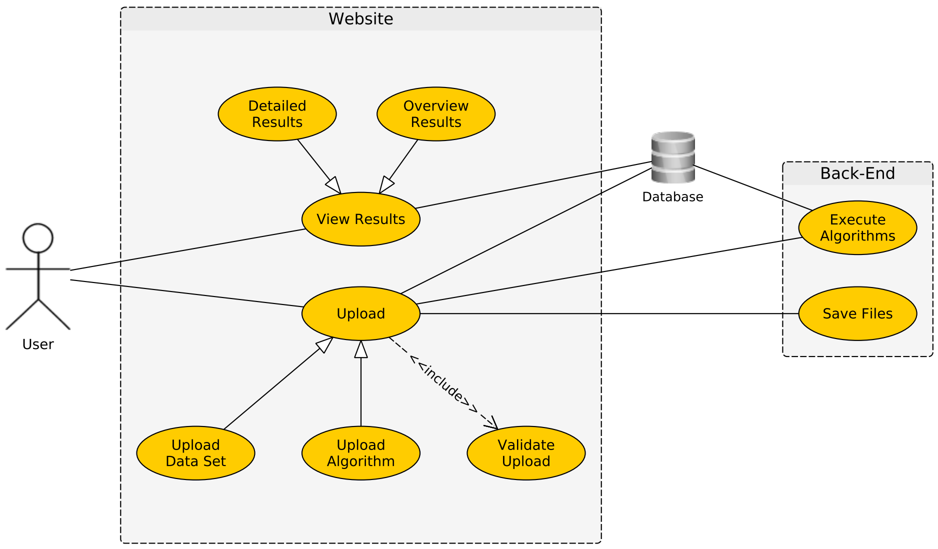


Figure 3.1: A user can interact with the system through a website allowing either viewing results, which are generalised by result overview and single detailed results, or by uploading detection algorithms or data sets, which have to be validated before it is uploaded to the back-end system and inserted into the database..

Should the user want to view the results of the detection algorithms, the "view results" is an generalisation of a complete overview of the results and a more detailed view of a single result, as "view results" can be either. The appropriate results are being fetched from a database connected to the front-end system. Should the user want to upload, either an algorithm or a data set, the upload will have to be validated first, before the upload is accepted, whereby the database will be updated, saving the data set or algorithm. Given that the upload is an algorithm, it will also be executed, and the results of the detection algorithm will be uploaded to the database, after being executed.

It is not intuitive from the use case, how the detection algorithm should interact with the system. There are two ways algorithms can be implemented into the system. The first is, where the algorithm will have to be checked and verified manually. This solution can potentially be extremely time consuming, as each software developer programs differently, and some are better at in-line documentation than others. The second is, where each algorithm is treated as a black box. Treating algorithms as complete unknowns, requires that the input and output of the algorithms are well defined. The system will only be able to interact with the algorithm with inputs and the algorithm interacts via the output result. In Figure 3.2 is a simplified view of how the inputs and outputs are defined. As previously stated in section 2.4 four folder with data sets will be made. Two folders containing malicious and non-malicious training data sets and two for malicious and non-malicious evaluation data sets.

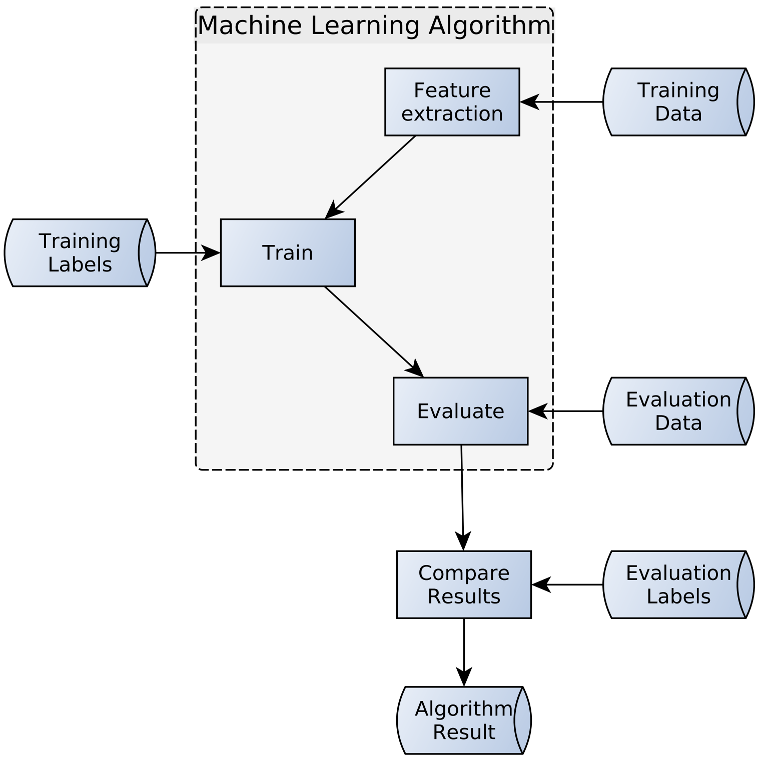


Figure 3.2: Input and output relation of the uploaded algorithms

The reasoning behind Figure 3.2 is, whether the MLA is supervised or unsupervised, that there are certain steps that needs to be taken. In the case of a supervised MLA, it takes in training data set and the correct labels of the training data. From this, the algorithm looks at selected features of the training data which is specified by the algorithms author.

When detecting botnet traffic, a statistical approach is made. The statistics will be based on the source/destination IP and port, as well as the protocols. From there the statistics change depending on the choices made by the author of the MLA. A few features that can be extracted: the connection period in milliseconds, mean packet length, number of bytes from source to destination, number of packet from destination to source, etc. When looking at the supervised MLA, these features are used to tell if a packet, flow or host is malicious or non-malicious. For unsupervised MLA, the labelling is not provided for the training data, but an unsupervised will try to put packets, flows or hosts together. These are based on which are most similar in respect to the features selected for the statistical purpose.

When the MLA has been trained using the training data, and labelling data if the algorithm is supervised, new data is being given as input. Where as the MLA will try to associate the new data, given statistical knowledge it has from the training period, to be either benign or malicious. The end result labelling should tell which hosts are malicious and benign according to item DataSet.5. in section 2.4

As the data sets are not merged, but located in their respective folder with training and evaluation both with malicious and non-malicious data sets defined in item DataSet.4.. The out of the MLA based detection method will have to be:

- <file>-<IP>
- <M/B>

This will also be restated in the system requirements in section 3.4. The comparison between the true labelling and the MLAs labelling, will produce four different outcomes depending on the labelling from both the MLA and the data sets. These are:

- True Positive. System labels states "Malicious", so does the detection method
- True Negative. System labels states "Benign", so does the detection method

- False Positive. System labels states "Benign", detection method states "Malicious"
- False Negative. System labels states "Malicious", detection method states "Benign"

3.3 Activity Diagram

From the use case, there are three components to the system:

- Users
- The website / front-end
- Back-end + database

In the following two subsections, a walkthrough of a high level activity diagram, is presented for both the "upload" and "view results" use cases. The activity diagrams is used to showcase a flowchart, depicting the actions each of the three components will have to do, in order to see "view results" and upload.

3.3.1 Activity Diagram: Upload

In Figure 3.3 the acticity diagram for the upload use case is presented.

In Figure 3.3, a users submits an upload, the upload is verified. If the upload is not verified the user is send back to the upload screen. Given that the upload was verified, the website processes the upload, gathering the information needed for the back-end system. The gathered information is stored in a database, the file (i.e. algorithm or data set) is saved locally on the server that runs the back-end file system. If the upload is a data set, the process has ended, as there is no additional steps to be made. If the upload is an algorithm, the back-end will try to execute said algorithm, with the available data sets. As explained in section 3.2 the MLA has to produce an output of <file>-<IP>, <M/B> these labels are compared to the system ground truth about the data sets labels. Lastly the results are uploaded to the database.

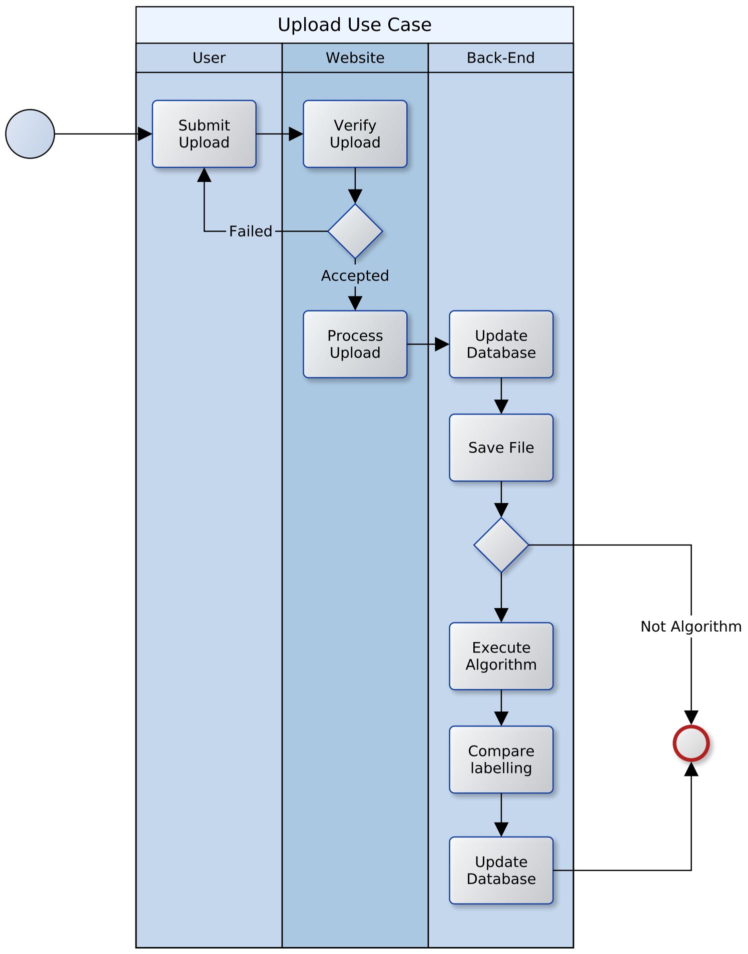


Figure 3.3: An activity diagram of the sysetm flow, with respect to the three components in the system, for uploading an algorithm or data set.

