Lightweight IPv6 network probing detection framework

Geana, A.

Award date:
2015

Lightweight IPv6 network probing detection framework

Alexandru Geana

Thesis submitted in partial fulfilment
of the requirements for the degree of
Master of Science
in
Information Security Technology
at the
Eindhoven University of Technology

Supervisors:
dr. Jerry den Hartog
ir. Mathias Morbitzer
dr. Mykola Pechenizkiy

Eindhoven, August 2015

Acknowledgements

This thesis report is the result of my graduation project, which concludes the master program in Information Security Technology, a special track of Computer Science and Engineering at the Technical University of Eindhoven. The project was carried out in collaboration with Fox-IT, Delft.

I would like to express my sincere gratitude to my supervisor, dr. Jerry den Hartog and my tutor, Mathias Morbitzer who guided and encouraged me throughout my graduation project. Their comments, suggestions and feedback were invaluable to the completion of this work. Whenever I had questions or requests, whenever I felt stuck, they were more than willing to help me get through. From the very beginning and up until the end, I knew I could count on their support.

Additionally, I also wish to thank Gordon "fyodor" Lyon, Daniel "bonsaiviking" Miller and David Fifield, the three members from the open-source nmap project, for overseeing my progress with regards to the implementation work on the nmap tool. They made me feel welcome when joining my first open-source project and for this experience I am truly grateful.

Lastly, I wish to thank my parents, as well as my significant other, for their moral support, encouragement and assistance over the last few months. Had it not been for them, the outcome of this project would not have been the same.

Den Haag,
August 2015

Bogdan Alexandru Geana

Abstract

Fingerprinting is the action of detecting which operating system or firmware is running on a network enabled device. There are a number of different methods for achieving this, such as active probing or passive sniffing. Active probing relies on interaction with the target device, while passive sniffing requires eavesdropping on the communication between the target and another device.

This thesis has two goals. The first is to improve the accuracy and reliability of active fingerprinting methods, for legitimate purposes such as detection of malicious devices on a private network. The second goal is to enable detection of fingerprinting attempts in a practical manner, meaning a high detection rate combined with low number of false positives.

Active probing is studied as performed by the open-source nmap tool. This tool makes use of machine learning when classifying the operating system of a device. Several methods are proposed to increase its efficiency, such as the addition of new probes, new features and pre-processing of the training dataset.

With enough knowledge and insight into how fingerprinting is performed, acquired from both the nmap project and existing scientific literature, an anomaly detection scheme is proposed which enables detection of probing attempts. The scheme works in two detection stages and is based partly on machine learning algorithms. A proof of concept is also implemented which is used to test the performance of the proposed detection scheme.

The results show that the scheme is able to achieve a high detection rate while keeping the number of false positives equal to 0. The source code is released under an open-source BSD license and is made public at github.com/alegen/master-thesis-code.

Keywords: nmap, OS probing, anomaly detection, machine learnin.

Contents

Chapter 1

Introduction

1.1 Overview of operating system probing

Remote operating system probing and detection is the action of identifying, through the use of a computer network, the underlying operating system kernel or firmware package installed on a device. The process of probing combined with analysis of the gathered data is also termed fingerprinting.

Detection does not target individual versions of an operating system. Instead operating systems with similar characteristics are grouped into families such as Microsoft Windows NT, Linux 3.X or FreeBSD 9.X. This grouping is indirectly based on implementation choices and design decisions which differ from one project to another.

The term operating system fingerprinting is synonymous to TCP/IP stack fingerprinting. The TCP/IP stack is the component of a kernel which deals with network connectivity. It is this component that allows for discrimination among operating systems based on behavior and responses to certain inputs.

The practice of fingerprinting has uses in both legitimate and illegitimate scenarios. Legitimate uses include the ability to build a detailed and up-to-date inventory of all devices connected to a network and use this information to push security updates to all affected hosts. Detection of unauthorized devices is also one of the legitimate uses, where an administrator may want to identify unknown equipment connected to the local network. For example, the presence of an access point in a network which should only contain printers may be indicative of malicious activity. Such an attempt could be mounted in order to infiltrate a secure network from outside the physical perimeter of a building.

Of course, for the legitimate uses there should be no reason why probing would be dangerous. The illegitimate uses on the other hand may help malicious actors with network reconnaissance and information gathering. In order to execute an attack as stealthily as possible, knowledge about the underlying operating system may prove very useful since it is not easy to determine remotely if a service running on a host is vulnerable or patched. Furthermore, trial and error of different attack vectors may cause crashes or a large amount of suspicious traffic, thus triggering alarms with intrusion detection systems.

One example of a platform-dependent vulnerability which requires additional information for successful exploitation is CVE-2014-0160, commonly known as *Heart Bleed*. This vulnerability affects OpenSSL versions spanning a two year period (i.e. from 1.0.1 to 1.0.1g). An attacker may be interested which operating system is present on a device, prior to commencing exploitation, since OpenSSL is much more common on Linux systems. An additional malicious use-case appears in the context of vulnerable cross-platform software, where one program may have the same bugs present across multiple operating systems. Exploiting such bugs usually requires custom tailored payloads in order to be successful, depending on the combination of CPU architecture and operating system.

Fingerprinting methods fall into one of two categories: active methods which assume interaction

with the target and passive methods which rely on analysis of traffic between the target and another device [Tro03].

The easiest and most straightforward active fingerprinting method involves connecting to services that leak information regarding the operating system (e.g. family, version). This leakage can be either explicit when a service is reporting system details, or implicit when the service is using platform-specific technologies (e.g. ASP.NET intended primarily for Windows).

A more powerful active fingerprinting approach makes use of specially crafted packets. These are sent to the target and fingerprinting is performed based on differences in the replies. Differences are a result of the ambiguity introduced by standardization documents which cannot sensibly and efficiently define the expected behavior that a host should adopt for every possible scenario. The reason is that the number of invalid combinations of packet headers and values rises significantly as protocols become more and more complex. Protocol complexity also leads to behavioral differences between different TCP/IP stack implementations, some of which are incorrect with respect to the available documentation [Pax97]. A packet which is meant to cause undefined behavior is called an anomaly or a probe.

Lastly, a fingerprinting approach based on active probing, but not relying on anomalies, makes use of TCP retransmission timeouts. The TCP protocol is connection oriented and reliable, allowing for lost packets to be sent multiple times if they are not acknowledged in a specified timeframe. The amount of time between retransmissions differs significantly enough to leak information about what operating system is running on a device.

If active methods are not possible, detection can still be accomplished by passive sniffing of the traffic going from the target to a second host. The party performing fingerprinting analyses the headers of network packets. As some of the values in these header do not have standard defaults, they vary from one operating system to another. This variation enables discrimination among multiple families of operating systems.

1.2 Research description

1.2.1 State of the art

Operating system fingerprinting is a topic being actively researched in the IT and information security fields, having two primary directions. On one hand it is desired to increase accuracy and reliability of fingerprinting methods and on the other hand to improve detection of probing attempts.

Implementations of fingerprinting tools are available and used in different setups such as active vs. passive detection or low network interaction vs. increased accuracy. Best known projects nowadays are nmap [nma] for active and p0f [Zal] for passive fingerprinting. The work presented in this report focuses primarily on active fingerprinting and thus on nmap.

The nmap tool has a clever mechanism for classification among multiple operating systems when probing over the IPv6 protocol. During execution, nmap sends a number of 18 probes to a target host and checks for the responses. These probes are partially invalid packets which are meant to generate errors in the TCP/IP stack implementation of the probed device and force responses that have subtle differences depending on the operating system. Specific values are first extracted from these responses, then processed and finally used by a machine learning back-end which takes care of the analysis steps.

The machine learning back-end constitutes the IPv6 fingerprinting engine of nmap. It is trained offline before the online probing process takes place. Multiple sets of responses from known operating systems are submitted by the community. A set of several responses is commonly referred to as a (finger)print. Submissions are integrated into the training set on a regular basis by the project developers. In time, the training set grows with prints added to either existing or new groups of operating systems.

Whenever a new print is submitted, it is labeled accordingly with the operating system that it belongs to. The decision of whether to place the print in a new or existing group is based on a

series of tests which aim to measure how much the print differs from all others in the group with the same label.

For the purpose of detecting and avoiding fingerprinting, tools such as *IP-Personality* and *IP-Morph* exist which disguise the operating system of a device. The first one aims to detect probes based on static signatures and then spoof responses which imitate a different operating system. *IP-Morph* on the other hand makes use of traffic normalization, a process through which certain packet values are modified to be equal for all devices on a network. IDS solutions (e.g. Snort and Bro) are also available, although their applicability to the problem of detecting probing attempts is shown to be inadequate.

Academic research on the topic of intrusion detection in network traffic has led to a number of approaches based either on static signatures of known attacks or on anomaly detection. For the latter, different algorithms have been adopted, some based on statistical analysis and others based on machine learning. The applicability of each method in the context of operating system fingerprinting is discussed in detail in chapter 4.

1.2.2 Scope and limitations

The work described in this report focuses only on fingerprinting via active probing with anomalous packets. From a practical point of view the use of anomalous packets offers the best tradeoff between accuracy and amount of generated network traffic. Thus active fingerprinting is most applicable in real world situations. From the perspective of detecting fingerprinting attempts, the choice to focus on active fingerprinting follows from the fact that passive fingerprinting does not generate any network traffic, nor does it modify the contents of packets. As a result, it is particularly difficult to detect and the cost outweighs the benefits.

Possible fingerprinting methods vary with respect to the internet and transport layer protocols that are used. The protocols considered in this report are IPv6 and ICMPv6 for the internet layer and UDP and TCP for the transport layer. In addition, certain control protocols based on ICMPv6 are also analyzed, such as multicast listener discovery.

The decision to focus on IPv6 is based on two reasons. Firstly, the implementation of the nmap IPv6 fingerprinting engine is a rather new addition to the project and still offers room for improvement. Secondly, the growth levels of the technology have been accelerating in recent years [Goo]. More and more devices are offering support for IPv6 in recent years, a trend resulting from 1) adoption by cloud server providers, 2) the imminent depletion of the IPv4 address space and 3) the rise of the Internet-of-Things movement which will require an increasing number of available addresses [JLS13; IoT].

1.2.3 Problem statement of issues with IPv6 fingerprinting

Scanning and probing over the vast internet is not an easy task. Many problems may occur as a result of latency, packet loss and intermediate devices which change the contents of packets in flight. This results in a lower accuracy level and incorrect classification of the operating system running on a device. A secondary result of these issues occurs during training. Not only does the training set contain inaccurate examples, but in some cases it also has missing values.

Changing the context to possible (mis)uses of nmap, increasing the accuracy and reliability of the tool could have negative side effects. Each connected device is a potential target for an attacker and, as previously mentioned, knowledge of the exact details of a device is valuable when mounting an attack. By identifying a network asset, an attacker automatically discovers its weaknesses and the means to attack it by generating as little network traffic as possible.

While it is true that some IDS systems may offer means to alert when fingerprinting attempts are discovered, they are not always practical especially for the problem that is discussed in this report. Projects such as Snort or Bro are powerful and though they can easily detect known anomalies, they may not be well suited for undiscovered ones. One reason is that Snort makes use of a rule based approach, which means that anomalies need to have clear descriptions in order to be detected. Bro on the other hand makes use of a scripting language that allows for security

policies to be programmed in a flexible manner. In theory, Bro is extensible enough to implement an anomaly detection technique for probes such as the ones used by nmap, but does not yet offer such capabilities.

Projects exist with the ability to counteract nmap fingerprinting attempts by intercepting individual probes and masquerading as other systems or by performing traffic normalization [Gaë; PVH10]. This report looks at the detection methods employed by these tools and shows them to also be problematic since they rely primarily on signatures for detecting probes, similar to Snort.

1.2.4 Goals

The goal of this work is two-fold. On one hand work is aimed towards improving the reliability and accuracy of the fingerprinting engine used by nmap. On the other hand, security measures are developed which offer control and detection of unwanted fingerprinting attempts.

The reasoning behind these two goals is that information security research should have a complementary aspect and focus should be equally spread on both offensive and defensive topics. Focusing too much on offensive research may lead to a dangerous situation in which no mitigations are available to existing attacks, while focusing too much on defensive research may lead to a false sense of security resulting from an apparent lack of vulnerabilities.

1.2.5 Hypotheses

H1. The accuracy and reliability of nmap can be improved by increasing the quality of the data used during training and by implementing new methods of extracting information from a network device (e.g. by re-using data from existing probes or adding new ones).

H2. Although there are a number of well-known probes used by the nmap tool, discovering new relevant anomalies for the purpose of fingerprinting is feasible. As a result of this, it would be possible to circumvent detection schemes that rely on signatures.

H3. Machine learning may offer a scalable solution to allow detection of network anomalies without decreasing the overall quality of the network connection.

1.2.6 Research questions

The research questions answered in this report fall into two different categories which analyze the same problem from different perspectives. From a certain point of view, one may argue that the two directions are competing, when considering their priorities.

The first research question can be summarized as:

How can we optimize operating system fingerprinting via active probing?

and can be further divided into

A1. *How is logistic regression applied in nmap?*
In order to understand how the fingerprinting engine works, the theoretical knowledge and methods in which it is applied are required.

A2. *What are the side-effects of inaccurate and/or incomplete data used for training the logistic regression model?*
A better understanding of the state of the training-set and information obtained while probing is required before attempting to improve these aspects.

A3. *What are the current techniques for probing and scanning hosts over the network?*
This knowledge is a prerequisite for developing a framework for detecting packets used in probing. Next to the knowledge obtained by working to improve nmap, research into the state-of-the-art has to be performed to understand the direction in which the field is progressing.

The second research question is formulated as:

How can we detect fingerprinting attempts which rely on active probing?

and can may further expanded into the following questions:

B1. *What is the best approach to detect packets used for probing a host?*
Such packets (may) contain anomalies which result in different behaviors depending on the operating system. The solution should be generic and not platform-specific.

B2. *How can detection be implemented to work in real-time?*
What are the best approaches for machine learning to allow real time detection of anomalies?

B3. *How can scalability be increased?*
This question primarily targets hosts which receive large amounts of incoming traffic and thus a large number of packets. Since the proposed solution will be based on verifying/scanning packets, it needs to be able to cope with such traffic intensive hosts.

1.3 Purpose and significance of the study

Improving the current state of the art in operating system fingerprinting has a direct beneficial effect to all security practitioners who rely on tools such as nmap for network discovery. With the ability to identify additional information about each host, possible system vulnerabilities can be determined earlier. Furthermore, devices which should not be connected to specific network segments may also be exposed if the firmware or operating system is unfamiliar.

Certain findings which resulted in increased accuracy and reliability of fingerprinting were integrated into the open-source nmap project. This includes the addition of new probes and a new approach to data pre-processing which leads to a higher quality of the trained model and thus better results during online probing. Furthermore, certain parts were also included into a submission to the *8th ACM Workshop on Artificial Intelligence and Security*[1]. The submission was accepted and will be presented in October 2015.

Detecting unwanted probing attempts offers insight and situational awareness to system administrators, allowing them to monitor closely what is happening within their networks. Situational awareness can help characterize the attacker models and threat indicators needed for defining a comprehensive security policy for an organization. Probing attempts also serve as precursors to more dangerous actions and can help correlate multiple seemingly unrelated events in case of a forensic investigation.

Preventing successful fingerprinting of a host may also be desired in certain cases. Although not the focus of this research, a good prevention method requires accurate detection of probing attempts. While security through obscurity is not the proper approach towards eliminating threats, it can be successfully included into a defense in depth strategy. Hiding certain details about a network device may prove beneficial against possible attackers, by increasing the amount of effort required for an offensive action.

1.4 Research outline

The study starts by outlining the definition of an anomaly in the context of network traffic. It then touches on the current methods for operating system fingerprinting and looks at some examples of anomalies that are being used by current software packages.

With a clearly defined concept of how anomalies work and what their effects are, the next step is to verify the hypothesis that previously undiscovered anomalies are practical and offer enough relevant information to distinguish between different operating systems. Furthermore, research is done into the current state of the art techniques in anomaly detection and prevention.

[1] AISec 2015 - www-bcf.usc.edu/ aruneshs/AISec2015.html

After comparing the advantages and disadvantages of several anomaly detection methods and taking into account the possibility of undiscovered anomalies, a new statistical method is defined, implemented and tested against a corpus of network traffic data. Research is also performed into the appropriate means of testing and validating a statistical model.

Emphasis is put primarily on detecting anomalies, while preventing them is a secondary objective provided enough time is available. While preventing an anomaly can be as straightforward as dropping an IP packet, research shows that responding to an intrusion can be a delicate topic which requires careful attention.

1.5 Target audience of this thesis

The work in this thesis is meant to be useful first and foremost to anyone who is working with computer networks from an information security perspective. The ideas presented in this report can offer a different view into how an attacker works his way through compromising a network. Attacker models and threat factors can be more efficiently defined when considering what information can be extracted remotely by a malicious agent. Furthermore, depending on the type of network and the volume of traffic that flows through it, administrators may receive a clear picture of what type of anomaly detection best suits them.

Machine learning researchers may also find interesting the results and design decisions related to different anomaly detection mechanisms. The advantages and disadvantages of several schemes are compared, taking into consideration the types of anomalies that are targeted, the sets of features that are being analyzed and the applicability in a given network environment. Additionally, specific parts discussing the quality of training data could also prove interesting.

Last but not least, open-source enthusiasts may also find this work relevant, considering that parts of it were directly implemented into the *nmap* network scanner.

1.6 Section organization

The rest of this report is structured as follows.

* Chapter 2 discusses the concepts behind active and passive fingerprinting and analyzes the advantages and disadvantages of each method. Furthermore, the operating system fingerprinting engine implemented in the nmap tool is discussed. Details of each of the 18 probes are given, together with reasoning about their structure. The core logistic regression algorithm is described, along with the features it is build upon.
* Chapter 3 presents the work done as part of improving the accuracy of the nmap fingerprinting engine. The three methods are discussed, namely the addition of new probes, the addition of new features to the logistic regression model and the imputation of missing data in the training set.
* Chapter 4 focuses on the current state of the art with regards to operating system probing and network intrusion detection schemes. A discussion is presented at the end of this chapter regarding which direction to follow with the design of the detection scheme against probes.
* Chapter 5 describes the design of the proposed anomaly detection scheme. All steps are discussed, from splitting the detection into two stages, the characteristics analyzed by each stage and the training of the second stage.
* Chapter 6 presents a proof of concept implementation of the anomaly detection scheme, as well as results gathered from testing the scheme.
* Chapter 7 concludes the thesis and chapter 8 discusses possible further directions for research.

Chapter 2

Preliminaries

This chapter presents relevant background information required for understanding the rest of the report. The topics include general operating system fingerprinting, differences between passive and active approaches, a discussion on active fingerprinting and the design and implementation details of the nmap fingerprinting engine.

2.1 Operating system probing

2.1.1 Active vs. passive detection

Fingerprinting the operating system of firmware that is running on a network-connected device can be achieved in various ways, but the main differentiation is whether the technique relies on interaction with a host or not. Each method has its own tradeoffs between accuracy, amount of generated network traffic and pre-requirements for being applicable in a given scenario.

Active fingerprinting relies on interaction with the targeted device [Lyo09] while passive fingerprinting is designed to sniff network traffic to and from the target. Whereas active fingerprinting relies on receiving responses to probes, passive fingerprinting attempts to opportunistically detect the operating system based on contents of captured traffic [Lip+03; Fal11].

Active fingerprinting has the advantage of increased accuracy since it is able to obtain much more relevant information from the target. The responses received to anomalous packets offer more insight into the implementation details of the TCP/IP stack and allow for more fine-grained distinction. The disadvantage is that, by sending anomalous packets, such attempts are easier to detect [GT07b].

