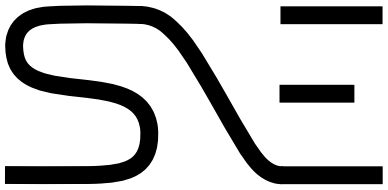


System and Network Engineering - University of Amsterdam

Research Project 2

Discriminating reflective (D)DoS attack tools at the reflector

July 9, 2017

Authors
M.W. Grim
max.grim@os3.nl

F. Mijnen
fons.mijnen@os3.nl

Supervisor
L. Haagsma
haagsma@fox-it.com

Abstract

This research focuses on a specific DDoS attack called a Reflective Distributed Denial of Service (RDDoS) attack. A RDDoS attack uses reflectors to direct vast amounts of traffic to a victim. Tools that generate these attacks can be found on the Internet and are accessible to anyone with knowledge about computers. The goal of this research is to analyse network packets captured at the reflector for behavioural patterns in order to identify the tool being used for the attack. This is done by performing both manual analysis and machine learning techniques. In the latter case both clustering algorithms and trained multiclass classifier models are used to determine the tool being used based on the incoming network data. This research will show that it is feasible to apply machine learning techniques on network captures to successfully identify different RDDoS tools used for an attack both with unsupervised and supervised learning.

Acknowledgements

We would like to thank Lennart Haagsma from Fox-IT, who not only provided us with data collected from multiple open DNS resolver honeypots but also gave us great supervision during our project. Without his data and expertise we would not have been able to perform this research.

Contents

1 Introduction **3**
- 1.1 Research questions . . . 4
- 1.2 Related work . . . 4
- 1.3 Report outline . . . 5

2 Background **6**
- 2.1 Distributed Denial of Service (DDoS) . . . 6
- 2.2 Reflective Distributed Denial of Service (RDDoS) attacks . . . 7
- 2.3 Amplification attacks . . . 9
- 2.4 Machine learning . . . 11
  - 2.4.1 Density-Based Spatial Clustering of Applications with Noise . . . 11
  - 2.4.2 Multiclass Neural Network . . . 12
  - 2.4.3 Multiclass Logistic Regression . . . 12
  - 2.4.4 Multiclass Decision Forest . . . 13

3 Methodology **14**
- 3.1 Lab generated data . . . 14
  - 3.1.1 DNS DDoS scripts . . . 14
  - 3.1.2 Test environment setup . . . 16
  - 3.1.3 Attacking & data collection . . . 16
  - 3.1.4 Multiclass classification . . . 17
  - 3.1.5 Training the model . . . 17
  - 3.1.6 Azure Machine Learning . . . 18

4 Results **20**
- 4.1 Fox-IT data . . . 20
  - 4.1.1 Dataset 1: 25 packets per PCAP . . . 20
  - 4.1.2 Dataset 2: 250 packets per PCAP . . . 22
- 4.2 Lab generated data . . . 27
  - 4.2.1 Accuracy results . . . 28
  - 4.2.2 Training with fewer features . . . 30
  - 4.2.3 Principal Component Analysis (PCA) . . . 30
  - 4.2.4 Decision tree . . . 32

5 Discussion **36**

6 Conclusion **39**

7 Future work **40**

A Scripts **42**

B DNS packets in Fox-IT dataset **47**

C Features generated from DNS packets **51**

CHAPTER 1

Introduction

In 1974 a 13-year-old launched the very first known Denial of Service (DoS) attack. He issued a console command to all machines in a classroom bricking all computers in the Computer-Based Education Research Laboratory at the University of Illinois Urbana-Champaig, effectively denying access to the machines for all students until the machines were power-cycled [1]. This rather harmless prank is a bleak comparison to the professionalised and malicious DoS attacks Internet users face today. In time, the DoS attacks transitioned to Distributed Denial of Service (DDoS) attacks where multiple systems are used in the DoS. The world first realised the colossal damage a DDoS attack can cause on February 7, 2000, when a 15-year-old Canadian known by the alias *mafiaboy* launched a DDoS attack on large Internet websites such as Yahoo! and Amazon causing an estimated 1.2 billion dollars in damages. Three years later, the SQL Hammer worm was powerful enough to launch a DDoS attack affecting South Korea in 2003. With the effective DoS of a country it was becoming more and more apparent that DoS attacks would be a big problem for the internet.

In the 43 years since the first DoS, the problem still persists today. To this day systems are attacked in large DDoS attacks which are still increasing in size (see Figure 1.1). As attacks are becoming more advanced and available to consumers the threat of being targeted by DDoS attacks increases. A disturbing trend in DDoS attacks is emerging where hackers provide their DDoS capabilities to consumers. A considerable amount of money can be made for these hackers, where reported earnings are hundred thousands of dollars every year [2]. DDoS attacks now often utilise reflection to increase the size of the attack. In these amplification attacks, the attacker uses legitimate services on the Internet and spoofed IP address to mislead the legitimate servers into sending traffic to a victim. The traffic sent from the legitimate servers is often larger than the request to send traffic and therefore amplifies the size of the attack.

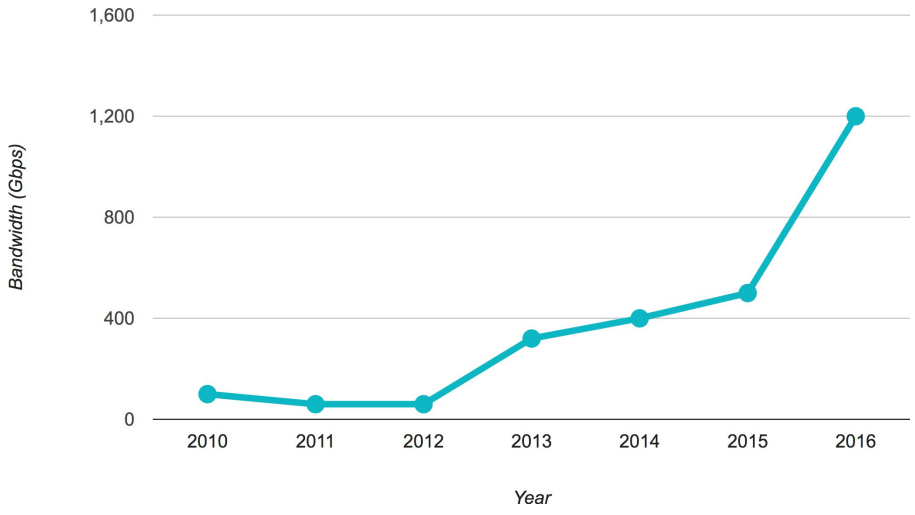


Figure 1.1: Growth of Distributed Denial of Service (DDoS) attacks in size over the last six years [3].

For forensic investigation, dealing with Reflective Distributed Denial of Service attacks is exceptionally hard since the attack hides the botnet behind legitimate services. No IP addresses can be found, and the traffic received is crafted by the legitimate service, which means deducting

the exact tool that has been used is difficult to find. In this research, we will attempt to perform this classification of RDDoS tools by capturing RDDoS requests and analysing the network packets.

1.1 Research questions

In this research we will propose and test a method to discriminate different Reflective Distributed Denial of Service (RDDoS) attacks which use DNS and identify and group different attack scripts. In order to do this, we will look into the following research questions.

1. Do Reflective Distributed Denial of Service (RDDoS) attacks leave distinctive traces
2. Can a fingerprint be build using distinctive traces left by reflective DDoS attacks
3. Is it possible to identify the tool / script / service used in a reflective DDoS attack
4. Can machine learning be utilised to automate the identification process

1.2 Related work

This section presents an overview of relevant work done in the field of Distributed Denial of Service (DDoS) and Reflective Distributed Denial of Service (RDDoS) taxonomies, detection and prevention.

Previous research on *"Distributed Denial of Service: Taxonomies of Attacks, Tools, and Countermeasures"* by Specht and Lee focused on proposing clear definitions and taxonomies for the broad landscape of (R)DDoS attacks [4]. This taxonomy makes clear distinctions between resource- and bandwidth depletion attacks. Additionally it differentiates between secondary victims (such as zombies and reflectors) and primary victims which are the actual target in an attack (see Figure 1.2).

Feinstein et al. (2003) calculated the entropy of captured packets and used distribution analysis to search for anomalies in their attributes. Their results show that they were able to successfully detect attacks [5]. This was partially because the tools often used by attackers were quite unsophisticated and did not attempt to properly hide their marks, allowing the researchers to quickly detect anomalies.

Considering each of the steps and their precursors of a starting DDoS attack, K. Lee et al. used the attributes of network captures as features in a hierarchical clustering model to proactively detect DDoS attacks in a network. This model proved to be effective and efficient in detecting the start of the actual attack, where older methods only focused on traffic generated during the

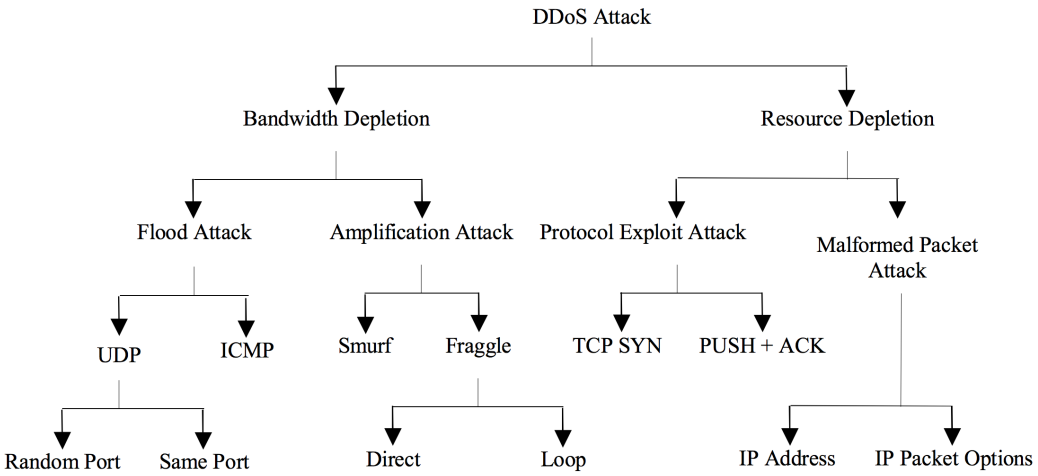


Figure 1.2: Taxonomy of DDoS attacks [4].

attack [6]. As a consequence DDoS attacks could be detected in its earliest stage, providing more time to commence countermeasures.

Other research has been done by Zi et al. into an adaptive clustering method using feature ranking to detect DDoS attacks. The algorithm is a modification of the k-means algorithm called the Modified Global K-means (MGKM) algorithm. The MGKM algorithm is described as an algorithm with an adaptive process, in that it uses feature ranking to adjust the working feature vector, allowing it to find different patterns in DDoS attacks and improve its own effectiveness while running [7].

Wei et al. (2013) used the inherent linear relationships of attacking and responsive flows to detect RDDoS attacks. The advantage of such an approach is that it is protocol independent. In order to detect these attacks a Rank Correlation based Detection (RCD) is used. RCD can be used to calculate the rank correlation between flows and fire alerts once a positive results comes back, as an indicator for RDDoS attacks [8].

Rozekrans and Mekking researched whether implementing Response Rate Limiting (RLL) is effective in defending against DNS amplification attacks [9]. This was done by tuning the RLL settings while sending queries and measuring the in- and outbound traffic. Their main conclusion was that RLL is a proper defence against current amplification attacks but not more sophisticated attacks.

To our knowledge, no research has been done before in determining which toolkit is used for an incoming Reflective Distributed Denial of Service, with or without using machine learning.

1.3 Report outline

This report will continue with a summary on relevant subjects in chapter 2. Firstly the Denial of Service (DoS), Distributed Denial of Service (DDoS) and Reflective Distributed Denial of Service (RDDoS) attack types will be described and defined in chapter 2. Secondly, this same chapter will review some concepts of machine learning and the algorithms relevant to this research. Next, chapter 3 portrays the methodology of the research and which choices were made in the experiments. Thereafter results are described in chapter 4, which is split up into two parts. Chapters 5 and 6 will discuss the results and draw conclusions from those. Finally, chapter 7 will discuss further research to be done in this field.

CHAPTER 2
Background

This research mainly focuses on two subjects: Distributed Denial of Service (DDoS) attacks (specifically RDDoS attacks) and the classification of tools that perform these attacks with the help of machine learning. Though these subjects are generally well known, it is important to clearly specify the context and terminology of these subjects as they play a central role in this research.

2.1 Distributed Denial of Service (DDoS)

In essence, a Denial of Service (DoS) attack is an attack on one or multiple systems where the system is made unavailable, denying service to legitimate users. In general, there are two flavours of DoS attacks: bandwidth depletion and resource depletion (see Figure 1.2).

Bandwidth depletion attacks aim at flooding the bandwidth of a victim in order to ensure legitimate traffic can no longer reach the victim. By sending out more traffic than a victim can handle, all other traffic can effectively be blocked out, resulting in a DoS. Usually a single entity has only a limited amount of bandwidth. Since an attacker has to generate more traffic than an attacker can receive it has to devise methods that increase their ability to generate vast amounts of traffic.

The second DoS attack type is a resource depletion attack. This attack, instead of bringing down a network by depleting its total bandwidth, depletes a specific resource of a service or server by exploiting a weakness. These services are often remotely available services like FTP, web-servers or other servers with open ports listening to network traffic. Sending a malicious malformed packet to such a service may cause some sort of internal error, effectively bringing down the server or degrading the functionality severely. This research will solely focus on the former type of Denial of Service attack, the bandwidth depletion attack.

As mentioned before, attackers need to send out more traffic to the victim than it can handle to perform a bandwidth depletion attack. Considering that one machine can only generate so much traffic, often not enough to bring down a professionally hosted server, attackers started to distribute their DoS attacks, which was the start of the well-known Distributed Denial of Service (DDoS) attacks (see Figure 2.1). As the name suggests, a DDoS attack distributes the attack over multiple machines attacking the victim all at once. Using multiple machines in an attack significantly increases the bandwidth and therefore the power of an attack.

A DDoS attack consists of many components working closely together. The entire infrastructure used to create a DDoS attack is known as the *attack platform*. This attack platform consists of all components required to perform the DDoS attack, including machines that generate traffic and a central or distributed control component. The attacker uses the attack platform to launch DDoS attacks against machines which are reachable from the Internet. The machine under attack is the *primary victim*, whereas the *secondary victims* are all machines infected and used by the attacker to generate traffic [4]. In a DDoS attack the infected machines (often refered to as *zombies*) are used to generate all the traffic in the attack. The collection of these infected machines is called a *botnet*. It is unclear how large the average botnet is or how powerful these botnets are, but botnets can grow to enormous sizes. In 2013 Symantec researched the ZeroAccess botnet, estimating the size of the botnet to an astonishing 1.9 million machines [10].

Because a DDoS attack uses large networks to generate traffic some degree of control over the bots is desirable. Configuring targets, starting and stopping attacks and other actions must be controllable by the attacker. In general, botnets are controlled by a one or more Command and Control (C&C) server(s) which control the bots in the botnet. The C&C can relay the attacker's

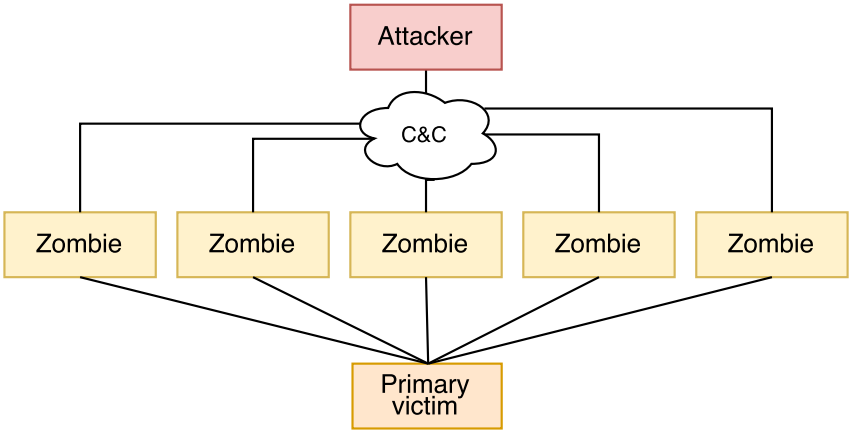


Figure 2.1: Distributed Denial of Service attack generated by multiple zombies. These zombies, which are secondary victims, listen to a Command and Control (C&C) server that is controlled by the attacker. Note that in this case the attacker is hidden.

commands to the bots and control the botnet. The infrastructure of a botnet is built as a simple agent-handler model where the bots are agents handled by the C&C. It is important to note that the C&C often uses covert channels in order to disguise the communication between the C&C and bots. Frequently used covert channels include IRC, HTTP, DNS, and ICMP.

2.2 Reflective Distributed Denial of Service (RDDoS) attacks

A Reflective Distributed Denial of Service (RDDoS) attack is a variant of a DDoS attack which abuses legitimate services to direct traffic at the victim. These legitimate services (often referred to as the *reflectors*) respond to requests containing a spoofed source IP, therefore directing traffic to the intended victim. By spoofing the source IP address an attacker can trick legitimate services into sending traffic to the victim. The use of reflection for a DDoS attack both hides the identity of the attacker from the primary victim and allows the attacker to amplify its attack significantly. Amplification is discussed in more detail in the next section.

One simple and classic reflection attack is the SMURF attack. Although the SMURF attack is not distributed in the same way as a RDDoS attack, it does make use of reflection and is a good starting point to explain the concept. SMURF attacks make use of Internet Control Message Protocol (ICMP) requests that are sent to the network's broadcast address (see Figure 2.2). The router receiving this request will broadcast it to all devices in the subnet. Since the attacker is able to spoof the source IP of the request, all devices to which the initial request is broadcasted by the router will respond to what they believe is the legitimate source IP. Because of the nature of the attack the responding devices have no way to verify the source IP and therefore have to trust it. Nowadays a SMURF attack is not feasible anymore as routers block ICMP request sent to the broadcast address [11].