3.3.2 Activity Diagram: Results

In Figure 3.4 the activity diagram for the "view results" use case is presented.

The user requests the overview results, as this will be the natural first step of the view results part. The data is fetched from the database, the website will before presenting the results, process the data, from where the user is able to see an overview of the results. From here the user can stop looking at the results, or request a specific test result from a specific MLA. The website will request new data from the database, which has to be processed by the website.

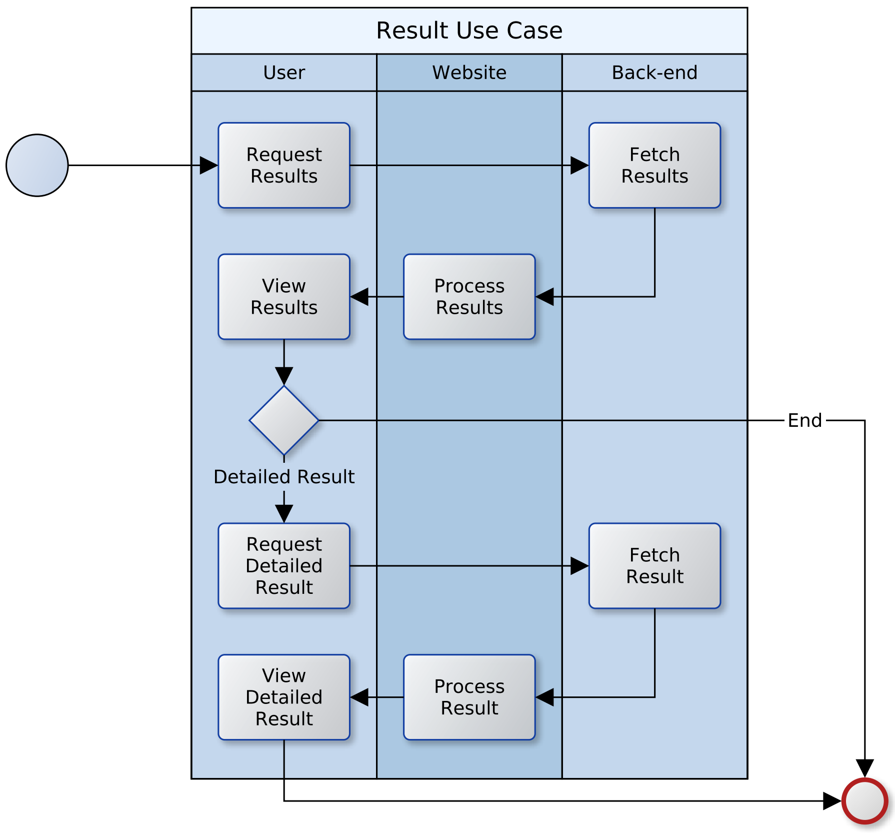


Figure 3.4: An activity diagram of the system flow, with respect to the three components in the system, for viewing the MLA results

The overview results will be presented as follows:

$$\text{True Positive rate} := \frac{TP}{TP + FN} \tag{3.1}$$

$$\text{False Positive rate} := \frac{FP}{FP + TN} \tag{3.2}$$

These will give a good indication of how good a MLA is, and according to [18] the most common performance metrics. With the detailed results, more performance metrics are presented. While the detailed result page will also show general information about the algorithm e.g. author, upload date, classifier type etc. Finally, the performance metric presented in the detailed result view:

$$\text{True Positive rate} := \frac{TP}{TP+FN} \tag{3.3}$$

$$\text{True Negative rate} := \frac{TN}{TN+FP} \tag{3.4}$$

$$\text{False Positive rate} := \frac{FP}{FP+TN} \tag{3.5}$$

$$\text{False Negative rate} := \frac{FN}{FN+TP} \tag{3.6}$$

$$\text{Accuracy rate} := \frac{TP+TN}{TP+TN+FP+FN} \tag{3.7}$$

$$\text{Error rate} := \frac{FP+FN}{TP+TN+FP+FN} \tag{3.8}$$

$$\text{Precision} := \frac{TP}{TP+FP} \tag{3.9}$$

3.4 System Requirements

The requirements for the system, have been specified throughout the report. The objective of this section is to collect all the requirements and create an overview of all the requirements. While there might be new requirements presented in the system requirement, the basis for these requirements have been presented.

The system requirements will be presented according to the MoSCoW model [25], in which the design principles are: mush have, should have, could have and won't have. As the MoSCoW model is used for full scale projects, the definitions of the MoSCoW model might be different here than from the standard definition.

Must have are the requirements that are essential for the system to operate. If all of these requirements are not fulfilled, the system will not be able to operate as intended.

Should have are requirements that will greatly improve the system, but not essential for operation. Should have requirements includes operation enhancement, information gathering etc.

Could have are requirements that are "nice to have" functions for the system. These requirements includes graphical design, easy to interpret information display etc.

Won't have are requirements, that will not be met. These will not be met due to lack of importance and/or time constraint.

Any of the requirements can be remarked with a statement of why a requirement is a "won't have".

3.4.1 Front-End Requirements

As stated in section 3.1, a webservice would be created, thus requiring an interface. Throughout section 3.2 and section 3.3, the need for uploading MLA and the possibility for uploaded data sets have been established. The same section also presented some requirements for viewing the detection method results, while there have been added a few to the system requirement in the presentation of results. These are part of a natural evolution, when restrictly looking at host based detection.

1. Presentation (Website)
    (a) The website must have a pure html based interface
    (b) The website could have a graphical web interface
2. User upload
    (a) The website must have the functionality for users to upload MLAs.
    (b) The website won't have the functionality for users to upload malicious evaluation data sets
    (c) The website won't have the functionality for users to upload non-malicious evaluation data sets.
    (d) The website won't have the functionality for users to upload malicious training data sets.
    (e) The website won't have the functionality for users to upload non-malicious training data sets.
3. Results
    (a) The website must be able to give an overview of all tested algorithms.
        i. True positive rate.
        ii. False positive rate.

(b) The website must be able to present detailed information about each algorithm tested.
   i. Accuracy rate.
   ii. Error rate.
   iii. Precision.
   iv. True positive rate.
   v. True negative rate.
   vi. False positive rate.
   vii. False negative rate.

(c) The website could be able to present family/sample classification test results.

(d) The website won't be able to show results for different evaluation traffic data sets.

(e) The website won't be able to show results for different training traffic data sets.

The requirements of uploading new data sets have been marked as won't have. From having a service that allows uploading of MLAs to allowing upload of new data sets, are not that far from each other. Allowing users to upload new data sets, has two different concerns to it. A data set can potentially be big, meaning that it requires a huge amount of storage to keep, one which the system does not have (Appendix B). Second is a trust issue, as the labelling of the traffic either has to be trusted completely upon upload. As stated in section 2.2 small mistakes in the labelling can provide wrong MLA predictions. There is also the possibility that each uploaded data set can have the labelling checked, verified and potentially corrected, but this can prove to be a long process if the data set is large. Due to this, the website won't be able to show any results using different data sets than those provided by the system.

3.4.2 Back-End Requirements

In section 3.2 the idea of saving information in a database was presented, and throughout section 3.2 and section 3.3 it was stated what information should be saved in the database. Some fields have been added to the database that was not presented in section 3.2 or section 3.3. These new requirements will help with operations of the system. In section 2.4, it was presented how the data sets for the system would be generated, and also hinted at how MLA should input the data sets into the system. This was further expanded upon in section 3.2, where it was explained, that every MLA is treated as a black box, and how the inputs are defined, as well as the output of the system.

1. Database
    (a) The database must be able to store/access general information about each MLA.
        i. Unique Identifier.
        ii. Algorithm Classifier.
        iii. Algorithm programming language.
        iv. Creator of the algorithm.
        v. Tested by requirement item Back-end.3.a.
    (b) The databaes must be able to store/access test results for each MLA.
        i. Unique Identifier (identical to item Back-end.1.(a)i).
        ii. True Positive.
        iii. True Negative.
        iv. False Positive.
        v. False Negative.
    (c) The database could be able to store file/folder locations for data sets.
        i. Malicious evaluation data set.
        ii. Non-malicious evaluation data set.
        iii. Malicious training data set.
        iv. Non-malicious training data set.
2. Test execution
    (a) For test execution there must be one programming language support.
    (b) For test execution the uploaded MLA must fulfil.
        i. Inputs
            A. String argument (arg[0]) for non-malicious training traffic data set file/folder path.
            B. String argument (arg[1]) for malicious training data set file/-folder path.
            C. String argument (arg[2]) for non-malicious evaluation data set file/folder path.

D. String argument (arg[3]) for malicious evaluation data set file/-folder path.

ii. Output

A. Labelled output of benign and malicious on a 1-to-1 host basis.

B. File location: Current file (the MLA) path.

C. Output file: "results.csv", with the csv defined as <file>-<IP>, <B/M>.

- <file>: Which file did <IP> come from.
- <IP>: The IP address of the host.
- <B/M>: B for benign host, M for malicious host.

(c) For test execution there should be ≥ 2 different programming languages supported.

3. Data

(a) There must be a folder with evaluation traffic data sets, divided into malicious and non-malicious data sets.

(b) There must be a folder with training traffic data set, divided into malicious and non-malicious data sets.

(c) Users won't be able to upload evaluation data sets

(d) Users won't be able to upload training data sets

Chapter 4

Implementation

This chapter presents a high level view of the system that has been designed to fulfil the system requirements (section 3.4). The back-end system is described in section 4.1. The back-end is comprised of the MySQL server and a script that interacts with the database, and is able to execute algorithms. In section 4.2, the front-end system will be presented. The front-end is compriced of a website, which can communicate with the MySQL server placed in the back-end system.

