Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Information Security and Communication Technology

Odin Jenseg

A machine learning approach to detecting malware in TLS traffic using resilient network features

Master's thesis in Information Security
Supervisor: Slobodan Petrovic
June 2019

NTNU
Norwegian University of
Science and Technology

Preface

This Master's thesis is research conducted at the Department of Information Security and Communication Technology at NTNU. It was carried out during the spring semester of 2019. The basis of this research stemmed from my interest in network security monitoring and my work experience as a security analyst at mnemonic. This research was carried out with mnemonic, a cybersecurity company and a Managed Security Service Provider (MSSP). The topic of this thesis was brought up in discussion with mnemonic. This work has been done to solve a growing problem that network security monitoring faces today. The content of this is aimed at readers with interests in network security monitoring and malware detection, and how machine learning can be used to extend existing detection capabilities.

June 1, 2019

Odin Jenseg

Acknowledgment

First and foremost, I would like to thank Even Sverdrup Augdal for guidance, valuable discussions, and always able to answer my questions or point me in the correct direction. Secondly, I would like to thank my supervisor, Slobodan Petrovic and Andrii Shalaginov for answering all of my questions, providing research ideas, and giving valuable help in writing this thesis. Further, I wish to express gratitude to colleges in mnemonic for sharing their experience and essential discussions. I will extend this gratitude to Stian and Henrik for reading my work and providing essential feedback. Then, I would like to acknowledge my classmates for scientific discussions and always eager to share their knowledge. Finally, I would like to thank my girlfriend Kari Anette Sand for all support during this thesis.

O.J.

Abstract

The growth of malware utilizing encrypted channels makes it challenging to detect malicious activity using current Network Intrusion Detection Systems (NIDSs). The current network intrusion detection systems utilize pattern matching algorithms to identify malware artifacts in the network traffic. In Transport Layer Security (TLS) encrypted networks, limited amount of data are available for the NIDS. This obstacle is exploited by malware authors to evade detection. In this thesis, we are looking into using machine learning classification algorithms to recognize malware communication within TLS channels without having to decrypt the network traffic. In the last few years, an increase in research has been looking into solutions for this problem using classification algorithms. We extend the existing research by identifying features in the TLS traffic that is resilient to evasion techniques used by more advanced types of malware. Advanced malware is more problematic to detect with traditional NIDSs since they try to evade these systems by generating traffic that is similar to ordinary corporate traffic. Features identified as resilient towards these evasion techniques are those describing the malware behavior in TLS encrypted traffic. Extracting behavior artifacts are performed with a mature NIDS and Network Security Monitoring (NSM) system called Suricata and Metadata Collector, a tool developed as a part of this thesis. With efficient classification algorithms, we can overcome some major challenges of NIDS face today.

Contents

List of Figures

List of Tables

Listings

Acronyms

AV Anti Virus.

C2 Command and Control.

CDN Content Delivery Network.

DNS Domain Name Server.

DoH DNS over HTTPS.

DoT DNS over TLS.

IDS Intrusion Detection System.

IoC Indicator of Compromise.

IPS Intrusion Prevention System.

NIDS Network Intrusion Detection System.

NSM Network Security Monitoring.

SNI Server Name Indication.

TAP Test Access Point.

TLS Transport Layer Security.

Glossary

classification performance Here: Metrics estimating the quality of a classification algorithm. These metrics are using the value of true positives, false positives, true negatives, and false negatives in their calculations.

1 Introduction

The first chapter of this thesis includes a brief introduction to the problem of detecting malware in encrypted traffic. Following up on the introduction, justification of the research, and the research questions this thesis is going to answer are given. At the end of this chapter are the contribution and thesis outline.

1.1 Topic covered by the Thesis

In a network infrastructure of a medium to large enterprise, network controls are commonly deployed that can detect security incidents. An accepted method to detect intrusions in network traffic is to intercept the traffic and look for malicious patterns using automated tools.

An example of malicious activity could be exfiltration of company secrets. In such scenarios it is crucial to detect and respond instantaneously. The time it takes to detect and respond can have a severe impact on the consequences of how much sensitive information that get exfiltrated. With active monitoring of network traffic, the mean time to detect can be relatively short, resulting in rapid response and consequence mitigation. That being said, network intrusion detection systems have become less effective due to the increase in encryption. In recent years more of the network traffic is being encrypted with Transport Layer Security (TLS). Current Intrusion Detection System (IDS) will only be able to detect malicious patterns if the traffic is not encrypted. More than 80% of the current web traffic is encrypted over HTTPs [6], and more than 37% of all malware is utilizing HTTPs for their communication channel [49].