In a report written by Brian Krebs, the author mentions that attacks are becoming easier to access and lower in costs: *"By offering a low-cost, shared distributed denial-of-service (DDoS) attack infrastructure, these so-called booter and stresser services have attracted thousands of malicious customers and are responsible for hundreds of thousands of attacks per year."* [12]. Because the financial gain of providing a good DDoS service has become noticeable to attackers infrastructures have become more professional. This financial incentive caused an emergence of "booter services", which aim to profit from users requiring an easy to use and powerful DDoS tool. On December 25 2014 an attacker group called the *Lizzard Squad* went as far as to bring down the Xbox Live and PlayStation network to market their platform. It is unmistakable that a lot of money can be made by providing these services. A small overview of a RDDoS attack platform is shown in Figure 2.3.

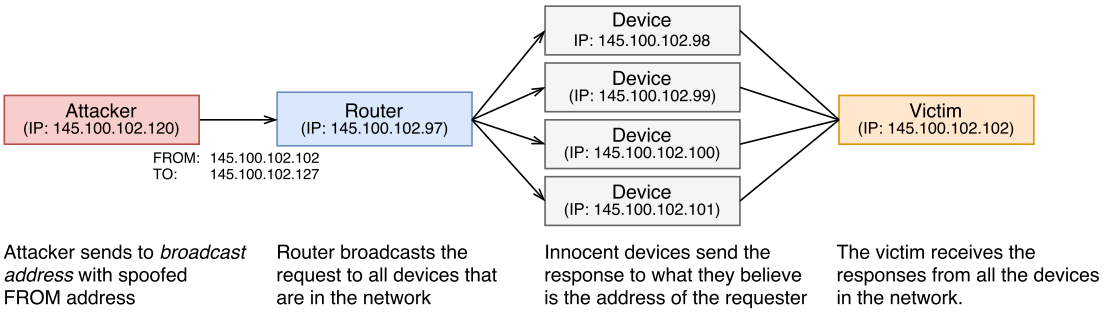


Figure 2.2: Example of SMURF attack. In this example, the attacker sends a request with a spoofed FROM address to the broadcast address. This is received by the router, which in turn forwards the message to all the devices in the network. These devices will respond to the request with the spoofed FROM address. In the end, the victim will receive all these responses (which it didn't request), flooding the victim with messages. [11]

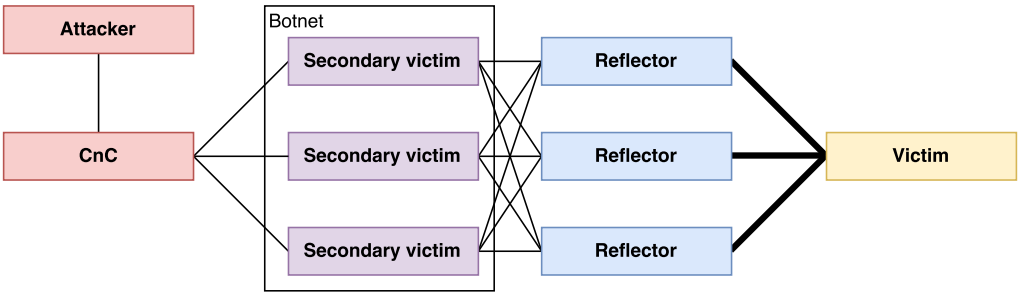
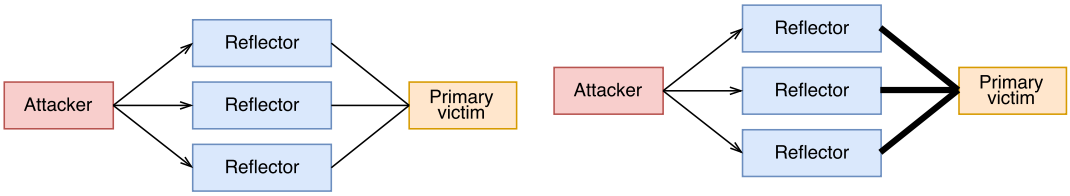


Figure 2.3: Infrastructure of a RDDoS attack using amplification.

2.3 Amplification attacks

Amplification attacks are attacks in which an attacker abuses a legitimate service as a reflector to amplify an attack. Imagine an attacker having a network with a decent upload speed of a 100Mbps. Without the use of amplification or a botnet the attacker is only able to perform bandwidth depletion attacks of that same bandwidth size. However, if an attacker could make a request to a legitimate service that would send a response four times larger and meanwhile allows for spoofing the source IP address, the attacker can significantly increase its attack size (see Figure 2.4). This is what is called an (reflective) amplification attack [13]. Cloudflare, one of the leading service providers in DDoS protection, notes that amplification is becoming more important in DDoS attacks: *"In order to increase your attack's volume, you could try and add more compromised machines to your botnet. That is becoming increasingly difficult. Alternatively, you could find a way to amplify your 100Mbps to something much bigger."* [11].



(a) Reflective attack without amplification.

(b) Amplified reflective attack triggering a large response.

Figure 2.4: Simplified illustration of a reflection attack vs. an amplified reflection attack. In the amplified reflection attack the attacker crafts its requests in such a way that a maximum response is generated (the amplification factor is as high as possible). Note that for reflection attacks the attacker spoofs the source IP. Therefore all responses are directed to the attacker.

In essence, a packet with a spoofed IP address is sent to a legitimate service using the victim's IP address as a spoofed source address. If the request packet is smaller than the response packet the attacker can effectively amplify the bandwidth of his attack. A small packet can often request a large response, for example requesting a large DNS record could amplify the response greatly. The small request will thus become a large packet sent to the victim effectively amplifying the size of the attack. Multiple protocols are susceptible to reflection attacks and almost all introduce an amplification factor that may be utilised by the attacker. In 2014 US-cert, the United States computer security response team released a list of possible reflection methods and their amplification factor as shown in Table 2.1 [14]. The actual amplification factor depends on many circumstances such as the configurations of the servers et cetera. The problem still persists today because of the large amount of open DNS resolvers (see Table 2.2).

One of the more common and modern amplification attack is a DNS RDDoS attack [18]. Figure 2.5 shows a simple example of DNS amplification, where the request is smaller than the triggered response. For Reflective Distributed Denial of Service (RDDoS) attackers it is important to maximise the victims incoming traffic while minimising the attackers outgoing traffic.

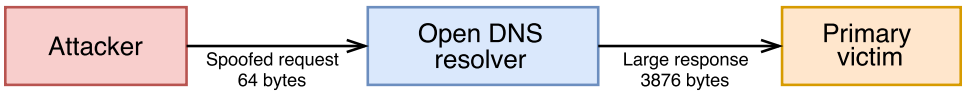


Figure 2.5: Simplified example of a DNS amplification attack with a single attacker [19].

Table 2.1: Table of reflection protocols and their amplification factor collected and combined from several sources [14, 15, 16].

Protocol	**Amplification factor**
NTP	556,9
CharGEN	358,8
DNS	1 - 179
QOTD	140,3
RIPv1	131,24
Quake Network Protocol	63,9
LDAP	46-55
BitTorrent	4,0 - 54,3
SSDP	30,8
Portmap	7 - 28
Kad	16,3
Multicast DNS	2 - 10
SNMPv2	6,3
Steam protocol	5,5
NetBIOS	3,8

Table 2.2: Top 20 Countries with open Recursive DNS servers [17].

Country	**Total**	**Country**	**Total**
China	1,498,166	Japan	60,731
United States	306,564	Indonesia	50,945
Republic of Korea	289,506	Islamic Republic of Iran	46,386
Brazil	200,980	Australia	44,455
Taiwan	184,089	Romania	43,928
Russian Federation	145,883	Bulgaria	43,087
India	98,895	Ecuador	35,676
Turkey	86,264	Argentina	35,578
Philippines	76,574	Canada	33,198
Poland	75,508	Mexico	33,108

2.4 Machine learning

Broadly there are two classes of machine learning called unsupervised learning and supervised learning. Supervised learning is used when the dataset does have an "expected output variable" such as a continuous value or class, which we will refer to as labelled data. Using this variable the machine learning algorithm can train its model on the input data, after which it is able to perform estimations on data where the output variable is unknown.

On the other hand, unsupervised learning is used when the dataset does not have an "expected variable" to train on, which we will refer to as unlabelled data. In this case, the data can be used to find patterns or clusters, but not to learn an estimated value as there is nothing to learn.

Classification can be used to let a computer decide which class a specific row in a dataset belongs to based on its trained decision boundaries. This is ideal for later research where we will try to classify attacks based on the RDDoS tool used to initiate the attack. As mentioned before this is a form of supervised learning. There are broadly two types of classification: binary classification and multiclass classification (see Figure 2.6). Binary classification is used when the data can only be one of two classes, whereas multiclass classifiers can classify any number of classes.

In the following sections four algorithms will be highlighted that are used in this research: Density-Based Spatial Clustering of Applications with Noise (DBSCAN), Multiclass Neural Network (MNN), Multiclass Logistic Regression (MLR) and Multiclass Decision Forest (MDF). The first algorithm is a clustering algorithm that is used for unsupervised learning while, on the other hand, the latter three algorithms are classification algorithms that are used for supervised learning.

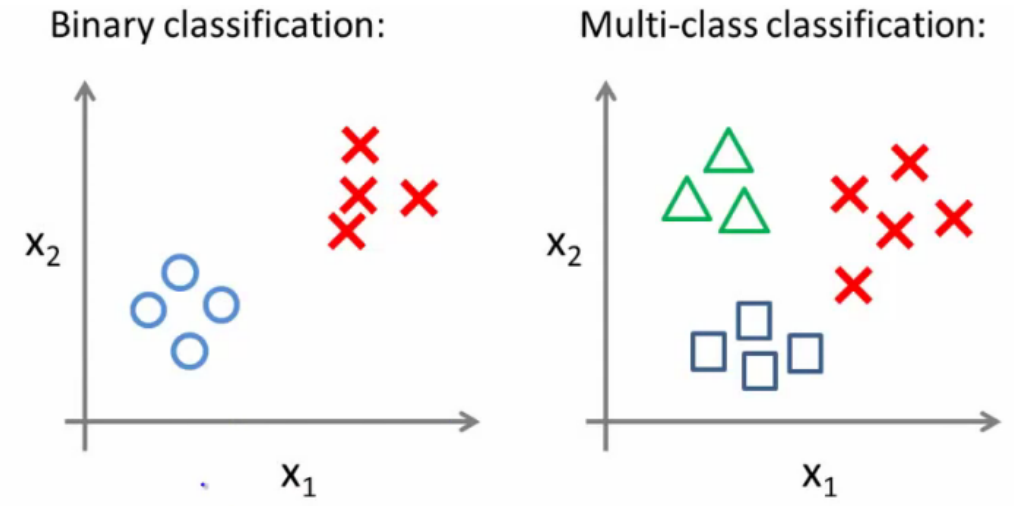


Figure 2.6: Simple 2D with two features: X1 and X2. On the left binary classification is shown where only two classes are classified, whereas on the left multi-class classification is shown with in this example three classes [20].

2.4.1 Density-Based Spatial Clustering of Applications with Noise

In order to cluster unlabelled data into a yet unknown number of clusters the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm can be used. DBSCAN clusters features into a multidimensional space where the distance between each feature is measured. If enough items are within a certain distance it is marked as an independent cluster. This method is able to cluster data without knowing the amount of clusters that are present and the clusters may be of any shape. Considering the fact that we do not know the number of clusters and the fact the data we will use contains more than two features DBSCAN seems to be a good fit for clustering our data.

It is important to note that DBSCAN requires two user defined parameters to perform the

data modelling. A user must specify the maximum distance ϵ and the minimum number of points in a cluster. The ϵ specifies the maximum distance two points may be apart to be considered neighbours. A spatial analysis is then done and, when enough items are within distance vector ϵ, a cluster is found (see Figure 2.7).

DBSCAN is therefore designed to find *core points*. These core points are points which have more or equal points in distance ϵ than the minimum points and therefore form a cluster. These core points are all identified in a certain dataset. If two core points are within one ϵ of each other they form a cluster together, including all border points of both core points. DBSCAN checks every core point for mutual core points and thus builds op a cluster without having to know the number of clusters or the shape of the clusters.

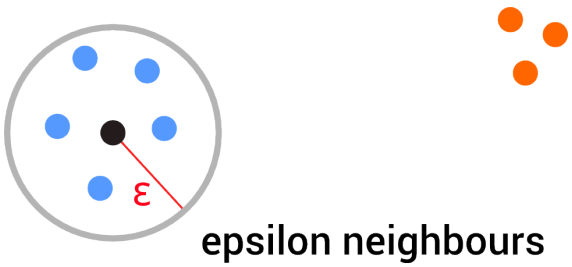


Figure 2.7: A demonstration of DBSCAN where a cluster is formed when points are within ϵ spacial distance [21].

2.4.2 Multiclass Neural Network

The Multiclass Neural Network (MNN) algorithm uses an Artificial Neural Network (ANN) to classify data into multiple classes. Artificial Neural Networks use terminology analogous to the human brain: neurons (nodes) are the basic foundation of an ANN and transform incoming values into outgoing values. An ANN consists out layers of multiple neurons that are interconnected by synapses, which are weighted edges [22]. There are three types of layers: the input layer, one or multiple hidden layers, and a single final output layer (see Figure 2.8). All these layers perform different kinds of transformations and the weights of the synapses are trained using the input data. Deep Neural Networks contain many hidden layers and perform difficult and computationally intensive tasks. However, often predictive tasks only need a few hidden layers [23]. Classification using neural networks is a supervised learning model, and therefore requires a labelled dataset that includes a label column. The advantages of using the Multiclass Neural Network algorithm are the high accuracy and complexity of the model allowing it to do difficult tasks, though this last property also can be a downside resulting in long training times.

2.4.3 Multiclass Logistic Regression

The Multiclass Logistic Regression (MLR) algorithm uses logistic regression to classify data into multiple classes. It is sometimes also referred to as Multinominal Logistic Regression or Polytomous Logistic Regression [24]. Just as the MNN algorithm it uses supervised learning to train the model and therefore requires a labelled dataset. Ordinary logistic regression can be used to predict the probability of a single outcome and can therefore be used for binary classification. Multiclass Logistic Regression is able to predict the probability of multiple outcomes and is therefore perfectly suitable for multiclass classification [25].

One advantage of the MLR algorithm is that after training the weight assigned to features can be reviewed. This is valuable data, because from this data it can be inferred which features are important to which classes, and which features are not of any value to the trained model. One of the biggest advantages of using the Multiclass Logistic Regression over a Multiclass Neural Network is that it is faster in training the model. However, the MLR model trains on linear relationships, making it less suitable for data showing only non-linear relationships.

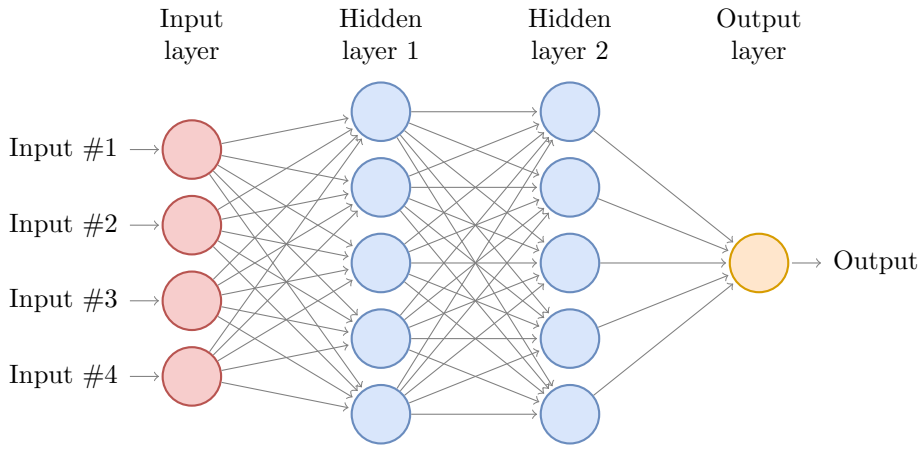


Figure 2.8: Example of an Artificial Neural Network (ANN) with one input layer consisting of four neurons, two hidden layers both consisting of five neurons and one output layer consisting of one neuron.

2.4.4 Multiclass Decision Forest

The Multiclass Decision Forest (MDF) algorithm, not to confuse with the Multiclass Decision Jungle algorithm, makes uses of decision trees to classify data. The decision forest algorithm builds up a series of decision trees during the training of the model. It builds multiple decision trees (see Figure 2.9) and then uses a voting mechanism to select the best N decision trees. The voting process makes use of non-normalised frequency histograms, which are then summed and normalised to get the probabilities for all labels. Trees with the highest prediction confidence will be selected as the trees to use [26].

Advantages of using the Multiclass Decision Forest algorithm are that it is efficient both in computation time and memory usage. Additionally, decision trees are not influenced by noise and can have non-linear decision boundaries. However, decision trees may be too simple and restrictive for more complex problems [27].

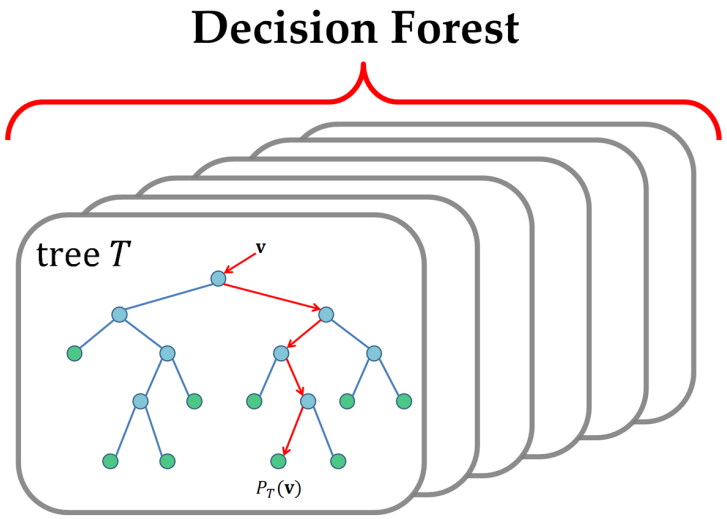


Figure 2.9: The Multiclass Decision Forest (MDF) builds multiple decision trees, after which a voting mechanism selects the N best trees based on the highest prediction confidence [28].