Passive fingerprinting does not interact at all with the target and is harder to detect. While eavesdropping on network traffic, there is no sign of interference between the target and any of the endpoints it communicates with. The disadvantage on the other hand is that differences between TCP/IP stack implementations are more subtle and harder to detect. All the information used for distinguishing between operating systems is extracted from legitimate traffic. As such, the accuracy may not be as high and the results not as reliable as in the case of active fingerprinting. Furthermore, the operational requirements are higher than for active fingerprinting, reducing the applicability of this method.

2.1.2 Active fingerprinting basics

The core principle used by any tool which performs active fingerprinting is that of ambiguous packets [Lyo09; sav]. Packets are ambiguous if no exact rules exist on how a networked host should interpret them. Rules are generally specified in RFC documents, but considering the total number of options, flags and fields values for each protocol, standardization bodies cannot practically define the exact behavior for every possible scenario. As a result, it is up to each TCP/IP stack implementation to deal with undefined cases as it sees fit. This results in different behaviors and

responses to the same ambiguous packet, in turn enabling accurate classification of implementations.

Traditionally, implementations of TCP/IP stacks have been included into kernels or firmware and are not handled by individual processes running on top of a system. Because of this, a clear connection can be made between operating systems and and TCP/IP stack implementations.

One example of active fingerprinting is based on the processing of the IPv6 routing header. The routing header was initially created as an extension header for IPv6 to specify a list of one or more intermediate nodes a packet should visit before reaching the final destination [RFC2460]. In the initial RFC, only the type 0 routing header (RH0) was defined.

Specifying multiple addresses more than once is not forbidden. This oversight in the RFC leads to malicious usage of the header and can result in congestion of traffic between two IPv6 routers. As a result, the usage of RH0 has been deprecated [RFC5095], a new behavior has been defined and the implementation of RH0 has become non-mandatory.

While testing against RFC5095 across multiple platforms in search of new possible probes, the following discoveries were made:

1. Microsoft Windows versions 7 through 10 do not reply at all when receiving an ICMPv6 packet with an RH0.
2. FreeBSD 10 and Linux 3.2 reply with an ICMP parameter problem message.
3. Linux 3.16 replies with an ICMP destination unreachable message only if the address of the host is placed last in the list of addresses to be visited. Otherwise no reply is sent at all.

With a single probe it thus becomes possible to distinguish between Windows or Linux/FreeBSD and between Linux 3.2/FreeBSD or Linux 3.16. Combined with other probes, it is possible to further increase the accuracy of detecting a range of versions of a specific operating system.

2.2 nmap's approach to active detection

Among the best known and most widely used fingerprinting tools available is nmap. In order to have a clear understanding of how it achieves its intended purposes, a detailed description of the underlying techniques is given.

2.2.1 Probe overview

In total, nmap sends a maximum of 18 probes for the purpose of fingerprinting. This does not include packets that are sent for other purposes (e.g. port scanning or service fingerprinting). These probes are as follows:

1. *Sequence generation probes (S1 - S6)* - A total of 6 TCP probes that are sent 100 milliseconds from one another in order to detect the algorithm for generating TCP sequence numbers and timestamps. Each packet has a different set of TCP options and sizes of the window.
2. *ICMP echo (IE1 and IE2)* - These are 2 ICMPv6 echo request messages with differing sets of options. The first one contains an incorrect value for the code field in the ICMPv6 header, 9 instead of 0 [RFC4443]. The second echo request has an invalid set of IPv6 extension headers. These headers are: Hop-By-Hop, Destination Options, Routing and again Hop-By-Hop. Per the standard the Hop-By-Hop header cannot appear more than once and it should always appear as the first one of all extension headers [RFC2460].
3. *Node Information Query (NI)* - A Node Information Query is used to ask a network connected device for its IPv4/IPv6 addresses or hostname [RFC4620]. The probe asks the target for its IPv4 addresses, but despite the request some operating systems respond with a hostname instead.
4. *Neighbor Solicitation (NS)* - This probe is based on the IPv6 Neighbor Discovery Protocol. Its purpose is to ask the target for its hardware address [RFC2461]. Since the protocol is

meant to work only on the local network segment, this probe is sent only to targets on the same network.

5. *UDP (U1)* - This probe consists of a UDP datagram sent to a closed port in order to force the target to respond with an ICMPv6 Port Unreachable message.
6. *TCP Explicit Congestion Notification (TECN)* - The probe is a TCP packet sent to an open port and has the Urgent Pointer field set, although the URG control bit is not set [RFC793].
7. *TCP (T2-T7)* - A set of 6 probes, each a TCP packet with a different set of options.

2.2.2 Design

The IPv6 fingerprinting engine used in nmap is based on a learning algorithm called logistic regression. This is a linear, unsupervised, classification algorithm often used for problems to which the answer is selected from a pre-defined set of possible outcomes. In our case the problem can be defined as

Which operating system is most likely to give this set of responses to the probes?

and the set of answers is represented by all known operating systems which nmap is able to detect. At the time of writing, there were 90 known operating system classes. Each class contains ranges of operating system versions with different patches and/or service packs installed.

Being an unsupervised algorithm means that training data needs to be labeled. The training set contains network packet headers from responses to the aforementioned probes, labeled with the operating system which replied. At the time or writing there were a total of 285 sets of responses. A set of responses is refered to as a fingerprint. Features (a total of 676) are then extracted from the headers and a training feature matrix is formed. For the purpose of training the model, an additional "bias" feature is added for which all values are equal to 1. The data can be graphically represented as shown in table 2.1 where $Feature_a$ (a$\in\{0, 1, \ldots, 676\}$) are the independent variables and $Label \in\{1, 2, \ldots, 90\}$ is the dependent variable whose values are being modeled.

Fingerprint #	$Feature_0$	$Feature_1$	$Feature_1$	...	$Feature_{676}$	*Label*
1	1	$x_{1,1}$	$x_{1,2}$	...	$x_{1,676}$	y_1
2	1	$x_{2,1}$	$x_{2,2}$	...	$x_{2,676}$	y_2
3	1	$x_{3,1}$	$x_{3,2}$	...	$x_{3,676}$	y_3
...	...	...	...	...	...	...
284	1	$x_{284,1}$	$x_{284,2}$	...	$x_{284,676}$	y_{284}
285	1	$x_{285,1}$	$x_{285,2}$	...	$x_{295,676}$	y_{285}

Table 2.1: graphical representation of training data.

A trained logistic regression model serves as a binary classifier, meaning that it only distinguishes between two different answers (usually boolean). In order to cope with this limitation, but still be able to perform multi-class classification (i.e. for 90 operating system classes), nmap applies a technique called *One-VS-All* classification which involves training a total of 90 logistic regression models.

Each model is trained to detect whether a given fingerprint belongs to a certain class or not. During the training phase of each model, the dependent variable is changed such that it equals 1 if the fingerprint belongs to the class the model is being trained for and 0 otherwise.

Training consists of calculating a set of 677 weight parameters θ such that the (squared error) cost function $\mathcal{J}(\theta)$ is minimized, where:

$$\mathcal{J}(\theta) = \frac{1}{2*285} \sum_{i=1}^{285} (\mathcal{H}_\theta(\mathcal{X}_i) - y_i)^2$$

$\mathcal{H}_\theta(\mathcal{X}_i) = \frac{1}{1+e^{-\theta^T * \mathcal{X}_i}}$ is called the hypothesis and sigmoid function

$$\theta = [\theta_0, \theta_1, \theta_2, \ldots, \theta_{676}] \qquad \mathcal{X}_i = [1, x_{i,1}, x_{i,2}, \ldots, x_{i,676}]$$

The sigmoid function $\mathcal{H}$ is constructed such that it always outputs values in the open interval (0, 1). The intuition behind the cost function $\mathcal{J}$ is that it quantifies how well the model can predict the correct values in the dependent variable, given the static training feature matrix and a set of weights θ.

The smaller the difference between $\mathcal{H}_\theta(\mathcal{X}_i)$ and y_i is, the better the weights fit the logistic regression model. As a result, whenever fingerprint i belongs to the targeted operating system class, the hypothesis function should output a value close to 1, otherwise 0. Finding adequate values for θ is performed using an optimization algorithm, such as gradient descent.

During operating system probing, nmap obtains a set of responses from the target. It then extract the same features as those used during model training and obtains a feature vector $\mathcal{X}_p$. In order to identify which class $\mathcal{X}_p$ belongs to, it calculates the prediction score (i.e. result of $\mathcal{H}_\theta(\mathcal{X}_p)$) for each of the 90 models and reports the one with the highest value.

All logistic regression computations performed by nmap are based on the open-source *liblinear* library [Fan+08].

2.2.3 Feature overview

The total number of 676 features used in the logistic regression model come from packet headers of different layers. From the IPv6 header which is part of the internet layer, the following features are extracted:

1. *Payload length*: 16 bit unsigned integer, the value of which is equal to the size in octets of the packet excluding the IPv6 header.
2. *Traffic class*: 8 bit field which holds two values; the first 6 bits are used for differentiated services and hold ID numbers used to classify packets [RFC3260] and the remaining 2 bits are used for explicit congestion notifications [RFC3168].
3. *Hop limit*: 8 bit unsigned field which is decremented by 1 by each node that the packet passes through; this is the IP equivalent of the time-to-live field.

More features are extracted from the TCP transport layer header:

1. *TCP window*: 16 bit value equal to the size of the receive window.
2. *TCP flags*: 12 individual bits which signal different events in a TCP connection.
3. *TCP options*: various options depending on the implementation of the TCP/IP stack and supported TCP extensions.
4. *TCP option length*: the lengths in octets of each TCP option.
5. *TCP selective acknowledgement*: TCP extension that allows a receiver to acknowledge non-continuous blocks of data.
6. *TCP maximum segment size*: the maximum amount of data that a host is able to receive in a single packet.
7. *TCP window scale*: used for modifying the size of the TCP window in order to benefit from high bandwidth network.

The last feature is the initial sequence number counter rate and is calculated from the 6 sequence probes. These are sent precisely 100 ms apart and the feature value is calculated by adding the differences between consecutive TCP initial sequence numbers divided by the total time.

This set of features was chosen by then nmap developers as optimal considering the information that they are able to convey for training the logistic regression model. Good features tend to have very little variance per operating system class, but high variance overall. Furthermore, some of them have little variance overall, but the values differ for only one operating system which in turn helps when training the model for that particular class.

Whenever the decision is made to use new features, their efficiency is tested by means of cross validation. Additionally, cross validation also offers the possibility of verifying how well the 90 logistic regression models operate. The type of cross validation that is used for nmap is k-fold, with 5 folds. The complete training set is divided into 5 equal parts and for each of 5 times, logistic regression models are trained on 4 parts and validated on the 5^{th}.

The splitting is done in a random fashion, without ensuring that the same percent of examples from each class is present in each fold (i.e. without stratification). This is not the most optimal approach since folds may not have adequate representatives of each class, but this is the best option for the nmap training set since it contains a total of 44 classes with only 1 example.

2.2.4 Result validation

As explained previously, during the process of fingerprinting nmap reports the class for which the logistic regression model outputs the highest score. By itself, this approach is not enough to ensure that the end-result is valid. Two models may report similar scores or the highest score may be very low and thus not representative. At the end of the probing phase, a number of further steps are taken:

1. Verify that the 2^{nd} highest score is smaller than 90% of the 1^{st} highest score. If this requirement does not hold, then the scores are presumed to be invalid and a failed detection message is reported.
2. Calculate the Mahalanobis distance between the set of features extracted from the probe responses and the sets of features belonging to the class that were used during training. The Mahalanobis distance is a metric which shows how much a point differs from all examples of a related distribution. Applied to the case of nmap, it shows how much the responses to the probes differ from the training responses. The distance needs to be smaller than a threshold value equal to 15, chosen after inspecting of the values that are usually returned for valid and invalid results. If the distance is above the threshold, the fingerprinting result is again presumed to be invalid.

2.3 Conclusion

This chapter discusses the differences between active and passive fingerprinting and emphasizes the advantages and disadvantages of each method. Active fingerprinting relies on direct network interaction with the target, whereas passive fingerprinting eavesdrops on traffic flowing to and from the target.

While passive fingerprinting has the added benefit that it is almost impossible to detect (unlike the active version which makes use of anomalous packets), active fingerprinting yields more accurate results and is more practical from a deployment perspective. It is much easier in a real world scenario to interact with a host directly than to access traffic flowing from it to another device.

Furthermore, the design details of the nmap fingerprinting engine are reviewed. The nmap project makes use of logistic regression which is a machine learning algorithm used for classification problems. Since it needs to determine which operating system is running on a device from a set of multiple choices, it makes use of *One-VS-All* classification and trains multiple logistic regression models. During fingerprinting, the model which gives the best score is chosen and a number of validation steps are performed to ensure that the result is correct.

Chapter 3

Improving accuracy in nmap

Improving the state-of-the-art on operating system fingerprinting of IPv6 network enabled devices offers two important benefits:

1. It increases the accuracy and quality of results offered by the nmap tool for legitimate purposes (e.g. detecting unauthorized devices connected to the local network). The reasons for working on nmap and not other implementations are the availability of the source-code and the ability to interact with the maintainers of the project.
2. Offers a different perspective of what turns a packet into a probe. It is not necessary for the packet to be malformed. Instead, even packets that are structurally valid can be used as probes (e.g. multicast listener discovery general query packet). Additional knowledge of what turns a packet into a probe was acquired. As a result, the scope and definition a network anomaly was broadened to include the findings presented in this chapter.

In this section we present the different directions which were pursued in order to increase the reliability of nmap, the experiments that were executed and their results.

3.1 Additional probes

One of the first characteristics that were sought to be improved was the set of probes used by nmap for the purposes of fingerprinting. Research was done into the possibilities of adding new probes that would allow for additional information to be obtained from network devices.

3.1.1 IPv6 fragment ID generation algorithm

IPv6 fragmentation preliminaries

The original IPv6 RFC specifies a set of four extension headers which can be optionally added to a packet based on certain criteria [RFC2460]. Of these four headers, the *Fragment Header* is used for the purpose of splitting a large packet into multiple parts such that they can fit within the maximum transmission unit (MTU) of a path between two devices. The MTU is calculated as the minimum of all link MTUs between every two consecutive nodes.

Amongst the fields present in this header, there is one used for grouping multiple fragments of the same packet, called the *Identification* (ID) field. Since fragmentation of packets can only be performed by nodes from which the packet originated and never by intermediate nodes, the choice of what values to use for IDs lies with the sender. The RFC document does not specify how these values should be generated, only that a newly generated value *"should be different than that of any other fragmented packet sent recently"*. As a result, it is possible to link the fragment ID generation algorithm to a TCP/IP stack implementation [Mor].

Operating System	Support for "atomic" fragments	ID generation algorithm
Linux 2.6.32	yes	incremental by 1
Linux 3.2	yes	incremental by random number
Linux 3.10	no	-
Linux 3.16	no	-
Linux 3.18	no	-
Windows 7	no	-
Windows 8.1	yes	incremental by 2
Windows 10	yes	incremental by 2
OpenBSD 5.6	yes	random
FreeBSD 10.1	yes	random

Table 3.1: support for IPv6 "Atomic" Fragments and ID generation algorithms.

Generation of *"Atomic" Fragments*

One of the most straightforward methods of obtaining packets with a *Fragment Header* from a host is via *"atomic" fragments*. This phenomenon is defined in the original IPv6 RFC and is triggered whenever a host receives a *"packet too big"* message, in response to a packet sent previously that was too large for the specific path. The message in this case requests all subsequent packets to be less than 1280 bytes in size (i.e. IPv6 minimum MTU). The purpose of this feature is to allow IPv6 packets to be forwarded to hosts which only support IPv4 and have a minimum MTU of at least 68 bytes. If this is the case, the sender does not need to split packets into parts smaller than 1280 bytes, but instead is required to generate *"atomic" fragments*. These consist of full packets which still include the header of interest.

The *"packet too big"* format is specified in the ICMPv6 RFC [RFC4443] as an ICMPv6 packet with type 2 and code 0. Additionally, the payload needs to be set to the contents of the packet which was too large in size to forward.

A python script that uses the *Scapy* library was created for the purpose of testing how different operating system follow the RFC specification[1]. The script generates a *"packet too big"* message and sends it to a host of choice. Further testing for *"atomic" fragments* is performed by sending ICMPv6 echo requests and checking whether the replies include a *Fragment Header*. Since the aim was to force a target host to generate *"atomic" fragments* without any prior traffic, a trial and error search was conducted to discover what ICMPv6 packets can be accepted as valid. We discovered the following requirements for a *"packet too big"* message:

1. The exact MTU specified in the *"packet too big"* message can vary between 600 and at most 1279, depending on the TCP/IP stack implementation. The choice was made to use a value of 1278.
2. The payload must be set to an IPv6 header for which the source and destination addresses are the opposites of those found in the IPv6 header of the encapsulating packet. This is enough and no upper layers are needed (e.g. UDP, TCP) for the encapsulated IPv6 header.

The initial testing showed positive results as performed on two virtual machines running older versions of Debian 6 and 7, Linux kernel versions 2.6.32 and 3.2 respectively, and newer versions of Windows. The fragment ID generation algorithms were incremental by 1, 2 or a small random number. OpenBSD and FreeBSD on the other hand use completely random values for consecutive packets. Table 3.1 shows the support for *"atomic" fragments* and the ID generation algorithms of tested operating systems.

Even so, support for *"atomic" fragments* is slowly being removed as a result of deprecation by newer standards. The behavior was first superseded by suggesting different counters be used for *"atomic"* and regular fragments if the ID generation is globally incremental [RFC6946]. A second

[1] fh_probe.py - github.com/alegen/code-snippets/blob/master/scapy/fh_probe.py

Operating System	ID generation algorithm
Linux 3.10	incremental by random number
Linux 3.16	incremental by random number
Linux 3.18	incremental by random number
Windows 7	incremental by 2

Table 3.2: fragment ID generation algorithms for fragmented echo replies.

modification is being discussed in a draft RFC which pushes for the deprecation of the complete *"atomic" fragments* specification [RFCDR1]. The reasons for deprecation are security-related and try to avoid problems caused by an attacker being able to guess future ID values used in packets from other connections.

Pursuing development of a probe based on *"atomic" fragments* becomes no longer reasonable when taking into consideration current developments. Already the Linux kernel has been patched in February 2015[2] and the change has been backported to versions as far back as 3.10.

Fragmentation of ICMPv6 echo replies

An alternative method of obtaining packets with *Fragment Headers* from a device is to force it to reply with packets larger than the maximum possible MTU. In this scenario, the headers are used for the intended purpose and are thus safe against deprecation. Testing the generation algorithms of ID values was performed by sending ICMPv6 echo requests with a payload of 2000 bytes. The value was chosen to be higher than the most common ethernet (OSI layer 2) MTU value of 1500 bytes. The ICMPv6 specification states that the echo reply needs to include the complete payload data from the invoking request message, meaning the reply will also be fragmented. In addition to the results from table 3.1, the ID generation algorithms shown in table 3.2 were found. As can be seen, it is possible to differentiate between the most common operating systems based on this algorithm. In the case of the Linux kernel, it is even possible to differentiate between release versions.

There is one draft RFC which discusses the impact of an attacker guessing fragment ID values for general fragments and suggests different ID generation algorithms [RFCDR2]. Although these algorithms have an impact on other attacks, they do not pose a risk for fingerprinting. The author proposes generation of ID values by using host-based counters (i.e. different starting values for each remote host), random ID values or a solution based on one-way hash functions. With such a variety of options, it is assumed that different vendors will pursue different variants.