4.1 Back-end System

The back-end system is in charge of data storage, this includes the file system, where new MLAs are being kept, and a database storing the MLAs results and information about each algorithm. The back-end system is also in charge of executing the uploaded MLAs, and comparing the labelling of the MLA to the ground truth of the data sets. This section is therefore be split into three different subsection. In subsection 4.1.1 a short description of the file system is explained. The database and the different tables required for the system is presented in subsection 4.1.2 and lastly, in subsection 4.1.3 an activity diagrams provides an understanding of how the back-end application functions.

4.1.1 File System

In Figure 4.1, a folder tree structure is presented. The project-root folder, stores the back-end application files which will be explained in subsection 4.1.3, it also links to the botnetApp folder. The botnetApp folder stores the uploaded algorithms in the folder uploads. The unique identifier that is given to each algorithm upon upload, provides the name of the folder, to which the algorithm is placed in, the algorithm it self, is also named after the unique identifier. Given that some MLA authors might name their algorithm the same, the system effectively nullifies this problem.

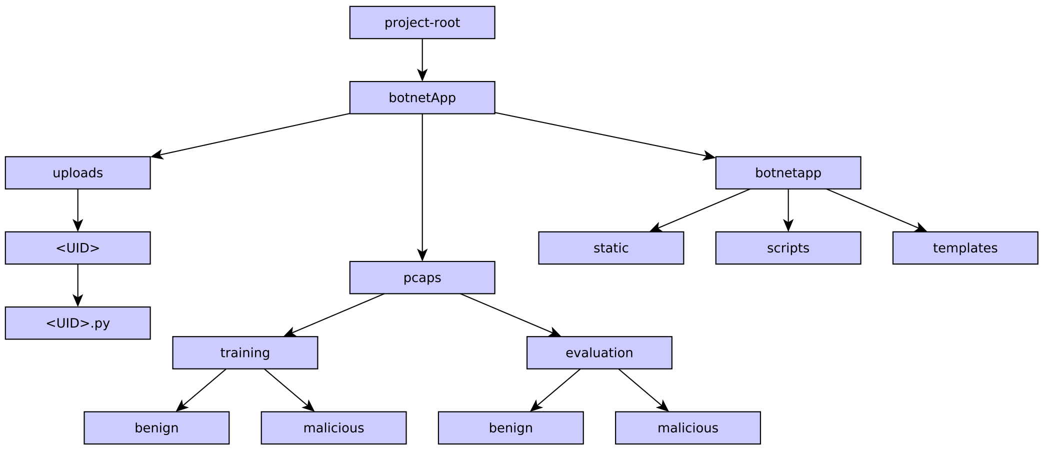


Figure 4.1: A simplified view of the file system, where buttom left corner shows a dynamic file location of MLAs

In the botnetApp folder, there is also a folder called pcaps, this folder is divided into a training and evaluation folder. Both of these folders are split into benign and malicious folders, each containing their respective data sets. These folder paths are fed into the MLAs, and are used for each respectively training and evaluation.

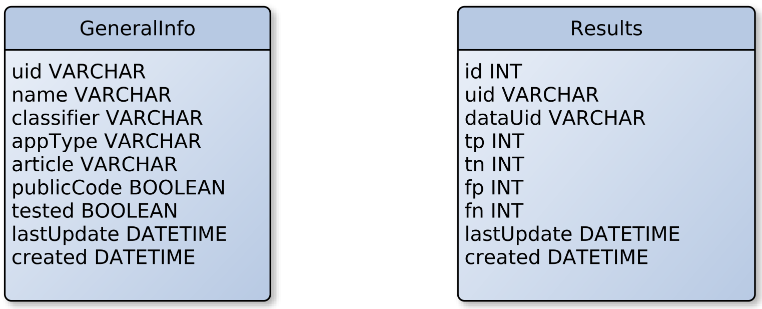
Lastly is the next botnetapp folder, which contains the front-end logic that was elaborated upon in subsection 4.2.1, html templates, css code in static and finally, database modules in scripts.

4.1.2 Database

The database has been created specifically with respect to the system requirements. The tables can be seen on Figure 4.2. There have been created two different tables that collaboratively fulfils the system requirements in respect to the information that needs to be stored for each detection approach.

In GeneralInfo a unique identifier is created. As mentioned in subsection 4.1.1, using an uid the system is able to distinguish the different detection approaches that are uploaded. The "name" variable keeps the original filename, as it might otherwise be unlikely for the algorithms authors to find their tested detection method. As there are many different detection methods based on MLA such as: X-means clustering, random forest classifier etc. The "Classifier" would help any reader to quickly get an overview of which methods a MLA would use. In order for the system to detect which language the MLA is written in: Be it Python, Java, C# or another programming language.

The author also has the possibility of attaching a website to the detecting method, this should be used to link to a published paper, where in the detection approach is presented, meanwhile the author also has the possibility to make the source code public for anyone to download, linking to the algorithm uploaded to the system. Lastly, the tested variable is used to check whether a new detection method has been uploaded to the system, which needs to be tested.



GeneralInfo
uid VARCHAR
name VARCHAR
classifier VARCHAR
appType VARCHAR
article VARCHAR
publicCode BOOLEAN
tested BOOLEAN
lastUpdate DATETIME
created DATETIME

Results
id INT
uid VARCHAR
dataUid VARCHAR
tp INT
tn INT
fp INT
fn INT
lastUpdate DATETIME
created DATETIME

Figure 4.2: View of the tables in the database, Where GeneralInfo stores generic information and Results stores the comparison results

The Results table uses the same unique identifier in uid, as the GeneralInfo uses. This allows distinguishing between the results, so the right result is linked to the correct MLA. The result data is simply stated in true positive, true negative, false positive and false negative. As any statistical information about the efficiency can be calculated based on these metrics.

4.1.3 Application Activity Diagram

Figure 4.3 illustrates what the back-end system is doing, when executing a MLA based detection method. The application consists of three modules, a SQL API which is only used for interacting with the MySQL server, these interactions are, due to the needs of the system, only select and update statements are used. The second component, is the "executor". This module executes all MLAs that has been uploaded to the system, afterwards it runs a script, that compares the labelling of the MLA to the true labels of the data sets. The last component is the main loop, which runs the program in a continuous loop.

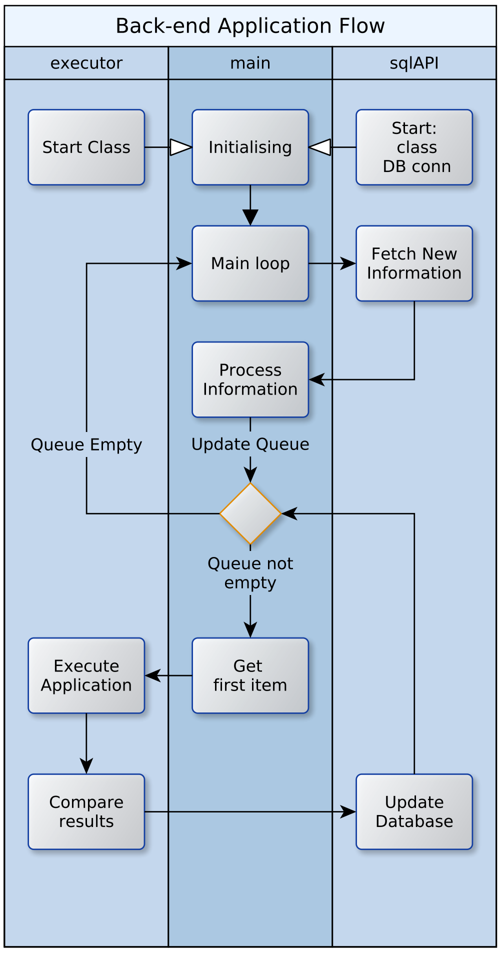


Figure 4.3: An activity diagram of the back-end application, that is used for executing uploaded MLAs

The main loop starts by initialising both the SQL API and the executor classes. The

SQL API is then called to fetch any new uploaded algorithms and is returned to the main loop, if there is nothing new in the database, then the execution queue is considered empty, the main loop shutdown for an arbitrary amount of time, before calling the SQL API again.

Given that the database has new entries, the API class will fetch all the different unique identifiers, in accordance to subsection 4.1.2, that has not yet been tested. Due to the setup of the file system (subsection 4.1.1), the back-end system is able to execute any applications based on the unique identifier, together with the information of which application type (e.g. Python or MATLAB). The executed algorithm will, per the system requirements, provide a result labelling file, which is compared to the ground truth labelling of the data sets.

The result of the comparison is then uploaded to the database, after this, the main loop will forward the next item in the queue. Given that the queue would be empty, the main loop would start over, and request any new information from the database through the SQL API.

4.2 Front-end

In order to make the system publicly available, the front-end system will, according to the system requirements, be a website. The website is created using a Python framework called Pyramid. To install and run Pyramid a small guide has been created and can be seen in Appendix C. In subsection 4.2.1 a short introduction to the framework Pyramid will be given.

The flow of the website is based on the activity diagrams presented in section 3.3, which provided a high level understanding of how the system should work. The activity diagram for viewing results, presented in subsection 3.3.2, is as in depth as it can be. Uploading new detection method will be presented more in depth, as there is more logic behind this method, than viewing results. This will be presented in subsection 4.2.2.

4.2.1 Pyramid Framework

Python pyramid is a non-full stack framework. A non-full stack framework means that, only the server application is provided, and a relatively small library on the

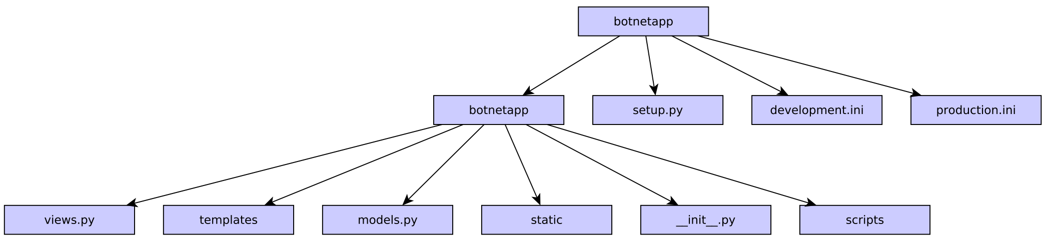


Figure 4.4: A basic overview of the file structure the pyramid framework requires

side, compared to one of the biggest python webframeworks Django. To put into context, Django is also a full-stack framework, and therefore has a much larger library than pyramid does. One of the main reasons that the Pyramid framework was chosen, was because it is a non-full stack framework. Though the library is smaller compared to Django, it also requires less time to get comfortable with, as there is less behind the scenes.