CHAPTER 3

Methodology

This research will be using two data sources of network packet captures. The first dataset will be real data provided Fox-IT from actual RDDoS attacks, which is collected from open DNS resolvers which intentionally serve as honeypots. The second dataset will be generated in a test lab. As mentioned in chapter 2 capturing the data at the victim's side is virtually useless considering the packets generated by the service are legitimate and have little connection barring the domain name and certain DNS query settings. All data used will be request packets from an RDDoS DNS tool to a reflector.

We will be performing two experiments. First, we will analyse the actual data as provided by Fox-IT. This will consist of a static analysis where we will identify and analyse all differentiating features between the attacks. Our second experiment will be performed on a different dataset. In order to test trained -and other machine learning algorithms on attacks we will be creating our own test dataset in a lab using different RDDoS DNS tools. With this dataset, we can verify whether we can identify different RDDoS attack tools since all PCAPs are labelled.

The data Fox-IT captures is based on multiple honeypots providing open services for a whole host of RDDoS services like DNS, NTP, Chargen etc. The requests used in these attacks are then stored in a MongoDB database where they are stored by an arbitrary attack ID. The attacks are grouped based on time of the attack and the victims IP address. Fox-IT will be providing us with two datasets. The first dataset contains 14807 PCAPs with 25 packets per PCAP. The second dataset consists of 1868 PCAPs with 250 packets per PCAP.

For this research, we choose to focus on DNS as an RDDoS attack protocol. We considered the limited time in which this research can be done and decided that it would be more beneficial to really investigate DNS rather than shallowly investigate and discuss multiple protocols.

3.1 Lab generated data

A mentioned in the introduction a part of the captures from RDDoS attacks will be generated in a lab environment. Because the data is generated it is known beforehand which tools are used. Therefore, the data can be labeled and used for supervised learning.

3.1.1 DNS DDoS scripts

In this research four different DNS RDDoS tools are used, which are ethan, flooder, saddam and tsunami (see Table 3.1. Flooder, Ethan and Tsunami are written in C, whereas Saddam is written in Python2. The scripts differ in functionality, some support multi-threading whereas others only support a single thread. Some allow a user to set the UDP source port, whereas other scripts have a fixed or random UDP source port.

One property of these scripts is that they try to utilise Domain Name System (DNS) amplification, such as described in section 2.3. The higher the amplification factor, the more effective the script is. In order to make sure that the scripts used actually do have an amplification factor higher than 1.0 a simple measurement was made before using them in the experiments. To measure the amplification factor of these scripts they are executed one by one, capturing the frame length of packets with respectively the UDP source port or the UDP destination port set to 53 on the reflector. This results in Table 3.2 and Figure 3.1, where the amplification factor is measured for each tool.

Table 3.1: Tool comparison

DNS tool	Source	Language	Threading
saddam	GitHub.com	Python 2	yes
flooder	Pastebin.com	C	yes
ethan	GitHub.com	C	no
tsunami	Infosec-Ninjas	C	yes

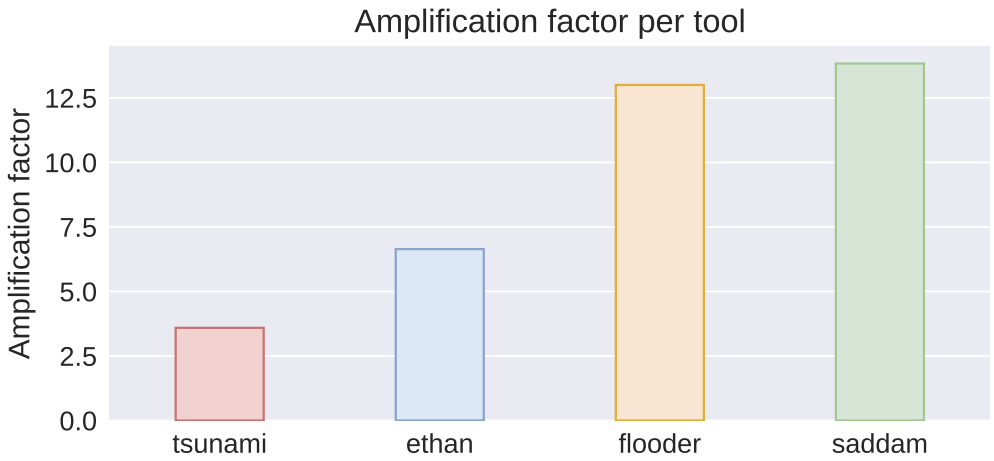


Figure 3.1: Amplification factor per tool

Table 3.2: Tool amplification measure. Frame lengths are in octets. The amplification factor is the outgoing frame length divided by the incoming frame length.

DNS tool	Incoming frame length	Outgoing frame length	Amplification factor
tsunami	78	280	3,59
ethan	67	445	6,64
flooder	83	1079	13,00
saddam	78	1079	13,83

3.1.2 Test environment setup

For the purpose of generating labelled RDDoS captures a test environment was devised. The setup consists out of three servers, connected with simple a switch. The three servers serve as the attacker that conducts the attack, the reflector which is configured as an open DNS resolver, and the victim that receives the attack. The setup is shown in Figure 3.2. The specifications of the setup are listed in Table 3.3.

Table 3.3: Test environment setup specifications.

Machine	**Type**	**CPU**	**Memory**	**OS**
Attacker (dijon)	Dell Poweredge R230	Intel Xeon CPU E3-1240L 2.10GHz	16 GB	4.10.0-22-generic Ubuntu 17.04
Reflector (nice)	Dell Poweredge R230	Intel Xeon CPU E3-1240L 2.10GHz	16 GB	4.10.0-22-generic Ubuntu 17.04
Victim (berlin)	Dell Poweredge R210 I	Intel Xeon CPU L3426 1.87GHz	16 GB	4.10.0-22-generic Ubuntu 17.04
Network switch	TP-Link TL-SG1008D	-	-	Unmanaged

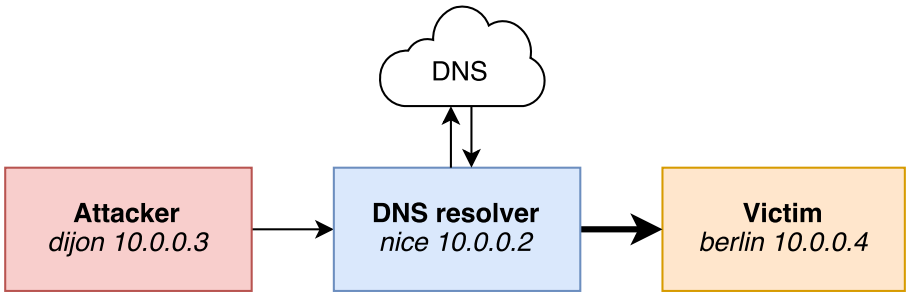


Figure 3.2: Example of the test lab performing a DNS RDDoS, where the DNS record is first resolved and then send from the reflectors cache to the victim Berlin.

3.1.3 Attacking & data collection

Executing the attacks and capturing the network packets is completely automated by a script (see A.1). The captures are collected and stored on the reflector (the open DNS resolver). First, script starts in a counting loop. In this case, this loop is done 250 times. At each instance, the script opens up an SSH session to the attacker host, launches the attack script with the correct parameters, and retrieves its PID. Then the script opens up a second SSH session to the reflector host, at which it starts `tcpdump` to capture the packets with on UDP destination port 53 and with the IP address of the victim. The `tcpdump` program is set to capture a million rows, after which it terminates. Because the PID of the monitoring SSH session also is stored, and the SSH session terminates as soon as a million captures are made, the script waits for this PID to finish, after which it kills the PID of the attack script at the attacker's host. Then, the loop can continue to the next attack. In total 1 million packets are captured per attack, and per tool 250 attacks are performed. Because in this research 4 different tools are used, this totals to an amount of 1 billion packet captures.

The `tshark` program is used to convert the binary PCAP files to Comma-separated values (CSV) files, where we select the columns that will be used. From these columns the features are generated, reducing a CSV from a million rows into a single tool Then, all 250-feature rows are merged into a single file with the attack tool as its label. The complete process is visualised in Figure 3.3. The script that converts PCAP to CSV is listed in appendix A.2, and the script that converts CSV files to features is listed in appendix A.3, with the helper functions listed in

appendix A.4 and appendix A.5. What features exactly are generated is listed in that particular appendix.

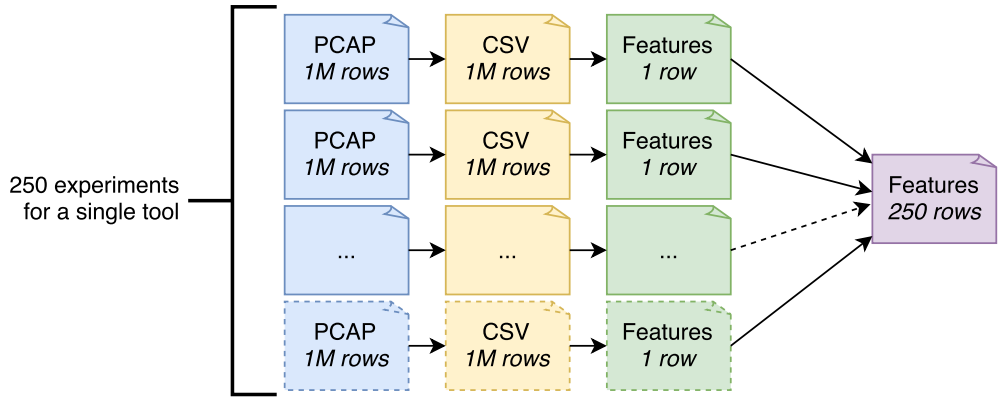


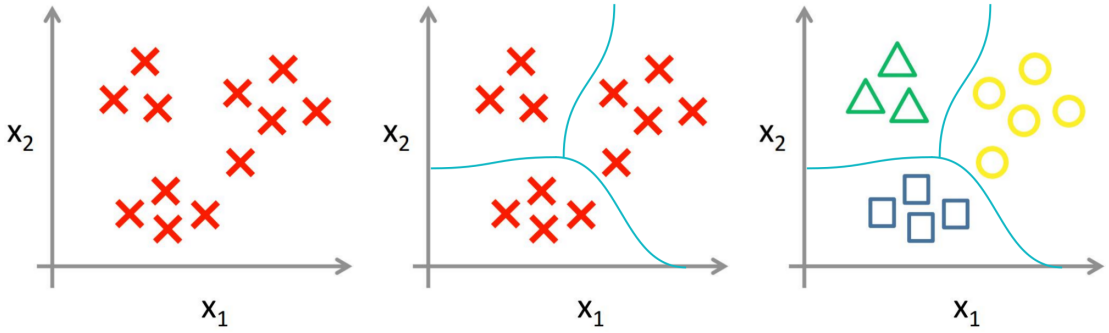
Figure 3.3: Illustration of data collection pipeline. First the network data is captured and stored as Packet Capture (PCAP) files. For each tool 250 experiments are done, each experiment capturing one million packets. These PCAP files are then converted into CSV files, only exporting the fields used for the features. Thereafter the features are generated from these CSV files, reducing one million captures into a single feature row. Finally all feature rows are merged into one large file containing the label of the attack tool used.

3.1.4 Multiclass classification

What we are trying to do is classify attacks based on the captured data. Figure 3.4a shows a simple 2D example of classification of unknown data with just two features: X1 and X2. When a model is trained beforehand on other data, those decision boundaries can be applied on this data (see Figure 3.4b). Then, if we feed the algorithm with data to be estimated, the algorithm can classify the data based on the decision boundaries, which is illustrated in Figure 3.4c.

3.1.5 Training the model

Now that the raw data is converted to features, it is time to train the model. The data is randomly split up into a training set and a test set, where the training set contains 90% of the data and the test set 10% of the data. The training set is fed into the machine learning algorithm. Once the model is trained, it is fed with the test data. For each row in the test data, the trained model will



(a) Unclassified data with features X1 and X2.

(b) Applying the decision boundaries of the trained model.

(c) Classifying the data based on the trained model.

Figure 3.4: 2D example of how a multiclass classifier conceptually works.

give an estimated class. This estimated class can be compared with the actual (known) class of the test row, which is how the model is evaluated overall. See Figure 3.5 for the whole process.

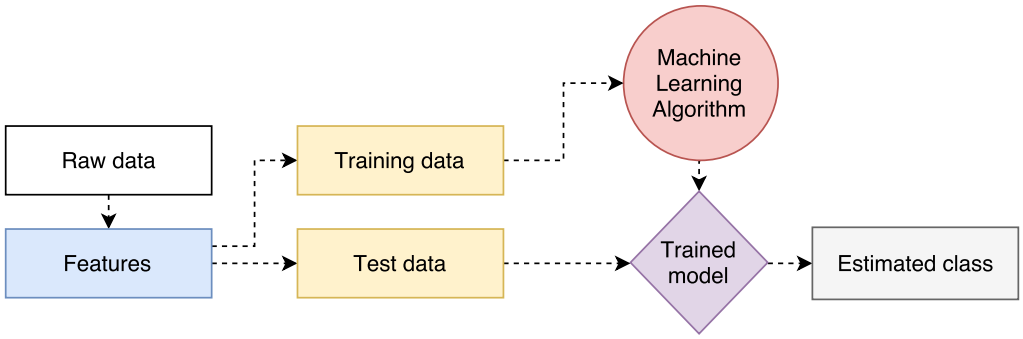


Figure 3.5: The pipeline for training a machine learning algorithm and using test data to evaluate the accuracy of the trained model. First, the raw data is converted into a set of features, which is then randomly split into a training dataset and a test dataset. In all the experiments of this report, the training dataset will contain 90% of the data whereas the test dataset contains 10% of the data. Feeding the training data into the machine learning algorithm results in a trained model, which can, in turn, be used to classify the test data, returning an estimated class for each row in the test data. Because the label is known beforehand, this can be used in combination with the estimated label to evaluate the accuracy of the model.

We also perform 10-fold cross validation. This reduces the odds of over-fitting on a specific train set, whereas now this experiment is repeated 10 times with other sub-sets of the data, making sure all the data is used both in training and in testing the dataset. This will execute the experiment 10 times with different sets of training and test data. The evaluations performed after each estimation are averaged into a total accuracy rate.

3.1.6 Azure Machine Learning

We chose Azure Machine Learning, which is a Software as a Service (SaaS) application, to train the models and evaluate the results. It has an intuitive interface that allows for fast prototyping without the need to write a lot of code. Also, supports visualisations and can import data from an HTTP web server, automating the entire process. See Figure 3.6 for an example pipeline in the Azure ML User Interface. This particular example first imports data from a storage account, after which it transforms the data into categorical features and renames some columns. Once finished, the resulting data is fed into a 10-fold cross validation module that uses the Multiclass Logistic Regression algorithm as a classifier. This module both trains and tests the model 10 times, after which the evaluation model calculates the statistics on accuracy and other performance measures.

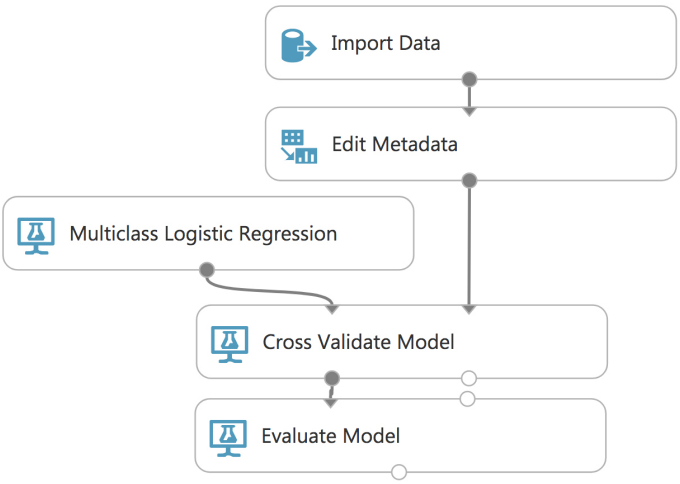


Figure 3.6: Example of the Azure Machine Learning interface that can be used to build pipelines that train, sore and evaluate models.

CHAPTER 4
Results

The results will be split in two distinct result sections. In the first section, we will review the results of the static analysis of the datasets provided by Fox-IT. The second section will look at the data generated by ourselves in the test lab.

4.1 Fox-IT data

Fox-IT provided us with two different data sets. The PCAPs contain DNS requests, UDP -and IP header data. These 3 parts of a network packet can then be used to try and discriminate between RDDoS attack tools.

4.1.1 Dataset 1: 25 packets per PCAP

The first dataset from Fox-IT contains 14807 PCAP's which contain a total of 368.442 packets to inspect. In order to differentiate between attacks we must find the differences between attacks. The first step was therefore to list all the fields in each packet and see where different values could be found. We have observed that the UDP source port, DNS ID, and IP ID often change both between PCAP's and inside PCAP's themselves. This could be an interesting feature when for example measuring the minimum and maximum values or variance of these fields.

Furthermore the TTL field was shown to vary between attack. However, we expect that this field will be detrimental to our efforts to identify RDDoS tools. Botnets may change from attack to attack. Some bots may be added where new machines are infected, or bots are lost where machines are cleaned. This means the TTL pattern could vary between attacks of the same tool. Therefore we will ignore the TTL in further experiments. Furthermore, the IP Header checksum must be correct for routers to forward the data so this field will be ignored as well since the way the field is used will be identical between all tools. The total length and source address were also found to vary. Obviously the source address is the address of the primary victim and can therefore vary between attacks. The total length was found to always be correct, since it is essential that the entire request is received correctly we expect that the total length will always be valid. We validated this assumption on the dataset and observed no invalid total length field. Therefore, we believe it is reasonable to assume that the total length field is not a noteworthy feature.