Implementation in nmap

Even though fragmentation of ICMPv6 echo replies is a reliable method for fingerprinting the operating system of a remote host, implementation of this technique in nmap was not possible. The reason was due to the internal architecture of the probing engine on which nmap relies. The current design only works with probes which consist of one packet and also receive a response of one packet. This is in contrast to the fragmented approach where both the probe and the response consist of two packets each.

3.1.2 Multicast Listener Discovery requests

Preliminaries

Multicast Listener Discovery (MLD) is a protocol meant to be used on the local network segment only. Its purpose is to allow IPv6 router nodes to detect whether hosts exist on the local segment which are listening to multicast addresses. The protocol specifies that IPv6 routers need to send MLD queries to the IPv6 multicast ethernet and link-local addresses for all nodes, meaning

[2]Linux kernel mailing-list - lkml.org/lkml/2015/2/24/880

33:33:00:00:00:01 and **ff02::1** respectively. Devices listening to any multicast addresses need to send MLD reports to the router specifying which addresses they are interested in.

A *general query* message is used to discover all multicast addresses which have listeners on the local segment. For the purpose of fingerprinting, such a message may be used as a probe.

The MLD protocol has two versions which are defined separately from one another [RFC2710; RFC3810]. Although the formats of certain messages differ, MLDv2 implementations must also support certain MLDv1 messages for backwards compatibility and in order to maintain a correct overview of the link-local state.

There are two known issues with both versions of MLD which help make the protocol a candidate for designing new probes. The first results from the RFC documents forcing nodes to answer queries sent to multicast as well as unicast addresses. Thus interaction between two single hosts is possible and MLD queries can be sent directly to a device. Less network traffic is generated this way during fingerprinting. The second issue is also design specific and results from the election method of which router to assume the role of querier. In order to simplify this decision, the router with the lowest IP address is chosen. There is no additional validation built into the protocol to check whether a query came from a legitimate router or not. As a result of these two flaws, any host on a network can pretend to be a router, send a *general query* message to a unicast address of another device and receive a response, without raising any alarms.

Discrimination based on multicast addresses

The choice of which multicast addresses a device is listening on does not belong to the TCP/IP stack implementation, but instead relies on the software which is running. Even so, certain operating systems listen to specific addresses when using default configurations. Based on the set of multicast addresses reported by a device, the operating system can potentially be inferred. In table 3.3, the default addresses are shown for the tested operating systems.

Operating System	Default multicast addresses
Windows 7 Windows 8.1 Windows 10	ff02::1:3 ff02::c
IOS	ff02::2 ff02::d ff02::16 ff02::1:2
FreeBSD	ff02::2:ff2e:b774
Ubuntu Linux	ff02::fb

Table 3.3: multicast addresses in default configurations.

The advantage of using MLD queries as probes is that one can potentially differentiate between different distributions of Linux. For example, a device reporting that it is listening to address **ff02::fb** can safely be assumed to be running the Ubuntu Linux distribution.

Implementation in nmap

Implementation of a new probe based on MLD *general queries* was difficult in the core nmap code. MLD version 1 reports generate one packet for each address a device is listening to. This results in a similar problem posed by fragmented ICMPv6 replies. A secondary reason for not implementing this probe as part of the main codebase was possible interference from installed software and the fact that it works only on the local-link, thus limiting its usability.

As a solution, fingerprinting using MLD queries was implemented in a script for the nmap tool. The script is written in the LUA programming language and is executed in the context of a nmap process. All available nmap functionality is available to scripts also.

Since there was an existing script which performed host discovery on the local segment using MLD *general query* messages, certain components required for a fingerprinting script were already in place. There were several challenges during implementation related to practical issues such as:

1. The standard *nmap scripting engine library* (nselib) offers various functions and routines for building raw packets, but none dedicated specifically to MLD. As a result, close attention had to be paid to the RFC specifications to ensure that the script always builds valid packets. The same applies to parsing MLD reports received over the network; whereas MLDv1 reports contain only one multicast address, MLDv2 are different and may contain multiple addresses making the parsing of such packets more difficult.
2. Silent bugs in the VirtualBox networking code delayed the implementation of the script. One of the fields in the MLD header specifies the amount of time for which a host may delay answering to a query. For the purpose of fingerprinting, the field was initially set to 0 to force devices to report immediately. This would in turn crash the VirtualBox TCP/IP stack implementation, leaving all testing virtual machines disconnected. The workaround to this solution was to set the field to 1 instead.

3.1.3 IPv6 extension headers fuzzing

Extension header preliminaries

The previously discussed two probes do not deviate from the relevant specifications, but instead try to follow them and use structurally valid packets. The next section presents the results that we obtained from fuzzing TCP/IP stack implementations of multiple operating systems. Invalid packets were purposefully sent in order to cause undefined behavior.

The original IPv6 RFC specifies a total of 6 extensions headers with various uses. The exact order in which they may appear in a valid IPv6 packet is: hop-by-hop options, destination options, routing (type 0), fragment, authentication and encapsulating security payload [RFC2460]. In addition, the RFC also requires that each extension header appear at most once, with the only exception being the destination options header which can appear at most twice.

As a result of recent developments, the routing (type 0) header has been deprecated for security reasons [RFC5095] and support for it has been removed from the Linux kernel [Edg].

Lastly, there is no properly defined behavior for how a device should respond if it receives an IPv6 packet with an invalid set of headers.

All these reasons led us to believe that it may be possible to differentiate between operating systems based on how they answer invalid packets.

Fuzzing methodology and implementation

Fuzzing was performed by grouping together different combinations of IPv6 extension headers from the following set:

1. Three destination options headers, one of which containing no options.
2. Two routing headers, the first having the target IP address as first in the list and the second one having it as last. The list of IP addresses in a routing header specify which hosts the packet must visit before reaching its destination.
3. Six hop-by-hop headers, one of which containing no options.
4. One fragment header.

The total number of headers in the set was 12 and the total number of packets sent to each fuzzed operating system was:

$$\binom{12}{0} + \binom{12}{1} + \cdots + \binom{12}{11} + \binom{12}{12} = 4096$$

In order to execute this experiment against multiple operating systems, two scripts based on *Scapy* were developed.

The first script[3] generates packets with all aforementioned combinations of headers and sends them to a network host. It then proceeds to save tuples of probe & reply to a pcap file. This allows for additional testing to be carried out easily, knowing what probe generated a certain behavior.

The second script[4] is used to enable easy differentiation between sets of responses of distinct operating systems. The script checks for common responses, strips values from fields which are not comparable (e.g. IP addresses, checksums etc.) and then checks whether the remaining values are equal.

Results

Major differences were discovered with regards to which packets obtain a response, based on the operating system vendor. While Linux version 2.6.32 replied to 1025 probes, version 3.18 replied to 765. Furthermore, the structure of ICMPv6 "parameter problem" messages also varied. Whereas Linux 2.6.32 sends back the complete invalid packet (i.e. the probe) as payload, version 3.18 removes some of the extension headers. Similar behaviors were seen also from Windows where version 7 responded to 918 probes and version 8 to only 16, although they are consistent in the responses they send to same probes. Discrimination is also possible against FreeBSD and OpenBSD, both in the number and format of the responses.

Discrimination between operating systems is possible based on these answers. Although the results were positive, none of the probes were implemented in nmap. The main reason stemmed from probes not receiving a reply from all operating systems. Multiple probes would be required in order to cover a wide range of possible scenarios, in turn increasing the total number of probes used by nmap. Additionally, in practice it would be hard to accurately detect whether a response was not sent or was not received due to network instability.

3.2 Additional features from existing probes

Besides adding new probes to from which to extract relevant information, the option of using only the existing training data was also considered. An additional method of improving the results of the logistic regression engine was to add more relevant features from existing fingerprints, which would further enable differentiation between operating systems. In this section we describe the new features that were added to nmap.

3.2.1 IPv6 hop limit

The IPv6 protocol uses a *hop limit* value for each packet which is equivalent to the IPv4 *time to live* value. When a packet is first sent, the field is given an initial value. Each subsequent node visited by the packet decremenets the value by 1. When the value reaches 0, the packet is discarded. The purpose of this field is to prevent packets from being forwarded indefinitely. Although existing RFC documents specify how the field must be used, none enforce a specific initial value. As a result, different TCP/IP stack implementations use different starting values.

The field inside the IPv6 header is 8 bits long, meaning it can take values in the interval 0 to 255. While analyzing the nmap training set and testing different operating systems, it was discovered that values cluster around 32, 64, 128, 255. For this reason it was decided to treat the new feature as categorical. A cateorical feature means that each observation can take a value from a predetermined set. Although this decision does not have a great impact for logistic regression, it had a greater impact on the imputation work (described in chapter 3.3).

One issue that came up while adding this feature to nmap was the fact that probe responses in the training set do not have the original hop limit value anymore. This is to be expected,

[3] exthdr_probes.py - github.com/alegen/code-snippets/blob/master/scapy/exthdr_probes.py
[4] pcaps_diff.py - github.com/alegen/code-snippets/blob/master/scapy/pcaps_diff.py

```
1     var ts_hl                                  # training set hop limit
2     var er_lim                                 # error limit
3     for each cc_hl in [32, 64, 128, 255]       # iterate over all cluster hop limits
4         if cc_hl - er_lim <= ts_hl and ts_hl <= cc_hl + 5
5             return cc_hl
6         end if
7     end for
8     return -1
```

Figure 3.1: pseudo-code for guessing initial hop-limit values.

since packets have to travel through a number of intermediate nodes before reaching their final destinations. In order to guess the original hop limit value, the algorithm in figure 3.1 was used.

The algorithm verifies if the hop limit of a packet from the training set is close enough to any of the four cluster values, within a given error range. If it is, then the training set values is replaced with the cluster value. Otherwise it is assumed that the initial hop limit value cannot be sensibly guessed. In this case the training set value is replaced with -1.

The reasoning for using error limits is that if a training set value is too far away from a cluster value, then it might have been modified by an intermediate node. If this is the case, the hop limit should be avoided when training the model, as it is not representative.

The upper limit is set to 5 so that we take into account asymmetric routes (i.e. the probes and responses travel through different routes between the same two hosts). The lower error limit is calculated based on additional information contained in nmap fingerprints, regarding the hop distance between the two hosts. This distance is calculated during probing in different ways depending on the scenario (e.g. traceroute, ICMP error messages).

The impact of adding this new feature was verified by means of cross validation (as explained in chapter 2.2.3). Incidentally, there is a decrease in the estimated accuracy of a model trained without and afterwards with the new hop limit feature. It is suspected that, during the non-stratified k-fold cross validation, certain folds do not have an adequate amount of examples for each class. As a result, the folds are not completely representative of the whole training.

3.2.2 TCP maximum segment size and receive window size correlation

An additional feature added to the logistic regression model was obtained from combining two existing features, namely the TCP maximum segment size and the TCP window size. Logistic regression models do not capture hidden correlations between existing features. These correlations have to be "encoded" in additional features which need to be calculated prior to the training phase of the model.

The TCP maximum segment size refers to the maximum size of each TCP packet. An important factor when deciding this value is the size of the packet receive window. As such, there is definite correlation between the two parameters when establishing a TCP connection.

With this feature added to the logistic regression model, differentiation between Linux kernels before and after version 2.6.39 became possible. With the release of the specified version, the initial congestion window was increased [Cor].

After inspecting the values for the TCP maximum segment and window sizes, it was decided to treat these two features as continuous variables. For a feature to be continuous, it means that values from observations fall in an open interval, instead of a predefined set.

The formula for calculating the new feature was chosen as

$$correlation \quad feature = \frac{TCP \quad window \quad size}{TCP \quad maximum \quad segment \quad size}$$

The results from cross validation applied without and afterwards with the new feature showed a slight increase in the estimated accuracy levels. One reason for the increase may be that correlation

between these two features is more consistent than the hop limit values in the classes of the training set. Even so, the same problems regarding the k-fold cross validation test still applies.

3.3 Imputation of missing data

Imputation in the context of statistics is a process which substitutes missing data with artificial values that try to emulate real world observations. Given a dataset with incomplete samples (fingerprints in the case of nmap), imputation tries to infer the missing values based on existing ones from related features.

3.3.1 Problem description

One of the difficulties with probing devices is the inherent network instability or interference on the transmission path which may result in probes or responses not arriving at their final destinations. These obstacles give way to an incomplete training set which leads to a decrease in the quality of the training data and thus a deterioration of the classification accuracy. The solution that nmap had in place for dealing with this problem was substitution of all missing values with -1. A better solution was needed such that missing values would be substituted with relevant values, preferably inferred from other similar features.

The reasons why data items are missing needs to be taken into consideration. The nmap scripts contain code which checks whether a value is missing due to non-response on behalf of a host (labeled MISSING) or because of reasons such as interference from firewalls or network instability (labeled UNKNOWN). Only values labeled UNKNOWN need to be imputed. The ones labeled MISSING are expected behavior which the logistic regression model needs to adapt to.

3.3.2 Dealing with missing data

There are multiple solutions to the problem of missing data in statistical analysis, although not all of them were equally applicable in the case of nmap. The most simple solution was to remove all fingerprints with missing data and use only the remaining ones. Analysis of the training set showed that around 90% of the fingerprints had at least one missing item. For practical reasons, removing these fingerprints from the training set was not feasible. The harm resulting from a much smaller training set would outweigh the possible benefits.

Further research led to solutions such as replacing all missing values of each feature with the mean of the existing values or with randomly selected values of the same feature. The former has been shown to be sub-optimal since it introduces bias into the dataset, whereas the latter may lead to datasets which are not representative of real world observations.

Among the most optimal imputation methods is the use of multiple imputation by chained equations (MICE). The main advantage this approach brings over single imputation methods (e.g. mean substitution) is a lower bias rate. Furthermore, MICE can be applied to different types of models and variables (i.e. continuous or categorical).

The applicability of MICE depends on the dataset and the nature of the missing data. MICE was designed to function on the assumption that data is missing at random (MAR), meaning the probability that a value is missing depends only on observed data and never on unobserved data [Azu+11]. In other words, the missingness may be explained by variables on which full information is available.

In the case of nmap, this assumption holds since fingerprints belonging to the same class or even to the same operating system vendor are similar. As a result, missing values of one fingerprint may be inferred exclusively from existing values of a different fingerprint from the same class.

The full method is described in figure 3.2 [Azu+11]. The required parameters for applying MICE are: 1) the number of imputed sets to be generated, 2) the maximum number of imputation iterations for each set and 3) the imputation model to be used for each each variable that has missing values.

```
1    simple_imputation(dataset)  # missing values imputed with placeholders
2    for each imputed_set
3        for each imputation_iteration or until convergence
4            for each imputed_variable in dataset
5                set placeholder values back to missing
6                infer the dependent variable imputed_variable from all other
                     independent variables using imputation_model
7            end for
8        end for
9    end for
```

Figure 3.2: pseudo-code for guessing initial hop-limit values.

For the imputation models, it was decided to use logistic regression for features with only two values for all observations (i.e. bit flags), multinomial logistic regression for categorical variables (e.g. hop limit) and linear regression for continuous variables (e.g. TCP window size).

The maximum number of imputation iterations does not need to be reached if the imputed set converges, meaning that it becomes stable and changes very little or not at all from one iteration to the next.

3.3.3 Implementation in nmap

There are not many available MICE implementations to choose from, the best supported options being *Amelia* and *mice*. Both of them are implemented in the R programming language. For practical reasons and general availability of online documentation, the mice library was chosen [BG11]. This posed an initial challenge since the nmap code which performs training is written in the python programming language. The solution came from the *rpy2* python library which acts as a bridge between Python and R processes and allows sharing data structures between the two programming languages.

A second issue was method in which mice returns the result of the imputation process. Instead of returning a single imputed matrix, it tries to fit a user-provided model and returns the coefficients for each feature. This was not optimal since the Python workflow was already integrated with the *liblinear* library for training of the logistic regression model.

In order to solve this problem, it was decided to use the intermediate imputed sets from *mice* and merge them based on the type of variable. For continuous variables, all imputed values for the same missing item are averaged. For categorical variables a "voting" mechanism was implemented for the imputed value that with the highest frequency.

The first version of the *mice* integration into the nmap model training system was not successful. The problem was that all features would be fed to the imputation routine, each one with a different imputation model attached. After additional consideration, it was decided to group features together based on the imputation model and perform multiple executions of the imputation routine.

The new version resulted in a partly working integration, though the complete process required extensive amounts of time. During integration of fingerprint submissions, the complete training process needs to be executed for each new fingerprint. An execution time of more than 2 to 3 minutes was not acceptable.

A secondary issue was discovered after comparing imputed sets from individual executions of the same imputation routine. The imputed values would sometimes greatly differ, in turn leading to completely different trained models, some of which worse than the beginning situation. This was undesirable and a different method that returned more "stable" results was needed.

The next step was to search for the features that were most beneficial towards the accuracy of the trained logistic regression model. The rationale was that, with a smaller set of features fed to the imputation routine, less time would be required for it to finish and less noise would be present in the features. In turn, the results would become more "stable", without major differences from

one imputation execution to another.

For this purpose, the *recursive feature elimination* (RFE) algorithm implemented in the *scikit-learn* [Ped+11] library was used. By specifying a logistic regression model to be fit to the dataset, the RFE algorithm can select the most relevant features. It does this by recursively considering smaller and smaller sets of features, pruning the ones with small weights. Cross validation is used to ensure that removing a certain feature does not negatively influence the dataset.

After using this library, a minimal set of roughly 60 features[5] was chosen. While testing the imputation routine on this set, a major decrease in the execution time was noticed, although the "stability" issue was still present.

The fix for the stability problem was to group features together, not based on their imputation models, but instead based on the relationships between them. Features have to be grouped together such that relevant information is encoded for the imputation model to produce meaningful values. Features were grouped based on the layer in the TCP/IP stack model (e.g. IPv6 features, TCP features) and the protocol header field from which they originated (e.g. IPv6 hop limit, IPv6 traffic class).

Additional testing with different values for the number of imputed sets and the number of iterations per set showed that imputed values were persistent among multiple executions of the imputation routine.

3.3.4 Results

The final testing results were obtained by performing classification on previously unseen fingerprints (i.e. not part of the training set), using a model trained on an imputed dataset. In order to quantify the performance gains, we measured the increase for the prediction score of each fingerprint.

In the case of a Linux 3.18 fingerprint from a Fedora 21 virtual machine, the score without imputation was 8% and with imputation 22%. For a Linux 4.0 fingerprint the score increased from 75% to 85%. For a Windows 8.1 fingerprint, the score remained the same at 99%.

In all cases, the novelty factor was slightly increased, though this problem can easily be fixed by an increase of the threshold.

3.4 Conclusion

There are multiple methods for improving the accuracy and reliability of fingerprinting, such as the use of additional probes, additional features or additional pre-processing of the dataset before the training phase. While some of these additions can be directly applied to nmap, others are incompatible with the internal design of the tool.

This chapter discusses the use of additional probes based on IPv6 fragment ID generation algorithms, multicast listener discovery (MLD) queries and probes discovered by fuzzing the TCP/IP stack implementations of multiple operating system vendors. The first two methods rely on network packets which are well-formed and structurally valid according to specifications, while the third method relies on inducing behavioral variation resulting from erroneous packets.

A different approach to enhancing nmap was via additional features added to the logistic regression model. The first feature, the hop limit field from the IPv6 header, differs based on the operating system since there is no value enforced in any of the specifications and vendors are free to choose their own. The second feature is the result of combining two existing features, namely the TCP maximum segment and receive window sizes. Since logistic regression models cannot capture the correlation between features, mixing features together can help improve this deficiency.