A proposed method for dealing with this problem is to utilize machine learning to learn patterns in the encrypted network traffic. Even though the traffic payload is encrypted, there are metadata and related information available in cleartext. This type of information can be difficult for a human security analyst to utilize, but computers can automatically make predictions based on previous observations by using machine learning algorithms [42, p. 1].

1.2 Keywords

Keywords covered for this thesis are chosen according to IEEE Computer Society: **I.2.6g** Machine learning, **C.2.0.f** Network-level security and protection, **C.2.3.b** Network monitoring, **C.2.2** Network Protocols, **K.6.5.c** Invasive software (viruses, worms, Trojan horses)

1.3 Problem description

Pattern matching-based Network Intrusion Detection Systems (NIDSs) are not efficient on detecting threats in TLS encrypted traffic. This intelligence is exploited by malware authors who make malware's communication blend in with the normal traffic in the network. A commonly used method to

counter this problem is to decrypt the network traffic before a network sensor analyses the traffic. This is a valid solution to the problem but introduces more complexity in the network infrastructure, and it requires all computers to trust the certificate used by the decryptor. In this thesis, we propose an alternative method for detecting malware without decrypting the traffic.

Without decrypting the content and analyzing the patterns, it is possible to look for known malware IP addresses and domains. However, such information can be easily modified by the malware author, and when dealing with a more advanced threat actor, these Indicator of Compromises (IoCs) have a short lifetime. Using these indicators can also introduce a lot of false positives and false negatives. For instance, malware authors often use shared hosting so the IP address may be used by other non-malicious activities too and will result in false positives. It is also impossible to possess an exhaustive list of bad IPs and domains, so it will get false negatives if it depends on such weak indicators.

Other research has been using machine learning in this field, and have shown that this is a valuable method. Existing research uses a variety of features, both static (e.g., known legitimate domain name), and behavior (e.g., time differences between network packets) [15].

1.4 Justification, motivation, and benefits

Being able to adapt the detection capabilities to identify new threats is crucial in the process of continuous security monitoring. Encrypted traffic is a challenge that is important to address as malware utilizes encryption to hide their traffic and evade detection.

The existing research in this field has been using as much information as possible in the network traffic. In this research, instead of using all available information, we are looking into information that is resilient to easy modification by the malware authors. In this thesis, we define resilient as information that is resistant to evasion attacks and change in the attacker's tactics. By focusing on behavior and information that are more difficult to modify, it is harder for the adversaries to evade our detection methods.

The motivation for this research is to develop an intelligent method that can detect malicious activity in encrypted network traffic without having to decrypt the traffic prior to inspection. If we can successfully develop a method like that, we can remove an important blind spot in current IDSs.

1.5 Research questions

To develop a machine learning classifier that can differentiate between normal and malware traffic, it is necessary to have information that can represent this goal. The main focus of this thesis will therefore be feature extraction and feature selection. This focus is reflected in the following research questions:

1. What are resilient network features for malware detection in TLS encrypted traffic?
2. How can these features be extracted from the network traffic?
3. Which feature selection and classification algorithms provide the best classification performance?

1.6 Scope and Contributions

The scope of this research is to look into detecting malware traffic in encrypted traffic, without having to decrypt, by utilizing classification algorithms. Our focus in this thesis is in the features used in the classification algorithms and not the algorithm itself. To answer the first two research questions, it is necessary to look into existing research of detecting malware in encrypted traffic and defining resilient features in the same field. Defining and finding resilient features is performed with a literate review of related research and tactics described the MITRE ATT&CK™ framework and the theory of the Pyramid of Pain model [20, 52]. Feature selection will be used to identify which of these features are most important to achieve the goal. The absolute goal of this research is to detect malware in TLS encrypted traffic, and it is therefore important to produce results about the classification performance. The tested classification algorithms are chosen based on earlier research.

1.7 Thesis outline

The thesis is divided into individual chapters covering their part of the structure. This section provides an overview of these chapters and contain:

Chapter 2 - Background (p. 4) provides an introduction to the essential topics used in this thesis. First, we start with an introduction to Network Security Monitoring (NSM). Then, we are giving a brief overview of malware taxonomy and evasion techniques for Command and Control (C2) in TLS encrypted traffic. Lastly, we introduce the field and processes of the machine learning field with a focus on classification.