The UDP header was also inspected for valuable features. The source port was found to often change. The checksum, though not changing in our dataset, could be legitimately used by some tools and not used at all by others. We therefore decided that it could be a differentiating feature. The source port, as mentioned above, was shown to frequently change.

When we set the DNS ID, IP ID and UDP source port to 1 we observed other variances in the data. After listing the variance of the data we observed a different IP Header in a small number of packets in the dataset. These PCAP's have at least one packet with a Differentiating Service (DS) field (also widely known as a DSCP field, however we will adhere to the naming specified in the RFC [29]) set to `0x40` as shown in Table 4.1. An example of these packets are found in appendix B.

We can identify two different headers, one with the DS field set to `0x40` or 16 which assigns the packet to DS Field class to 2. This class sets the priority to *immediate forwarding of the packet* [30]. It is possible that the simple fact that the DS Field is set to 0x40 indicates two different attack tools. In order to verify this hypothesis we will split the data set into two. One set where the DS Field is set to 0x40 and one where the field is set to 0x00. Furthermore, it is also important to see if the fact that the DS field is uniform in a PCAP. It could be possible that

a PCAP contains a mix of DS field set to `0x40` and `0x00`. We observe that 39 PCAP's contain only packets with a DS field set to `0x40`, and 178 which contain one or more packet(s) with a DS field set to `0x40`. This indicates that a large number of attacks actually vary in the usage of the DS field.

In order to determine whether these differences indicates a different attack tool or not the datasets should be compared. The first observation we can make is that only 0,8% of the packets have the DS Field set to 0x40. This could indicate that only a small amount of attacks were made with a attack tool that uses this setting. However we should look further into the data and see if the attacks are actually different. When looking at the PCAPs containing at least one packet with a DS Field set 0x40, we observe that the DNS ID is repeated more times than the average repeat in the data set. When looking at the entire dataset we observe that out of the 14.807 PCAPs from the dataset, 74% of the PCAPs contain either 2 or 3 unique DNS ID's. However, when we filter to only look at PCAPs that contain at least one DS Field set to 0x40 we observe that the 56% of the PCAPs now have 1 unique DNS ID. Where 2 or 3 unique ID's makes up 74% of the PCAPs in the entire set, it only represents 34% of the this subset. All datasets and unique DNS ID values are represented in Table 4.2.

We observed some other interesting characteristics when analysing the subset of PCAPs containing at least one packet with a DS field set to `0x40`. In the entire dataset, we found 8 different domain names used for the queries. However, when only looking at the subset where the DS field is set to 0x40 we find 4. Moreover of the 217 PCAPs in the subset, 127 contained 2 specific domain names which were capitalised. When the data is again divided into capitalised and non-capitalised

Table 4.1: A representation of the IP header fields found in the DNS dataset 1.

Version	**Header length**	**DS**	**Flags**	**Frag. offset**	**Prot**	**Frequency**
4	20	`0x00000000`	`0x00000000`	0	17	365532 (99.2%)
4	20	`0x00000040`	`0x00000000`	0	17	2905 (0.8%)

Table 4.2: Collection of tables showing the occurrences of Unique DNS IDs per PCAP.

(a) Total data set.

Unique DNS ID	**Frequency**
1	2455 (16,5%)
2	6105 (41,1%)
3	4848 (32,6%)
4	1277 (8,6%)
5	157 (1%)
6	14 (0,09%)
7	1 (0,006%)

(b) Subset of a consisting of PCAPs containing at least 1 packet with DS Field set to 0x40.

Unique DNS ID	**Frequency**
1	123 (56,7%)
2	39 (18%)
3	36 (16,6%)
4	16 (7,4%)
5	3 (1,4%)
6	0 (0,00%)
7	0 (0,00%)

(c) Subset of B where query domains are capitalised.

Unique DNS ID	**Frequency**
1	105 (82,7%)
2	21 (16,5%)
3	0 (0,00%)
4	1 (0,8%)
5	0 (0,00%)
6	0 (0,00%)
7	0 (0,00%)

(d) Subset of B where query domains are not capitalised.

Unique DNS ID	**Frequency**
1	18 (20%)
2	18 (20%)
3	36 (40%)
4	15 (16,7%)
5	3 (3,3%)
6	0 (0,00%)
7	0 (0,00%)

domains an even clearer difference between PCAPs is observed. For capitalised domain names 82% of the PCAPs contain only one DNS ID. We believe it is likely that the PCAPs containing capitalised domains are from a different tool. Other differences were not found, the packets are shown in Appendix B.

Since the fields are all the same, and the only difference in the attacks are marginal, we will require more packets in order to truly check the patterns in the data. The actual data is almost uniform across all PCAP's. Only the DS field is different among all these PCAPs as well as the queried domain. Since we have only been able to identify two tools it will be necessary to do pattern recognition and further analysis on the data. However, Fox-IT provided us with 25 packets per attack which makes it impossible to do accurate pattern analysis. Therefore we were provided with a second dataset from Fox-IT for DNS attacks containing 250 PCAPs per attack. This should allow us to perform more in-depth analysis of the captures.

4.1.2 Dataset 2: 250 packets per PCAP

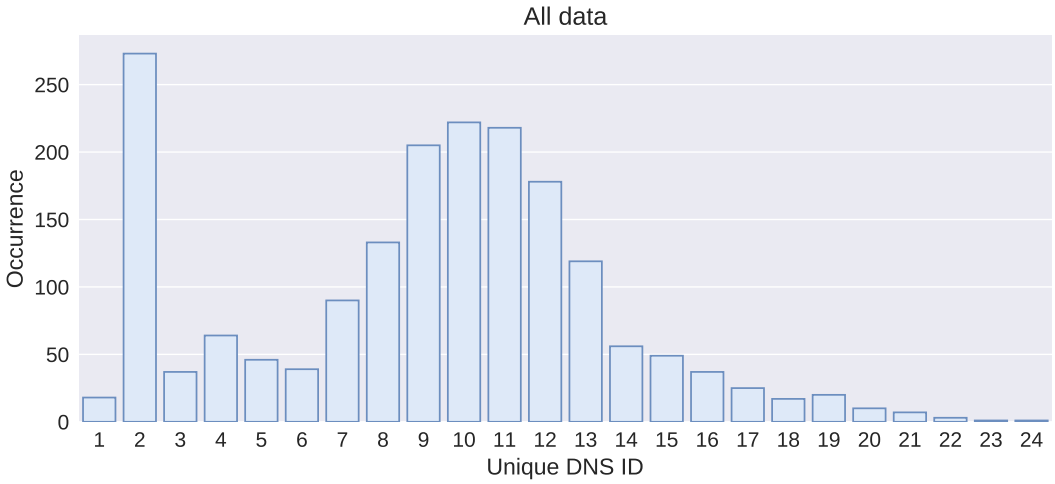


Figure 4.1: A chart plotting the unique DNS ID's found per PCAP in the entire dataset.

This dataset consists of 1.868 PCAPs with each containing approximately 250 packets. By repeating the experiments done on the previous dataset we can make some observations. First, the tool using capitalised letters seems to be missing in this dataset. We can still differentiate and compare the attacks containing at least one packet with the DS field set to 0x40. After splitting the dataset we can see a very distinct count of unique DNS IDs depending on the data.

Observing the entire dataset it becomes clear that a large number of PCAPs contain a unique DNS ID count from 7-13 make up 62% of the attacks as shown in Figure 4.2. However, we can observe that plenty of PCAPs have only 2 unique DNS IDs. Interesting is that between 2 and 7-13 the number of unique DNS IDs found is relatively low, which shows that PCAPs with 2 unique DNS IDs and 10-13 are different tools. Furthermore, when only looking at PCAPs where unique DNS ID count < 7 we observe that a unique DNS ID count of 2 represents 60% of the data set. The distribution of unique DNS IDs is is shown in Figure 4.2. A clear difference between the complete dataset in Figure 4.1 and this dataset can be observed.

We can conclude that PCAPs containing at least one packet with a DS field set to `0x40` behave differently compared to the entire dataset when looking at the unique DNS ID count. We can validate this the hypothesis that this is indeed a different tool by looking at the behaviour of the other frequently changing data fields like UDP source port and IP ID. When we look at the unique IP IDs in the PCAPs we can observe that different behaviour is shown much like the DNS ID as shown in Figure 4.3 and Figure 4.4.

When we plot all the unique IP IDs found in Figure 4.3 we can observe a chart that vaguely resembles a Gaussian distribution. However when we narrow the dataset down to only include PCAPs where the DS field is set as shown in Figure 4.4 we can observe an entirely different

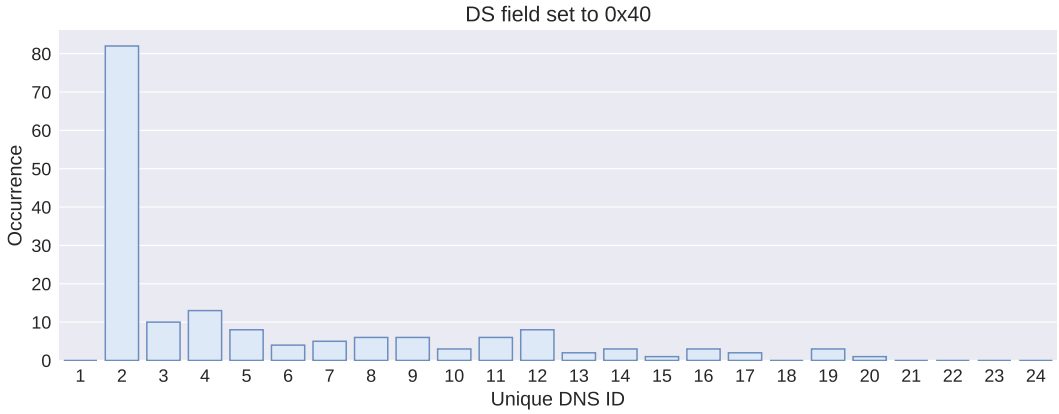


Figure 4.2: Unique DNS IDs in PCAPs where at least one packet contains a DS field set to 0x40. A significant higher relative presence of 2 unique DNS IDs can be observed here.

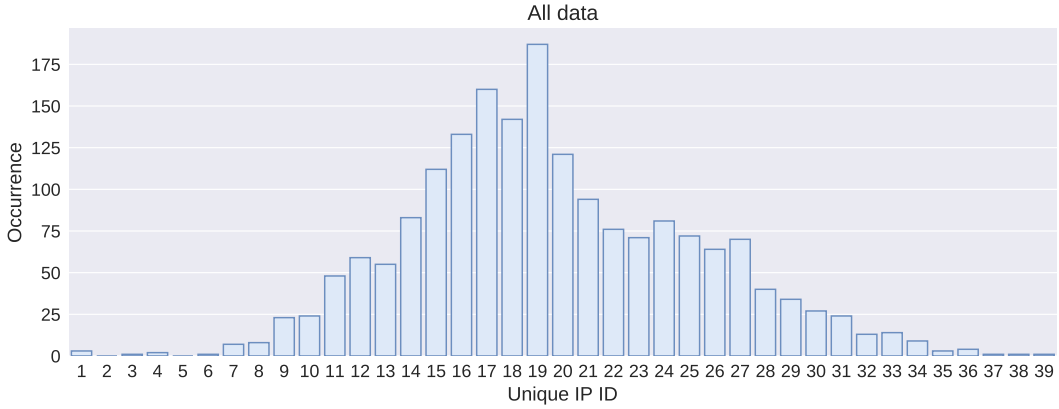


Figure 4.3: Unique IP ID observed in all PCAPs

distribution of the data. This different behaviour further justifies our belief that the PCAPs containing a set DS field are from a different tool. It should be noted here that the distribution of the last frequently changing field, the UDP source port, shows very similar behaviour as the IP ID. We will therefore not look into it further in this report.

Furthermore, we observed some PCAPs contain malformed packets as shown in appendix B. This could also be an indicator of a different tool. However, when we again try and recognise different patterns in frequently changing fields we find no major differences between the PCAPs containing malformed data and all data. If we compare the unique IP IDs from Figure 4.5 and all data in figure Figure 4.3 we observe that the distribution appears to be identical. We even observed packets that both have malformed packets and a DS field set to 0x40.

This led us to believe that perhaps, PCAPs with no malformed packets could lead to a new tool. When we plot the unique DNS ID's of tools with no malformed packets we can see slight differences. However, we do not believe this is conclusive enough to identify an attack. We can make one interesting observation here though, PCAPs with only one unique DNS ID never have the DS field is set to `0x40` and never contain a malformed packet. This could indicate a different tool.

However, there are other subtle differences we can use to identify the tools. When the frequently changing fields change, and how they change compared to each other can also be used to identify tools. We observed that when the DNS IDs are reused for multiple domain names, the combination of DNS ID, UDP source port and IP ID are never unique. This means that the DNS ID is not only used with multiple domain names, but also with multiple UDP source ports and IP IDs. We

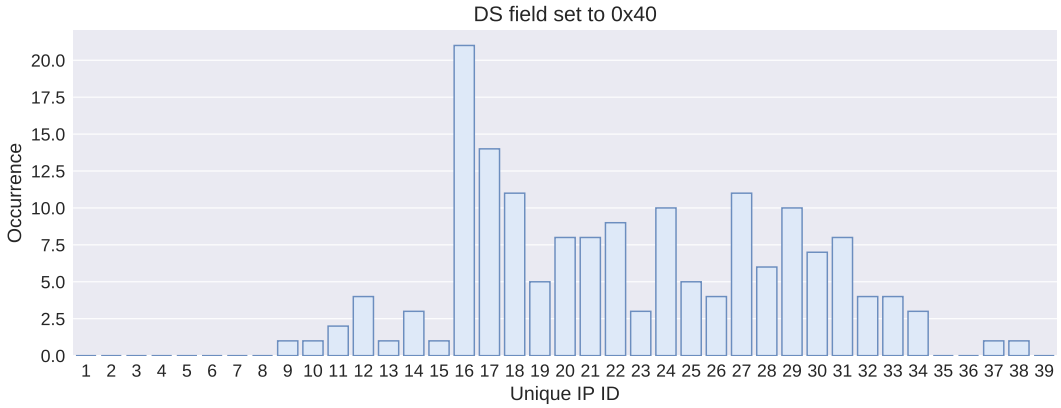


Figure 4.4: Unique DNS IDs in PCAPs where at least one packet contains a DS field set to 0x40. A significant higher relative pressence of 2 unique DNS IDs can be observed here

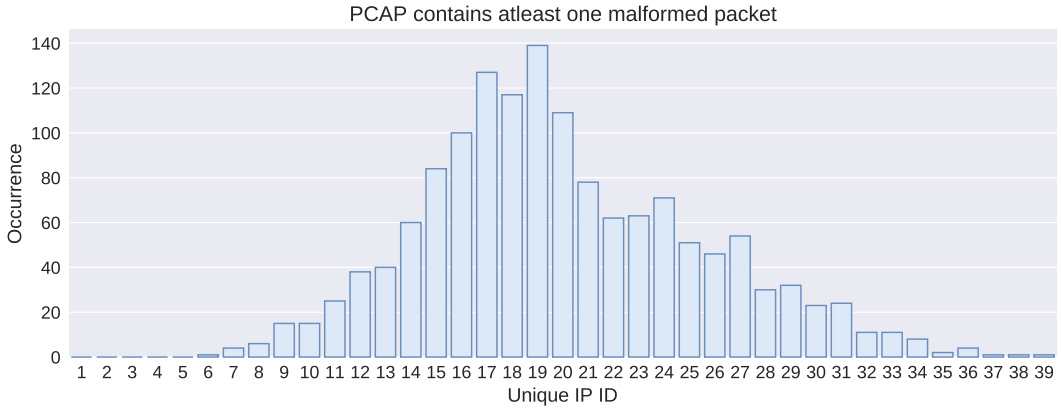


Figure 4.5: Unique IP ID in PCAPs with malformed packets. Distribution remains roughly equal compared to the entire dataset.

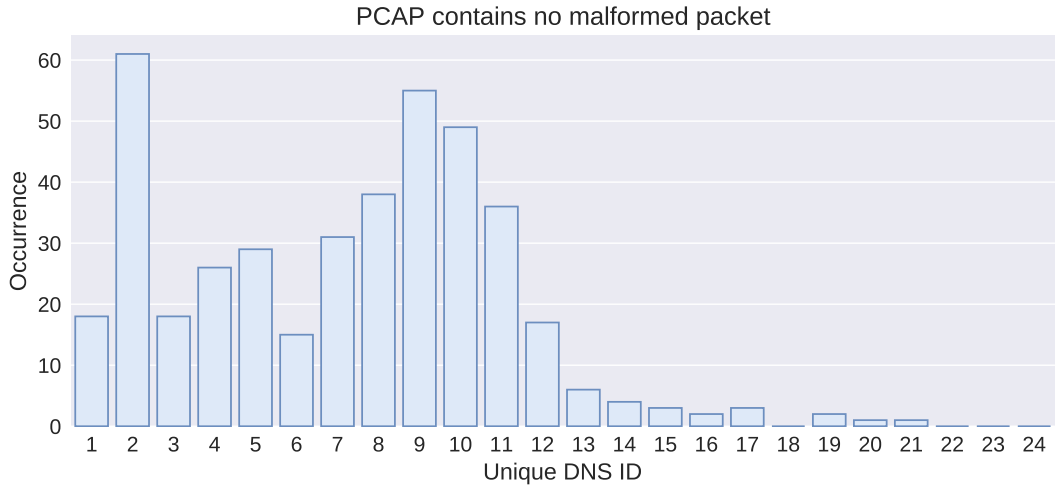


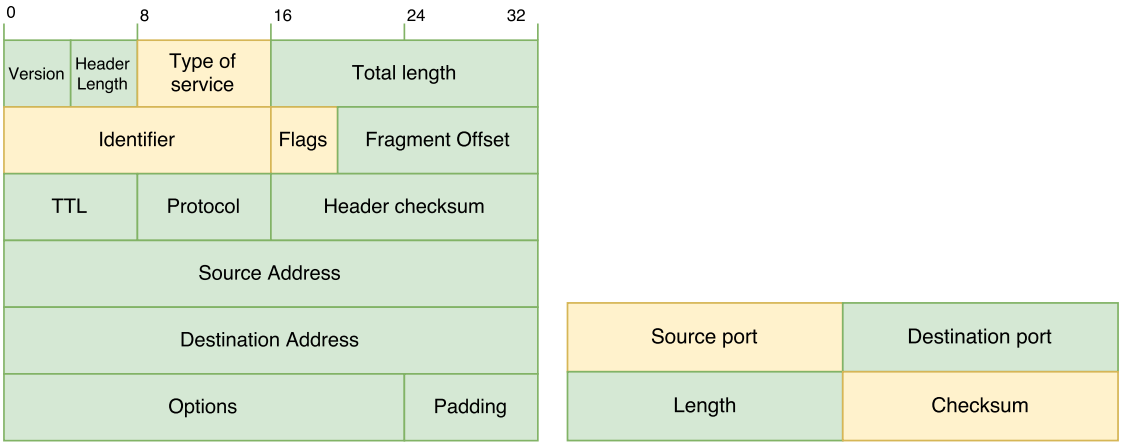
Figure 4.6: Unique DNS IDs in PCAPs where no malformed packets are present. Distribution remains roughly the same.

believe this behaviour is a sign of a different tool.