Lastly, additional pre-processing of the dataset by means of imputation led to major increases in the model performance. This method has the advantage that no additional data needs to be added gathered, unlike the previously discussed methods. Instead it aims to increase the amount of information from the existing data by inferring missing values from existing ones.

[5]Features from the RFE algorithm - github.com/alegen/nmap/blob/imputation/ipv6tests/impute_filter.cfg

Chapter 4

Related work

This chapter discusses the current state of the art and existing research on the following topics:

1. operating system probing and detection
2. methods of identifying fingerprinting attempts and
3. intrusion response.

A detailed overview is given ranging from software implementations of fingerprinting detection and preventions systems to academic research in the field. Based on the findings presented in this chapter, further decisions are made regarding the architecture and design of the proposed anomaly detection scheme.

4.1 Active operating system fingerprinting

The work presented in this report focuses mainly on active fingerprinting (i.e. involving interaction between prober and probee). Following is an overview of the related work in this field, based on the current research and literature.

One of the problems with active fingerprinting is the network traffic it generates as a result of sending many anomalous packets. The default configurations in open-source tools, such as nmap, generate a predefined set of probes that are easy to detect. Intrusion detection systems often interfere with probes and therefore a smaller number of less frequent probes are desired.

In [GT07a; GT07b] the authors provide a theoretical framework alongside a practical implementation of a fingerprinting mechanism which minimizes the number of probes that are sent to a target. The research focuses on the information gain of a possible probe, calculated as the difference in classification accuracy before and after receiving a response to the packet.

An additional improvement is the use of decision tree classifiers which aid in choosing the exact sequence of probes to send during fingerprinting. The sequence is constructed dynamically, based on intermediate responses to probes. The sequence also takes into consideration the possibility of a probe to be detected, conveyed as a metric which represents the cost of the probe.

Some of the important discoveries regarding probe structure are:

1. Probes which contain a TCP header with the SYN flag set are less likely to be discovered and blocked by detection systems.
2. Malformed TCP packets have a higher probability of being detected.
3. ICMP packets often raise alarms.

Applied to the nmap set of probes, the results show successful classification of three major operating systems with as few as 4 anomalous packets.

A different approach is presented in [Sha+14] which discusses a theoretical framework for operating system fingerprinting with information extracted from one single probe. The concept is based

on splitting the set of features extracted from a probe into two sets, depending on how the values are modified in transit. One set contains features modified by the network (e.g. retransmission timeouts) and the other modified by the target host (e.g. TCP window size).

The aim then becomes to optimize the information gain from both sets of features. The authors present their own sets of features and results from testing this method. They achieve classification accuracy levels of up to 99%.

4.2 Avoiding information leakage

4.2.1 IP Personality and TCP/IP stack spoofing

IP Personality [Gaë] is a project which aims to modify the behavior of the TCP/IP stack. It was designed to work with Linux by loading a kernel module which takes control over certain aspects of the network-packet processing pipeline. By emulating different characteristics found in other TCP/IP stacks, IP Personality can mislead active fingerprinting tools and prevent successful detection.

The project was developed to detect and protect against nmap fingerprinting attempts. In order to achieve this, the kernel module includes detailed signatures of the nmap probes. The detection mechanism is based on static signatures. Since all of the probes are anomalous and rely on pushing the TPC/IP stack into undefined states, it is straightforward to define rules that detect these packets.

Furthermore, the project distributes rules of how other TCP/IP stacks behave to every individual anomaly. Whenever such a packet is encountered, IP Personality sends back a response based on which TCP/IP stack it is imitating. The characteristics it can change are: TCP initial sequence number, TCP window, TCP options and complete answers to some UDP and TCP packets

Although IP Personality can be considered a potent solution against nmap fingerprinting, it is by no means a definitive solution against fingerprinting in general. From a theoretical perspective, the methods employed by this project cannot scale as research into the field of fingerprinting advances. The project relies heavily on signatures which are not enough to generalize beyond the set of currently known probes.

In addition to the theoretical issues, a number of practical issues were also discovered. Because of the fact that IP Personality needs direct access to the network packet processing pipeline, operation of the kernel module requires extensive modification of the Linux source code. As a result, the only sensible method to distribute this software is either to merge it into the official Linux source tree, or provide a separate code patch.

The patching method was adopted by the developers and each patch file needs to be applied to the exact version of the source code that it targets. Considering that Linux is an active project which is being heavily developed, maintaining a set of patches for multiple versions of the source code is a daunting task. At the time of writing, the latest version of Linux supported was 2.4.18 which is more than 10 years old and thus obsolete.

4.2.2 IP Morph and traffic normalization

IP Morph [PVH10] is a project that, same as IP Personality, aims to hide the exact details of the operating system from active fingerprinting tools. Unlike IP Personality though, IP Morph does not rely on signatures of probes, but instead makes use of a technique called packet normalization.

The aim is to create an intermediary component through which all network traffic passes, analogous to a network gateway. Whenever a device transmits network traffic, IP Morph substitutes certain values inside packet headers with preset values. The preset values are used consistently for multiple types of operating systems, thus the term packet normalization.

The component through which network traffic passes also makes use of auxiliary timers in order to protect against detection methods based on TCP retransmission delays. Since probe responses

contain hardly any information specific to the exact operating system, it becomes almost impossible to accurately perform fingerprinting. An additional benefit of using IP Morph is protection against passive fingerprinting tools. Since packet are normalized in a continuous manner, the opportunistic approach does not hold.

Unlike IP Personality, the IP Morph system does not perform any detection of probes. Instead, all traffic is normalized, regardless of the structure of incoming packets. With regards to prevention, IP Morph is not a definitive solution against operating system probing. The scheme fails to protect against probing attempts based on control protocols which cannot be modified without negative consequences. As a result, certain protocols fall out of the scope of this project, one example being the multicast listener discovery (MLD) protocol. Thus, the probe discussed in chapter 3.1.2 can bypass the prevention mechanism.

Additional problems posed by traffic normalization are also discussed in [HPK01]. Normalization of packet headers do not work against fingerprinting attacks which extract information from TCP/IP stack behavior (i.e. completely different responses to the same probe). In order to solve this issue, the component that performs traffic normalization would have to keep track of the exact state in which each device is. This is an impractical requirement considering the lack of implementation details for non-open operating systems.

4.2.3 Snort, Bro and other anomaly detection systems

Snort is an network intrusion detection system that makes use of static rules which are applied against incoming network traffic. Rules for each type of probe need to be available in order to detect fingerprinting attempts. There are a set of rules for the nmap probes which have a similar effect as IP Personality [Lyo].

Bro, another intrusion detection system, follows a different design philosophy than Snort. Instead of using static rules, it makes use of its own scripting language to define routines for analyzing packet streams. Although more extensible than Snort, Bro does not currently offer any solution against nmap fingerprinting.

General purpose anomaly detection systems tend to focus on detecting multiple types of intrusions. Over the years, both commercial and open-source projects had to evolve to meet an ever-growing list of attack vectors [jmc; Cis05] such as: worms, trojans, protocol attacks at different OSI layers (e.g. ARP, IP, TCP) and application layer attacks (e.g. buffer overflows, SQL injections). As a result, defeating against fingerprinting alone is not a top priority for these projects and tradeoffs are usually part of their design decisions.

4.3 Intrusion detection methods

4.3.1 High level classification

The taxonomy related to network intrusion detection is wide and methods vary according to the underlying principles. Based on state of the art literature, an attempt is made to classify different schemes and extract the relevant methodologies to be considered when using anomaly detection methods.

Most detection schemes can be placed into one of three main categories [BSW02; SM07; MA12]. These are signature based, anomaly based and stateful protocol analysis oriented. Furthermore schemes may be combined in order to achieve better results by leveraging advantages from multiple approaches.

Signature based

Signature based intrusion detection solutions focus on defining malicious activities and building a knowledge base to be later used in the detection process [SM07]. These are the simplest detection methods since they only compare a single unit of activity to a list of signatures. Signatures can be thought of as patterns of characteristics which correspond to known threats. Such methods are

sometimes also referred to as expert systems in literature. Examples specified previously include IP Personality and Snort.

By making use of fine tuned data and a large enough set of signatures, these methods can achieve high accuracy rates by detecting even the most subtle intrusion attempt. In addition, an extra advantage is the minimal number of false positives resulting from the fact that signatures for malicious behavior can be defined with high precision.

The major drawbacks of these systems result from the inability to detect actions for which no signature is available [SM07; Lüs08]. This means that even small deviations of known attacks can result in successful intrusions. Furthermore, signature based detection technologies have little or no understanding of the underlying events which need to be distinguished. They also lack the ability to remember previous events when processing the current one. These limitations prevent signature based methods from detecting attacks that that span over multiple steps if none of the steps contains a clear indication of an attack. A final drawback is the fact that expert knowledge is expensive and the acquisition process may take a long period of time.

Anomaly based

A different category of intrusion detection schemes consists of anomaly based methods. These employ statistical techniques in order to differentiate between legitimate and malicious behavior [ZYG09; BBK14; Gar+09]. The approaches rely on the basic concept that anomalous activity is indicative of an attempted attack.

In order to develop an anomaly detection system, a baseline normality model first needs to be established. This model represents normal behavior against which events are compared. The system analyzes each event and classifies it as legitimate or not, depending on how much it differs from the normality model. A comprehensive set of characteristics (referred to as features) is defined to differentiate anomalies from normal system events [MA12].

The normality model is defined from a collection of event samples, referred to as the training dataset. The better the quality of the dataset, the better the model is defined. Unlike signature based systems, expert knowledge is not needed. While some systems require the training data to be available before before the live detection stage, others can define normality from training data acquired during execution [CMO11].

Anomalies can be classified into three different categories: point anomalies, contextual anomalies and collective anomalies [CBK09; BMS14]. Point anomalies result from individual events or data instances, when compared to the normality model. Contextual anomalies also result from individual events, but are only valid when certain conditions are met [Son+07]. The idea of integrating context information into the detection process is aimed towards decreasing the number of false positives. Event characteristics are treated differently based on circumstantial factors. The context is defined from domain knowledge as well as the structure of the data and is part of the problem argumentation. Each data instance requires both contextual attributes (i.e. to determine the context in which an event occurs) and behavioral attributes (i.e. metrics which define the particular event). Collective anomalies do not result from single data instances, but from several observations taken over a predefined period of time.

The main advantage of anomaly detection methods stems from their ability to detect larger sets of anomalies, including deviations from known attacks. It is possible to identify malicious events even if they are not present in the initial data used for defining the normality model [Gar+09; CBK09].

The disadvantage of these methods are their high computational requirements [BBK14]. As a result, it is harder to process events which have a high frequency such as incoming network packets on a high speed connection [GHK14]. Moreover, an anomaly based system may fail to detect well known attacks if they do not differ significantly from what the system establishes to be normal behavior. A final remark is the higher the false negative rate, resulting in more alerts being triggered for events which are not malicious in nature [CBK09].

Stateful protocol analysis

Stateful protocol analysis methods are designed by creating profiles for legitimate behavior of how protocols should execute [SM07; MA12]. This approach is similar to signature based methods considering that events are compared to a set of rules. The difference is that protocol analysis methods have a deep understanding of how the protocol is supposed to be executed. To achieve this, such methods need to keep track of previous events when processing the current one. These methods are not meant to be used on their own, but are generally incorporated into mixed solutions, working alongside signature or anomaly based methods in order to increase the overall performance.

The main disadvantage of such approaches stems from the difficulty of formally describing protocols. This may lead to circumvention by intrusion attempts which stay within the limits of acceptable behavior.

4.3.2 Classification of anomaly based methods

Considering the disadvantages of using static rules and the new anomalies discovered while working on nmap (described in chapter 3.1), the decision was made to focus on anomaly based intrusion detection methods. For this reason, the scope was centered around existing work done in the field of detecting anomalies in network traffic.

In this section we present a sub-classification of anomaly detection methods and look at the advantages and disadvantages of proposed approaches. Although methods vary both conceptually and in the level of complexity, there are mainly three classes, namely: statistical, knowledge and machine learning based.

Statistical based

Anomaly detection schemes based on statistical methods define the baseline normality model as a probability density function which calculates a score denoting the likelihood that a new and unseen example (i.e. not part of the training dataset) is generated by a stochastic process [Ans60; Gar+09]. Events with a low score are considered to be anomalies.

Each of the examples in the initial training dataset is considered to be non-anomalous. This is a disadvantage of such methods, since obtaining datasets without any attacks can be expensive and time consuming. As a result, implementations of such systems may have a longer bootstrapping period. Furthermore, if the training set contains hidden attacks or anomalies, the resulting normality model may be unable to detect certain attacks.

Applied to computer networks, these methods work by first generating the model from network traffic data captured for the purpose of training. For the best results, the data should not contain any traces of intrusions or anomalies, since the resulting stochastic model is considered to represent the normal behavior of agents in the network. This features are based on characteristics such as the number of packets per connection, number of flows between two given agents or structure of packets. The anomaly detection system then tries to measure the likelihood that new packets are generated from the same applications and protocols.

In [Ye+02], the authors developed a multivariate model which uses 284 different variables obtained from the Basic Security Module of a Sun SPARC 10 host. Anomalies are detected by observing events in a time-window of pre-configured length. The variables are given different weights by using an exponentially weighted moving average technique, meaning that newer observations have a greater impact when deciding whether an anomaly has occurred or not. The scheme also makes use of Hotelling's T^2 test which quantifies the deviation of an event from the in-control population used for training. The results show that distinguishing anomalies from the rest of the events is difficult. As a result accurate detection of anomalies brings a 2% false positive alarm rate, a number that may be considered too large if the total number of events is high. By tuning the alarm threshold, a 0% false positive rate can be achieved, but this in turn reduces the accuracy of detecting anomalies to 16%.

Applied to the problem of detecting network port scans (i.e. probing which searches for ports which accept connections), in [Kim+04] the authors consider a detection method based on two

dynamic chi-square tests. The variables used in their model, depending on the IP packet payload, are: direction, status of TCP 3-way handshake, termination of a connection, duration, flags and ports. An anomaly is collective [CBK09] and is represented as a sequence of observed packets which may be used in port scanning. The results of this scheme are 10% and 100% detection accuracy for 9 and 10 scanned ports respectively. Regarding false positives, the values are between a minimum of 5% and maximum of 20%, again depending on the number of scanned ports.

A different scheme is proposed in [KMS12] that differentiates between normal and attack network events using a Hidden Naïve Bayes classifier. The method assumes that all features of a given data point are independent of each other. As a result, the accuracy drops if there are complex dependencies between events. This is the case for the KDD99 network intrusion dataset, which the authors used for testing. An additional disadvantage of the proposed scheme comes from the fact that Naïve Bayes does not scale well for large datasets [Koh96]. Depending on the 1) feature selection algorithms for pruning irrelevant features, 2) discretization methods for converting feature values from continuous to categorical and 3) classifier algorithms, the overall performance was between 92% and 93% detection rate. Considering that operating system fingerprinting can be successful with as little as one probe, this detection rate is not practical for our purposes.

Using hidden Markov models, the authors of [TPS13] apply the problem of quickest change point detection in order to infer when anomalies occur. This method works by observing a set of random observations which follow a known probability density function and then detecting when this function has changed. The difficulty lies in detecting the change after a minimal number of observations. This scheme was tested on a trace which contained a denial of service attack. The initial distribution, prior to the attack follows a Gaussian process, yet after the attack starts, this distribution is not similar to Gaussian anymore. The results are 0.007% false positive rate and the detection delay is 0.14 seconds. Although this method looks promising, it requires a large number of anomalous events to be present in a short period of time. This is in contrast to fingerprinting methods which are possible with small number of probes.

Knowledge based

Knowledge based methods attempt to detect anomalous events by making use of rules derived either from other sources of information (i.e. protocol specifications, vulnerabilities reports etc.) or by applying logical constraints specific to the environment which require protection [BBK14].

A model for storing the data in the knowledge base is presented in [Mor+12], where the authors propose indexing anomalies based on three properties: the means, the consequences and the targets. The knowledge base is extended either with already structured data, or by analyzing unstructured text which contains relevant information [Mul+11]. A "reasoner" module then uses the knowledge base as well as information from incoming data streams to detect anomalies.

A study related to anomaly detection, but focusing on anomaly extraction (i.e. the problem of separating benign events from anomalous ones) is presented in [Bra+12]. Data for the knowledge base is received from multiple anomaly detection systems deployed on a network. The advantage of adopting such a system on top of already deployed anomaly detection infrastructure is the reduction of false positives. The proposed method makes use of association schemes, based on the assumption that anomalies are present in larger numbers of points with equal features.

The feature set is rather limited, as this method focuses on analyzing traffic flows instead of individual packets. The feature set consists of seven variables: source and destination IP addresses, source and destination ports, the transmission protocol, number of packets per flow and the number of bytes per packet. The distributions of each feature are analyzed by creating histograms every 15 minutes. The Kullback-Leibler distance is then used to detect anomalies.

One major advantage of this approach is the fact that anomalies can be detected on a per-feature basis, which offers the possibility of giving precise feedback to an operator regarding the exact reason an alarm was triggered. The achieved results vary between 80% accuracy with 3% false positives and 100% accuracy with 5-8% false positives.

Machine learning based

This class of detection schemes makes use of machine learning, a family of algorithms commonly associated with computational learning and artificial intelligence. Several machine learning algorithms have been created for specific types of problems such as:

1. Regression algorithms which aim to predict a numerical value based on existing historical data. One example of algorithm in this category is linear regression.
2. Classification algorithms which try to identify to which class or category a new event belongs to. Most notable classification algorithms are logistic regression, neural networks and support vector machines.
3. Clustering algorithms aimed at grouping data points based on similarities. The k nearest neighbors algorithm is designed for problems which require clustering.

A second differentiating factor for machine learning algorithms is the type of training data. The data may be either labeled (i.e. for supervised algorithms) or unlabeled (i.e. for unsupervised algorithms). In a labeled dataset, the each example is given a label conveying a certain aspect which the algorithm is trying to learn. Labels are often obtained after analyzing the datasets by humans who make judgements about a given piece of unlabeled data. Thus, labeling can be seen as a data pre-processing step. Depending on the size and complexity of the dataset, labeling is often viewed as an expensive and time-consuming process.

For the purpose of anomaly detection, the most used algorithms are those from the classification and clustering categories. When applying a classification algorithm, two classes are defined and each new event is marked as either benign or anomalous. In the case of clustering algorithms, events from the training set are grouped together based on similarities and new events are marked as benign or anomalous depending on "distance" metric. The metric describes how much a new event differs from the cluster center.

A study on the applicability of machine learning techniques for the problem of anomaly detection is presented in [SP10]. In this work the authors argue that the task of detecting new and previously unseen anomalies is different than then applications of machine learning in other fields. Machine learning algorithms have a high degree of accuracy when applied to the problem of matching a new item with a precise description to something previously encountered. As a result, machine learning approaches require a comprehensive training set with carefully selected data.

When using classification algorithms, an adequate number of representative examples is needed for each class. This means that anomalies have to be well defined in order to be correctly classified. This is in contrast to datasets used for training anomaly detection systems which either assume all examples to be positive (non-malicious) or have a small subset of malicious examples, resulting in skewed classes. If clustering algorithms are used, then the dataset needs to contain as few anomalous examples as possible, similar to the case of statistical methods.

In order to achieve positive results when using a machine learning approach for anomaly detection, the authors in [SP10] argue that it is important to gather in-depth knowledge about the system that needs protection. This in turn leads to well defined features which have a greater impact towards discrimination between benign and anomalous events. It is thus more sensible to focus on the exact features than on the exact combination of machine learning techniques that are applied.