Chapter 3 - Related Work (p. 16) consist of the related research for detecting malware in TLS encrypted traffic. The knowledge provided in this chapter is building stones for this thesis.

Chapter 4 - New methodology for detecting malware in TLS traffic (p. 22) explains the methodology used in this thesis. We start with defining resilient network features for detecting malware in TLS traffic down to the choice of different classification algorithms.

Chapter 5 - Experimental setup and classification results (p. 34) we explain the lab environment and results from data exploration, feature selection, and classification.

Chapter 6 - Discussion (p. 42) we analyze our findings regarding our research questions. Further, we look into the limitations of our research.

Chapter 7 - Conclusion (p. 45) concludes the thesis based on the discussion.

Chapter 8 - Future work (p. 46) looks into the subproblems identified throughout the work of thesis, and define new exciting research fields in this research area.

2 Background

The background chapter present concepts of network monitoring, then a brief introduction to malware taxonomy and up to date challenges with detecting C2 traffic in TLS encrypted network traffic. Finally, machine learning theory is introduced with a focus on classification.

2.1 Network security monitoring (NSM)

The primary goal of NSM is to detect and respond to intrusion before they damage the business goals. Principles included in NSM to achieve this goal are: collecting data, analyzing the data, and escalation of indicators of intrusion [19]. Further in this section, the focus is on collecting and analyzing raw network traffic. However, this process also includes the collection of network application logs and other logs relevant for detecting and responding to intrusions. One important success criterion of the NSM process is to identify where it is essential to collect network traffic. Places to collect network traffic is from the internet, the DMZ (externally exposed services), and the internal network. A system that collects and analyzes network traffic for potential malware traffic should, at a minimum, analyze the network traffic from the internal network. The internal network is where clients run and pose the highest risk of being infected with malware. In the following subsections, two of the most relevant principles are described: Collecting network traffic and analyzing the traffic. Escalation of possible incidents are an essential part of NSM but is out of scope for this thesis since our focus is to collect, analyze and detect malware in TLS encrypted networks. However, escalation processes need to be defined when analyzing events in a corporate network.

2.1.1 Collecting network traffic

The first choice that has to be made in the collection phase is to identify where to capture the network traffic. Then, it is necessary to decide if a network sensor is going to be deployed in inline or passive mode. Inline means that the actual network traffic is passing through the sensor, and can automatically respond to intrusion by dropping malicious traffic. An intrusion detection system deployed in this way is called a Intrusion Prevention System (IPS). The second mode is passive, and the concept is that a network sensor gets a copy of the network traffic. In this thesis, the passive mode is seen as the most suitable method because of computationally heavy machine learning algorithms are not able to meet the expectation of real-time detection that is required by a IPS. In the research by Hallstensen [35] on multi-sensor fusion in intrusion detection, they included this is a limitation of their method due to the design of distributed sensor fusion. In this thesis, a single network sensor is used, but it is required to correlate data from different sources, and process the correlated data.

Three different approaches to collecting traffic in passive mode are highlighted: port mirroring on a network switch, a network Test Access Point (TAP), and directly capturing the traffic on a client or server.

- **Port mirroring** is a method that can be used on enterprise network switches by sending a copy of the seen traffic to another port [19, p. 48]. A drawback of this method is that the switch makes a programmatic copy of the traffic, and when the port gets oversubscribed, packets can be dropped or sent out of order [32]. This problem can lead to a false representation of the observed network packets.
- **Network TAP** is a network device that is placed inline between two networks links and makes an exact copy of the observed traffic [19, p. 48]. This method does not suffer from the disadvantage of port mirroring, as the traffic is mirrored as an exact copy.
- **Capturing on the host** is a method that can be used to collect the traffic that is observed on the host or the traffic that passes through the host.

2.1.2 Analyzing the network traffic

Analyzing the network is performed by a network sensor which can interpret and analyze different types of data. In this section, three different types are discussed: Session Data, Packet String (PSTR) Data, and alert data.

Session data does not include detailed information about the traffic, but instead it includes a summary of communications between two network devices. Information that is collected with this type is: Source and destination IP address, source and destination network ports, which protocol, the number of packets and bytes, TCP flags, and direction of the traffic [72].