Furthermore, when we further look into the PCAPs containing only one DNS ID we have observed some PCAPs containing a static IP ID, DNS ID and UDP source port. We expect that this is also a different tool considering the static nature of these normally changing fields.

Moreover, we expect that the large peak around 2 unique DNS IDs is a different tool compared to the more equal distribution of around 10 unique IDs. However, we have found no further substantial evidence to substantiate this claim. Without clear differences in behaviour, we believe it is incorrect to determine these small differences as different attacks.

After this static analysis, we believe we have identified the important features that can be used to identify an attack. For the IP header, we believe that the DS Field or Type of Service field, identifier (the IP ID) and flags can show substantial differences. For the UDP header, we expect that the source port and checksum could be interesting. The headers and their fields are shown in Figure 4.7 For the DNS payload, we believe that every field of the packet could be interesting. Furthermore as mentioned we will be using metadata to determine the attack tool. Every feature we will be using is specified in appendix C.



(a) IP header

(b) UDP header

Figure 4.7: A diagram of an IP header (4.7a) and a UDP header (4.7b). Yellow fields will be used for fingerprinting where green fields will be ignored.

We would like to be able to recognise the different tools without having to perform a cumbersome static analysis of the data. By identifying features that could be interesting we believe that we can cluster the data using DBSCAN specified in subsection 2.4.1. First, we plotted the Fox-IT data simply using every feature. In this example we set the ϵ to 1,5 and the plot is shown in Figure 4.8.

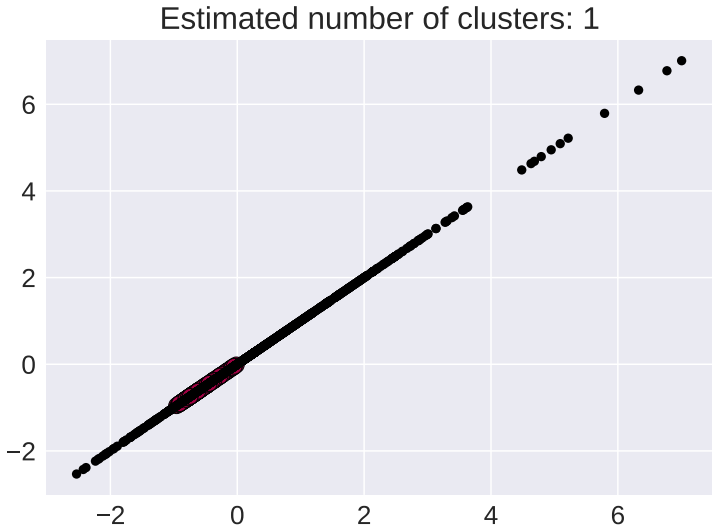
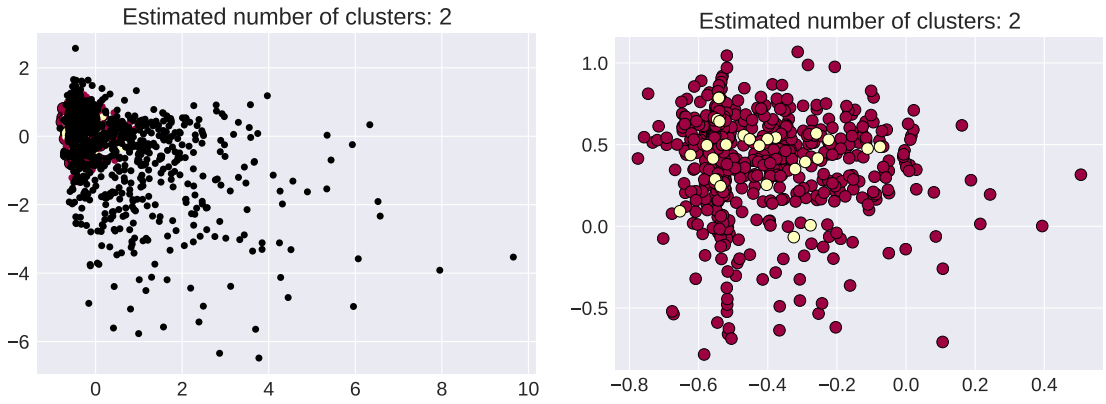


Figure 4.8: Scatter plot of all features from the Fox-IT dataset 2. Black dots displays noise.

We expect that the linear form of the clustering is due to a number of features being identical. As we mentioned all packets are almost identical and therefore hard to cluster using all features. We, therefore, expect that using only the features that differ from each other should give different results as shown in Figure 4.9. Note that DBSCAN plots multiple features to a 2D figure which can make the clustering unclear as is the case here.



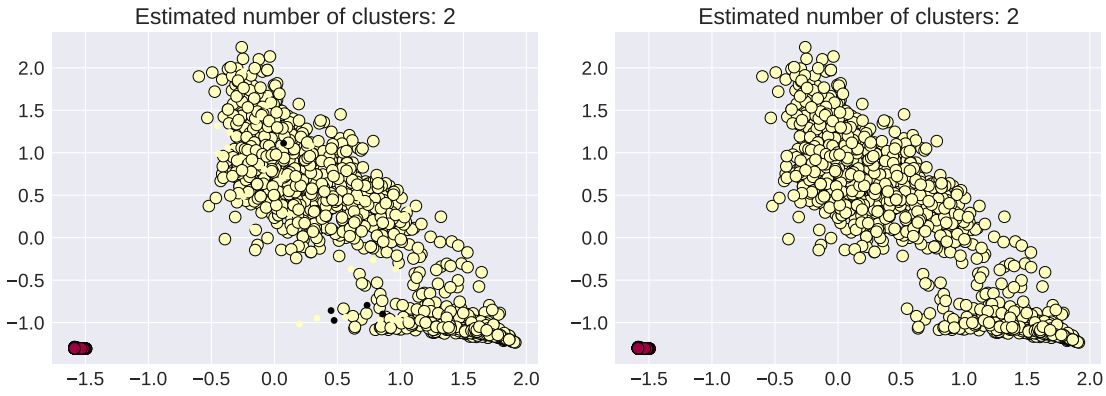
(a) Unique features with noise

(b) Unique features without noise

Figure 4.9: Clustering of the Fox-IT dataset based on unique features, one with noise (4.9a) and one clustering with noise removed (4.9b).Black dots indicate noise, while clusters are grouped by colour.

In order to further increase the accuracy of the clustering algorithm, we will attempt to narrow down the features used to features we have identified to differentiate the attacks. The features used in this case are as followed: DNS ID longest repeat, DNS ID unique length, DNS RR UDP payload minimum size, IP ID longest repeat, IP ID unique length, IP DS field unique length, UDP source port longest repeat, and UDP source port unique length as shown in Figure 4.10

Furthermore, we would like to test if the clustering algorithm is capable of detecting new tools. The tools used in the generation of our lab data could be interesting to add to the data. For this we took 250 packets per PCAP of lab-generated data and merged it with the dataset from Fox-IT. We can then compare and see if the tools we use show new clusters. Since we are now unsure whether the selected features are enough to identify the data we will use unique features to identify

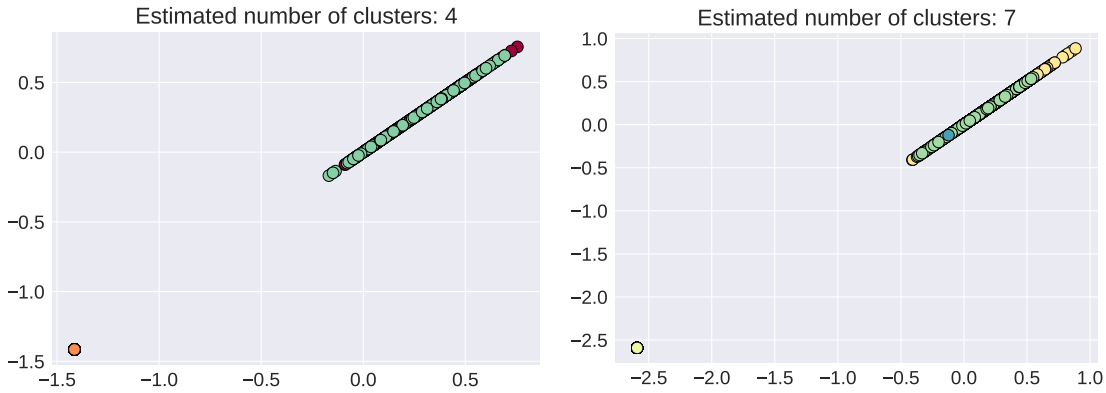


(a) Selected features with noise

(b) Selected features without noise

Figure 4.10: Clustering of the dataset based on selected features, with noise (4.10a) and without noise (4.10b). Black dots indicate noise.

the clusters. First, to test the clustering algorithm we clustered our lab data expecting to find 4 clusters.



(a) Cluster of lab generated data based on the unique features

(b) Cluster of lab generated data combined with Fox-IT data based on the unique features

Figure 4.11: Clustering of unique features of the lab generated dataset (4.11a) and the lab generated data combined with Fox-IT data (4.11b).

We can observe new clusters showing up in Figure 4.11b. However, we would expect to find between four and six clusters instead of seven since the clustering in Figure 4.9b shows two clusters when the Fox-IT dataset is clustered based on unique features. A reasonable explanation for this would be a change in the unique columns which could change the clusters. Another plausible explanation is that outliers of our lab generated data could, together with Fox-IT data outliers, form a cluster which is larger or equal to the minimum points. A complete overview of our findings in the Fox-IT data can be found in Table 5.1.

4.2 Lab generated data

This section describes the results produced by training and evaluating several multiclass classifiers. The data used, which in total consists of 250 billion captured packets, is collected in a closed environment spawning automated Reflective Distributed Denial of Service (RDDoS) attacks using the methodology described extensively in section 3.1.

4.2.1 Accuracy results

Initially, two algorithms were used to test the multiclass classifiers. These algorithms were the Multiclass Neural Network (MNN) algorithm and the Multiclass Logistic Regression (MLR) algorithm, which are both described in subsection 2.4.2 and subsection 2.4.3. Table 4.3 shows the first evaluation for both algorithms. For this first evaluation the lab generated PCAP data was used, where each PCAP signifies one attack initiated by one script and holds 1 million packet captures.

Table 4.3: The first classifier evaluation results after training the multiclass classifier using a MNN and a MLR algorithm. For these results the lab generated PCAP data was used containing 1 million packets per PCAP file. Because both the precision and recall are at 100% this results in a 100% accuracy.

Evaluation measure	**Multiclass Neural Network**	**Multiclass Logistic Regression**
Precision	100%	100%
Recall	100%	100%
Accuracy	100%	100%

Because of these highly accurate results, it is attractive to look further into the data. Perhaps it is possible to reduce the overall number of captures collected from an attack or to reduce the number of features that are used while training the model. Reducing the number of captured packets or the number of features reduces the storage required, training time for the model, and the amount of time needed to capture the data before it can be fed into the classifier.

To verify the possibility to reduce these components the model was retrained with fewer packets per attack, specifically with 10.000 packets and with 1.000 packets. The results of the reduction in captures per attack are shown in Table 4.4. We can observe that reducing the number of packets per attack to 10.000 of 1.000 does not influence the accuracy of the classifier.

Table 4.4: The classifier evaluation results after training the multiclass classifier with lower numbers of captures per attack. As can be seen reducing the captures per attack to 1.000 does not influence the accuracy of the classifier.

Captured packets per attack	**MNN accuracy**	**MLR accuracy**
1.000.000	100%	100%
10.000	100%	100%
1.000	100%	100%

The question remains whether this reduction in captured packets has other consequences than the accuracy. Both the MNN and the MLR algorithms have the property to express the probability for each class. Table 4.5 shows an excerpt for the evaluation results for the MLR algorithm. Note that this table is heavily condensed both in columns and rows in order to fit on the page. In this table it is clear that the MLR algorithm scores each feature of rows with a certain probability. The first row, for example, has a score of probability score of 0,9864 for the class tsunami. Because this probability is the highest, the model will classify this row as tsunami. Because the original label is also known, it can be compared to this known label to evaluate the correctness of the model.

Table 4.5: Example of scored dataset for MLR with 1.000 captures. Features are omitted to fit the table on the page.

label	dns.id_longest_repeat	dns.id_max	dns,id_max	P(flooder)	P(ethan)	P(saddam)	P(tsunami)	Scored label
tsunami	1000	32037	32037	0,0060	0,0046	0,0030	0,9864	tsunami
ethan	1000	3667	3667	0,0017	0,9896	0,0023	0,0064	ethan
flooder	5	65490	65490	0,9821	0,0029	0,0098	0,0052	flooder
ethan	1000	19330	19330	0,0017	0,9896	0,0023	0,0064	ethan
flooder	5	64218	64218	0,9821	0,0029	0,0098	0,0052	flooder
ethan	1000	22118	22118	0,0017	0,9896	0,0023	0,0064	ethan
tsunami	1000	26663	26663	0,0062	0,0051	0,0031	0,9857	tsunami
saddam	1	65494	65494	0,0095	0,0025	0,9852	0,0028	saddam
tsunami	1000	18797	18797	0,0060	0,0047	0,0030	0,9863	tsunami
flooder	5	59540	59540	0,9820	0,0029	0,0099	0,0052	flooder
tsunami	1000	52228	52228	0,0062	0,0051	0,0031	0,9856	tsunami
tsunami	1000	58719	58719	0,0058	0,0043	0,0029	0,9870	tsunami
flooder	9	61002	61002	0,9821	0,0029	0,0098	0,0052	flooder
flooder	5	49737	49737	0,9821	0,0029	0,0098	0,0052	flooder
flooder	5	59058	59058	0,9821	0,0029	0,0098	0,0052	flooder
ethan	1000	30098	30098	0,0017	0,9896	0,0023	0,0064	ethan
saddam	1	65533	65533	0,0094	0,0025	0,9853	0,0028	saddam
saddam	1	65488	65488	0,0096	0,0026	0,9850	0,0029	saddam
ethan	1000	16519	16519	0,0017	0,9896	0,0023	0,0064	ethan
tsunami	1000	35850	35850	0,0059	0,0045	0,0030	0,9867	tsunami
flooder	5	63435	63435	0,9822	0,0029	0,0097	0,0052	flooder
saddam	1	65519	65519	0,0096	0,0025	0,9851	0,0029	saddam
tsunami	1000	35368	35368	0,0059	0,0046	0,0030	0,9865	tsunami
tsunami	1000	7281	7281	0,0061	0,0049	0,0031	0,9859	tsunami
tsunami	1000	40566	40566	0,0062	0,0052	0,0031	0,9854	tsunami
flooder	10	60818	60818	0,9821	0,0030	0,0098	0,0052	flooder
flooder	5	62222	62222	0,9820	0,0030	0,0098	0,0052	flooder
tsunami	1000	8287	8287	0,0059	0,0044	0,0029	0,9868	tsunami
tsunami	1000	18732	18732	0,0060	0,0047	0,0030	0,9863	tsunami
saddam	1	65499	65499	0,0097	0,0026	0,9849	0,0029	saddam

Because these probabilities are stored these can be averaged for the correct label, which signifies a certain amount of "certainty" of the trained model. Table 4.6 shows these average scored probabilities respectively using a MNN and a MLR while reducing the number of captured packets. The MNN turns out to be more confident in general with an average probability of 99% compared to 98% for the MLR algorithm. However, both these values are exceptionally high and therefore the differences are insignificant for the final result, especially since both algorithms have an accuracy of 100%. Additionally, we can note that decreasing the number of captures for a feature row barely influences the confidence.

Though we deem the differences to be insignificant, it is interesting to note that in 7 out of 8 cases the average scored probability for the correct class increases instead of decreases by a minimal amount (the exception is the saddam class in the MLR). Because we also consider these differences as insignificant we can conclude that reducing the number of captures has no meaningful impact on the classification and can safely be done without influencing the result.

Table 4.6: Average scored probabilities for classes using MNN and MLR.