Applied to the problem of detecting port scans on mobile devices, [PDN14] presents a method based on a decision tree combined with a cascade correlation neural network. The features used in this scheme are extracted from TCP and IP headers and include bit flags, address and port information and frequencies of incoming and outgoing packets. The decision to use two detection mechanisms was based on the fact that the decision tree could detect the simpler scanning methods with high accuracy and little resource usage, but failed to detect the complex ones. The decision to use cascade correlation neural networks was based on their ability to retrain on new data without having to add the previous data in the training set.

Different from the supervised model of neural networks, the work in [BBK12] focuses on unsupervised anomaly detection. The detection scheme is based on a subspace clustering approach, where one example may be part of multiple clusters depending on which subspace is being analyzed. Relevant features are chosen based on their ability to form stable clusters that have as few points in common as possible. The results range from 66.2% to 99.9% detection rate and 0.0016% to 0.197% false positive rate.

4.3.3 Traffic aggregation and sampling

Instead of analyzing traffic on a per-packet basis, some of the proposed methods employ aggregation of network traffic data into flows [SSP08; JL14]. A flow is a logical combination of packets which travel in either direction between two or more hosts. The packets are combined on characteristics such as the IP addresses, the ports and the protocol types. Information from multiple packets is extracted, such as average packet size or duration of the flow. In addition, the authors in [HH05] present their study into different sampling techniques. These methods analyze only a portion of the packets for anomalous events.

In [SSP08] the work focuses on identifying anomalies in time-series generated from data gathered from independent packets and flows. Furthermore, the authors analyze the effects of using sampling. The results show that certain classes of attacks can be detected only by analyzing flow data, particularly port scans. On the other hand, denial of service attacks based on the UDP protocol cannot be traced without additional features, namely the amount of network data passing through a detection point.

Aggregation of traffic on the internet backbone is presented in [JL14]. The amount of traffic passing between internet autonomous systems is of magnitudes much greater than enterprise networks. Autonomous systems are large internet networks which perform under a single administrative authority. Multiple autonomous systems are connected to each other to form the global internet. The proposed detection scheme performs aggregation of traffic data from each neighboring autonomous system. Testing of this scheme showed accurate detection of three denial of service attacks. Additionally, it was possible to trace the attack back to the originating autonomous system. On the other hand, the scheme was unable to detect a user-to-root attack that installed and executed malicious files on a vulnerable host.

As can be seen, one characteristic of detection based on aggregated data is that anomalies are discovered within a certain time-frame, usually after then event has occurred. Furthermore, anomalies that are detected easily consist mainly of volumetric features which rely on the size of a certain characteristic. Although aggregation is helpful in certain situations such as forensic analysis, anomaly modeling or detection in high-speed networks, the proposed methods are not responsive enough to detect small anomalous probes in a timely manner.

4.4 Intrusion response

A recurring challenged in the field of anomaly detection is the general inability to convey results in a human friendly manner. This issue is commonly referred to as the "semantic gap" [SP10]. Most of the schemes are constructed to detect deviations from a normality model using a mathematical approach, but the designs rarely focus on offering relevant insight into what the results actually mean. Whenever an alarm is triggered, system maintainers are not given relevant information as to why an anomaly has occurred. When coupled with frequent false positives, it may lead to a state where alarms are not analyzed appropriately. As a result, the practical usability of the anomaly detection system decreases.

The two issues described above are tackled in [Cos+14] by employing two novel concepts: a white-box behavioral-based model and a feedback loop. The white-box model is based on defining profiles and detection rules which describe normal behavior and which are obvious from the detection context. Thus, whenever an alarm is raised, the system maintainer can understand the

underlying cause. This concept is coupled with the feedback loop which allows system maintainers to mark false positives as benign, thus decreasing the number of incorrect alarms in the future.

4.5 Discussion

The topic of active fingerprinting has been met with relevant developments and is more mature than what nmap implements. Whereas nmap sends the same 18 probes regardless of the host that is being targeted, one of the proposed methods discussed above is able to limit the number of required probes by taking intermediate results into consideration. Furthermore, there are proposed methods for probabilistic approaches which may limit the number of probes down to one.

These developments have a direct impact on detection. Detecting probing attempts requires a small resolution for analyzing the network traffic as individual packets have to be tested for (ab)normality.

Finding new yet to be discovered probes, although challenging, is not an impossible task, as described in chapter 3.1. Implementations of tools which aim to distort fingerprinting attempts detect probes by means of static signatures. As such, it is possible to circumvent these tools. An additional aspect which makes detection of probing attempts even more ambitious, is the ability to generalize beyond static signatures and detect undiscovered probes.

For the stated reasons, the decision to follow an anomaly detection based approach was made. Anomaly detection is a wide term that applies to a multitude of different schemes and algorithms. While reading state of the art literature, one may get insight into some of these schemes, but it is difficult to accurately evaluate each and every one of them. It is a known problem in the anomaly detection and machine learning communities that a newcomer may become overwhelmed by the sheer number of different applications and possible options [Dom12]. Furthermore, based on the contents of the white papers alone, it may not always be enough to fully grasp the effectiveness of a proposed scheme. Important information is sometimes left out, such as the set of features a scheme is based on. Lastly, choosing to apply an existing detection scheme to a different problem than what it was created for may not be the most optimal approach. Schemes are developed for well-defined scenarios and not all of them can be used to detect anomalies encoded in network packet headers.

Starting with a proposed scheme and adapting it for detection of probes which target the TCP/IP stack was not a suitable option. It was decided to follow a custom-tailored approach and build from the ground up a scheme that would fit the problem at hand.

In order to have positive results with anomaly detection, the problem definition and scope need to be narrowed. In combination with a relevant set of features, this leads a well-defined normality model. By following this methodology, the detection scheme can obtain high detection rates while keeping the false positive rates low.

Whereas some probes try to push the TCP/IP stack into undefined behavior, others try to discover implementation issues resulting from not respecting the latest versions of RFC documents. Thus, the detection scheme may be split into two parts, one based on stateful protocol analysis that performs checks against existing RFC documents and another which uses a baseline normality model that can generalize beyond static checks. This differentiation enables further narrowing of the problem description for the second detection stage. Additionally, the use of protocol analysis has the added benefits that 1) it has a higher detection and lower false positive rates and 2) it can offer additional context as to why an alarm is triggered.

The second stage requires further consideration for which type of scheme to employ. The choice comes down to one of three possibilities: 1) statistical or machine learning, either 2) supervised or 3) unsupervised.

Supervised methods for the purpose of anomaly detection have few advantages over the other two choices. When unknown attacks exist, not part of the training set, their performance and ability to generalize is affected [Las+05; SP10]. Most of the supervised machine learning algorithms try to classify between benign and anomalous examples. This faces greater difficulty than general classification problems since training sets often contain much more benign examples than anom-

alous ones (i.e. skewed classes). In addition, greater numbers of related anomalous examples are required for the models to get a sense how they "look like". During the detection phase, anomalies are identified only if they are similar to exampled used for training. Lastly, at the packet detection level that is of interest, it is sometimes impossible to unambiguously label training data correctly [Las+05]. Pursuing a supervised learning scheme is thus not an optimal solution.

Unsupervised methods have the advantage that data does not need to be labeled. This makes the process of data acquisition easier than in the case of supervised learning algorithms. On the other hand, the disadvantage of unsupervised methods lies with the fact that these are generally complex and more parameter dependent, meaning that more work is required for fine tuning to a specific environment [BBK14].

Statistical methods have the similar advantages to unsupervised methods, considering the fact that they do not need labeled data. Additionally, they can show good results with smaller training sets. Lastly, these schemes can generalize well to many different types of anomalies [Gar+09]. Even so, their main disadvantages are that the training data needs to contain only benign examples and, depending on which probability model is chosen, parameterization may also become an issue.

Choosing between statistical and unsupervised machine learning schemes is not straightforward as both have rather similar advantages and disadvantages. One additional aspect to consider, outside the scope of which scheme to use, is the choice of features. Choosing appropriate features can have a much greater impact than fine tuning a complex algorithm with bad features [Dom12]. Feature engineering is a domain specific task and relevant information needs to be "encoded" into the set of features on which the scheme is ultimately based on.

With this last condition in mind, the choice was made to use a simpler statistical method with a relevant set of features which may yield better results. Not only is it easier to design, implement and test, but also the attention is placed on how to best approach the problem and define anomalous events (i.e. via relevant features).

4.6 Conclusion

This chapter discusses the current state of the art on TCP/IP stack fingerprinting and anomaly detection methods. These two topics are related by their common attention to probes used in fingerprinting, which are equivalent to anomalies in network traffic.

There is extensive research into improving fingerprinting by lowering the required number of probes. Academic literature proposes new techniques that can bring the total number of probes down to one. This can have a direct impact by making detection more difficult.

Current tools and software implementations of projects that aim to disrupt fingerprinting are mostly based on static signatures of known probes. These techniques are not enough to detect all possible fingerprinting attempts, given that new probes may be discovered for which no signature is available.

An analysis of proposed methods for anomaly detection was conducted. Schemes usually follow one of three approaches: static signature checks, anomaly detection or stateful protocol analysis. After considering the type of anomalous events which need to be detected (i.e. TCP/IP stack probes), appropriate schemes were chosen for designing a custom-tailored approach.

Detection is performed in two stages, the first based on stateful protocol analysis and the second on statistical analysis. The use of two detection stages poses a number of advantages. The stateful protocol analysis stage has a low number of false positives, since it is meant to follow the existing RFC documents. The second stage allows the scheme to generalize beyond static checks and detect unknown probes.

Chapter 5

Anomaly detection system design

This chapter describes the conceptual design of the proposed anomaly detection scheme. It starts by reiterating the relevant findings from previous chapters which guided the decision making process. The two stages are presented afterwards, the first based on stateful protocol analysis and the second based on statistical analysis of benign network traffic. Next, anomalous characteristics to be detected by the scheme are defined and tied to either of the two stages. In the case of the second detection stage, each characteristic is interpreted as a feature to be used when defining the baseline normality model for network traffic. Lastly a discussion is done regarding the most optimal network deployment strategies for such a detection system.

5.1 Essential findings

The current state-of-the-art on anomaly detection applied to network traffic was presented in chapter 4.3. The advantages and disadvantages of several directions were discussed and one of the first major differentiating factors was either to pursue a signature based or an anomaly based method. Whereas the former has it's main disadvantage in the fact that it can only detect known anomalies for which signatures are available, the latter is able to generalize beyond what is currently known.

As part of our stated goal to improve the accuracy and reliability of nmap, a number of additional probes were discovered. While some of these probes were malformed packets of existing protocols (i.e. point anomalies), others were structurally valid. What turned the latter category into anomalies were contextual characteristics (e.g. MLD queries sent by non-router hosts). It was thus possible to discover previously unknown probes for which no signature was available. As a result, an approach based on anomaly detection using statistical analysis was considered.

One of the major disadvantages of anomaly detection is the generally higher number of false positives (i.e. benign events treated as anomalies). In order to reduce this concern, it was decided to minimize the subset of packets on which to apply anomaly detection. The characteristics which could only result from probing would be detected by a different approach, based on stateful protocol analysis.

In the following chapters, both methods will be described together with the characteristics that are analyzed at by each stage.

5.2 Static protocol analysis stage

5.2.1 Design

The static protocol analysis is the first detection stage of the proposed approach. Applied to the problem of detecting anomalous packets, this stage aims to enforce adherence to the existing standards and specifications relating to network traffic. This stage focuses primarily on point

anomalies which result from malformed packets and contextual anomalies based on the validity of certain messages when accounting for the identity of the sender. To some degree, some of these checks can be performed by high-end filtering devices such as industrial network switches and routers. In order to achieve its goal, this stage inspects the headers of packets that come from parties initiating contact (e.g. begin TCP handshake, send ICMPv6 echo requests, send MLD queries). The packets coming from responding parties are considered benign and are not fed to the analysis process. Even so, they are inspected in order to maintain correct state such as termination of a TCP connection signaled by a TCP packet with the FIN flag set.

Relevant rules are extracted from RFC requirements tagged with the "MUST" and "SHOULD" keywords and focus on valid message formats and appropriate values for fields in packet headers. Although standardization documents often leave room for interpretation regarding how correct behavior should be defined, some hard requirements do exist in order to ensure compatibility between different vendors and implementations.

5.2.2 Traffic characteristics

Deprecated and insecure IPv6 features

As a result of the increased deployment of IPv6 enabled devices in the past years, more and more attention and research has been put into the technology. Consequently, the IPv6 standard has been refined and in the process, certain features have been deprecated for security reasons. Since not all vendors implement the latest version of RFC documents, discrimination among multiple operating systems becomes possible based on newly defined behavior.

At a minimum, the following IPv6 RFC documents should be enforced by a protocol analysis component:

1. *Deprecating Site Local Addresses* [RFC3879] - The concept of a "site" is not clearly defined and, in addition, these addresses add complexity at the expense of little benefit. The detection system should trigger an alarm whenever packets with destination addresses in the range FEC0::/10 are encountered.
2. *Deprecation of Type 0 Routing Headers in IPv6* [RFC5095] - This header can be used in distributed denial-of-service attacks and must not be supported any longer. Alarms should be triggered when packets with such a header are encountered.
3. *Handling of Overlapping IPv6 Fragments* [RFC5722] - The IPv6 fragment header has a *fragment offset* field which specifies the offset of the data relative to the start of the original packet. IP fragmentation attacks can be launched by using incorrect offsets, allowing the bypass of firewalls and intrusion detection systems which perform deep packet inspection. The algorithm in figure 5.1 should be implemented. This would be in sync with the latest developments regarding treatment of this feature [1].
4. *Deprecating the Generation of IPv6 Atomic Fragments* [RFCDR1] - The specified maximum MTU in *"packet too big"* messages should be at least 1280 bytes to prevent the behavior from chapter 3.1.1.

```
1   amount_data = 0
2   for each new fragment
3       if fragment offset < amount_data then
4           raise alarm
5       end if
6       amount_data += fragment_offset
7   end if
```

Figure 5.1: pseudo-code for enforcing RFC5722.

[1] ipv6hackers mailing list - lists.si6networks.com/pipermail/ipv6hackers/2012-February/000357.html

Order of IPv6 extension headers

The exact order required for IPv6 extension headers is specified in RFC2460. In order to prevent fingerprinting attempts based on different error messages as explained in chapter 3.1.3. The protocol analysis system should verify that the IPv6 extension headers are placed in the order: hop-by-hop options, destination options and fragment headers.

ICMPv6 state

The component executing the first stage of detection should also keep state of outgoing and incoming ICMPv6 requests and responses so that these can be linked. By linking such messages to one another, the detection system may discover malicious attempts of changing the configuration of networked devices (e.g. inform hosts of fake IPv6 routers on the network). The informational protocols built on top of ICMPv6 which work in a request-response manner have the "type" field set to a value of at least 128 [RFC4443].

Additionally, the detection system should keep track of which roles a host has on the network. Certain ICMPv6 messages can only be sent by hosts with certain roles, such as *Router Advertisements* or *MLD Queries*.

One external solution which may be adopted against such malicious behavior is the *SEcure Neighbor Discovery Protocol* (SEND). This extension of the *Neighbor Discovery Protocol* (NDP) enables authentication of NDP messages and certification of hosts together with their roles on the network [RFC3971].

TCP connection state

As described previously, relevant information for discriminating between operating systems can be extracted from responses to certain TCP packets [2]. Some of these packets may be considered anomalous as they are sent to closed ports. Moreover the TCP protocol makes use of bit flags placed in the header for different actions, such as initiating or terminating a connection. The existing documentation specifies exactly how these flags can be combined for different purposes. The protocol analysis component should be able to detect when invalid combinations of flags are present in a header.

External solutions exist with regards to this problem as well. One that is straightforward and easier to implement is the use of a firewall configured to drop packets which are either invalid or are sent to closed ports.

Length and checksum fields

Certain protocols have fields in the header for calculating checksums values in order to prevent packets from being modified accidentally during transmission. Both the TCP and ICMPv6 protocols have such fields and they are mandatory. The UDP protocol has a checksum field that is optional when using IPv4, but also mandatory when using IPv6.

5.3 Statistical analysis stage

5.3.1 Problem generalization

Anomaly detection using statistical analysis can be regarded as an unsupervised learning algorithm since data does not have to be explicitly labeled. Instead all m training examples are considered benign. For this reason, the training set should be as clean of anomalies as possible.

A set of n features is defined from characteristics which are influenced by anomalous events. Features are combined together in order to define the baseline normality level, referred to as the model. From a computational point of view, the model is a probability density function (PDF, in

[2] nmap TCP probes - nmap.org/book/osdetect-methods.html#osdetect-probes-t

formulas denoted $\mathcal{P}$), whose value represents the likelihood that a new observation x is generated by the same stochastic process which also generated the examples from the training set. In our case, the stochastic process is the network traffic. A threshold value ε is chosen such that if the result of the aforementioned function is less, the observation is flagged as an anomaly. These elements are summarized in figure 5.2.

Training set of m examples $\{x^1, x^2, x^3, \ldots, x^m\}$
$\mathcal{P}$ model trained from training set
x_{test} new observation
if $\mathcal{P}(x_{test}) < \varepsilon$ then flag anomaly
else if $\mathcal{P}(x_{test}) \geq \varepsilon$ then treat as benign

Figure 5.2: elements of the anomaly detection scheme

5.3.2 Model

The anomaly detection scheme designed against TCP/IP stack probes is based on the Gaussian distribution, commonly denoted as $\mathcal{N}(\mu, \sigma)$. The plot of the PDF takes the form of a bell-shaped curve as shown in figure 5.3. The area under the curve is constant and always equal to 1.

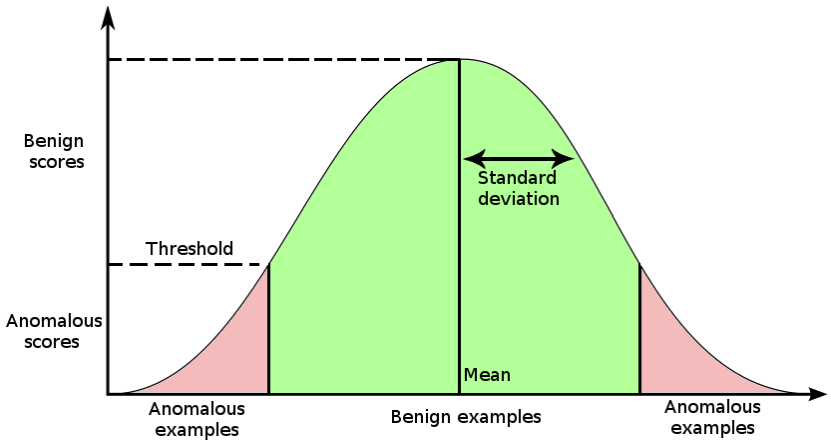

Figure 5.3: generic plot of Gaussian probability density function.

This distribution has two parameters, called the mean μ and the standard deviation σ. The mean specifies the point where examples converge, while the standard deviation represents the amount of variation in the training set. Benign examples are considered to be more numerous and are thus closer to the mean. On the other hand, anomalies tend to occur less frequently and are further away from the mean.