PSTR data includes human-readable data extracted from the network traffic and can be seen as a method to normalize network protocols [70]. Examples of normalized fields are the URL and User-Agent string in HTTP traffic, but also the SNI and certificate in TLS traffic. Suricata and Zeek (former Bro) are two accepted applications that can analyze and normalize a variety of network protocols; HTTP, TLS, SMTP, SMB, etc. [61, 10].

Alert data, in the context of network traffic, is generated by a NIDS, and its detection capability can be divided into two primary categories: signature-based and anomaly-based detection [71]. Signature-based detection is the oldest form of intrusion detection and is performed by searching the network traffic for specific patterns. Snort and Suricata are systems that provide this type of functionality by using a similar rule format [69, 10]. In comparison with signature-based detection, anomaly-based detection relies on observing network occurrences and discerning anomalies traffic through heuristics and statistics [71].

2.2 Malware taxonomy and C2 detection

To understand how NIDSs can detect malware traffic in the network, it is important to look into the differences between malware and their different capabilities. First, in this section, is a brief introduction to the malware taxonomy and malware analysis. We then list the relevant C2 techniques for malware in TLS encrypted traffic.

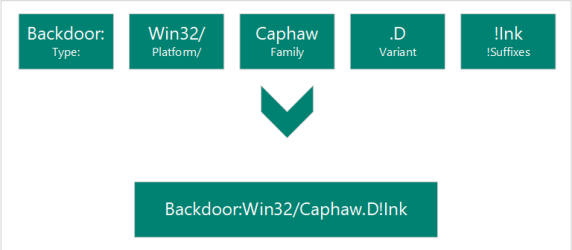


Figure 1: Computer Antivirus Research Organization (CARO) malware naming scheme [51]

2.2.1 Taxonomy

Malware is a general term for computer code that is developed with the purpose of executing harmful actions. Since various types of malware have different functionality, a set of generic groups is created with the purpose of grouping malware [75].

- **Backdoor** allows unauthorized access for an attacker to a compromised host.
- **Botnet** allows an attacker to instruct a set of computers to perform the same actions.
- **Downloader** the only action performed is to download additional malware.
- **Information-stealing malware** collect and exfiltrates information from the compromised host.
- **Launcher** used to launch other malware.
- **Rootkit** used to conceal the existence of other code.
- **Scareware** scare the users to buy something.
- **Spam-sending malware** used to send spam from the infected host.
- **Worm or virus** copy itself and propagate to other machines.

These groups give a brief description of the malware capabilities, although they tend to fall short when describing the full functionality and purpose. Instead of only using these types, anti-virus vendors often provide their own naming schemes for malware. It has also been proposed to use a standardized naming scheme called Caro [2].

Based on the scheme in Figure 1, it is possible to extract five information fields that are used to describe the malware [51].

1. **Type** is the type of malware, and can be correlated with the list above.
2. **Platform** is the type of computer platform and CPU architecture that the malware targets. In this example, the malware targets Windows platform running on a 32 bit CPU architecture.
3. **Family** is the given family name for the malware.
4. **Variant** is the variant of the malware, as there may exist several pieces of malware in the same family with some differences in functionality.
5. **Suffix** is the common filename suffix used for the malware.

This framework is mostly used by Microsoft [51] but is good for presenting important metadata for

malware.

2.2.2 Malware analysis

Analyzing malware is an art of dissecting the malware to understand how it works, how to defeat and eliminate it [75]. The analysis results can be used to develop host and network-based signatures that can detect an infected host, and respond with actions to mitigate the threat. To acquire these signatures, it is possible to perform two different analysis methods.

Static analysis is examining the malware without executing the actual code. Usually, the analyst does not have access to the raw source code and will analyze the raw binary instead. When analyzing compiled binary files, you are limited to look into CPU instructions of the malware and data presented in the file. This type of information can reveal information like C2 domains, but since malware authors often obfuscate their code with encryption, this type of information can also be encrypted [79].

Dynamic analysis is examining the behavior of the malware when it is executed. Information that is extracted from this type of analysis include changes the malware performs on the infected host and network traffic from the malware. Same as with static analysis, malware authors are developing functionality that makes it challenging to analyze the malware, even when it is executed. An example of such functionality is to look for malware analysis tools in the environment the malware is executed. It is observed that malware checks for files, registry keys, or processes related to malware analysis tools [31].