As pyramid is a Python framework, it is also possible to use all the standard libraries that normally come with Python. This makes a Python webframework like Pyramid, as powerful as a normal Python development process. As mentioned in subsection 4.1.1 Pyramid requires a certain file structure in order to work. In Figure 4.4 the botnetapp folder that was presented in subsection 4.1.1 has been highlighted.

The `setup.py` installs any packages that needs to be installed, such as a database API in the form of sqlalchemy, and html templates that Pyramid uses.

The `production.ini` and `development.ini` files are used as configuration files. In the configuration it is possible to set which extensions that are included. In the `development.ini` it is favourable to set a debugging tools as the website would be running the development configurations while under constructions. In the production configuration it would not wise to leave the debugging tool for normal users to use.

Inside the botnetapp folder the `__init__.py` holds the basic configuration of the website it self. The configuration consists of valid path locations, dynamic path locations, initialisation of the connection to the database. In `models.py` a hardcoded version of the database tables is created, this means that inside models, both tables (subsection 4.1.2) needs to be defined in terms of variable types. `models.py` is closely related to the files located inside scripts, as they also have to do with the database and the connection in between. In `views.py` the logic of the website is placed, it contains the instructions of what should happen given a GET request or a POST request. From `views.py` data is sent to the html templates located inside the folder templates, that takes css arguments from the static folder.

4.2.2 Uploading MLAs

Although the high level presentation of the uploaded flow in the activity use case diagram, there are some alteration that is worth pointing out. In Figure 4.5 the new activity diagram is presented.

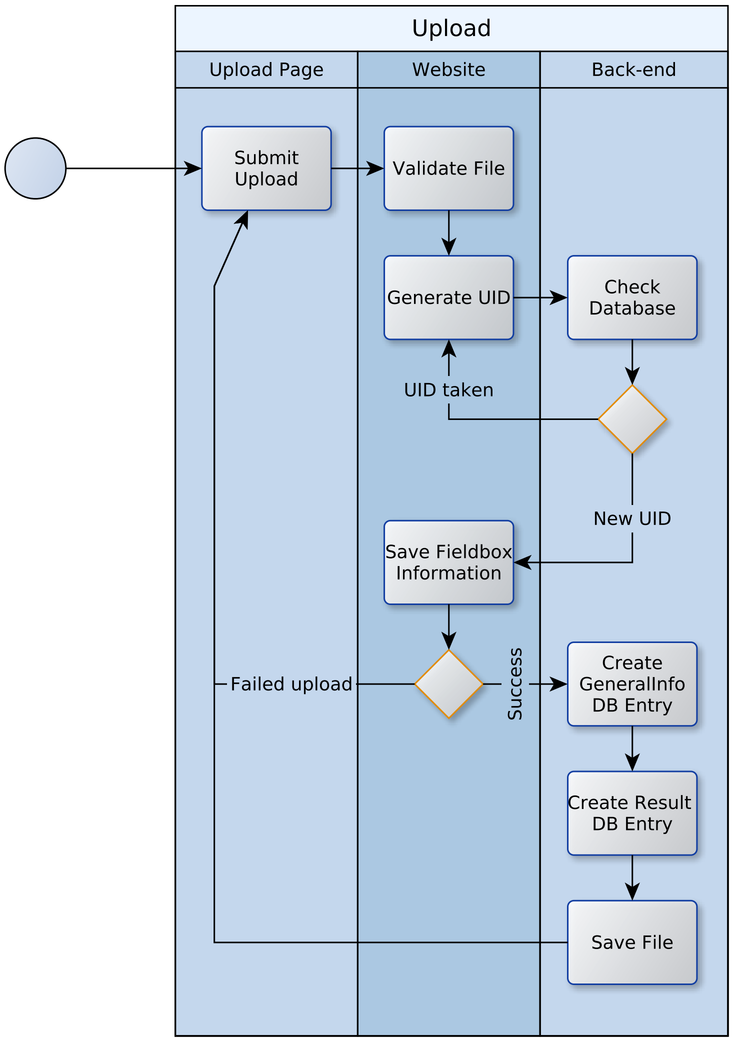


Figure 4.5: An in depth activity diagram for uploading new detection approaches

When a user submits a detection method, the website first checks if the back-end system is capable of running the detection approach, based on the programming language, this is the step where the upload is validated. Once the detection approach has been validated, the upload is given an unique identifier. The website then requires any information with the new identifier, if the database returns nothing. The identifier is available and the system continues saving the other information the user put into the field boxes.

If everything has gone problem free to this point the information provided by the user is now stored. A table entry in the result database has also been created, and lastly the MLA file it self is stored in the file system.

Chapter 5

Testing and Validation

The test specification and test validation will be carried out throughout this chapter. In section 5.1 a formal test specification will be presented. From the test specification, four tests will be defined. Each test will verify or disprove that the system works. A MLA based detection method in the form of three different classifiers, will also be tested, further verifying or disproving that the system works together.

5.1 Test Specification

To verify that the system requirements in section 3.4 have been fulfilled, four tests will be performed in order to showcase that every *must have* requirements are completed. The *should have* requirements should be completed, but are not essential for the system to run. The *could have* requirements are not necessary for the system, but would be "nice to have".

All four tests will be carried out on a single machine, with specifications listed in Appendix B. On Figure 5.1 a small depiction of the machine and what it is running is presented. The source code for both the front-end and back-end is available online.

The test specification will have four mandatory tests, in order to verify the system. If all must have requirements are fulfilled all four tests will be successful.

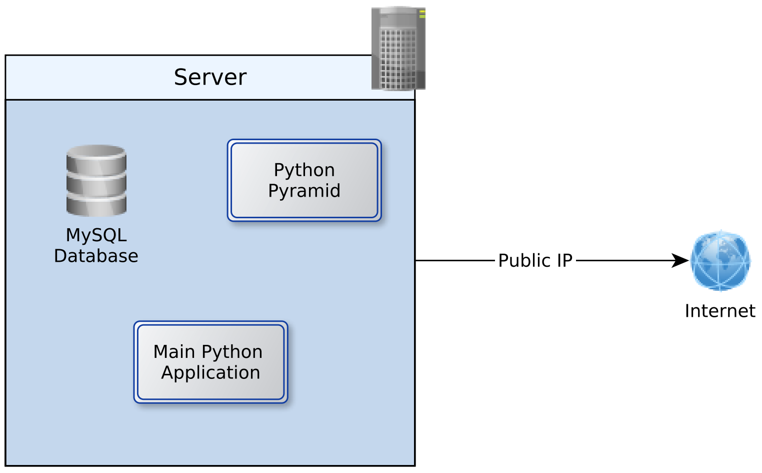


Figure 5.1: Depiction of the system setup, a single computer running the database, the server application for pyramid and the back-end application

- Upload detection method
- Executor detection method
- Present result for detection methods
- Present detailed results for a specific detection method

Along with these four is three additional tests that are outside the system requirements, as three MLA based detection methods have been provided by Matija Stevanovic, a Ph.d student at Aalborg University. The detection approaches, that has been provided are both using random forest MLA to classify the data sets. One classifies according to the TCP traffic, while the second one classifies according to UDP traffic and the last one classifies based on DNS traffic.

The four mandatory tests, are presented sequential, so that the first test does not require any of the other tests to be successful. The second test presented requires, that the first test is completed, before the test can be initiated and so on. The last three test that are based on the MLA provided by Matija Stevanovic, are only used to verify that the system is working, the result of the detection approaches are not important for this project.

Each test will follow the same approach:

Scenario A small introduction to the test, and how it will most likely be performed from a users perspective.

System requirements This will specify all the system requirements that can be tested in for a given test. This will be presented as e.g.Back-end.3.a, this referring to system requirement for the back-end system, and the data sets. Further more each item in the system requirements, will be coloured as, green are completed, orange are almost completed and red are failed.

Requirements remarks This will be short remarks to reasoning behind the completed, almost completed and failed system requirements.

Verdict The end result of the tests, will be shown as either accepted or failed.

5.1.1 Test 1: Upload algorithm

A user want to upload a MLA based botnet detection method to the website, to compare the efficiency of detecting botnet traffic compared to other MLA, already uploaded to the system. Once the algorithm has been uploaded, the programming language, in which the algorithm is written, will be verified. In this scenario, the algorithm is verified and information about the algorithm and the MLA it self will be stored on a local server, and new entries will be made in the database.

The system requirements which can be placed on the uploading of algorithms.

Note: Requirements marked green are completed, orange are almost completed and red are failed.

Must have

1. Front-end.1.a: The website must have a pure html based interface
2. Front-end.2.a: The website must have the functionality for users to upload MLAs
3. Back-end.1.a: The database must be able to store/access general information about each MLA
4. Back-end.2.a: For test execution there must be one programming language support
5. Back-end.3.a: There must be a folder with evaluation traffic data sets, divided into malicious and non-malicious data sets

6. Back-end.3.b: There must be a folder with training traffic data set, divided into malicious and non-malicious data sets

Should have

7. Back-end.2.c: For test execution there should be ≥ 2 different programming languages supported

Could have

8. Front-end.1.b: The website could have a graphical web interface

Additional requirements are posed on the creator of the MLA based detection method, as detection approaches are treated as an unknown. Certain requirements to the algorithm needs to be met, for it to function with the system.