Applied to the plot, μ sets the location of the curve apex, while σ controls the height and the slope of the curve. Since the area is constant, a curve with a smaller σ is more steep and thus taller than a curve with a greater σ. These parameters are calculated with the formulas

$$\mu_i = \frac{\sum_{j=1}^{m} x_i^j}{m} \qquad \sigma_i = \sqrt{\frac{\sum_{j=1}^{m} (x_i^j - \mu_i)^2}{m}}$$

$$i \in \{1, 2, \ldots, n\}$$

The anomaly detection scheme uses one pair of parameters μ and σ for each of the n features that compose the model. As a result, the PDF is calculated multiple times. The single variant is applied to one feature at a time and has the formula

$$\mathcal{P}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

$$x,\ \mu,\ \sigma \in \mathbb{R}$$

One option of combining several features (i.e. pairs of μ and σ) is by applying the above formula multiple times and multiplying the results. A better appraoch is to use the multivariate Gaussian PDF which can be applied to multiple features at a time. The formula for the multivariate PDF is

$$\mathcal{P}(x) = \frac{1}{\sqrt{|\Sigma|}\sqrt{(2\pi)^n}} e^{-\frac{1}{2}(x-\mu)'\Sigma^{-1}(x-\mu)}$$

$$x,\ \mu \in \mathbb{R}^n \qquad \Sigma \in \mathbb{R}^{n*n}$$

where Σ is the covariance matrix calculated over the set of n features and $|\Sigma|$ is the matrix determinant.

The advantage of this method lies in its ability to automatically capture correlations between the features. Figure 5.4 is a good representation of how the two combinations methods behave when applied to two different features.

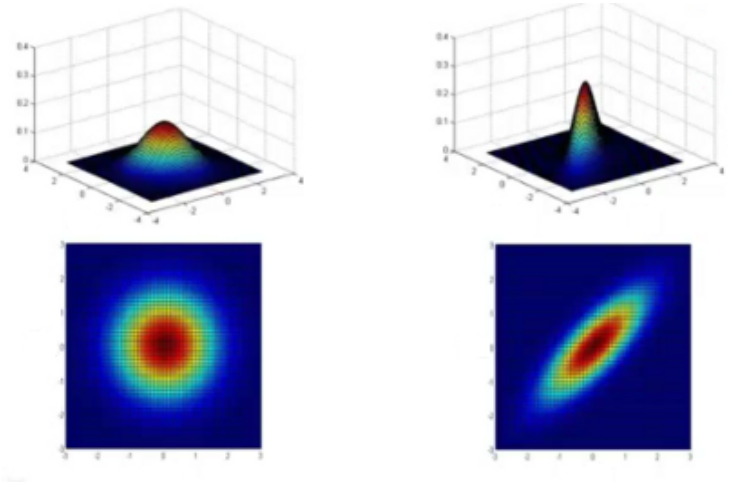

Figure 5.4: plots of univariate (left) and multivariate (right) Gaussian PDF.

For example, from a UDP datagram transmitted over IPv6, two features can be extracted which have a direct connection. The IPv6 payload length is a 16 bit field that specifies the amount of data following the the IPv6 header in octets. The UDP length field specifies the number of octets comprising the user datagram including the header. An increase in the IPv6 field results in a direct increase in the UDP field. While the the univariate PDF treats these two values in isolation, the multivariate PDF checks the combination for anomalous behavior.

5.3.3 Feature engineering

Feature set

Features are extracted from packet headers at different layers (e.g. network, transport). The full list of features that the anomaly detection system makes use of is presented in table 5.1. This set of features was chosen by analyzing which header fields may be indicative of probing attempts.

One of the initial problems that was faced during the design of the scheme was the fact that the Gaussian distribution requires numeric values. While features extracted from single header fields are directly numeric and allow for direct computation of μ and σ, other features require additional

Feature	Data sources	Utility
IPv6 headers *	frequencies of header combinations	Indicative of 1) TCP/IP stack behavior and 2) protocols used on the network.
IPv6 payload length	payload lengths vector	Usual amount of data carried per packet.
IPv6 hop-by-hop options *	frequencies of option combinations	Normal behavior for options per type of header.
IPv6 destination options *		
ICMPv6 type and code *	frequencies of value combinations	Indicative of normal usage for protocols built on top of ICMPv6.
ICMPv6 packet too big MTU	MTU values vector	Regular MTU for paths in the network.
UDP length	datagram lengths vector	Usual amount of data in UDP datagrams (correlated with IPv6 payload length)
TCP SYN optios *	frequencies of option combinations	Indicative of host capabilities; options only appear at certain stages of the TCP handshake.
TCP SYN-ACK options *		
TCP ACK options *		
TCP Non-SA options *		
TCP flags *	frequencies of flag combinations	Flags have different uses with regards to connection state and packet processing options.
TCP window size	window sizes vector	Indicative of receive window size; devices with different capabilities vary with regards to how long the window is.
TCP data offset	data offsets vector	Indicative of amount of data in the TCP header.
TCP maximum segment size	MSS values vector	The maximum amount of data that may be transported in a TCP packet.
Nr. simultaneous flows	frequency of nr. of simultaneous flows	TCP probes have better results if they have the SYN flag set and initiate a connection.

Table 5.1: features used by the anomaly detection component.

pre-processing (i.e. those marked with * in table 5.1). The solution to this problem is to label each feature value and calculate μ and σ based on the label occurrence frequency.

To serve as an example, in the case of IPv6 extension headers the combination of fragment and hop-by-hop options headers receive a different label than the combination of fragment and destination options headers. The same concept is applied to combinations of options in the *hop-by-hop* and *destination options* headers, combinations of ICMPv6 types and codes, combinations of TCP options and flags. The order of the elements in each combination is not relevant and thus multiple combinations have the same label. The algorithm for this initial pre-processing step is explained in figure 5.5.

After pre-processing each example in the training set, an array of frequency scores is obtained. These scores are numeric and can thus be used with the Gaussian PDF. The most optimal set of parameters μ and σ for the PDF need to ensure that the label with the highest frequency score obtains the highest result. Additionally, lower frequency scores must have proportionally smaller PDF results.

```
1     var frequency   # array of frequency scores
2     for each example in the training set
3         extract feature value from example
4         apply label to value
5         frequency[label] += 1
6     end for
```

Figure 5.5: pseudo-code for frequency calculation during pre-processing stage.

The highest result a PDF can achieve is at the mean, as can be seen from figure 5.3. For this reason, we sort the bins from the highest to lowest and when calculating σ we duplicate all frequency scores symmetrically of the mean. The algorithm which calculates the mean μ and standard deviation σ is described in figure 5.6.

```
1     var frequency   # same array of frequency scores
2     var tr_frequency = transform(frequency)
3
4     sort tr_frequency by score from highest to lowest
5
6     var mean = 0        # mean is the index of the highest label
7     var std = 0
8     var denom = 0
9
10    for idx in range 0 to len(tr_frequency) - 1
11        std += tr_frequency[idx] * pow( idx, 2 )
12        denom += tr_frequency[idx]
13        if idx != 0
14            std += tr_frequency[idx] * pow( -idx, 2 )
15            denom += tr_frequency[idx]
16        end if
17    end for
18    std = sqrt( std / denom )
19
20    return (mean, std)
```

Figure 5.6: pseudo-code for parameter calculation during training stage.

The array is sorted from the most to the least frequent label. Then a transformation function is applied, explained in more detail in chapter 5.3.3. The plot of the IPv6 extensions headers frequency scores is at this step is shown in figure 5.7.

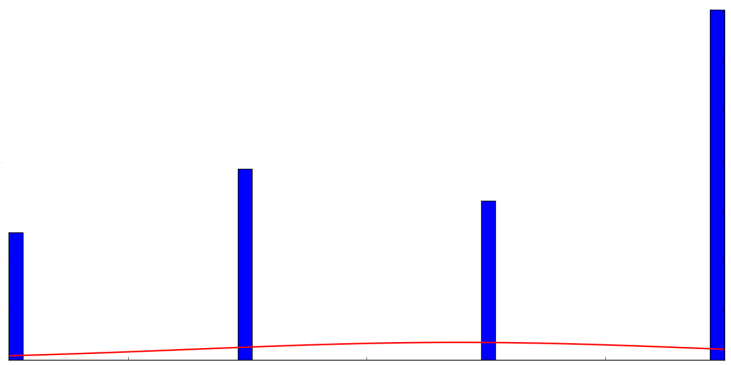

Figure 5.7: step one of the training phase for the IPv6 headers feature.

Next the sorting of the labels is performed in reverse order, based on the frequency score. The plot of this next step is shown in figure 5.8.

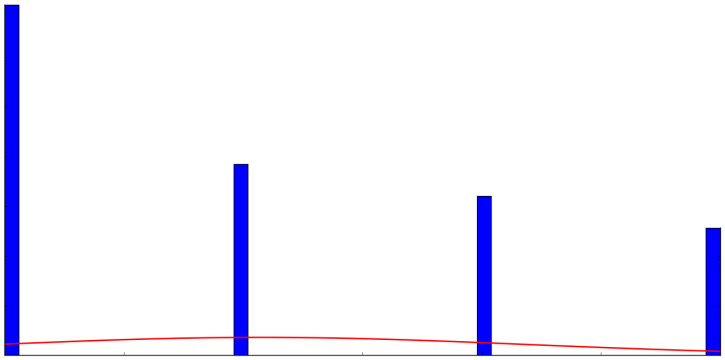

Figure 5.8: step two of the training phase for the IPv6 headers feature.

The final step is the mirroring of the scores in a symmetrical manner against the highest score. In the pseudo-code, this is achieved in lines 10 through 17. The plot of the data is shown in figure 5.9.

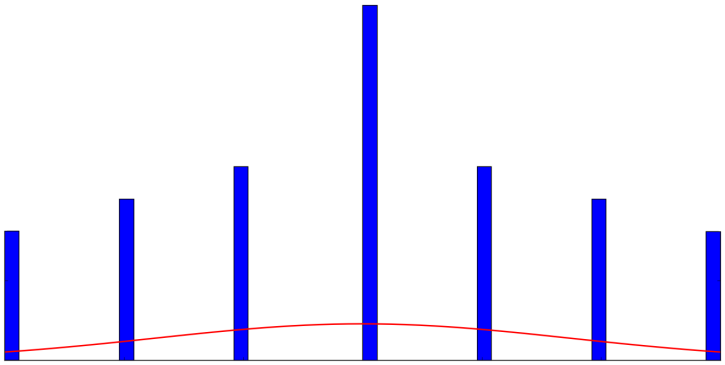

Figure 5.9: step three of the training phase for the IPv6 headers feature.

As can be seen in figure 5.9, the data follows the Gaussian distribution and the bell shaped red curve is visible. Because of the way in which the standard deviation was calculated, the results of the PDF are proportional to the frequency scores denoted by the heights of each bin.

During the detection stage, for a given observation, the index of its label in the sorted frequency score list is used at the input value for the PDF function. If an observation is encountered for which no label was created in the training phase, it is considered to be an anomaly. The PDF calculation is skipped and a value of 0 is used instead. This approach ensures a value below the below the threshold ε.

Data transformations

In order for the scheme to have positive results, the data extracted for each feature should be Gaussian distributed. If this does not occur and the bell-shaped curve is not visible when plotting the data, then the parameters μ and σ are not optimal for the purpose of anomaly detection. In order to achieve an ideal representation of the data, transformation functions are used which have the following two goals: 1) ensure the mean is in the center of the curve and 2) push the possibly anomalous examples further from the mean. In this form, the value for the threshold ε can be chosen such that examples are marked more appropriately and the number of false positives and false negatives kept to a minimum.

The transformation functions are applied to numerical feature values, first time prior to commencing the training stage and a second time during the detection stage, prior to calculating the PDF.

The functions have to be defined on a per case basis after inspection and analysis of the training data. This is because the baseline normality model may differ from one network to another, depending on the type of traffic that passes through. The training data may have subtle differences which would make transformation functions inapplicable in different networks.

While designing the scheme, transformation functions were chosen for each of the features listen in table 5.1. In principle, there are two types of functions: those for features which were based on single header fields (i.e. directly numerical) and for features which were based on frequency scores.

Examples of transformations functions and their results, based on testing data, are presented in the next paragraphs. They should only be taken as guidance for implementing functions specific for the network environment that is to be protected.

The IPv6 payload length is a 16 bit field that specifies the amount of data following the the IPv6 header in octets. It is thus directly numerical and can be used with the Gaussian distribution. A plot of the raw values for this feature can be seen in figure 5.10.

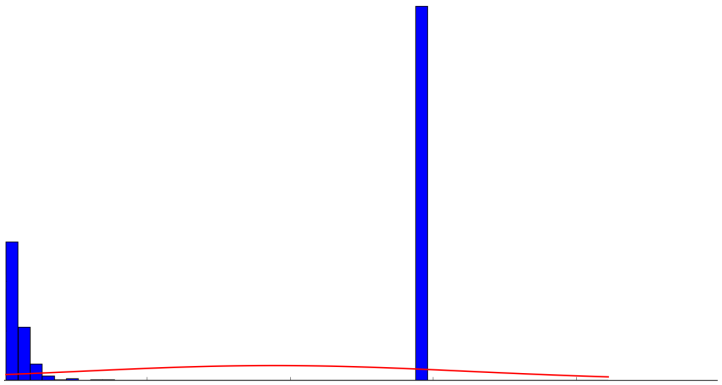

Figure 5.10: histogram plot of IPv6 payload length values.

The issue here stems from the fact that data is not Gaussian distributed. The most frequent value of 1460 bytes is represented by the rightmost bin, while other frequent values are on the left side. As a result, the bell shaped curve failes to form when plotting the data. In order to fix this, a remapping function was created which. Pseudo-code of the algorithm is described in figure 5.11.

The algorithm starts by sorting the bins based on height, from lowest to highest. It then proceeds to move each bin from the edges of the curve to the mean. As a result, the bins that are shortest are at the edges and the ones that are higher are gradually placed closer and closer to the mean. The result of this transformation step can be seen in figure 5.12.

The number of bins is an important choice for this algorithm, as values which fall in the same bin are mapped to the same value. As a direct result of this, deviations of very common values are interpreted as being common values themselves. This is evident in the case of the IPv6 payload

```
1     var idx_bin_left = 0
2     var idx_bin_right = len(bins)
3
4     sort bins by height from lowest to highest
5
6     for idx_cur_bin in range 0 to len(bins) - 1
7         if idx_cur_bin is odd
8             idx_dst_bin = idx_bin_left
9             idx_bin_left += 1
10        else
11            idx_dst_bin = idx_bin_right
12            idx_bin_right += 1
13        end if
14
15        move bin at index idx_cur_bin to location idx_dst_bin
16    end for
```

Figure 5.11: pseudo-code for transforming data to Gaussian distribution.

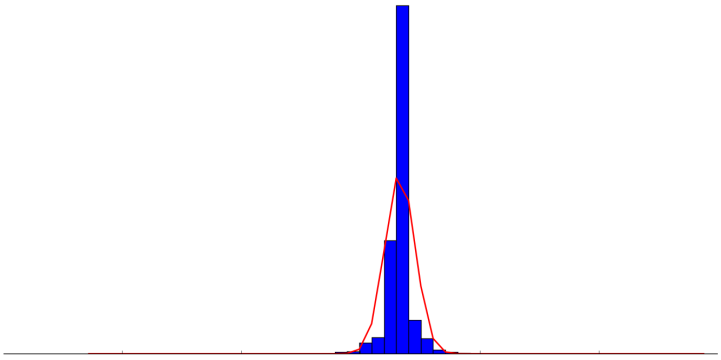

Figure 5.12: histogram plot of IPv6 payload length values after applying the log function.

length where values in an open interval around the value 1460 (i.e. most common value) are treated the same. The size of this interval is linked to the number of bins. The chosen number of bins was set to 100, meaning that the deviation from the from very values can be at most 1% of the domain.

A different, simple approach is taken for features that are based on frequency scores. A review of the transformation function applied to the IPv6 headers feature is presented below. Without any pre-processing, the plotted data is shown in figure 5.13.

Although the training data has 4 different labels, the frequency of one label in particular dwarfs all the others. As a result, the standard deviation σ is not big enough for the bell encompass any of the other labels. For the anomaly detection scheme, this would mean that benign observations would be marked as anomalous. In order to fix this issue, the log function is applied to the frequencies. In pseudo-code figure 5.6, this transformation function is applied in lines 5 to 7. The plot of the resulting frequencies is shown in figure 5.9.

Multiple classifiers

As previously mentioned, the anomaly detection scheme uses one pair of parameters μ and σ for each feature. During the detection stage, it is desired to combine multiple pairs together (i.e. multiple features) to obtain a single classifier that can be evaluated against a single threshold value.

In the case of the proposed scheme, using one single classifier for all features is not possible.

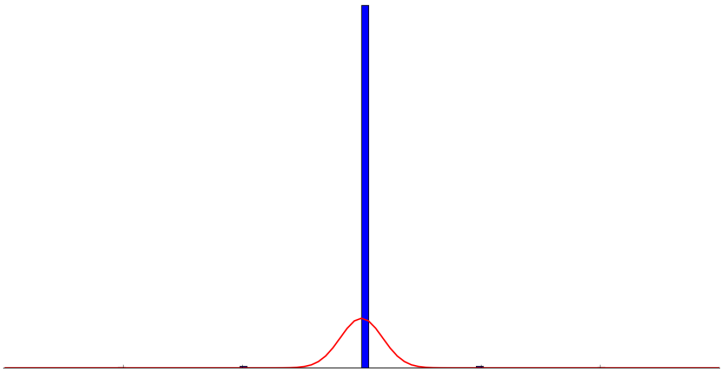

Figure 5.13: histogram plot of IPv6 headers feature.

This is because of the fact that certain features are mutually exclusive (e.g. those extracted from the TCP and UDP headers) and certain features can only be extracted provided that the needed header is present.

A solution to this problem is to group multiple features together and have a set of classifiers which can be applied to a packet based on its characteristics. Two methods for combining parameters are used: multiplication of several univariate PDF results or the use of the multivariate PDF. The groups, together with their applicability and combination methods are presented in table 5.2.

Features	**Applicability**	**Combination method**
IPv6 headers	all packets	-
IPv6 hop-by-hop options	IPv6 with hop-by-hop options header	-
IPv6 destination options	IPv6 with destination options header	-
IPv6 payload length ICMPv6 payload length	ICMPv6 packets	multivariate PDF
ICMPv6 type and code	ICMPv6 packets	-
ICMPv6 packet too big MTU	ICMPv6 packet too big messages	-
IPv6 payload length UDP length	UDP packets	multivariate PDF
Nr. simultaneous flows TCP SYN options TCP flags TCP MSS	TCP SYN packets	multiplication of univariate PDF results
TCP SYN-ACK options TCP flags TCP MSS	TCP SYN-ACK packets	multiplication of univariate PDF results
TCP ACK options TCP flags	TCP ACK packets	multiplication of univariate PDF results
TCP Non-SA options TCP flags	TCP packets without the SYN or ACK flags	multiplication of univariate PDF results
TCP window size TCP data offset	TCP packets	multivariate PDF

Table 5.2: feature groups.

The groups are made up of features which 1) are not mutually exclusive and 2) can always be

extracted from the applicable packets. In addition, the main driving factor for creating each group was the combination method.

The multivariate PDF is preferred over the multiplication of univariate PDF results as it is able to capture correlations between features. Unfortunately, it is not always possible to apply this method because of the mathematical formula and the way in which features based frequency scores are used.

The multivariate PDF formula requires the covariance matrix Σ calculated over the set of features in the group. Since features based on frequency scores are pre-processed prior to the training stage, the data cannot be used for calculating Σ anymore. For such features, the univariate PDF is calculated multiple times for each feature in the group and the results are multiplied.

5.4 Location of detection system within the network

An important decision is the placement of the detection system within a network. There are numerous setups and locations where it may be located, each with its own advantages and disadvantages. In order to maximize the potential for discovering anomalies in a timely manner, multiple factors need to be acknowledged.