Attack label	**Algorithm**	**1.000.000 captures**	**10.000 captures**	**1.000 captures**
flooder				
	MNN	0,9999231884	0,9999231884	0,9999321089
	MLR	0,9852063487	0,9852063487	0,9852958462
ethan				
	MNN	0,9999454031	0,9999454031	0,9999536729
	MLR	0,9882222996	0,9882222996	
saddam				
	MNN	0,9999306398	0,9999306398	0,9999332861
	MLR	0,9875003358	0,9875003358	0,9869602409
tsunami				
	MNN	0,9999223037	0,9999223037	0,9999306374
	MLR	0,9841269312	0,9841269312	0,9875378556

4.2.2 Training with fewer features

The Multiclass Logistic Regression algorithm stores the weights it assigns to features for different classes, as mentioned in subsection 2.4.3. These weights indicate the importance of the features for different classes and thus signifies what features might be irrelevant and can be left out. I.e. when features score a weight of 0 among all classes they are not contributing to the model. Because of this useful property some additional experiments were done using the MLR algorithm with only a selected set of features.

Using the feature weights, the number of features can be limited to only the ones used in the model. Table 4.7 shows the output for the MLR algorithm. From the in total 71 features present in the dataset, only 21 are used by the classifier. Other unused features have a value of 0 over all the classes and are therefore omitted in the table. Using these results we can re-train and evaluate the MLR classifier with a thousand captures per attack and only these 21 features. Again, reducing the features saves both storage and training time for the model, as fewer features have to be calculated, stored and evaluated. After reducing the dataset to only these features, evaluation again results in an accuracy of 100% (see Figure 4.12 for the final confusion matrix). Table 4.8 shows the average probabilities of this newly trained model compared to the model using all the features. Reducing the number of features shows no significant differences in average probabilities over all the classes. Therefore we can conclude that in this case, with only four attack tools, calculating and storing 21 features to train the classifier on suffices.

4.2.3 Principal Component Analysis (PCA)

To visualise the data in a scatter plot Principal Component Analysis (PCA) was performed on the top 10 set of weighted features provided by the Multiclass Logistic Regression (MLR) algorithm

Table 4.7: Feature weights from MLR. Features with a weight of 0 are omitted, but can be found in the appendix C

Feature	**flooder**	**ethan**	**saddam**	**tsunami**
Bias	0.622728	2.57913	-1.90491	-1.29728
dns.qry.class_unique	-0.79392	0	1.90643	0
dns.id_unique_len	-0.761273	0	1.87811	0
dns.qry.type_unique	0.117726	-1.15283E-7	0	-1.79597
ip.dsfield.dscp_unique	-0.122946	0	0	1.79175
udp.srcport_unique_len	-0.117052	0	1.53162	0
ip.id_longest_cons	-1.4457	0	0.421945	0.0336367
udp.checksum_used	0	1.07789	0	-0.249253
dns.count.add_rr_max	0	-1.07032	0	0.241751
dns.count.add_rr_min	0	-1.07032	0	0.241751
dns.count.add_rr_unique	0	-1.07032	0	0.241751
udp.checksum.status_unique	0	-1.07032	0	0.241751
dns.rr.udp_payload_size_max	0.969494	-0.250661	0	0
dns.rr.udp_payload_size_min	0.969494	-0.250661	0	0
dns.rr.udp_payload_size_unique	0.969494	-0.250661	0	0
dns.id_shortest_repeat	-0.504329	0.105602	-0.140917	0.541862
udp.srcport_shortest_repeat	-0.504329	0.105602	-0.140917	0.541862
dns.id_longest_repeat	-0.490286	0.103192	-0.148379	0.537796
udp.srcport_longest_repeat	-0.48792	0.102797	-0.14966	0.537126
udp.srcport_max	0	-0.16494	0	0
udp.srcport_min	0	0	0	0.132509
dns.id_max	0	-0.0262525	0	0

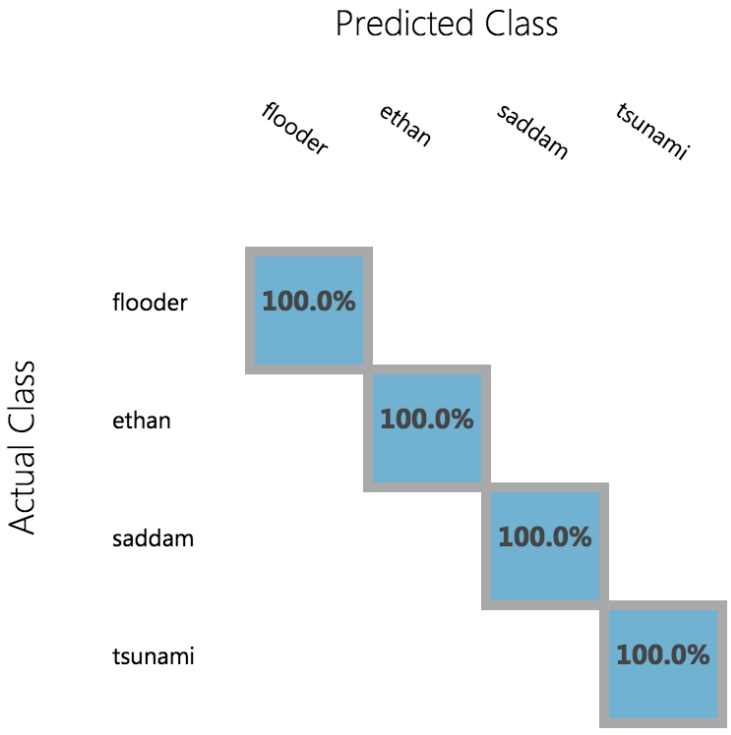


Figure 4.12: Confusion matrix from the evaluation results of the MLR classifier on the dataset of 1.000 captures per attack, trained only with the 21 selected features that were weighted by the same MLR model.

Table 4.8: Average scored probabilities for classes using MLR before and after applying feature selection on the dataset of 1.000 captures per attack.

Attack label	**1.000 captures (all features)**	**1.000 captures (selected features)**
ethan	0,9906537163	0,9906696107
flooder	0,9852958462	0,9855052562
saddam	0,9869602409	0,9867325054
tsunami	0,9875378556	0,9874639342

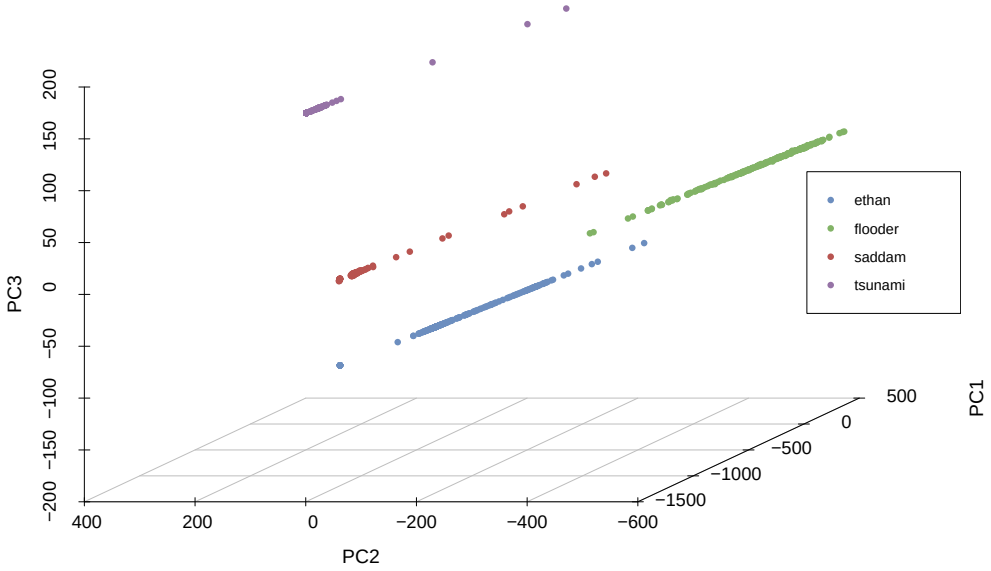
(see Table 4.7). This reduces the number of dimensions from 10 to 3, and can thus be used in a 3D scatter plot. The results are shown in Figure 4.13. In this figure clearly separated clusters of categorised points are visible, which are separable by the human eye. The plot is shown from two angles to correctly display the dimensions. This visualisation indicates that the data is easily distinguishable, confirming the earlier high accuracy numbers of both the MNN and MLR algorithms.

4.2.4 Decision tree

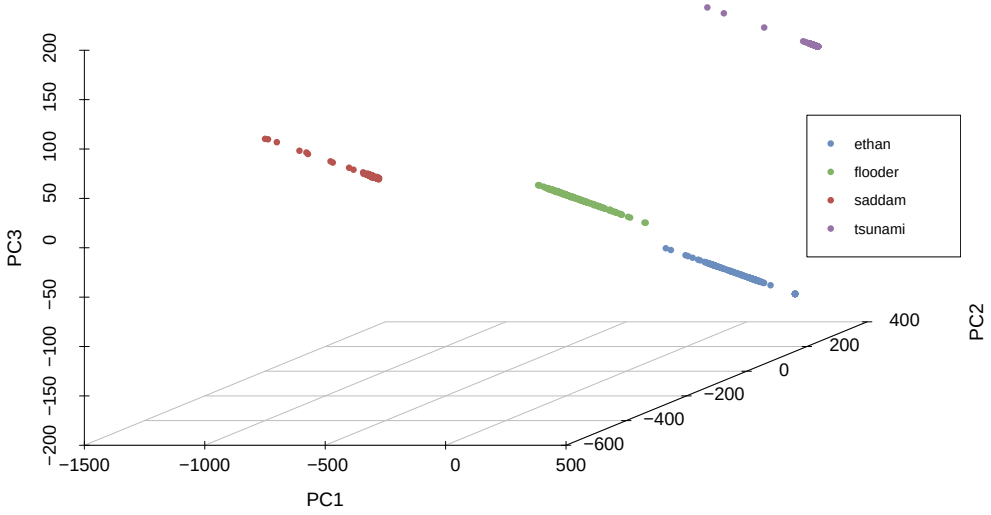
Because of the high accuracy obtained by using the MNN and MLR algorithms we experimented with another algorithm called the Multiclass Decision Forest (MDF) algorithm (earlier described in subsection 2.4.4). After setting the number of trees to be generated to one, 100% accuracy was still obtained with the dataset of 1.000 captures per attack and with the selected features from MLR in place. This decision tree is shown in Figure 4.14a.

In all of the previous results features used are pattern sensitive. That is, in our attacks we only use one attacker. However, if multiple attackers use the reflector at the same time for the same target, some of these features such as the longest repeat and the longest consecutive increasing subset are influenced by a second or third simultaneous attacker. Therefore we also attempted to construct a decision tree without these fields, which is shown in Figure 4.14b. This tree also has an accuracy of 100%.

Because there are only three decision nodes and four leafs, it is trivial to write a small script that performs the classification of the RDDoS data. This script is listed in Code 4.1.

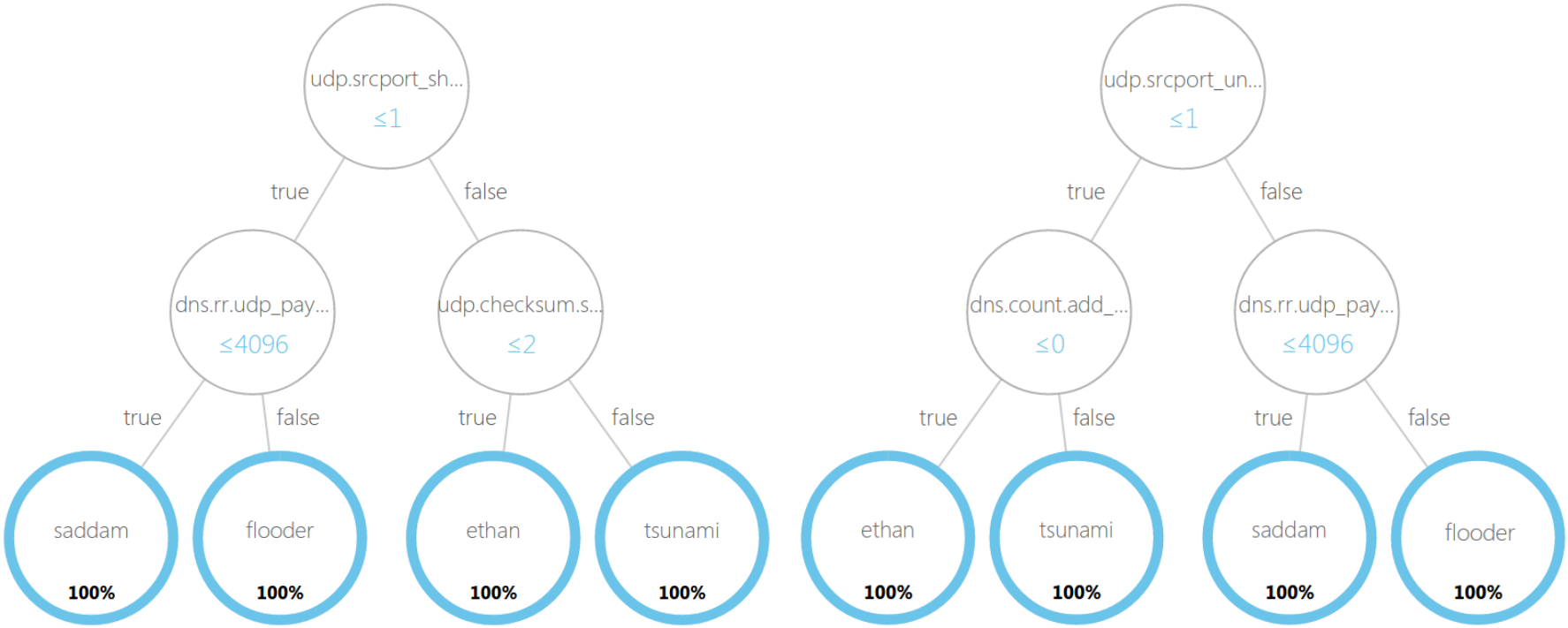


(a)



(b)

Figure 4.13: 3D scatter plot of the data points categorised by attack tool after performing Principal Component Analysis (PCA) on the top 10 features weighted by the Multiclass Logistic Regression (MLR) algorithm. A and B show different angles of the same plot.



(a) From left to right, top to bottom: udp.srcport_shortest_repeat, dns.rr.udp_payload_size_min and udp.checksum.status_unique_len.

(b) From left to right, top to bottom: udp.srcport_unique_len, dns.count.add_rr_min and dns.rr.udp_payload_size_min.

Figure 4.14: Two decision trees generated using the Multiclass Decision Forest (MDF) algorithm. Both trees are generated on the dataset containing 1.000 captured packets per attack, with the 21 selected features discussed in subsection 4.2.2. However, the tree on the right has also eliminated all features that infer patterns.

Code 4.1: Script written in Python3 that uses the knowledge obtained from the Multiclass Decision Forest (MDF) algorithm to classify the data from an CSV file. The `classify_tree` function consists out of only three if statements, which makes classification fast. This script loops trough all the lines in the CSV feature file, calling the `classify_tree` function and printing whether the classifier is correct or not for each line.

```
import os, csv

def classify_tree(o):
    a = int(o['dns.count.add_rr_min']) <= 0
    b = int(o['dns.rr.udp_payload_size_min']) <= 4096

    if int(o['udp.srcport_unique_len']) <= 1:
        return 'ethan' if a else 'tsunami'
    return 'saddam' if b else 'flooder'

all_files = filter(lambda x: x.endswith('csv_feature'), os.listdir('.'))
for filename in all_files:
    data = list(csv.DictReader(open(filename, 'r')))[0]
    print(data['label'] == classify_tree(data))
```

CHAPTER 5
Discussion

Do reflective DDoS attacks leave distinctive traces

We have observed that reflective attacks can leave distinctive traces. From the 4 different tools we have tested in the lab environment, we can clearly differentiate these attacks from each other. However, we must address the fact that identifying these attacks in lab environments is a different problem than identifying attacks in the wild on honeypots. We expect it is quite likely that different tools will have minute differences. However, it is also not impossible that two tools will look exactly the same.

In dataset 1 provided by Fox-IT we expect to have identified 2 different tools. However, we have concluded that a larger amount of packets than 25 are required to perform data analysis. In Fox-IT dataset 2 we are confident that we have identified at least 4 different tools. We have found evidence for more tools but are uncertain if these indicate actual attacks. An overview of the tools can be found in Table 5.1. The degree of certainty is a guide to display how much evidence for the fact that these are indeed different attack tools is found. For fairly certain and uncertain we expect it to be plausible that more than one tool falls in this dataset. Also, note that some of the PCAPs remain unidentified in these classifications. These PCAPs could be outliers of the identified tools or completely different attack tools.

Table 5.1: Overview of tools identified in the Fox-IT dataset 2

Tool	**Features**	**Certainty**
A	$\approx$ 2 unique DNS IDs / 250 packets, DS Field set to 0x40	Certain
B	Static DNS ID, UDP source port and IP ID per attack	Certain
C	DNS ID, UDP source port, and IP ID always change together, and the DNS ID is never used for different domains.	Certain
D	$\approx$ 1 unique DNS ID / 250 packets with no DS Field set to 0x00 and no malformed packets	Certain
E	$\approx$ 10-13 unique DNS IDs / 250 packets and DS Field set to 0x00	Fairly certain
F	2 unique DNS IDs and DS Field set to 0x00	Fairly certain
G	PCAPs contain malformed packets	Uncertain

We can provide a good estimate that PCAPs which show consistent differences are from different tools. However, it is impossible to disprove that two similar PCAPs are not from two different tools when the PCAPs are not labelled. This is essentially the problem with identifying these RDDoS attacks from an unlabeled dataset. We believe it is likely that each tool has different features and can, therefore, be identified. However, we also believe it is possible that two independent attackers create the exact same tool. Two particularly lazy programmers could, for example, create two different tools which both have random DNS ID, UDP source port, and IP ID generated once per attack. These generated features can then be used statically per attack. Considering the fact that most DNS packets appear to be crafted the same across tools in the Fox-IT dataset, the DNS packets are mostly the same, and that these frequently changing fields are now also used the same way it is impossible to discriminate between two different tools.