Performing these methods can be achieved with manual analysis, but also in a malware sandbox that performs automatic analysis of the file. A sandbox is commonly used when it is necessary to analyze a large amount of malware. When performing dynamic analysis of malware, it is important to follow a routine that ensures the integrity of the malware results and make sure that the computer being infected is not infected in the subsequent analysis. Figure 2 describes such a routine, and by following this routine, it is possible with a high degree of certainty that the result of the malware analysis is not affected by previous analysis and the result is produced by a standardized method. Using a virtual machine in this process makes it easy to define a baseline (a clean state) and revert to the baseline, and extract important artifacts. There exist both public sandboxes and open source tools that perform this process, which automatically provides a report of the analyzed malware [4, 60]. These platforms can also extract artifacts such as network capture, memory, and disk dump, that can further be analyzed by humans and machines. Different tools have been developed during the years to make this process easy, such as Cuckoo, an open source tool for automatic malware analysis [60].

2.2.3 C2 traffic and evasion techniques

The command and control traffic performed by malware is defined as the fifth phase in the cyber kill chain model [48]. In this phase, the adversaries can communicate with systems under their control in victims' networks – also known as C2 traffic. The characteristics of C2 traffic are often

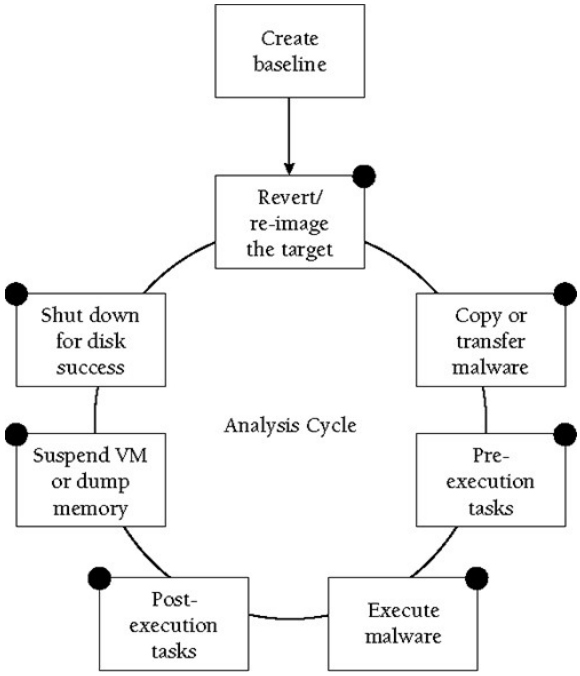


Figure 2: The analysis cycle[45]

linked to the specific adversary, as they use various levels of covertness to evade detection. The MITRE ATT&CK™ framework includes a list of 22 common techniques for command and control traffic. All of these techniques are not relevant in the scope of this thesis, as the analyzed traffic is mostly related to TLS. This thesis focuses on the techniques listed below. The name of techniques is chosen according to Mitre, and Mitre's technique identifier is presented in the parenthesis.

- Commonly Used Port (T1043) and Standard Application Layer Protocol (T1071) - Adversaries use network ports and protocols, such as 443 (HTTPS), to bypass firewall restrictions and intrusion detection systems [53, 54]. By using HTTPS, the traffic blends with normal user traffic.
- Domain Fronting (T1172) - is a technique that exploits the routing scheme in Content Delivery Networks (CDNs), or other types of services that host multiple domains, to obfuscate the intended destination of the HTTPS traffic [55]. This method is carried out by using a legitimate domain name in the TLS SNI and using the host header in HTTP as the actual destination domain. If both of these domains are under "control" of the CDN, the CDN will most likely forward the request to the target domain.
- Remote Access Tools (T1219) - Adversaries use legitimate tools for remote administration, such as Team Viewer, Go2Assist, and LogMeIn, to establish a persistent command and control channel [56]. Many of these tools go undetected in a network, as they might also be used for

legitimate purposes.

- Web Service (T1102) – Adversaries are using legitimate web services to relay commands to a compromised host [57]. A variant of this technique is to utilize social media platforms that are commonly used in organizations to deliver the commands, e.g., twitter posts [38].

2.3 Machine learning

Machine learning is a subfield of artificial intelligence, which can be explained as a method to automatically create models of the underlying data of a given problem. Two common subfields of machine learning are supervised learning and unsupervised learning. The difference between these is that supervised learning utilizes data labeled with the desired target condition, while unsupervised learning does not. In simple terms, this means that in supervised learning the algorithm can calculate its estimated target, compare it to the actual target, and subsequently correct itself.