The topology of the network is an important factor which influences multiple subsequent decisions. The different segments and the relationships between them can have a great impact for the behavior of the anomaly detection sensor. Certain segments are more prone to attacks than others, one example being the DMZ in contrast to other segments deeper inside the network. Another distinction results from defining the threat-model for each network segment. Internal and external actors differ with regards to knowledge of the infrastructure, their goals and levels of trust. This means that each segment has a different profile with respect to the traffic that passes through it. As mentioned previously, large variations within the problem that is being solved may have a negative impact on the learning algorithm and the quality of the trained model.

Different network segments also play a role for anomalies based on protocols which work only on the local segment (i.e. multicast listener discovery and neighbor discovery protocols). The scheme can only be applied to such probes provided that the detection system is placed on relevant network segments.

The amount of traffic passing through the anomaly detection system combined with the amount of computational resources required by the detection algorithm further impacts the efficiency of the detection scheme. When large amounts of data need to be analyzed, choke-points appear resulting in a decreased connection quality. Although this problem can be partially solved using aggregation of traffic data from different flows, this is not a solution for the scheme proposed in this work. A better approach is to have multiple detection systems and spread the workload evenly.

In order to address all these issues and limitations, it is preferred to have one sensor for each segment of the network. This setup allows for sensors to be more accurately trained for specific profiles of network traffic, to protect against probes built upon link-local protocols and to have a minimal impact of the quality of the network connections.

5.5 Conclusion

This chapter discusses the main design decisions and overall architecture of the anomaly detection system. The anomaly detection scheme is split into two stages, the first based on static protocol analysis and the second based on statistical analysis.

For the static protocol analysis, a review of the relevant RFC documents results in a minimal set of characteristics which need to be analyzed by the detection system. These characteristics target deprecated and insecure IPv6 features, the order of IPv6 extension headers, context of ICMPv6 messages, TCP connection states and the validity of length and checksum fields.

The section about the statistical analysis stage opens with a soft introduction to the Gaussian distribution. It then delves into the topic of feature engineering and explains how the training set

is processed in order to obtain an accurate definition of the baseline normality model. Features fall into one of two categories: numerical features extracted from single header fields and non-numerical features which require further pre-processing before the training stage. This pre-processing step involves calculation of frequency scores for each non-numerical feature.

Transformation functions are required in order to optimize the data such that it follows the Gaussian distribution. The algorithms for transforming the data are different for the two types of features. For numerical features the algorithm involves swapping positions of histogram bins. For non-numerical features, the log function is applied to frequency scores in order to increase the variance of the curve which has the effect of decreasing the amount of false positive alarms.

Multiple classifiers are created, each resulting from a different combination of features. The reason for having multiple classifiers stems from mutually exclusive features (e.g. TCP and UDP features) or features which may be missing in certain circumstances (e.g. ICMPv6 packet too big MTU).

Chapter 6

Implementation and testing

This chapter discusses implementation details of the anomaly detection scheme presented in the previous chapter. Only the second stage of the detection scheme, based on statistical analysis is implemented. First the libraries and tools used are listed and afterwards, testing of the system and relevant results are discussed.

The source code is released under the BSD license, and can be found online at the address github.com/alegen/master-thesis-code.

6.1 Programming language and libraries

The scheme is implemented using the Python programming language. This choice was made after considering the availability of tools and libraries for manipulating network traffic and performing statistical calculations. Python has seen a wide adoption in the scientific and academic communities in recent years, making it a strong candidate for the task.

For network traffic manipulation, the *Scapy*[1] library was used. It enables low-level interaction with the underlying TCP/IP stack of the host and allows capturing packets in promiscuous mode (i.e. packets that are not specifically meant for the anomaly detection program).

The statistical formulas are implemented in the *SciPy*[2] library. The routines and functions required to calculate Gaussian probability density functions as well as covariance matrices are provided by this project. The numerical structures used by *SciPy* (e.g. matrices) are implemented in the *NumPy*[3] library.

Plots for data visualization created for the purpose of analysis were generated with *matplotlib*[4]. Most of the figures in the report were generated with this library.

Fast prototyping of functions and algorithms (e.g. for transformation functions) as well as unit-testing was done with the help of an interactive *IPython*[5] shell.

6.2 Program architecture

The implementation of the scheme consists of three major components:

1. Data processor (source file *processor.py*) - the logic of the code which iterates over network packets and extracts relevant data. This component is used to extract feature data from the training set and also values from packets received during online detection.

[1] Scapy - secdev.org/projects/scapy
[2] SciPy - http://www.scipy.org/
[3] NumPy - numpy.org
[4] matplotlib - matplotlib.org
[5] IPython - ipython.org

2. Model trainer (source file *trainer.py*) - this component is tasked with training the model, meaning that it calculates the mean and standard deviation values, for each feature, from the data provided by the processor.
3. Baseline normality model (source file *model.py*) - performs the actual calculations for the Gaussian probability density function, with parameters calculated by the model trainer, on data generated by the data processor.

Apart from these, other source files are present which contain classes used for representing data structures (e.g. connection flow in file *flow.py*, packet traces in file *packet_trace.py*).

Because of the fact that computations can be time-consuming and they need to be executed rather frequently, especially while debugging and testing code, a caching mechanism has been created which saves intermediate results that can be reused.

6.3 Testing methodology

For the purposes of testing, training data was artificially generated using a number of virtual machines, each with a different operating system. In total, nine operating systems were used to generate traffic data, out of which four were Linux, one was OpenBSD, another one was FreeBSD and three were Microsoft Windows.

There are three reasons for using artificially generated data for the purpose of training and testing the model for the anomaly detection scheme:

1. It is difficult to obtaining training data in the form of packet capture files (commonly known as *pcap* files) that containing IPv6 packets. Most of the sets that are available online contain few number of packets, not enough to train a comprehensive model.
2. Combining multiple online *pcap* files is not optimal, as the model should be trained for a specific network in order to have positive results. Mixing multiple *pcap* files can lead to a combination of multiple different network behaviors and baseline normality levels.
3. By generating artificial traffic and not using *pcap* files from unknown sources, we can ensure that the "attack-free set" assumption required for statistical-based anomaly detection schemes is not violated.

In order to test the model, anomalous packets were also required. The anomalous packets used for testing are presented in table 6.1.

Packet index	Protocols	Description
0, 1, 2	ICMPv6	nmap probes IE1, IE2 and NI
3 - 8	TCP	nmap probes S1 to S6
9 - 14	TCP	nmap probes T2 to T7
15	TCP	nmap TECN probe
16	ICMPv6	MLD query probe
17	ICMPv6	packet too big message
18	ICMPv6	1^{st} fragment of an ICMPv6 echo request
19	ICMPv6	2^{nd} fragment of an ICMPv6 echo request
20	ICMPv6	nmap NI probe
21	UDP	nmap U1 probe

Table 6.1: anomalous packets used for the purpose of testing.

Testing was performed with an approach based on calculating ROC curves. The detection and false positive rates were calculated for multiple values of the threshold and then plotted.

For some of the plots it was not possible to calculate the two rates for all values of the threshold because of the very small values of anomalous scores and the long interval in which benign values exist. In these cases, instead of calculating the rates for all values of the threshold, the calculation

would stop whenever the false positive rate rose above 0.05. This decision was taken in an attempt to decrease the amount of time required to perform testing. The reason for stopping at this threshold was that, in a real world scenario, 5 false alarms for every 100 incoming packets would already make the system unusable.

An ROC curve was calculated for each of the groups in table 5.2 presented in the previous chapter. Whenever a classifier is referred to by a number, it is for the group with the same entry in the aforementioned table.

6.4 Testing results

The 1^{st} classifier looks at the chain of headers in an IPv6 packet. The ROC plot is presented in figure 6.1. With a false positive rate of 0, the classifier is able to correctly detect packets 1, 17, 18 and 19 as anomalous.

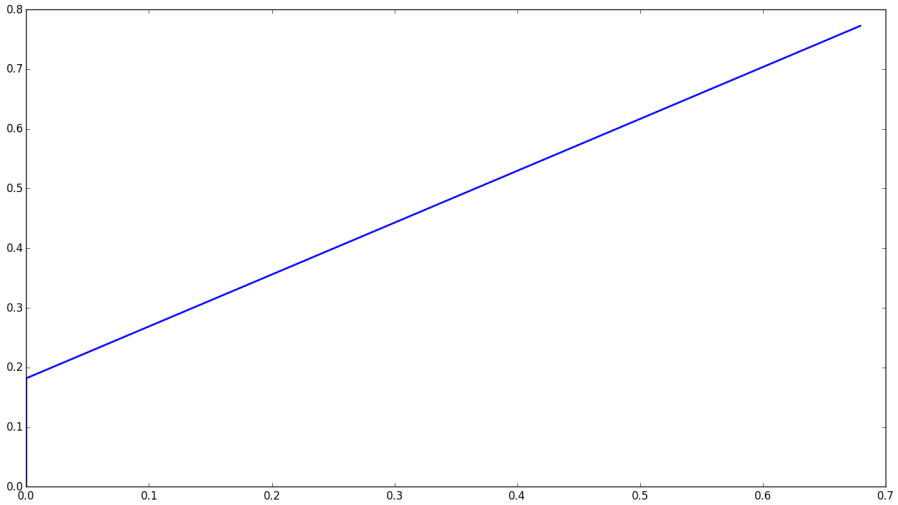

Figure 6.1: ROC for classifier targeting feature group 1.

The classifier for the 4^{th} group of features is applied only to ICMPv6 packets and analyzes the IPv6 payload length and the amount of data in the ICMPv6 packet. Keeping the false positive rate of 0, the classifier labels packets 0, 1, 2, 17, 18, 19 and 20 as being anomalous. These packets are very different than the baseline normality model and obtain the scores presented in table 6.2. All of the scores are lower than the smallest score for a benign packet, equal to 0.1864.

Packet	Score
0	0.0
1	0.0
2	9.6001E-268
17	5.3934E-46
18	0.0
19	0.0
20	0.1495

Table 6.2: scores of packets after applying the 4^{th} classifier.

The 5^{th} classifier detects probes 17 and 20 as anomalous while stil keeping the false positive rate at 0. The ROC plot is shown in figure 6.2.

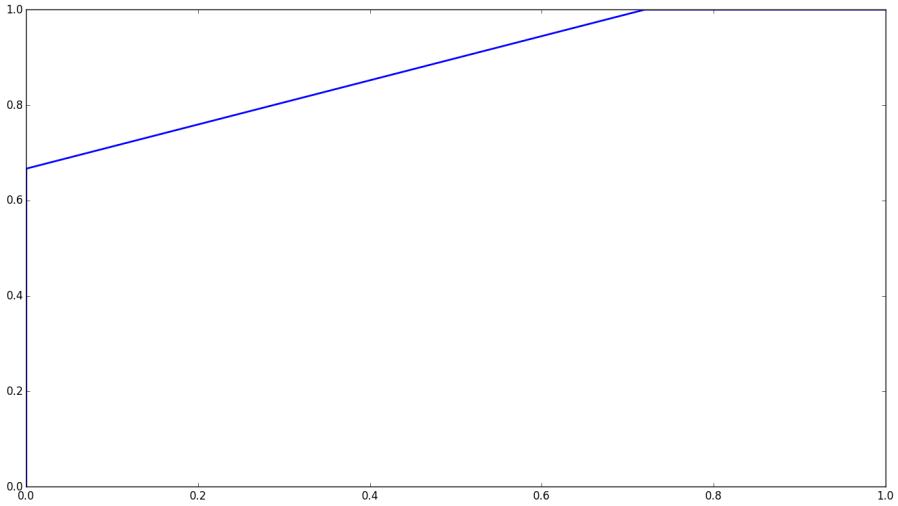

Figure 6.2: ROC for classifier targeting feature group 5.

The 8^{th} classifier detects packets 3, 4, 5, 6, 7, 8, 10, 12 and 15 as anomalous with 0 false positive rate. All these packets obtain a score of 0, while the scores for benign examples fall in the interval from 9.7751E-232 to 0.1820.

The 10^{th} classifier is not able to detect any probes in a meaningful manner. While the scores for benign packets fall in the interval from 2.9341E-18 to 0.04822, the smallest score for an anomalous example is 1.0281E-7 for packet 11.

The 11^{th} classifier correctly detects packets 9, 10, 14 and 15 as anomalous with 0 false positive rate. The scores for benign packets fall in the interval from 2.6888E-6 to 0.17605, while the scores of the detected anomalies are equal to 0. The ROC plot is shown in figure 6.3.

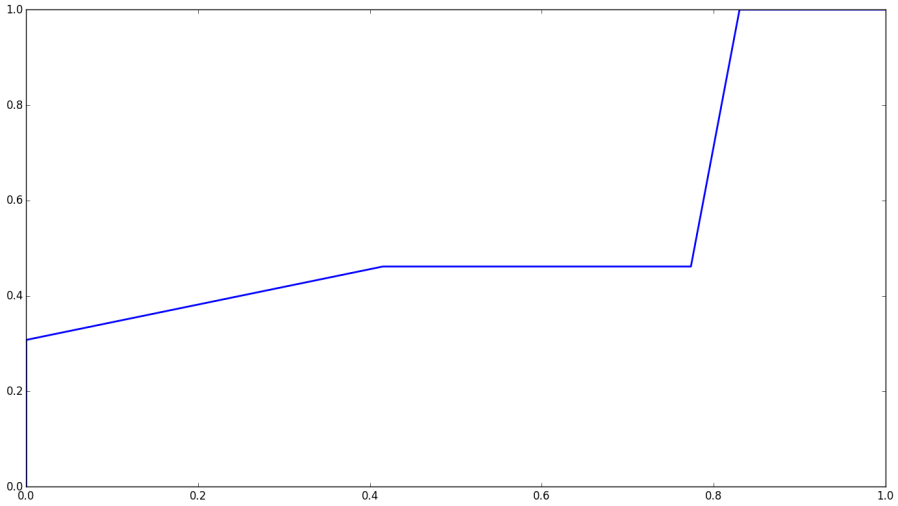

Figure 6.3: ROC for classifier targeting feature group 11.

The 12^{th} classifier detects packets 6, 8, 12 and 13 as anomalous. The interval of scores for benign examples is from 1.3306E-7 to 0.0043. The ROC plot is shown in figure 6.4 while and the scores for the detected anomalies are shown in table 6.3.

While generating the artificial training set, no packets were obtained that could be used for training classifiers for groups 2, 3 and 6. As a result, testing for features in these groups was not

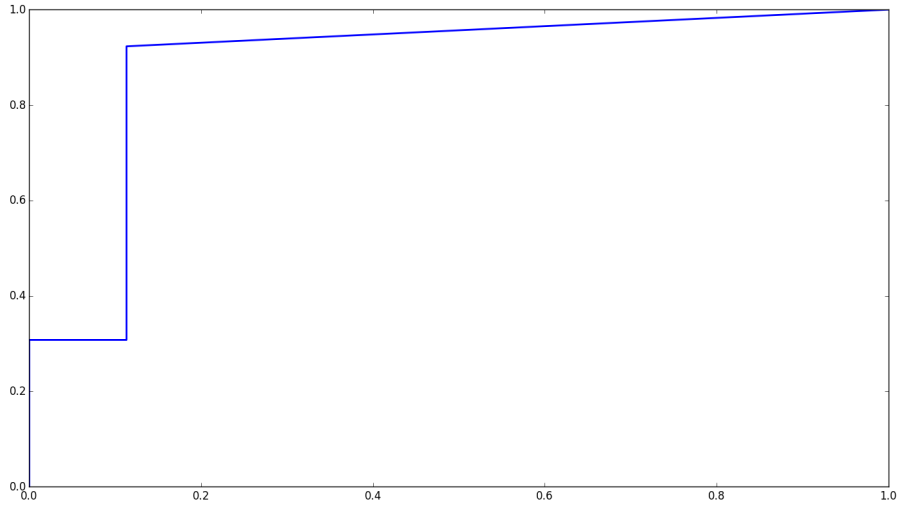

Figure 6.4: ROC for classifier targeting feature group 12.

Packet	Score
6	0.0
8	0.0
12	5.6655E-21
13	5.6151E-33

Table 6.3: scores of packets after applying the 12^{th} classifier.

performed. Even so, these should be implemented and used in networks which see traffic with the required characteristics.

The 7^{th} classifier fails to detect the only UDP probe in the test set, namely the U1 probe from nmap. This again results from lack of sufficient training data. Even so, in a real world situation, the probe is sent to a closed port and would thus be detected during the first stage of the detection scheme.

6.5 Results discussion

The results of the tests discussed in the previous chapter show that the scheme is usable in a real world environment. The detection rate can achieve positive results while keeping the false positive at 0. With the exception of packets 11 (nmap T4 probe) and 16 (MLD general query probe), all others are correctly labeled as being anomalous by at least one of the 12 classifiers.

Even though the first part of the detection scheme (i.e. based on static protocol analysis) was not implemented, it would have been able to detect both probe 11 and 16. Probe 11 is sent by nmap to an open port, but it does not have the SYN flag set which is required to initiate the TCP handshake procedure. As a result, it is distinguishable from benign packets which attempt a legitimate connection. Probe 16 can be detected by checking that the source address for the packet belongs to a router on the local network segment. This check can be enforced automatically on networks which use the SEND (SEcure Neighbor Discovery) protocol. Although probe 16 can be detected when it is received from a non-router host, it cannot be sensibly labeled as anomalous when the sender is a router with malicious intentions.

6.6 Conclusion

This chapter discusses the implementation details and testing results of the second stage in the proposed detection scheme. The scheme is based on anomaly detection via statistical analysis and makes use of the Gaussian distribution to score packets based on their likelihood to be probes.

A description of the libraries and tools used throughout the implementation process is offered in order to enable easier replication of the results. The source code is written in Python and makes use of the *Scapy*, *SciPy*, *NumPy* and *matplotlib* libraries.

Testing of the implementation is done with the use of ROC curves which plot the false positives against the detection rates for different values of the threshold. The results show that it is possible to detect all but two probes and maintain a false positive rate of 0.

The two probes (one based on UDP and the other on MLD) can be detected more efficiently in the first stage of the detection scheme which is based on static protocol analysis.

Chapter 7

Conclusion

The work presented in thesis focuses on remote probing and detection of operating systems running on network enabled devices. Probing is defined as the action of sending anomalous packets (referred to as probes) to a device which aim to generate undefined behavior in the TCP/IP stack implementation of the device's operating system. By analyzing the responses of the anomalous packets, relevant information can be obtained which enables discrimination between multiple operating systems. The act of probing combined with analysis of the responses is known as fingerprinting.

There are multiple ways of detecting the operating system of a network enabled device. The most common two methods are based on active and passive sniffing. The former relies on interaction with the targeted device, while the latter eavesdrops on network traffic between the target and other devices. The work performed as part of this project focused on fingerprinting via active probing.

The aim of the project was two-fold, stemming from the dual use of fingerprinting in real world scenarios. On one hand, it was desired to increase the accuracy and reliability of fingerprinting. This helps with legitimate uses, such as detecting rogue devices connected to a private network. On the other hand, improving a technology without implementing security measures may lead to an undesired situation where safeguards are not available against malicious usage. As a result, research has been performed into methods which allow detection of probes used for the purpose of fingerprinting.

In order to enhance fingerprinting methods, the open-source nmap tool was chosen as reference. When performing fingerprinting, nmap sends a total of 18 anomalous packets. From the responses it receives, it extracts a number of 676 features which are then used with a logistic regression model. Logistic regression is a machine learning algorithm which is used in classification problems. In the case of nmap, it attempts to classify the operating system which is most likely to send the specific set of responses. The model is trained offline, prior to the online probing stage. Training of the model is performed with the help of a dataset which contains known sets of responses from several operating systems.