- Back-end.2.a: For test execution there must have one programming language support
- All Back-end.2.b: For test execution the uploaded MLA must fulfil

All of the *must have* requirements have been successfully implemented in the project. The user is able to upload an algorithm from a pure HTML based interface. The database is uploaded with a new entry while inserting the information collected based on the system requirements. Support for a single programming has been implemented, being Python 2.7.11. This was also the language and version the provided detection approaches were coded in.

Multiple supported languages has not been implemented as stated in Requirement 7, as Python2.7 MLAs where provided. Therefore no effort where put into making support for more programming languages. It is worth mentioning, that added support for a programming language is fairly easy, as most programming languages are executable from the command line, such as Python, MATLAB, Java etc. As long as the developer of a detection method abides by the input/output requirements for the script.

In Requirement 8 is labelled as orange, this means the requirements is almost completed. The graphical design has been made and is available at the online repository, however the html and css needed for the graphical design, has not been made yet. Due to this, the requirements is marked as almost complete, but as not everything within this requirement has not yet been completed it can not be marked as completed.

The user requirements has not been marked as completed, almost completed or failed. These requirements are not some, that can be forced upon the system it self, as these requirements are placed on the users whom wishes to upload a detection method to the website.

This test has been approved. As a user is able to upload an algorihtm from the web interface. It can also be noted again, that uploading detection methods and uploading data sets are not that different. As stated in the system requirements, uploading data sets provides a the problem of labelling, trusting and verifying the labelling provided by the user.

5.1.2 Test 2: Automatic execution of algorithms

Once a user has uploaded a detection method to the website, the back-end fetches the information now available through the database, in order to execute the newly uploaded detection algorithm. In the case, that several algorithms have been uploaded in a short amount of time, creating a queue, the back-end executes each of these new algorithms one at a time, before rechecking the database for new entries.

Note: Requirements marked green are completed, orange are almost completed and red are failed.

Must have

1. Back-end.1.a: The database must be able to store/access general information about each MLA
2. Back-end.1.b: The database must be able to store/access test results for each MLA
3. Back-end.2.a: For test execution there must be one programming language support

Should have

4. Back-end.2.c: For test execution there should be ≥ 2 different programming languages supported

Could have

5. Back-end.1.c: The database could be able to store file/folder locations for data sets

As stated by the *must have* requirements, the system has to supports a single programming language. As stated in section 5.1, three different MLA based detection approaches where provided, each where based on the same programming language, there where no incentive to make support for several programming languages. This was also explained in the previous test for uploading MLA. The back-end application is able to fetch information from the GeneralInfo table (subsection 4.1.2), which adds a new entry to an internal queue of execution. After a detection method has been executed the results of the comparison is uploaded to the Results table in the database.

Requirement 5 was not made, as it was decided in the system requirements that users should not be allowed to upload data sets. Although this requirement could have been made, the paths to the folder locations with training and evaluation data sets where hardcoded into the executor module.

This test has been approved, as when a user uploads a detection algorithm the system is able to execute the algorithm and compare the labelling of the detection method to the ground truth labelling of the data sets. It however, worth mentioning that algorithms that had errors accrued during run time, forces not only the algorithm to stop, but also the whole back-end application.

5.1.3 Test 3: Result overview

After a user has uploaded a detection algorithm, and after said algorithm has been executed by the back-end. The performance of the detection method is available through the result section of the website. The user accesses this site, and the website fetches all results from the database and processes the information accordingly. Thereby presenting the results to the user.

Note: Requirements marked green are completed, orange are almost completed and red are failed.

Must have

1. Front-end.1.a: The website must have a pure html based interface

2. Front-end.3.a: The website must be able to give an overview of all tested algorithms
3. Back-end.1.a: The database must be able to store/access general information about each MLA
4. Back-end.1.b: The database must be able to store/access test results for each MLA

Could have

5. Front-end.1.b: The website could have a graphical web interface

The website fetches information from the GeneralInfo and Results table in the database. The information fetched from the database is used to present the results in accordance to Requirement 2. The each detection approach is presented by the original name given by the creator of the algorithm, and links to more specific result details, followed by the required overview information.

As explained in the test in subsection 5.1.1, Requirement 5 is again, marked as almost completed as the graphical design of the site was done. The HTML and css required to make the site operational with the design was not made.

This test has been approved, as when a user uploads a detection algorithm to the systemk, testing said algorithm, and uploading the results of the labelling of the detection approach to the Results table in the database. The user is now able to can access the result information through the website, and gain an overview of the MLA based botnet detection algorithm performance.

5.1.4 Test 4: Detailed Result

When the users has been presented for the results of every detection methods available. The users might wish to see more information about a specific detection approach. The website will at this point, fetch specific test results and process more statistical features and fetch information about the detection algorithm. This test is fairly similar to the "Result overview", but will provide a more specific insight to a single detection method.

Note: Requirements marked green are completed, orange are almost completed and red are failed.

Must have

1. Front-end.1.a: The website must have a pure html based interface
2. Front-end.3.b: The website must be able to present detailed information about each algorithm tested
3. Back-end.1.a: The database must be able to store/access general information about each MLA
4. Back-end.1.b: The database must be able to store/access test results for each MLA

Could have

5. Front-end.1.b: The website could have a graphical web interface
6. Front-end.3.c The website could be able to present family/sample classification test results

Directly from subsection 5.1.3, the user clicks on a specific set of results for a detection method, the user is directed to a more detailed view, where more statistics are being presented to the user. Requirement 6 where not met, due to the fact that the end result labelling where based on malicious or benign hosts. if the labelling had been flow based or packet based, this requirement could have presented some interesting results, as it could showcase, which bot malware specifically where found. As well as if the detection method where better at finding the newest version of an bot family malware.

This test has been approved, as the user is, directly after going through the test in subsection 5.1.3, able to see more information about a detection algorithm, as defined by Requirement 2.

5.2 Testing with Existing Detection Methods

Three different MLA based botnet detection algorithms has been provided by Matija Stevanovic. Each of the three algorithms are based on the same program, which uses an Random Forest MLA to classify the traffic in the data sets. The algorithms bases their labelling purely on either TCP, UDP or DNS traffic. [15] states that the majority of the Internet traffic in Europe and Asia from application that uses TCP or protocols that utilises TCP, with the only application that uses UDP being Skype. From this it can be expected that not every hosts in the data sets, will have either DNS or UDP, while most, if not all, hosts in the data sets have produced TCP traffic.

As the botnet detection approaches classifies with respect to TCP, UDP or DNS. It should be expected that the TCP classifier will find more malicious but also benign hosts. Compared to both the DNS and UDP classifier, as each of the algorithms only look at the respective traffic.

Each test have been executed according to the same methodology. Each algorithm has been uploaded through the websites upload function (subsection 5.1.1), where the back-end system found the newly uploaded algorithms and added them to an execution queue. Once an algorithm completed the labelling process, the results have been compared to the ground truth labelling of the data sets. The result of the labelling was uploaded to the MySQL database, and from there it can be presented to the user.

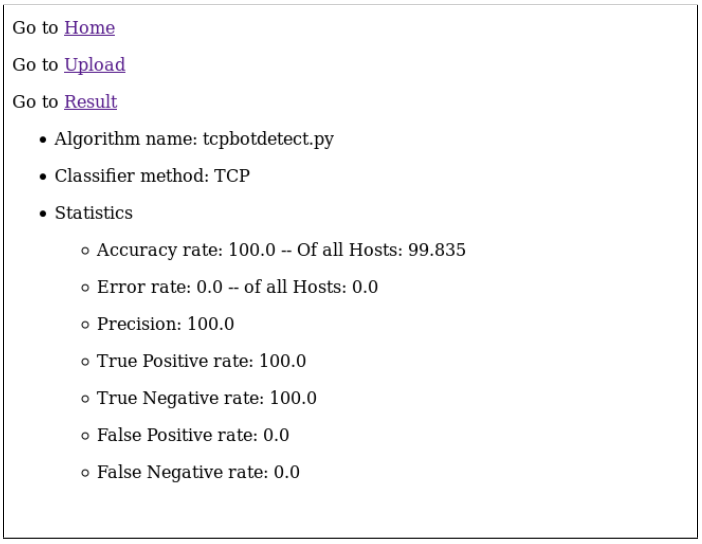
For testing the detection methods, 100% of the data sets where used for both training and evaluation of the algorithms. Although the detection methods performance, provides an unrealistic view of how the detection algorithms would fair in actual use, as it is expected that all three of the detection methods will predict 100% of the labels correct.

As this system should treat each uploaded detection algorithms as a black box, the nature of the unrealistic test result are not important. These tests showcases the system ability to automatically, upon upload of a detection approach, is able to execute the algorithm, compare the results of the labelling and present these results. Where each MLA based detection method has been tested using the same data sets.

5.2.1 Machine Learning Algorithm: TCP

The TCP classifier found almost all of the IPs in the data sets available for the system, and labelling every IP the detection method did find, correctly as presented on Figure 5.2, a screenshot of the labelling results of the MLA.

In Figure 5.2 it is seen that the accuracy of the algorithm is perfect, as it was predicted as the training data were also used for evaluation data. Next to the accuracy is a "of all hosts", this referrers to the accuracy of all the hosts in all of the data sets. Since the accuracy of "all hosts" is slightly lower than the other accuracy, is because some hosts in the data sets where not using TCP traffic at all.