Identifying RDDoS attacks relies on performing in-depth pattern recognition and other advanced distinction methods. Therefore multiple packets are required to identify attacks. Furthermore, we expect that even pattern recognition can be a flawed detection method for some tools. This limitation is based on the very fact that an RDDoS attack is distributed. If a botnet of 50 bots sends requests to every reflector based on some pattern, the pattern might be different with 100 bots. Longest consecutive DNS IDs, count of unique DNS ID, and other factors can then start to change from attack to attack. This would disturb the pattern, and thus we conclude that behavioural analysis can change when the botnet changes. The severity of this limitation is completely dependent on the tool used, however. We have observed some tools that select a single reflector per thread and tools that simply loop over every reflector in every thread.

Additionally, it is possible for an attacker to use different tools on different bots. Once this happens one attack will have several patterns from several attack tools that intermingle. This could create entirely new patterns which could lead researchers to believe that new attacks are in fact a combination of existing tools. However, we expect that this limitation is less severe than the different patterns caused by changing botnet structures. We suspect that the tools an attacker uses are less frequent to change than a botnet, which changes frequently as bots get cleaned up or new bots get infected. Furthermore, a certain collection of tools can still create a distinct pattern. Thus, a collection of tools can be grouped into a single tool and still be identified per attack. Exactly how much these limitations hinder recognition is unclear.

It should be mentioned that the similarity of tools in the Fox-IT dataset could be caused by a loss of servers. Fox-IT has notified us that it lost two of its most used honeypots in recent months. The new honeypots which of course have a different IP address are still being discovered and put into reflector lists of attackers. If only a select group of attackers are using the honeypots it could explain the limited attack tools found. We are uncertain to which degree this factors into the results.

Can a fingerprint be build using distinctive traces left by reflective DDoS attacks

We expect that fingerprinting RDDoS attacks is impossible. The concept of a fingerprint, in this context, is some combination of unique identifiers that can be used to pinpoint to a single RDDoS tool used in an attack. However, these tools do have to follow a specific protocol, i.e. DNS, and are therefore quite alike in their behaviour. We believe that tools can be identical in their network behaviour, yet still be completely different programs. Therefore, fingerprinting is impossible considering the fact that two different tools could yield the exact same fingerprint.

Is it possible to identify the tool / script / service used in a reflective DDoS attack

We have been successful in identifying four distinctive DNS RDDoS tools based on network captures with the help of a machine learning classifier. However, this does require a labelled dataset containing all the features used beforehand, as a tool cannot be classified as such if the classifier is unaware about its behaviour. Additionally, just as for the previous question, two tools can be 100% identical in their networking behaviour and still be other tools. In those cases not a single machine learning algorithm would be able to keep those tools apart using the current methodology.

Can machine learning be utilised to automate the identification process

We have shown that clustering can achieve a certain degree of success when applied to real data. However, clustering with an unknown number of expected clusters is hard. Using DBSCAN allows us to perform clustering, but it is uncertain what the correct ϵ is to create correct clusters. Furthermore, when verified on our lab data the clustering appears to be working correctly as shown in Figure 4.11a.

We expect that using clustering together with static analysis could be effective in research. Gathering a general idea in what ways the dataset differs and then clustering based on those features, comparing it with the unique features and then selecting the right input for DBSCAN could be valuable. We have shown that DBSCAN works with our lab generated data, and it appears that new clusters indeed show up when new tools are added to a dataset.

As for supervised learning we can say with confidence that it is possible to correlate network captures to the tools being used for the attack. However, there are some side notes here. Just as for the previous question the same limitations hold here. When two different tools exhibit exactly the same network behaviour it becomes hard, if not impossible, for a classifier to distinguish between them. Additionally all tools have to be known beforehand by the classifier, as the classifier is not able to classify a tool it was not trained on before.

However, despite these limitations, we believe that the supervised learning algorithms have shown that machine learning can successfully automate the detection process to some degree. This result confirms our suspicion that different RDDoS tools leave distinctive traces in most cases. Furthermore, the classification of these tools could be condensed to only look at features which do not require pattern recognition and thus bypass the limitations specified in this chapter. We expect that the trained algorithms will remain accurate when adding more tools to the list of tools. We therefore believe that the results of the supervised learning algorithms substantiate our claim that DBSCAN can be used to cluster data.

CHAPTER 6
Conclusion

In this research, we have shown that RDDoS attacks using DNS reflectors can leave distinct traces and can be identified based on certain behavioural patterns. Using data created in a lab we have successfully identified 4 different attack tools using machine learning. However, when analysing data gathered from actual honeypots in the real world it appears that identification is a lot harder.

We expect that though it is probable that RDDoS attacks leave distinct traces, it is far from impossible that two tools leave identical traces. Thus we believe that creating fingerprints of these RDDoS attacks is not possible. It is possible to see the difference between two different attacks, and it is possible to prove that two different tools leave two different traces. However, when using unlabelled data it is impossible to prove that two apparently identical attacks, and thus assumed the same tool, are in fact the same tool. Therefore, we conclude that though identification of different tools is likely possible, it is not impossible to incorrectly identify two extremely similar tools as the same tool.

We have shown that identification likely requires a large set of captures packets. It is improbable that individual PCAPs can be identified as different attacks. This identification method relies on pattern recognition and metadata about the attacks like the unique DNS IDs found in an attack, the minimum and maximum value of a UDP port and other data that can be yielded from the attacks. We acknowledge that some of this data can be different based on outside factors like the size and shape of an attacking botnet. Though we expect that this could affect the recognition, we believe that pattern recognition and looking at metadata, in combination with features gained from the IP header, UDP header and DNS payload is the best method for identification.

We expect to have found at least 4 and possibly 7 different tools as shown in Table 5.1. The identification process is based on manual inspection of approximately 1800 different attacks. When performing machine learning on this data it is possible to plot the data based on unique features and selected features. We have observed that the best results of plotting are achieved when the data is plotted by using selected features. These selected features are based on features that are identified as important features in the manual inspection.

As for the lab data we have shown that it is possible within a small test environment to successful identify RDDoS tools with a high accuracy using a trained machine learning multiclass classifier. The process of gathering the network captures can be fully automated. Newly emerged scripts can be added to the automation process where the attacks are generated, data gathered and added to the training set of the model. The results of these classifiers may prove useful in network monitoring tools, forensic investigations and statistical analysis of network traffic.

Furthermore, the results gathered with supervised learning shows that RDDoS attacks can indeed, and often do, leave distinctive traces. This result further builds on our hypothesis that it is indeed likely that these distinctive traces are left by attack tools. For the four tools we used all algorithms have shown to be 100% accurate in classifying the correct tool. Because of this we argue that the Multiclass Decision Forest algorithm is the best choice in this case as it can be expressed using only three if statements.

Considering the results of the static analysis, unsupervised machine learning, and supervised machine learning we are confident that enough prove has been gathered to show that RDDoS attack tools which use DNS leave distinctive traces that can be used to identify these tools.

CHAPTER 7
Future work

Instead of solely focusing on DNS the feasibility of the same approach could be tested on other protocols such as NTP, SNMP, SSDP, and other protocols. These protocols do allow for the same type of RDDoS attacks as DNS and could show similar distinctive behaviour. Apart from that this research only trained the classifier on four tools that we were able to find in the limited amount of time we had. Adding more tools may influence the accuracy results of the classifier in a negative way. Other factors that might influence the accuracy are multiple attacks being executed at once, possibly from multiple tools or attackers simultaneously. Yet another interesting approach may be to include other features such as captures from the victim's side. This data, combined with the data from the reflector's side, may lead to other interesting patterns.

Finally a different experiment could be done where the trained classifier is fed with the unlabelled data received from Fox-IT. In this case it is impossible to evaluate the model as the estimated label can not be verified against a known label. However, there could be some interesting results in the probabilities assigned by the classifier to each of the trained classes giving some indication of the input data to those trained classes.

Abbreviations

Additional RRs Additional Resource Records.

ANN Artificial Neural Network.

Authority RRs Authority Resource Records.

C&C Command and Control.

CSV Comma-separated values.

DBSCAN Density-Based Spatial Clustering of Applications with Noise.

DDoS Distributed Denial of Service.

DF Don't Fragment.

DNS Domain Name System.

DoS Denial of Service.

DSCP Differentiated Services Code Point.

ICMP Internet Control Message Protocol.

MDF Multiclass Decision Forest.

MF More Fragments.

MGKM Modified Global K-means.

MLR Multiclass Logistic Regression.

MNN Multiclass Neural Network.

PCA Principal Component Analysis.

PCAP Packet Capture.

RB Reserved Bit.

RDDoS Reflective Distributed Denial of Service.

RLL Response Rate Limiting.

SaaS Software as a Service.

UDP User Datagram Protocol.

APPENDIX A

Scripts

Code A.1: Bash script that automates the execution of DNS RDDoS attacks by controlling the execution of the attack and `tcpdump` processes on the remote hosts (attacker and reflector). First the attack is initiated on the attacker host, after which the process ID is stored as a variable. Once the attack is started, `tcpdump` is launched on the reflector server. Because `tcpdump` stops after one million captures the script will wait for it to end, and once this happens the script will kill the earlier stored process ID of the attack script at the attacker host, completing one successfull round of attack.

```
attacker_host="attacker_root"; reflector_host="reflector_root"
begin=0; end=249; current=$begin

function attack() {
    echo "> Starting attack $1"

    ssh -T $attacker_host <<EOF > tmp_pid 2> /dev/null
cd '/var/attack_scripts/dns_ddos_scripts/flooder'
nohup ./dns_amp_tool '10.0.0.4' '12345' 'reflection_file' '8' '1000000'
 ↪  > /dev/null 2>&1 < /dev/null &
echo \$!
EOF

    attack_pid=$(tail -n1 tmp_pid); rm tmp_pid
    echo "> Attack started with PID $attack_pid on $attacker_host"

    # Monitoring
    ssh -T $reflector_host "tcpdump -i eno2 udp port 53 and dst host
     ↪  10.0.0.2 and not src 10.0.0.2 -w /mnt/captures/flooder_$1.pcap
     ↪  -c 1000000" > /dev/null 2>&1 &
    monitor_pid=$!
    echo "> Monitoring started with PID $monitor_pid on $reflector_host"

    wait $monitor_pid; echo "> Monitoring finished"
    echo "> Stopping attack"
    ssh -T $attacker_host "kill -9 $attack_pid" > /dev/null 2>&1
}

while [ $current -le $end ]
do
    attack $current; ((current++)); echo ""
done
```

Code A.2: Bash script that loops through all the Packet Capture (PCAP) files in the working directory, converting all the files to a Comma-separated values (CSV) file using `tshark`. This script forks the `tshark` processes and makes sure only five processes run at simultaneously. Only fields that are of interest for the multiclass classifier are exported.

```
pids=()
running=0

trap ctrl_c SIGINT
function ctrl_c() {
    kill -9 $pids
    exit
}

for file in *.pcap
do
    newfile=`basename "$file" .pcap`.csv
    echo $newfile
    tshark -r $file -Tfields > $newfile -e ip.id -e ip.version -e
    ↪  ip.dsfield.dscp -e ip.dsfield.ecn -e ip.flags.rb -e ip.flags.df
    ↪  -e ip.flags.mf -e ip.proto -e udp.srcport -e udp.checksum -e
    ↪  udp.checksum.status -e dns.id -e dns.flags.response -e
    ↪  dns.flags.opcode -e dns.flags.truncated -e dns.flags.recdesired
    ↪  -e dns.flags.z -e dns.flags.checkdisable -e dns.count.queries -e
    ↪  dns.count.answers -e dns.count.auth_rr -e dns.count.add_rr -e
    ↪  dns.qry.type -e dns.qry.class -e dns.resp.name -e
    ↪  dns.rr.udp_payload_size -e dns.resp.type -e dns.resp.ext_rcode
    ↪  -e dns.resp.edns0_version -e dns.resp.z -e dns.resp.z.do -e
    ↪  dns.resp.z.reserved -e dns.resp.len &

    pids+=($!)
    sleep 1

    running=$(pgrep tshark -c)

    # Run 5 conversion scripts simultaneously
    while [ $running -ge 5 ]
    do
        echo "Converting..."
        sleep 5
        running=$(pgrep tshark -c)
    done
done

wait $pids
```

Code A.3: This Python script opens a Comma-separated values (CSV) file provided as a user parameter and from this generates all the features used for the classifier. Once ready, the resulting CSV feature file is written to the specified output path. The helper functions such as `unique_len` and `min_max` are listed in Code A.4 and Code A.5.

```
df = pd.DataFrame.from_records(iter(reader))
df = df.loc[df['dns.id'] != '']
df = df.apply(lambda x: x.str.strip()).replace('', -1)

data = {**{}, **min_max('ip.id', df), **repeat_stats('ip.id', df),
    ↪  **unique_len('ip.id', df), **unique_len('ip.dsfield.dscp', df,
    ↪  True), **unique_len('ip.flags.rb', df, True),
    ↪  **unique_len('ip.flags.df', df, True), **unique_len('ip.flags.mf',
    ↪  df, True), **unique_len('ip.proto', df, True),
    ↪  **min_max('udp.srcport', df), **repeat_stats('udp.srcport', df),
    ↪  **unique_len('udp.srcport', df), **unique_len('udp.checksum', df),
    ↪  **unique_len('udp.checksum.status', df, True), **min_max('dns.id',
    ↪  df), **repeat_stats('dns.id', df), **unique_len('dns.id', df),
    ↪  **unique_len('dns.flags.response', df, True),
    ↪  **unique_len('dns.flags.opcode', df, True),
    ↪  **unique_len('dns.flags.truncated', df, True),
    ↪  **unique_len('dns.flags.recdesired', df, True),
    ↪  **unique_len('dns.flags.z', df, True),
    ↪  **unique_len('dns.flags.checkdisable', df, True),
    ↪  **min_max('dns.count.queries', df),
    ↪  **unique_len('dns.count.queries', df, True),
    ↪  **min_max('dns.count.answers', df),
    ↪  **unique_len('dns.count.answers', df, True),
    ↪  **min_max('dns.count.auth_rr', df),
    ↪  **unique_len('dns.count.auth_rr', df, True),
    ↪  **min_max('dns.count.add_rr', df), **unique_len('dns.count.add_rr',
    ↪  df, True), **unique_len('dns.qry.type', df, True),
    ↪  **unique_len('dns.qry.class', df, True),
    ↪  **min_max('dns.rr.udp_payload_size', df),
    ↪  **repeat_stats('dns.rr.udp_payload_size', df),
    ↪  **unique_len('dns.rr.udp_payload_size', df, True) }

data['ip.id_longest_cons'] =
    ↪  len(longest_consecutive_increasing(list(df['ip.id'])))
data['udp.checksum_used'] = (len(df.loc[df['udp.checksum'] != 0]) > 0)
data['udp.srcport_longest_cons'] =
    ↪  len(longest_consecutive_increasing(list(df['udp.srcport'])))
data['dns.id_longest_cons'] =
    ↪  len(longest_consecutive_increasing(list(df['dns.id'])))
data['dns.rr.udp_payload_size_longest_cons'] =
    ↪  len(longest_consecutive_increasing(list(df['dns.rr.udp_payload_size'])))
data['label'] = args.l

with open(args.o, 'w') as output_file:
    writer = csv.DictWriter(output_file,fieldnames=sorted(data.keys()))
    writer.writeheader()
    writer.writerow(data)
```

Code A.4: Helper functions that are part of the Python3 code listed in A.3.

```python
def shortest_repeat(data):
    return str(min(sum(1 for i in g) for k,g in groupby(data)))

def longest_repeat(data):
    return str(max(sum(1 for i in g) for k,g in groupby(data)))

def min_max(name, data):
    return {
        name + '_min': str(data[name].min()),
        name + '_max': str(data[name].max())
    }

def repeat_stats(name, data):
    return {
        name + '_shortest_repeat': str(shortest_repeat(data[name])),
        name + '_longest_repeat': str(longest_repeat(data[name]))
    }

def unique_len(name, data, include=False):
    unique = sorted(list(data[name].unique()))
    unique = [str(i) for i in unique]

    data = {}
    data[name + '_unique_len'] = str(len(unique))

    if include:
        data[name + '_unique'] = str(','.join(unique))
    return data
```

Code A.5: Helper functions that is part of the Python3 code listed in A.3. This function calculates the longestt consecutive increasing subset from a list, such as `[3,4,5,6,7]` but not `[3,4,5,7,8]` or `[3,4,5,5,6]` [31].

```
def longest_consecutive_increasing(intvect):
    max_so_far = 0
    curr_count = 1
    max_pos = -1

    #iterate through all integers in the vector
    for counter1, integer in enumerate(intvect):
        #case where we just started. No comparison needed yet.
        if counter1 == 0:
            pass
        #case where we have an increase
        elif intvect[counter1] == intvect[counter1-1] + 1:
            curr_count += 1
        #case where we dont have an increase
        else:
            if curr_count > max_so_far:
                max_pos = counter1-curr_count
                max_so_far = curr_count
            curr_count = 1
    #final case after exiting loop
    if curr_count > max_so_far:
        max_pos = len(intvect)-curr_count
        max_so_far = curr_count
    return intvect[max_pos:max_pos+max_so_far]
```

APPENDIX B

DNS packets in Fox-IT dataset

Packet 1 - Common DNS packet

```
0000   00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00  ..............E.
0010   00 40 fc 78 00 00 f5 11 44 5a 6c 1c 38 a9 7f 00  .@.x....DZl.8...
0020   00 01 1c 60 00 35 00 2c 00 00 b3 23 01 00 00 01  ...`.5.,...#....
0030   00 00 00 00 00 01 04 6c 65 74 68 02 63 63 00 00  .......leth.cc..
0040   ff 00 01 00 00 29 23 28 00 00 00 00 00 00        .....)#(......
```

```
Internet Protocol Version 4, Src: 108.28.56.169, Dst: 127.0.0.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 64
    Identification: 0xfc78 (64632)
    Flags: 0x00
    Fragment offset: 0
    Time to live: 245
    Protocol: UDP (17)
    Header checksum: 0x445a
User Datagram Protocol, Src Port: 7264, Dst Port: 53
Domain Name System (query)
    Transaction ID: 0xb323
    Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 1
    Queries
        leth.cc: type ANY, class IN
            Name: leth.cc
            [Name Length: 7]
            [Label Count: 2]
            Type: * (A request for all records the
                server/cache has available) (255)
            Class: IN (0x0001)
    Additional records
        <Root>: type OPT
            Name: <Root>
            Type: OPT (41)
            UDP payload size: 9000
            Higher bits in extended RCODE: 0x00
            EDNS0 version: 0
            Z: 0x0000
```