Improving the reliability of nmap was achieved through several different methods. The first method was by adding additional probes which enable additional information to be obtained from the operating system of the remote device. New probes were discovered based on 1) fragmented ICMPv6 echo request packets, 2) the multicast listener discovery (MLD) protocol and 3) structurally invalid packets. The second method for improving nmap was based on additional features extracted from existing data. The new features are 1) the hop-limit field in IPv6 packets and 2) a combination of two existing features which captures the correlation between them. The third improvement brought to nmap was in the form of additional pre-processing of the training set by means of data imputation. Imputation was performed using the multiple imputation by chained equations technique.

Research into detection of probes used for the purpose of fingerprinting started with an evaluation of the current state of the art on the topic of network intrusion detection. Detection schemes follow three main methodologies, namely: the use of static signatures, anomaly detection and

stateful protocol analysis. As a result of the fact that undiscovered probes exist, some of which were found while enhancing nmap, the decision was made to focus on anomaly detection. Further research into detection methods showed that a simple algorithm with a comprehensive set of features can have better results than a highly parametric algorithm with sub-optimal features. Therefore, the choice was made to focus on feature engineering and use a simpler algorithm based on statistical analysis.

In order to increase the detection rate and at the same time decrease the amount of false positives, the scheme based on statistical analysis was combined with stateful protocol analysis. When a packet first arrives, it is analyzed against several RFC documents by the stateful protocol analysis scheme. These checks are aimed to verify the structural integrity of the packet as well as its conformance with the latest version of standards. If all checks are passed, the packet is analysed in the second stage with the anomaly detection scheme.

The anomaly detection scheme is based on the Gaussian distribution. Certain features are extracted from characteristics of the packet and the probability density function of the distribution is applied. If the result is less than a certain threshold value, then the packet is tagged as anomalous and an alarm is raised.

An offline training stage is used to calculate pairs of parameters μ and σ for the Gaussian density function, for each feature. Prior to the training stage, data needs to be pre-processed, as certain features are either non-numerical or do not follow the Gaussian distribution. Transformation functions are defined for each of the two cases.

Features of packets are combined together before comparing the result against a threshold. Two combination methods are used, one which calculates multiple values of the univariate density function and multiplies these results and another which uses the multivariate version of the density function. The latter is more optimal, as it is able to automatically capture correlations between features, but is not always applicable. Certain features make use of additional pre-processing which changes the structure of the data, while other features are cannot always be extracted from a packet.

A proof of concept implementation of the second detection stage, based on statistical analysis, was created. The code is available at github.com/alegen/master-thesis-code, released under an open-source BSD license. The implementation makes use of the Python programming language along with several libraries such as: Scapy, SciPy, NumPy and matplotlib.

The proof of concept implementation was used to analyze the efficiency of the scheme. Artificial data was generated for the purpose of training. The nmap probes, as well the probes discovered as part of this project were used in order to generate ROC curves. The results showed that it is possible to detect all probes with a false positive rate of 0, with the exception of a probe based on the multicast listener discovery (MLD) protocol.

Even though the anomaly detection scheme cannot detect the MLD probe, there exist security checks which may be enforced on a network if prevention of fingerprinting with the help of this probe is desired.

Chapter 8

Future directions

This chapter discusses possible further directions which may be pursued in order to build upon and advance the findings presented in this report. The maturity of the nmap project is visible from the amount of functionality and the performance of existing features. Even so, enhancements may be added to the IPv6 fingerprinting engine. Furthermore, although the results from testing the anomaly detection scheme are positive, there is still room for improvements.

Possible directions for future research are split into two categories, the first aiming to improve fingerprinting, while the second focusing on detecting probes.

8.1 Improving fingerprinting

One topic which can be followed is the discovery of new fingerprinting probes, based on other protocols than the ones currently being used. Possibilities include the dynamic host control protocols version 6 (DHCPv6), neighbor discovery protocol (NDP), multicast router discovery (MRD) and the mobile IPv6 extensions. Fuzzing header fields belonging to these protocols may result in behavioral differences and the possibility of discriminating between operating systems.

The current state of the imputation work performed on the training set, prior to the training stage, is definitely an improvement over the previous approach where missing values were set to -1. Still, the current imputation parameters (i.e. feature groups, number of imputation sets and iterations per set) are not the most optimal. Research into extending this data pre-processing step may lead to an increase in the quality of the training set. In turn, the reliability and accuracy of the results would also further improve.

A study with regards to the tradeoff between the quality of the imputed set (i.e. the increase in prediction score) and the amount of time required for imputation (i.e. the number of imputed sets and iterations) may prove valuable considering the number of times the algorithm is executed. During integration of fingerprint submissions from the nmap community, imputation of the complete dataset is performed after including each new submission. Decreasing the required time, while still maintaining practical results can have a positive impact on the applicability of imputation.

The search for optimal parameters for the imputation process could be more automated, with the use of additional scripts. This way, the imputation process may be executed multiple times and automatically pick the best set of parameters.

The training set may also be improved by analyzing the current groups of fingerprints and ensuring the overlap is minimal. Having multiple groups with very similar characteristics can lead to a poorly trained model. The reason is that the logistic regression model may have smaller weights for features which are relevant for classification, but fail to discriminate between fingerprints which are improperly placed in sepparate groups. In order to solve this problem, a clustering algorithm should be used to group fingerprints together according to similarity.

8.2 Improving detection of probes

A more comprehensive list of packets may be defined for the first stage of the detection process. In order to find relevant packets which have become obsolete, research into existing documentation should be performed. There are a number of RFC documents that summarize design changes to existing protocols and standards. One such example is RFC7414 - *A Roadmap for Transmission Control Protocol Specification Documents* which summarizes changes made to the TCP protocol. These documents may prove more useful and practical than analyzing individual RFC documents for a specific protocol (some of which may already be obsolete).

With regards to the second detection stage, two related improvements may be sought. A better approach is desired for integrating non-numeric features into the anomaly detection scheme. The solution should attempt to extract one value from each training example and only use it once for the calculation of the parameters μ and σ. With such a method in place, more features could be combined using the multivariate Gaussian PDF, bringing down the number of feature groups (and thus classifiers) from table 5.2. This would result in a simplification of the scheme with a smaller number of threshold values that require tuning.

As seen in chapter 6.5, the probe based on the multicast listener discovery protocol (MLD) is not detected by the second stage of the detection scheme. One partial solution is to ensure that all hosts on the network implement the Secure Neighbor Discovery Protocol (SEND) and do not accept MLD queries from non-router hosts. Even so, this solution does not defend against hosts which have the role of router on the local segment and have a malicious intent. If detection is to be integrated with prevention of successful fingerprinting, a study should be performed into the exact uses-cases of MLD and the implications of disabling it on the local segment. A starting point would be RFC4890 - *Recommendations for Filtering ICMPv6 Messages in Firewalls* which discusses the desired behavior firewalls should adopt with regards to ICMPv6 messages.

A more comprehensive test set could be gathered in order to verify the performance of the anomaly detection scheme in a better simulated environment. Currently, the testing results show that the scheme can achieve positive results, although this may be partly caused by the artificial nature of the test data.

A last research direction can be the integration of a feedback loop into the anomaly detection system, similar to the one discussed in [Cos+14]. This would allow further lowering of the false positives rate

Bibliography

Academic Sources

[Ans60] Frank J Anscombe. 'Rejection of outliers'. In: *Technometrics* 2.2 (1960), pp. 123–146.

[Azu+11] Melissa J Azur et al. 'Multiple imputation by chained equations: what is it and how does it work?' In: *International journal of methods in psychiatric research* 20.1 (2011), pp. 40–49.

[BBK12] Monowar H Bhuyan, DK Bhattacharyya and Jugal K Kalita. 'An effective unsupervised network anomaly detection method'. In: *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*. ACM. 2012, pp. 533–539.

[BBK14] Monowar H Bhuyan, DK Bhattacharyya and Jugal K Kalita. 'Network anomaly detection: methods, systems and tools'. In: *Communications Surveys & Tutorials, IEEE* 16.1 (2014), pp. 303–336.

[BG11] Stef Buuren and Karin Groothuis-Oudshoorn. 'mice: Multivariate imputation by chained equations in R'. In: *Journal of statistical software* 45.3 (2011).

[BMS14] Ismail Butun, Salvatore D Morgera and Ravi Sankar. 'A survey of intrusion detection systems in wireless sensor networks'. In: *Communications Surveys & Tutorials, IEEE* 16.1 (2014), pp. 266–282.

[Bra+12] Daniela Brauckhoff et al. 'Anomaly extraction in backbone networks using association rules'. In: *IEEE/ACM Transactions on Networking (TON)* 20.6 (2012), pp. 1788–1799.

[BSW02] Douglas J Brown, Bill Suckow and Tianqiu Wang. 'A Survey of Intrusion Detection Systems'. In: *Department of Computer Science, University of California, San Diego* (2002).

[CBK09] Varun Chandola, Arindam Banerjee and Vipin Kumar. 'Anomaly detection: A survey'. In: *ACM Computing Surveys (CSUR)* 41.3 (2009), p. 15.

[CMO11] Pedro Casas, Johan Mazel and Philippe Owezarski. 'Steps towards autonomous network security: unsupervised detection of network attacks'. In: *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on*. IEEE. 2011, pp. 1–5.

[Dom12] Pedro Domingos. 'A few useful things to know about machine learning'. In: *Communications of the ACM* 55.10 (2012), pp. 78–87.

[Fal11] Petter Bjerke Falch. 'Investigating passive operating system detection'. In: (2011).

[Fan+08] Rong-En Fan et al. 'LIBLINEAR: A library for large linear classification'. In: *The Journal of Machine Learning Research* 9 (2008), pp. 1871–1874.

[Gar+09] Pedro Garcia-Teodoro et al. 'Anomaly-based network intrusion detection: Techniques, systems and challenges'. In: *computers & security* 28.1 (2009), pp. 18–28.

[GHK14] Mario Golling, Rick Hofstede and Robert Koch. 'Towards multi-layered intrusion detection in high-speed networks'. In: *Cyber Conflict (CyCon 2014), 2014 6th International Conference On*. IEEE. 2014, pp. 191–206.

[GT07a] L Greenwald and T Thomas. 'Evaluating Tests used in Operating System Fingerprinting'. In: *LGS Bell Labs Innovations Technical Memorandum TM-071207* (2007).

[GT07b] Lloyd G Greenwald and Tavaris J Thomas. 'Toward Undetected Operating System Fingerprinting.' In: *WOOT* 7 (2007), pp. 1–10.

[HH05] Guanghui He and Jennifer C Hou. 'An in-depth, analytical study of sampling techniques for self-similar internet traffic'. In: *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*. IEEE. 2005, pp. 404–413.

[HPK01] Mark Handley, Vern Paxson and Christian Kreibich. 'Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics.' In: *USENIX Security Symposium*. 2001, pp. 115–131.

[JL14] Thienne Johnson and Loukas Lazos. 'Network anomaly detection using autonomous system flow aggregates'. In: *Transit* 3.4 (2014), p. 5.

[JLS13] Antonio J. Jara, Latif Ladid and Antonio Skarmeta. 'The Internet of Everything through IPv6: An Analysis of Challenges, Solutions and Opportunities'. In: *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)* 4.3 (Sept. 2013), pp. 97–118.

[Kim+04] Hyukjoon Kim et al. 'Detecting network portscans through anomoly detection'. In: *Defense and Security*. International Society for Optics and Photonics. 2004, pp. 254–263.

[KMS12] Levent Koc, Thomas A Mazzuchi and Shahram Sarkani. 'A network intrusion detection system based on a Hidden Naïve Bayes multiclass classifier'. In: *Expert Systems with Applications* 39.18 (2012), pp. 13492–13500.

[Koh96] Ron Kohavi. 'Scaling Up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid.' In: *KDD*. 1996, pp. 202–207.

[Las+05] Pavel Laskov et al. 'Learning intrusion detection: supervised or unsupervised?' In: *Image Analysis and Processing–ICIAP 2005*. Springer, 2005, pp. 50–57.

[Lip+03] Richard Lippmann et al. 'Passive operating system identification from TCP/IP packet headers'. In: *Workshop on Data Mining for Computer Security*. Citeseer. 2003, p. 40.

[Lüs08] Cécile Lüssi. 'Signature-based extrusion detection'. MA thesis. Eidgenössische Technische Hochschule Zürich, Institut für Technische Informatik und Kommunikationsnetze, 2008.

[MA12] David Mudzingwa and Rajeev Agrawal. 'A study of Methodologies used in Intrusion Detection and Prevention Systems (IDPS)'. In: *Southeastcon, 2012 Proceedings of IEEE*. IEEE. 2012, pp. 1–6.

[Mor+12] Sumit More et al. 'A knowledge-based approach to intrusion detection modeling'. In: *Security and Privacy Workshops (SPW), 2012 IEEE Symposium on*. IEEE. 2012, pp. 75–81.

[Mul+11] Varish Mulwad et al. 'Extracting information about security vulnerabilities from web text'. In: *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2011 IEEE/WIC/ACM International Conference on*. Vol. 3. IEEE. 2011, pp. 257–260.

[Pax97] Vern Paxson. 'Automated packet trace analysis of TCP implementations'. In: *ACM SIGCOMM Computer Communication Review* 27.4 (1997), pp. 167–179.

[PDN14] Christo Panchev, Petar Dobrev and James Nicholson. 'Detecting Port Scans against Mobile Devices with Neural Networks and Decision Trees'. In: *Engineering Applications of Neural Networks*. Springer, 2014, pp. 175–182.

[Ped+11] F. Pedregosa et al. 'Scikit-learn: Machine Learning in Python'. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[PVH10] Guillaume Prigent, Florian Vichot and Fabrice Harrouet. 'IpMorph: fingerprinting spoofing unification'. In: *Journal in computer virology* 6.4 (2010), pp. 329–342.

[Sha+14] Zain Shamsi et al. 'Hershel: single-packet os fingerprinting'. In: *The 2014 ACM international conference on Measurement and modeling of computer systems*. ACM. 2014, pp. 195–206.

[SM07] Karen Scarfone and Peter Mell. 'Guide to intrusion detection and prevention systems (idps)'. In: *NIST special publication* 800.2007 (2007), p. 94.

[Son+07] Xiuyao Song et al. 'Conditional anomaly detection'. In: *Knowledge and Data Engineering, IEEE Transactions on* 19.5 (2007), pp. 631–645.

[SP10] Robin Sommer and Vern Paxson. 'Outside the closed world: On using machine learning for network intrusion detection'. In: *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE. 2010, pp. 305–316.

[SSP08] Anna Sperotto, Ramin Sadre and Aiko Pras. 'Anomaly characterization in flow-based traffic time series'. In: *IP Operations and Management*. Springer, 2008, pp. 15–27.

[TPS13] Alexander G Tartakovsky, Aleksey S Polunchenko and Grigory Sokolov. 'Efficient computer network anomaly detection by changepoint detection methods'. In: *Selected Topics in Signal Processing, IEEE Journal of* 7.1 (2013), pp. 4–11.

[Tro03] Chris Trowbridge. 'An overview of remote operating system fingerprinting'. In: *SANS InfoSec Reading Room-Penetration Testing* (2003).

[Ye+02] Nong Ye et al. 'Multivariate statistical analysis of audit trails for host-based intrusion detection'. In: *Computers, IEEE Transactions on* 51.7 (2002), pp. 810–820.

[ZYG09] Weiyu Zhang, Qingbo Yang and Yushui Geng. 'A survey of anomaly detection methods in networks'. In: *Computer Network and Multimedia Technology, 2009. CNMT 2009. International Symposium on*. IEEE. 2009, pp. 1–3.

RFC Sources

[RFC2460] R. Hinden S. Deering. *Internet Protocol, Version 6 (IPv6)*. Dec. 1998.

[RFC2710] B. Haberman S. Deering W. Fenner. *Multicast Listener Discovery (MLD) for IPv6*. Oct. 1999.

[RFC3168] D. Black K. Ramakrishnan S. Floyd. *The Addition of Explicit Congestion Notification (ECN) to IP*. Sept. 2001.

[RFC3260] D. Grossman. *New Terminology and Clarifications for Diffserv*. Apr. 2002.

[RFC3810] L. Costa R. Vida. *Multicast Listener Discovery Version 2 (MLDv2) for IPv6*. June 2004.

[RFC3879] B. Carpenter C. Huitema. *Deprecating Site Local Addresses*. Sept. 2004.

[RFC3971] P. Nikander Ed. Arkko J. Kempf. *SEcure Neighbor Discovery (SEND)*. Mar. 2005.

[RFC4443] Ed. M. Gupta A. Conta S. Deering. *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*. Mar. 2006.

[RFC4620] Ed. B. Haberman M. Crawford. *IPv6 Node Information Queries*. Aug. 2006.

[RFC5095] G. Neville-Neil J. Abley P. Savola. *Deprecation of Type 0 Routing Headers in IPv6*. Dec. 2007.

[RFC5722] S. Krishnan. *Handling of Overlapping IPv6 Fragments*. Dec. 2009.

[RFC6946] F. Gont. *Processing of IPv6 "Atomic" Fragments*. May 2013.

[RFC793] University of Southern California Information Sciences Institute. *Transmission Control Protocol*. Sept. 1981.

[RFCDR1] T. Anderson F. Gont W. Liu. *Deprecating the Generation of IPv6 Atomic Fragments*. July 2015.

[RFCDR2] F. Gont. *Security Implications of Predictable Fragment Identification Values*. July 2015.

Books

[Cis05] Inc. Cisco Systems. *Server Farm Security in the Business Ready Data Center Architecture v2, Chapter 8*. 2005.

[Lyo09] Gordon Fyodor Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, 2009.

Online Sources

[Cor] Jonathan Corbet. *Increasing the TCP initial congestion window*. `https://lwn.net/Articles/427104/`. Accessed 18^{th} May 2015.

[Edg] Jake Edge. *IPv6 source routing: history repeats itself*. `https://lwn.net/Articles/232781/`. Acessed 17^{th} May 2015.

[Gaë] Jean-Marc Saffroy Gaël Roualland. *IP Personality*. `http://ippersonality.sourceforge.net`. Accessed 5^{th} March 2015.

[Goo] Google. *IPv6 statistics*. `https://google.com/intl/en/ipv6/statistics.html`. Accessed 2^{nd} May 2015.

[IoT] IoT6. *IPv6 advantages for IoT*. `http://iot6.eu/ipv6_advantages_for_iot`. Accessed 3^{rd} May 2015.

[jmc] horizon ¡jmcdonal@unf.edu¿. *Defeating Sniffers and Instrusion Detection Systems*. `http://phrack.org/issues/54/10.html`. Accessed 12^{th} May 2015.

[Lyo] Gordon "Fyodor" Lyon. *Advanced Network Reconnaissance with Nmap*. `https://nmap.org/presentations/Shmoo06/`. Accessed 12^{th} May 2015.

[Mor] Mathias Morbitzer. *Remote OS Detection with IPv6*. `https://www.troopers.de/events/troopers14/389_remote_os_detection_with_ipv6/`. Accessed 17^{th} May 2015.

[nma] nmap.org. *nmap - Network MAPper*. `https://nmap.org`. Accessed 26^{th} February 2015.

[sav] savage@apostols.org. *Introduction to QueSO*. `https://web.archive.org/web/19981202101717/http://apostols.org/projectz/queso/`. Accessed 14^{th} March 2015.

[Zal] Michal Zalewski. *p0f v3*. `http://lcamtuf.coredump.cx/p0f3`. Accessed 22^{nd} March 2015.