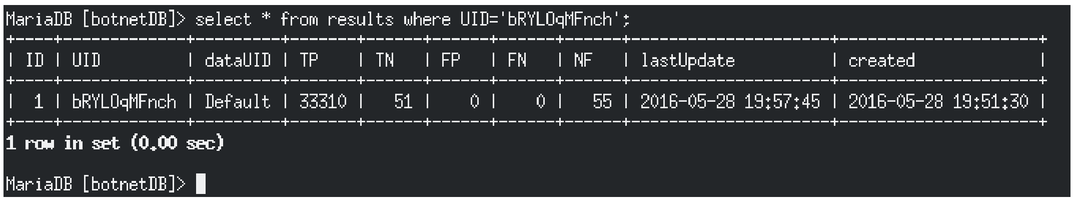
Go to Home

Go to Upload

Go to Result

- Algorithm name: tcpbotdetect.py
- Classifier method: TCP
- Statistics
    - Accuracy rate: 100.0 -- Of all Hosts: 99.835
    - Error rate: 0.0 -- of all Hosts: 0.0
    - Precision: 100.0
    - True Positive rate: 100.0
    - True Negative rate: 100.0
    - False Positive rate: 0.0
    - False Negative rate: 0.0

Figure 5.2: A screenshot of the detailed result page of the TCP classifier MLA



```
MariaDB [botnetDB]> select * from results where UID='bRYLOqMFnch';
+----+-------------+---------+-------+------+------+------+------+---------------------+---------------------+
| ID | UID         | dataUID | TP    | TN   | FP   | FN   | NF   | lastUpdate          | created             |
+----+-------------+---------+-------+------+------+------+------+---------------------+---------------------+
|  1 | bRYLOqMFnch | Default | 33310 |   51 |    0 |    0 |   55 | 2016-05-28 19:57:45 | 2016-05-28 19:51:30 |
+----+-------------+---------+-------+------+------+------+------+---------------------+---------------------+
1 row in set (0.00 sec)

MariaDB [botnetDB]> 
```

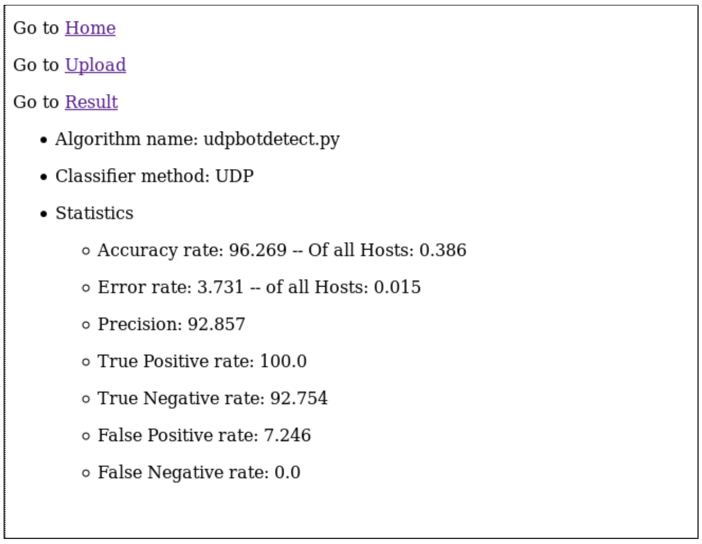
Figure 5.3: An SQL select statement showcasing the TCP classifier labelling results

As stated previously, the TCP classifier found almost all of the malicious and benign IPs in the data sets, as evident on Figure 5.3, where it found 33361 out of 33412 unique IP addresses.

5.2.2 Machine Learning Algorithm: UDP

As predicted previously the UDP classifier did not find as many malicious and benign hosts as the TCP classifier. Of the three MLAs tested through the system, the UDP was the only one to make false predictions, this illustrated in Figure 5.4, which presents the screenshot of the labelling results.

The results of the found hosts were just above 95% accuracy of the algorithm, this was expected to be 100%. It shows that it is not always easy to label malicious and non-malicious traffic, as they can seem very similar in nature. When the accuracy of all the hosts were calculated it is easy to see that, the UDP classifier did not find

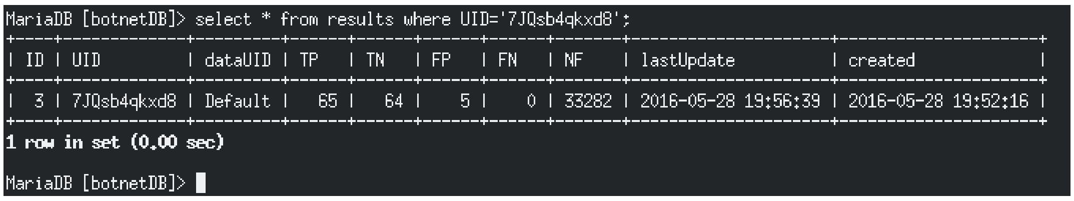

Go to Home

Go to Upload

Go to Result

- Algorithm name: udpbotdetect.py
- Classifier method: UDP
- Statistics
  - Accuracy rate: 96.269 -- Of all Hosts: 0.386
  - Error rate: 3.731 -- of all Hosts: 0.015
  - Precision: 92.857
  - True Positive rate: 100.0
  - True Negative rate: 92.754
  - False Positive rate: 7.246
  - False Negative rate: 0.0

Figure 5.4: A screenshot of the detailed result page of the UDP classifier MLA


```
MariaDB [botnetDB]> select * from results where UID='7JQsb4qkxd8';
+----+-------------+---------+------+------+------+------+-------+---------------------+---------------------+
| ID | UID         | dataUID | TP   | TN   | FP   | FN   | NF    | lastUpdate          | created             |
+----+-------------+---------+------+------+------+------+-------+---------------------+---------------------+
|  3 | 7JQsb4qkxd8 | Default |   65 |   64 |    5 |    0 | 33282 | 2016-05-28 19:56:39 | 2016-05-28 19:52:16 |
+----+-------------+---------+------+------+------+------+-------+---------------------+---------------------+
1 row in set (0.00 sec)

MariaDB [botnetDB]>
```

Figure 5.5: An SQL select statement showcasing the UDP classifier labelling results

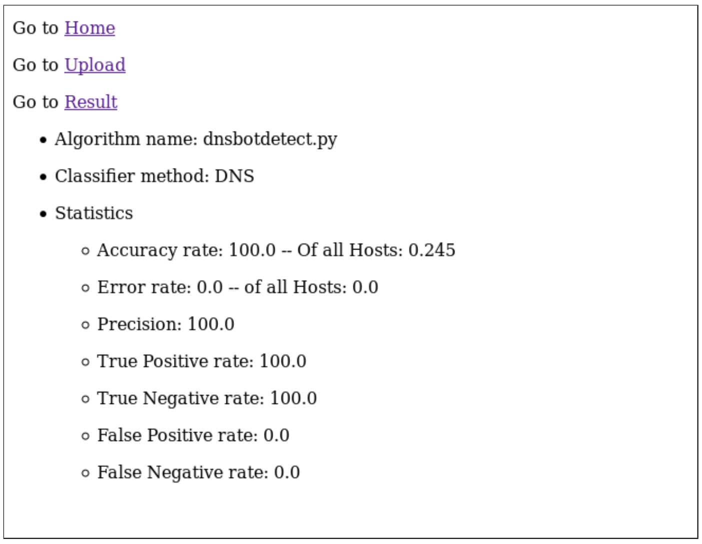
that many unique IP addresses. It can be seen that the classifier falsely labelled some hosts malicious while the hosts were non-malicious.

From the MySQL database screenshot in Figure 5.5, it is seen that the labelling resulted in 5 false positive, and the algorithm only found 134 out of the total of 33412 unique IP addresses.

5.2.3 Machine Learning Algorithm: DNS

As with the UDP classifier, the DNS classifier did not find many unique IPs in the data sets which contained DNS traffic. The detection method for DNS did not make any false predictions, as expected. The results can be seen in Figure 5.6

By comparing the two accuracy calculations, it is easy to see that there a fairly scarce distribution of DNS traffic in the data sets. The SQL statement on Figure 5.7 also confirms, that there where less DNS traffic compared to UDP traffic. A total

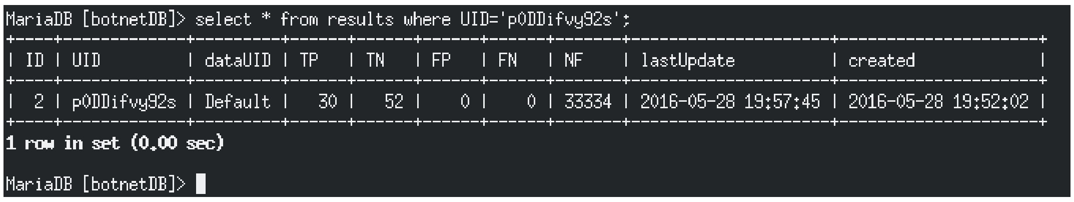

Go to Home

Go to Upload

Go to Result

- Algorithm name: dnsbotdetect.py
- Classifier method: DNS
- Statistics
  - Accuracy rate: 100.0 -- Of all Hosts: 0.245
  - Error rate: 0.0 -- of all Hosts: 0.0
  - Precision: 100.0
  - True Positive rate: 100.0
  - True Negative rate: 100.0
  - False Positive rate: 0.0
  - False Negative rate: 0.0

Figure 5.6: A screenshot of the detailed result page of the DNS classifier MLA


```
MariaDB [botnetDB]> select * from results where UID='pODDifvy92s';
+----+-------------+---------+------+------+------+------+-------+---------------------+---------------------+
| ID | UID         | dataUID | TP   | TN   | FP   | FN   | NF    | lastUpdate          | created             |
+----+-------------+---------+------+------+------+------+-------+---------------------+---------------------+
|  2 | pODDifvy92s | Default |   30 |   52 |    0 |    0 | 33334 | 2016-05-28 19:57:45 | 2016-05-28 19:52:02 |
+----+-------------+---------+------+------+------+------+-------+---------------------+---------------------+
1 row in set (0.00 sec)

MariaDB [botnetDB]>
```

Figure 5.7: An SQL select statement showcasing the DNS classifier labelling results

of 82 IPs where discovered out of the 33412 different IPs in the data sets.

Chapter 6

Conclusion and Discussion

In this final chapter, the conclusion and possible future work is presented in section 6.1 and section 6.2 respectively.

6.1 Conclusion

The current state-of-the-art botnet detection methods are based on machine learning, in which researchers are investing time to perfecting. Botnet is one of the biggest computer security risks to this day. Bot malware is able to gather keystrokes through keyloggers, make Distributed Denial of Service attacks against anyone on the Internet, and is one of the main contributed to SPAM mails to name a few.