Packet 2 - Malformed EDNS header

```
0000   00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00  ..............E.
0010   00 45 d7 32 00 00 f7 11 67 9b 6c 1c 38 a9 7f 00  .E.2....g.l.8...
0020   00 01 1b 48 00 35 00 31 00 00 7f 9d 01 00 00 01  ...H.5.1........
0030   00 00 00 00 00 01 04 6c 65 74 68 02 63 63 00 00  .......leth.cc..
0040   ff 00 01 62 65 29 23 28 00 01 ff 00 29 23 00 00  ...be)#(....)#..
0050   00 00 00                                         ...
```

```
Internet Protocol Version 4, Src: 108.28.56.169, Dst: 127.0.0.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 64
    Identification: 0xfc78 (64632)
    Flags: 0x00
    Fragment offset: 0
    Time to live: 245
    Protocol: UDP (17)
    Header checksum: 0x445a
User Datagram Protocol, Src Port: 7264, Dst Port: 53
Domain Name System (query)
    Transaction ID: 0xb323
    Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 1
    Queries
        leth.cc: type ANY, class IN
            Name: leth.cc
            [Name Length: 7]
            [Label Count: 2]
            Type: * (A request for all records the
                server/cache has available) (255)
            Class: IN (0x0001)
    Additional records
        <Root>: type OPT
            Name: <Root>
            Type: OPT (41)
            UDP payload size: 9000
            Higher bits in extended RCODE: 0x00
            EDNS0 version: 0
            Z: 0x0000
            Data length: 0
```

Packet 3 - Differentiated services set to 0x40 and capitalised

```
0000   00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 40  ..............E@
0010   00 43 f9 44 00 00 f6 11 d0 ed b2 fe f6 53 7f 00  .C.D.........S..
0020   00 01 e5 17 00 35 00 2f 00 00 ab 9e 01 00 00 01  .....5./........
0030   00 00 00 00 00 01 06 41 52 43 54 49 43 03 47 4f  .......ARCTIC.GO
0040   56 00 00 ff 00 01 00 00 29 23 28 00 00 00 00 00  V.......)#(.....
0050   00                                               .
```

```
Internet Protocol Version 4, Src: 178.254.246.83, Dst: 127.0.0.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x40 (DSCP: CS2, ECN: Not-ECT)
    Total Length: 67
    Identification: 0xf944 (63812)
    Flags: 0x00
    Fragment offset: 0
    Time to live: 246
    Protocol: UDP (17)
    Header checksum: 0xd0ed
User Datagram Protocol, Src Port: 58647, Dst Port: 53
Domain Name System (query)
    Transaction ID: 0xab9e
    Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 1
    Queries
        ARCTIC.GOV: type ANY, class IN
            Name: ARCTIC.GOV
            [Name Length: 10]
            [Label Count: 2]
            Type: * (A request for all records the
                server/cache has available) (255)
            Class: IN (0x0001)
    Additional records
        <Root>: type OPT
            Name: <Root>
            Type: OPT (41)
            UDP payload size: 9000
            Higher bits in extended RCODE: 0x00
            EDNS0 version: 0
            Z: 0x0000
            Data length: 0
```

Packet 4 - Differentiated services set to 0x40 and not capitalised

```
0000   00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 40  ..............E@
0010   00 47 30 b2 00 00 f4 11 ca 8a ae 6d cb d6 7f 00  .G0........m....
0020   00 01 4c 69 00 35 00 33 00 00 d8 08 01 00 00 01  ..Li.5.3........
0030   00 00 00 00 00 01 0b 68 6f 66 66 6d 65 69 73 74  .......hoffmeist
0040   65 72 02 62 65 00 00 ff 00 01 00 00 29 23 28 00  er.be.......)#(.
0050   00 00 00 00 00                                   .....
↪  .
```

```
Internet Protocol Version 4, Src: 174.109.203.214, Dst: 127.0.0.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x40 (DSCP: CS2, ECN: Not-ECT)
    Total Length: 71
    Identification: 0x30b2 (12466)
    Flags: 0x00
    Fragment offset: 0
    Time to live: 244
    Protocol: UDP (17)
    Header checksum: 0xca8a
User Datagram Protocol, Src Port: 19561, Dst Port: 53
Domain Name System (query)
    Transaction ID: 0xd808
    Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 1
    Queries
        hoffmeister.be: type ANY, class IN
            Name: hoffmeister.be
            [Name Length: 14]
            [Label Count: 2]
            Type: * (A request for all records the
                server/cache has available) (255)
            Class: IN (0x0001)
    Additional records
        <Root>: type OPT
            Name: <Root>
            Type: OPT (41)
            UDP payload size: 9000
            Higher bits in extended RCODE: 0x00
            EDNS0 version: 0
            Z: 0x0000
            Data length: 0
```

APPENDIX C

Features generated from DNS packets

Once the DNS packets are captured and the relevant fields are exported to a CSV file, features are inferred from this CSV file. The features and a short description are shown in Table C.1. The first part of the name is the name as it was given by tshark, whereas the part after the underscore indicates the type of feature that was generated. These indicators are as follows:

_unique
Samples a comma separated list of the unique values in a collection.

_unique_len
Calculates the length of all the unique values in a collection.

_shortest_repeat
Determines the shortest repeating subset of a collection.

_longest_repeat
Determines the longest repeating subset of a collection.

_min
Determines the minimum numerical value in a collection.

_max
Determines the maximum numerical value in a collection.

_used
Indicates whether a field is used or set to `NULL`.

_longest_cons
Calculates the longest consecutive increasing subset of a collection.

Name	Description
`ip.dsfield.dscp_unique`	Comma separated list of the collection of unique values of the Differentiated Services Code Point (DSCP) field.
`ip.dsfield.dscp_unique_len`	The number of items in the list mentioned above.
`ip.flags.df_unique`	Comma separated list of the collection of unique values of the Don't Fragment (DF) flag.
`ip.flags.df_unique_len`	The number of items in the list mentioned above.
`ip.flags.mf_unique`	Comma separated list of the collection of unique values of the More Fragments (MF) flag
`ip.flags.mf_unique_len`	The number of items in the list mentioned above.
`ip.flags.rb_unique`	Comma separated list of the collection of unique values of the Reserved Bit (RB) field.
`ip.flags.rb_unique_len`	The number of items in the list mentioned above.
`ip.id_longest_cons`	The longest consecutive increasing subset of the IP identifier field.
`ip.id_longest_repeat`	The longest repeating subset of the IP identifier field.
`ip.id_max`	The maximum value of the IP identifier field.
`ip.id_min`	The minimum value of the IP identifier field.
`ip.id_shortest_repeat`	The shortest repeating subset of the IP identifier field.
`ip.id_unique_len`	The number of items in the list mentioned above.
`ip.proto_unique`	Comma separated list of the collection of unique values of the IP proto field.
`ip.proto_unique_len`	The number of items in the list mentioned above.

Name	Description
udp.checksum.status_unique	The UDP checksum field is verified by the PCAP parser, which results in a status field. This status field is collected as a Comma separated list of the collection of unique values of this field.
udp.checksum.status_unique_len	The number of items in the list mentioned above.
udp.checksum_unique_len	The number of items in the list mentioned above.
udp.checksum_used	This fields indicates whether the (optional) UDP checksum field is used.
udp.srcport_longest_cons	The longest consecutive increasing subset of the UDP source port.
udp.srcport_longest_repeat	The longest repeating subset of the UDP field.
udp.srcport_max	The maximum value of the UDP source port field.
udp.srcport_min	The minimum value of the UDP source port field.
udp.srcport_shortest_repeat	The shortest repeating subset of the UDP source port field.
udp.srcport_unique_len	The number of items in the list mentioned above.
dns.count.add_rr_max	The maximum value of the DNS Additional Resource Records (Additional RRs) count field.
dns.count.add_rr_min	The minimum value of the DNS Additional Resource Records (Additional RRs) count field.
dns.count.add_rr_unique	Comma separated list of the collection of unique values of the DNS Additional Resource Records (Additional RRs) count field.
dns.count.add_rr_unique_len	The number of items in the list mentioned above.
dns.count.answers_max	The maximum value of the DNS answers count field.
dns.count.answers_min	The minimum value of the DNS answers count field.
dns.count.answers_unique	Comma separated list of the collection of unique values of the DNS answers count field.
dns.count.answers_unique_len	The number of items in the list mentioned above.
dns.count.auth_rr_max	The maximum value of the DNS Authority RRs field.
dns.count.auth_rr_min	The minimum value of the DNS Authority RRs field.
dns.count.auth_rr_unique	Comma separated list of the collection of unique values of the DNS Authority RRs field.
dns.count.auth_rr_unique_len	The number of items in the list mentioned above.
dns.count.queries_max	The maximum value of the DNS query count field.
dns.count.queries_min	The minimum value of the DNS query count field.
dns.count.queries_unique	Comma separated list of the collection of unique values of the DNS query count field.
dns.count.queries_unique_len	The number of items in the list mentioned above.
dns.flags.checkdisable_unique	Comma separated list of the collection of unique values of the DNS “Non-authenticated data” flag field.
dns.flags.checkdisable_unique_len	The number of items in the list mentioned above.
dns.flags.opcode_unique	Comma separated list of the collection of unique values of the DNS opcode flag field.
dns.flags.opcode_unique_len	The number of items in the list mentioned above.
dns.flags.recdesired_unique	Comma separated list of the collection of unique values of the DNS “Recursion desired” flag field.
dns.flags.recdesired_unique_len	The number of items in the list mentioned above.
dns.flags.response_unique	Comma separated list of the collection of unique values of the DNS response flag field.
dns.flags.response_unique_len	The number of items in the list mentioned above.
dns.flags.truncated_unique	Comma separated list of the collection of unique values of the DNS truncated flag field.
dns.flags.truncated_unique_len	The number of items in the list mentioned above.

Name	Description
dns.flags.z_unique	Comma separated list of the collection of unique values of the DNS "Z" flag field
dns.flags.z_unique_len	The number of items in the list mentioned above.
dns.id_longest_cons	The longest consecutive increasing subset of the DNS identifier field.
dns.id_longest_repeat	The longest repeating subset of the DNS identifier field.
dns.id_max	The maximum value of the DNS identifier field.
dns.id_min	The minimum value of the DNS identifier field.
dns.id_shortest_repeat	The shortest repeating subset of the DNS identifier field.
dns.id_unique_len	The number of items in the list mentioned above.
dns.qry.class_unique	Comma separated list of the collection of unique values of the Domain Name System (DNS) query class field.
dns.qry.class_unique_len	The number of items in the list mentioned above.
dns.qry.type_unique	Comma separated list of the collection of unique values of the Domain Name System (DNS) query type field.
dns.qry.type_unique_len	The number of items in the list mentioned above.
dns.rr.udp_payload_size_longest_cons	The longest consecutive increasing subset of the UDP payload size field.
dns.rr.udp_payload_size_longest_repeat	description
dns.rr.udp_payload_size_max	The maximum value of the UDP payload size field.
dns.rr.udp_payload_size_min	The minimum value of the UDP payload size field.
dns.rr.udp_payload_size_shortest_repeat	The shortest repeating subset of the UDP payload field.
dns.rr.udp_payload_size_unique	Comma separated list of the collection of unique values of the User Datagram Protocol (UDP) payload size field.
dns.rr.udp_payload_size_unique_len	The number of items in the list mentioned above.

Table C.1: List of 71 features generated from DNS packet captures. The first part of the name is the name of the field that was given by tshark, whereas the part after the underscore indicates what type of data was extracted from the field.

Bibliography

[1] *History of DDoS Attacks*. https://security.radware.com/ddos-knowledge-center/ddos-chronicles/ddos-attacks-history/. Accessed 08-06-2017. Mar. 2017.

[2] Brian Krebs. *Israeli Online Attack Service vDOS Earned $600,000 in Two Years*. https://krebsonsecurity.com/2016/09/israeli-online-attack-service-vdos-earned-600000-in-two-years/. Accessed 08-06-2017. Sept. 2014.

[3] Kanishk. *What Businesses In India Can Learn From Recent DDoS Attacks*. https://blogs.haltdos.com/2017/02/22/businesses-india-can-learn-recent-ddos-attacks/. Feb. 2017.

[4] Stephen M Specht and Ruby B Lee. "Distributed Denial of Service: Taxonomies of Attacks, Tools, and Countermeasures". In: *ISCA PDCS*. 2004, pp. 543–550.

[5] Laura Feinstein et al. "Statistical approaches to DDoS attack detection and response". In: *DARPA Information Survivability Conference and Exposition, 2003. Proceedings*. Vol. 1. IEEE. 2003, pp. 303–314.

[6] Keunsoo Lee et al. "DDoS attack detection method using cluster analysis". In: *Expert Systems with Applications* 34.3 (2008), pp. 1659–1665.

[7] Lifang Zi, John Yearwood, and Xin-Wen Wu. "Adaptive clustering with feature ranking for DDoS attacks detection". In: *Network and System Security (NSS), 2010 4th International Conference on*. IEEE. 2010, pp. 281–286.

[8] Wei Wei et al. "A rank correlation based detection against distributed reflection DoS attacks". In: *IEEE Communications Letters* 17.1 (2013), pp. 173–175.

[9] Thijs Rozekrans, Matthijs Mekking, and Javy de Koning. *Defending against DNS reflection amplification attacks*. http://rp.delaat.net/2012-2013/p29/report.pdf. Feb. 2013.

[10] Symantec Security Response. *Grappling with the ZeroAccess Botnet*. https://www.symantec.com/connect/blogs/grappling-zeroaccess-botnet. Accessed 07-06-2017. July 2013.

[11] Matthew Prince. *Deep Inside a DNS Amplification DDoS Attack*. https://blog.cloudflare.com/deep-inside-a-dns-amplification-ddos-attack/. Accessed 07-06-2017. Aug. 2012.

[12] Brian Krebs. *Stress-Testing the Booter Services, Financially*. http://krebsonsecurity.com/2015/08/stress-testing-the-booter-services-financially/. Accessed 07-06-2017. Aug. 2017.

[13] Randal Vaughn and Gadi Evron. "DNS Amplification Attacks". In: (Mar. 2006).

[14] *Alert (TA14-017A) UDP-Based Amplification Attacks*. https://www.us-cert.gov/ncas/alerts/TA14-017A. Accessed 07-06-2017. Nov. 2016.

[15] Roland van Rijswijk-Deij, Anna Sperotto, and Aiko Pras. "DNSSEC and its potential for DDoS attacks: a comprehensive measurement study". In: *Proceedings of the 2014 Conference on Internet Measurement Conference*. ACM. 2014, pp. 449–460.

[16] Florian Adamsky et al. "P2P File-Sharing in Hell: Exploiting BitTorrent Vulnerabilities to Launch Distributed Reflective DoS Attacks." In: 2015.

[17] The Shadowserver Foundation. *Open Resolver Scanning Project*. https://dnsscan.shadowserver.org/. Accessed 07-06-2017. June 2017.

[18] Marek Majkowski. *Reflections on reflection (attacks)*. https://blog.cloudflare.com/reflections-on-reflections/. May 2017.

[19] David Cornell. *DNS Amplification Attacks*. https://umbrella.cisco.com/blog/blog/2014/03/17/dns-amplification-attacks/. Accessed 07-06-2017. Mar. 2014.

[20] Alex S. Holehouse. *Stanford Machine Learning - 06: Logistic Regression*. http://www.holehouse.org/mlclass/06_Logistic_Regression.html.

[21] Gagarine Yaikhom. *Implementing the DBSCAN clustering algorithm*. http://yaikhom.com/2015/09/04/implementing-the-dbscan-clustering-algorithm.html. Accessed 28-6-2017. 2015.

[22] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[23] Microsoft Azure. *Multiclass Neural Network*. https://msdn.microsoft.com/en-us/library/azure/dn906030.aspx. Mar. 2016.

[24] J Engel. "Polytomous logistic regression". In: *Statistica Neerlandica* 42.4 (1988), pp. 233–252.

[25] Microsoft Azure. *Multiclass Logistic Regression*. https://msdn.microsoft.com/en-us/library/azure/dn905853.aspx. June 2016.

[26] Microsoft Azure. *Multiclass Decision Forest*. https://msdn.microsoft.com/en-us/library/azure/dn906015.aspx. June 2016.

[27] Azure Machine Learning Team. *Microsoft Azure Machine Learning: Algorithm Cheat Sheet*. http://download.microsoft.com/download/A/6/1/A613E11E-8F9C-424A-B99D-65344785C288/microsoft-machine-learning-algorithm-cheat-sheet-v6.pdf. Mar. 2017.

[28] MathWorks. *Decision Tree and Decision Forest*. https://nl.mathworks.com/matlabcentral/fileexchange/39110-decision-tree-and-decision-forest. May 2014.

[29] Kathleen Nichols et al. *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*. RFC 2474. http://www.rfc-editor.org/rfc/rfc2474.txt. RFC Editor, Dec. 1998. URL: http://www.rfc-editor.org/rfc/rfc2474.txt.

[30] *DSCP and Precedence Values*. http://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus1000/sw/4_0/qos/configuration/guide/nexus1000v_qos/qos_6dscp_val.pdf. Accessed 21-06-2017. Mar. 2016.

[31] *sub sharela sequence of increasing integers*. http://pythonfiddle.com/sub-sequence-of-increasing-integers/. Accessed 20-06-2017.