As the state-of-the-art method of detecting botnet traffic is using machine learning algorithm, the data used to train and test the detection approaches should be a fairly straight forward process. Time and time again researchers are forced to create new training and testing data sets, as they are often not publicly available on the Internet. Many papers state how these data sets are made. This also means that almost every research paper on botnet detecting using machine learning uses different data sets, making it hard to compare performance between the different detection methods, as the used data sets often only contains a small number of bot malware variations.

This leading up to the problem statement of this project:

> *"In order get a better comparison between botnet detection using machine learning algorithms. How to design a system that is publicly available and provides the same data sets for every algorithm, enabling easy comparison?"*

From the problem statement another question needed to be answered:

> *"As data sets are a fundamental requirement and challenge of any machine learning algorithm botnet detection method. What is/are the best practice for generating data sets?"*

Throughout chapter 2, three different methods for traffic capturing were presented. Capturing "in the wild", creating a controlled setup, where machines would either be infected purposely with malware or reinstalled before usage. Each of these three methods has their own advantages and disadvantages. Capturing traffic "in the wild" will produce the most realistic traffic data sets, as they by nature are real representations of network traffic. However, it is very hard to label traffic as malicious and non-malicious when data sets are captured "in the wild", and many legal and ethical concerns arises. Creating small controlled environment will not be as realistic as capturing "in the wild", but most of the traffic is known to be either malicious or non-malicious, depending on the setup, and many of the legal and ethical aspects are not present in a controlled environment.

The effect of the label levelling were also presented in chapter 2. By labelling malicious traffic to either malicious packets, flows or hosts. Each of these three levels of labelling have advantages and disadvantages.

In chapter 3 a high level design of a system that would be able to fulfil the problem statement was presented. The system was designed according to UML use case, and enhanced flowchart diagrams known as activity diagrams. The system be based on a front-end and a back-end syste. The front-end system consisting of a website, and the back-end system consisting of an application and a database. From the system design and the traffic capture chapter a set of MoSCoW system requirements where presented, that should allow for a system to fulfil the problem statement.

The implementation of the system was presented in chapter 4, were the front-end system is based on a Python webframework called Pyramid. The back-end system consisting of a Python application as well as a MySQL database, that would link the front-end and back-end system together. From the implementation, the system would allow users to upload new detection methods to the service. The website would update the database and save the detection algorithm in a file system. The back-end system would pick up on the new entry in the database, and execute the newly uploaded detection algorithm, were the resulting comparison of the detection methods labelling and the labelling of the data sets, were uploaded to the database.

Testing the system in chapter 5, proved the system was able to do as intended, throughout all stages. Although almost every non-must have requirements were not met, the system requirements were designed as such, completing every must have requirement would make the system fully functional. For the project Matija Stevanovic a Ph.D of Aalborg University provided machine learning based botnet detection approaches. That could classify according to TCP, UDP or DNS traffic. This algorithm was split into three, and showed for proof-of-concept that a system as the one that has been made can be done on a larger scale.

All of the tests were completed and approved, and every *must have* system requirements were met, the problem statement has been completed and proved.

6.2 Future Work

To wrap this project up, the last section will focus on the possible future work that can be done on this system.

Some of the more complex work, could be put into the creation of a system that automatically took data sets, as they are in the current version of the system, where each malicious pcap represents a different botnet samples. The system would have to take an arbitrary amount of malicious and benign pcap files, merge them together to a single pcap file for finally splitting the pcap file up into two, one as a training data set and the other as a evaluation data set. This would eliminate the trust issue of MLA based botnet detection creators as the input to the algorithm would be changed to <training pcap + training labels> and <evaluation pcap>. An automated system could greatly improve this project system, as it would also open up for dynamically creating pcaps, where the users of the system would specify which botnet types they wish their detection algorithm would be tested with.

Another improvement, would be automatically download any required library that an uploaded detection method would need, across all supported languages. For Python this could be done using the Pip installer. Trying to execute an algorithm

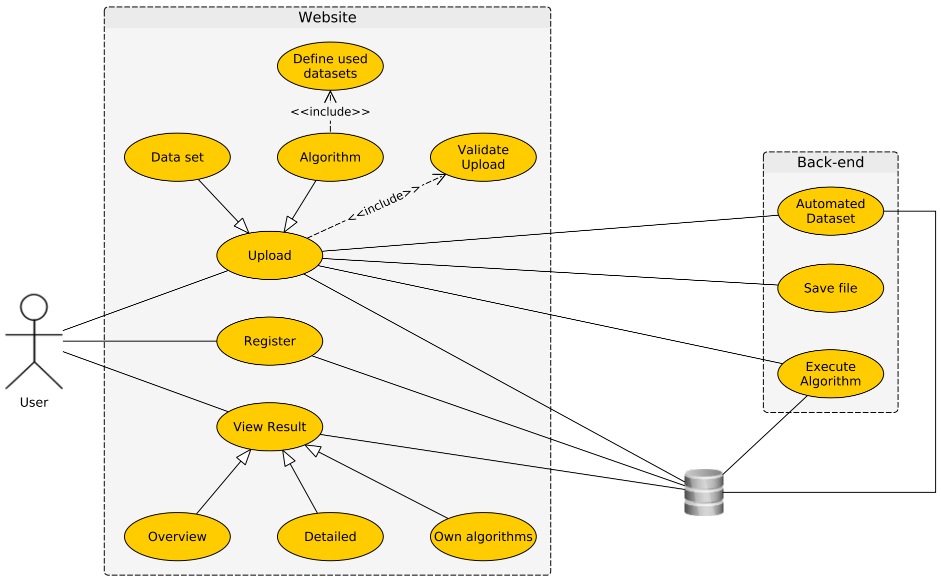


Figure 6.1: A possible use case diagram of the system if users where allowed to register to the site

while missing some libraries will force an error upon compilation.

One of the most obvious things that can improve the functionality and professionalism for the front-end is to have a proper graphical design deployed, a design has already been made, but lacks the html and css code to support it.

For further development for the system in general can be seen on Figure 6.1, that extends on the use case diagram presented in Figure 3.1

On the new use case diagram, a user is allowed to register, this means that a lot of trivial information about the user does not need to be included on every upload as it already available. This would also make it possible for users to only get tests results for their own uploaded algorithms. From [18] a lot of tailored detection approach where presented, as they are tailored towards specific botnet families/samples, upon upload the user could be asked to specify if the detection method should only be tested with the "default" data set, or if the algorithm should also be tested with a user defined data set. Implementing this could take full advantage of the automated system for merging several pcaps, defined by the user, to a single training and single evaluation data set

All of this would require a new database structure, one that would allow for users to be registered, and secondly allowing user defined data sets. A possible way of

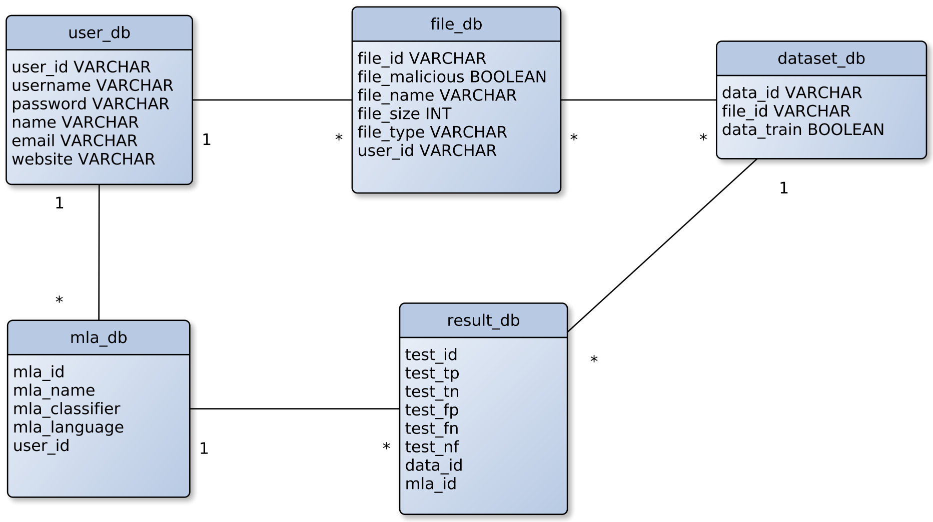


Figure 6.2: A possible database structure, where relations to each table is also shown. This database also takes into account that data sets information is stored in the database

implementing this could be as presented in Figure 6.2

The database representation a more scalable database structure than the one presented in subsection 4.1.2, as each table contains less fields, and is more organised. The `user_db` would provide that basics for user registration and user contact information, the `user_db` would be linked together with `mla_db` that stores information such as the original name of the file upload, the classifier type and programming language. The second table `user_db` is linked to is the `file_db`, this table would store uploaded data set information, whether or not the data set contains malicious hosts or only benign hosts (this could be further expanded upon, to have a table that stored labelling data for each data set), as well as the type of e.g. botnet family that is being uploaded, or benign office, home or public wifi traffic. The `dataset_db` would link these merged pcap files together, so that the system would know which data sets are merged together to form a training and evaluation data set, this would also allow for "fast" reconstruction of data sets if storage space is limited. Lastly the `result_db`, which is very identical to the results table defined in subsection 4.1.2.

As the last point of potential future work, would be to take the labelling of the data sets to either flow based labelling or packet based labelling. This could have a positive effect on for example the research community that uses flow based labelling of data sets, and not host based as presented in this project.

Bibliography

[1] Tomasz Bujlow, Valentín Carela-Español, and Pere Barlet-Ros. *Comparison of Deep Packet Inspection (DPI) Tools for Traffic Classification*. UPC-DAC-RR-CBA-2013-3. Universitat Politècnica de Catalunya, June 2013.

[2] Cisco. *What Is the Difference: Viruses, Worms, Trojans, and Bots?* `http://www.cisco.com/c/en/us/about/security-center/virus-differences.html`. Feb. 2015.

[3] Felix Leder Daniel Plohmann Elmar Gerhards-Padilla. "Botnets: Detection, Measurement, Disinfection & Defence". In: *The European Network and Information Security Agency* (Mar. 2011).

[4] Christopher Elisan. *Malware, Rootkits and Botnets: A Beginners Guide Malware*. ISBN: 9780071792066. McGraw Hill, Aug. 2012.

[5] Loras R. Even. *Honey Pot Systems Explained*. `https://www.sans.org/security-resources/idfaq/what-is-a-honeypot/1/9`. July 2000.

[6] Anand Ajjan James Wyke. *The Current State of Ransomware*. `https://www.sophos.com/en-us/medialibrary/PDFs/technical%20papers/sophos-current-state-of-ransomware.pdf?la=en`. Accessed: 11/03/2016. Dec. 2015.

[7] Lawrence Berkeley National Laboratory. *LBNL/ICSI Enterprise Tracing Project*. `http://www.icir.org/enterprise-tracing/Overview.html`. Accessed: 30/5/2016.

[8] Thuy TT Nguyen and Grenville Armitage. "A survey of techniques for internet traffic classification using machine learning". In: *Communications Surveys & Tutorials, IEEE* 10.4 (2008), pp. 56–76.

[9] Sang-Kyun Noh et al. "Detecting P2P botnets using a multi-phased flow model". In: *Digital Society, 2009. ICDS'09. Third International Conference on*. IEEE. 2009, pp. 247–253.

[10] pingdom.com. *IRC is dead, long live IRC*. http://royal.pingdom.com/2012/04/24/irc-is-dead-long-live-irc/. Accessed: 02/03/2016. Apr. 2012.

[11] Malware Capture Facility Project. *Malware Capture Facility Project*. http://mcfp.weebly.com/. Apr. 2016.

[12] J. Quittek et al. *IPv6 Flow Label Specification*. https://tools.ietf.org/pdf/rfc3917.pdf. Mar. 2004.

[13] J. Rajahalme et al. *Requirements for IP Flow Information Export (IPFIX)*. https://tools.ietf.org/pdf/rfc3697.pdf. Oct. 2004.

[14] S. Saad et al. "Detecting P2P botnets through network behavior analysis and machine learning". In: *Privacy, Security and Trust (PST), 2011 Ninth Annual International Conference on*. 2011, pp. 174–180. DOI: 10.1109/PST.2011.5971980.

[15] Sandvine. "Global Internet Phenomena Asia-Pacific & Europe". In: (Sept. 2015).

[16] Ali Shiravi et al. "Toward developing a systematic approach to generate benchmark datasets for intrusion detection". In: *Computers & Security* 31.3 (2012), pp. 357–374. DOI: 10.1016/j.cose.2011.12.012.

[17] Aron Stefánsson et al. *AAU HoneyJar containment: Real-time classification of CnC traffic*. 2013.

[18] Matija Stevanovic and Jens Myrup Pedersen. "On the Use of Machine Learning for Identifying Botnet Network Traffic". In: *Journal of Cyber Security, Vol. 4, 1–32.* (Aug. 2015).

[19] Nadezhda Demidova Tatyana Shcherbakova Maria Vergelis. *Spam and Phishing in the First Quarter of 2015*. https://securelist.com/analysis/quarterly-spam-reports/69932/spam-and-phishing-in-the-first-quarter-of-2015/. Accessed: 10/03/2016. June 2015.

[20] M. Tavallaee, W. Lu, and A. A. Ghorbani. "Online Classification of Network Flows". In: *Communication Networks and Services Research Conference, 2009. CNSR '09. Seventh Annual*. 2009, pp. 78–85. DOI: 10.1109/CNSR.2009.22.

[21] Florian Tegeler et al. "BotFinder: Finding Bots in Network Traffic Without Deep Packet Inspection". In: *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*. CoNEXT '12. Nice, France: ACM, 2012, pp. 349–360. ISBN: 978-1-4503-1775-7. DOI: 10.1145/2413176.2413217. URL: http://doi.acm.org/10.1145/2413176.2413217.

[22] Dr. Hamadoun I. Touré. *Cybersecurity Global Status Update*. http://www.un.org/en/ecosoc/cybersecurity/itu_sg_20111209_nonotes.pdf. Accessed: 02/03/1026. 2011.

[23] International Telecommunication Union. “ICT Facts and Figures – The world in 2014”. In: (Aug. 2014).

[24] John R. Vacca. *Computer and Information Security Handbook*. ISBN: 978-0-12-374354-1. Morgan Kaufmann, May 2009.

[25] Wikipedia. *MoSCoW method*. `https://en.wikipedia.org/wiki/MoSCoW_method`. Mar. 2016.

[26] David Zhao et al. “Botnet detection based on traffic behavior analysis and flow intervals”. In: *Computers & Security* 39 (2013), pp. 2–16.

Appendix A

TotalVirus Used Sites

When using `https://www.virustotal.com/` the different sites that are used to check files for malicious code are shown in Table A.1.

ALYac	BitDefender	GData	Rising
AVG	Bkav	Ikarus	SUPERAntiSpyware
AVware	ByteHero	Jiangmin	Sophos
Ad-Aware	CAT-QuickHeal	K7AntiVirus	Symantec
AegisLab	CMC	K7GW	Tencent
Agnitum	ClamAV	Kaspersky	TheHacker
AhnLab-V3	Comodo	Malwarebytes	TrendMicro
Alibaba	Cyren	McAfee	TrendMicro-HouseCall
Antiy-AVL	DrWeb	McAfee-GW-Edition	VBA32
Arcabit	ESET-NOD32	eScan	VIPRE
Avast	Emsisoft	Microsoft	ViRobot
Avira (no cloud)	F-Prot	NANO-Antivirus	Zillya
Baidu	F-Secure	Panda	Zoner
Baidu-International	Fortinet	Qihoo-360	nProtect

Table A.1

Appendix B

Production PC

The computer used for hosting the website, database and network traces is running Arch Linux, with the kernal version of 4.5.0-1-ARCH.

The specifications of the computer is done using **lshw**, which is not a standard application. In Listing B all commands for using lshw is shown and in Table B.1

```
# Download lshw, pacman takes care of dependencies, just accept installation
sudo pacman -S lshw
# sudo lshw provides more information than had sudo not be provided
sudo lshw -short
```

Device	Class	Description
	system	Desktop Computer
	bus	DH67CF
	memory	64KiB BIOS
	processor	Core i7 (To Be Filled By O.E.M.)
	memory	32KiB L1 cache
	memory	1MiB L2 cache
	memory	8MiB L3 cache
	memory	8GiB System Memory
	memory	4GiB DIMM DDR3 Synchronous 1333 MHz (0,8ns)
	memory	4GiB DIMM DDR3 Synchronous 1333 MHz (0,8ns)
	bridge	2nd Generation Core Processor Family DRAM Controller
	display	2nd Generation Core Processor Family Integrated Graphics Controller
	communication	6 Series/C200 Series Chipset Family MEI Controller #1
eno1	network	82579V Gigabit Network Connection
	bus	6 Series/C200 Series Chipset Family USB Enhanced Host Controller #2
usb1	bus	EHCI Host Controller
	bus	Integrated Rate Matching Hub
	input	USB Optical Mouse
	input	Dell USB Entry Keyboard
	multimedia	6 Series/C200 Series Chipset Family High Definition Audio Controller
	bridge	6 Series/C200 Series Chipset Family PCI Express Root Port 1
	bridge	6 Series/C200 Series Chipset Family PCI Express Root Port 2
	bus	uPD720200 USB 3.0 Host Controller
usb3	bus	xHCI Host Controller
usb2	bus	xHCI Host Controller
	bus	6 Series/C200 Series Chipset Family USB Enhanced Host Controller #1
usb4	bus	EHCI Host Controller
	bus	Integrated Rate Matching Hub
	bridge	H67 Express Chipset Family LPC Controller
	storage	6 Series/C200 Series Chipset Family SATA AHCI Controller
	bus	6 Series/C200 Series Chipset Family SMBus Controller

Table B.1

Appendix C

Pyramid setup

Through this appendix, the approach will be based on a Linux machine running Arch Linux with the the machine with specifications according to Appendix B.

C.1 Setting up pyramid

Before installing pyramid, python3 needs to be installed, for Arch Linux the default python is python3, for other Linux distributions such as Ubuntu, python2 is the default python application.

Installing python3 and python3 setuptools that are needed in order to install pyramid. Listing C.1 shows the command for installing python on Arch Linux.

```
# Install python3
sudo pacman -S python python-setuptools
```

Listing C.1: Installing python

Once both python packages have been installed, pyramid can be installed, this can be done with the following command, the optional command added in Listing C.2 will also install some of the most common packages used by pyramid, although they are not needed in order for pyramid to function, they provide additional functionality.

```
# Installing pyramid
sudo easy_install pyramid
# Optional pyramid packages, that are used for this project
sudo easy_install nose webtest deform sqlalchemy pyramid_chameleon pyramid_debugtoolbar
     waitress pyramid_tm zope.sqlalchemy
```

Listing C.2: Installing pyramid

Now that pyramid has been installed, pyramid provides templates. Using the templates, ensure that all files have been created, and the templates also run "out of the box". Using the command in line 1 in Listing C.3 will create a template that functions with sqlalchemy. Sqlalchemy is a highly functional API for a sqlite database, but also works with more advanced databases such as MySQL. Se

```
1 # Create a pyramid template location
2 pcreate alchemy
```

Listing C.3: Creating a pyramid template to work from

Lastly before being able to run the pyramid application two last commands needs to be run. These commands can be seen in Listing C.4. Firstly cd into the root directory of the folders created by the template creation in Listing C.3, then run a standard setup.py script followed by develop, this will install any packages specified in setup.py that is required for the application to run. The last function "pserve" will run a pyramid instance, accessible from localhost on port 6543 by default.

```
# Setting up the environment
cd <into/pyramid/template/root> && sudo python setyp.py develop
# Running python pyramid service
pserve development.ini --reload
```

Listing C.4: Installing the required packages to run the pyramid instance and running an instance of pyramid