The most frequently used method in machine learning and a subgroup of supervised learning is classification [42, p. 5]. This method involves an object that is described with a set of related features. The purpose is to assign this object with a classification label from a defined set of two or more labels. With only two labels in the set, the problem is called binary classification.

In unsupervised learning, the most popular method is clustering [42, p. 13]. This method does not take advantage of the object's target but determines similar clusters of the learning data. The number of similar clusters can be predefined, or be determined as a part of the clustering algorithm.

Further in this section, the focus is on the field of classification since the dataset used in this thesis is labeled, and we have chosen to utilize these labels with classification algorithms. However, several of the methods apply to the whole field of machine learning.

2.3.1 Classification

The main idea of classification is to extract data from observations and use that data to classify (label) the object. The problem of performing an activity with certain weather conditions is a common problem illustrated with a learning algorithm called decision trees [66].

Table 1: Example of training data for a decision tree

Features				Label
Outlook	Temperature	Humidity	Windy	
Sunny	hot	high	false	Play golf
Sunny	hot	high	true	Don't play golf
overcast	hot	high	false	Play golf
rain	mild	normal	false	Don't play golf
rain	cool	normal	true	Don't play golf

In this problem, there are different weather conditions used as features and the target label is to play golf or not to play golf. Table 1 includes training data for an example of this problem. Using this data, it is possible to build a decision tree, such as Figure 3, that can be traversed to predict if it is a good day to play golf or not. This problem has few features and a small dataset, but in other

problems, it can be more than 1000 features and millions of learning examples, making it more difficult to interpret the results of the learning algorithm.

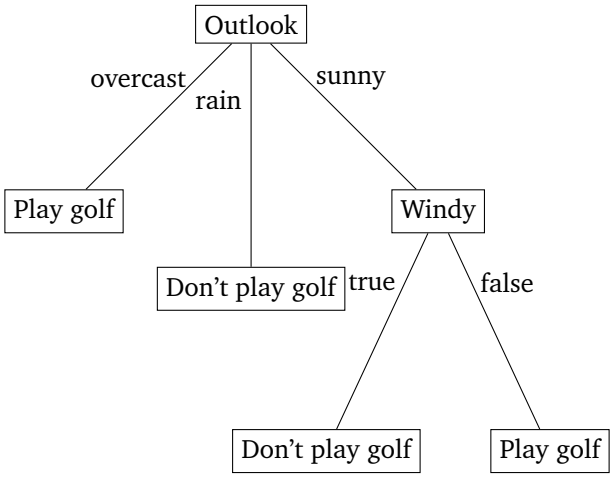


Figure 3: Example of decision tree

As problems are often much more complicated than the golf example, it is important to follow a process that can be used to improve the classifier and interpret the results. Seen in Figure 4, the classification process starts by dividing the data into two sets, a training set and a testing set. As data from observations include data that do not represent the target, it is important to perform pre-processing. The purpose of pre-processing is to remove noise, normalize the data, handle features with missing values, and other operations that contribute to defining a compact representation of the pattern [41].

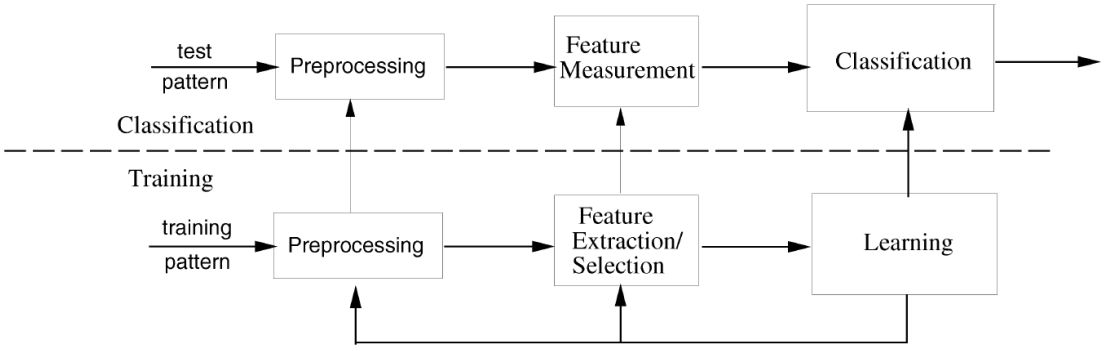


Figure 4: The process of classification[41]

In the training part of Figure 4, the feature extraction and feature selection are performed to extract the appropriate features for input to the classification [41].

Feature selection is important to evaluate the quality of the features, and further, show the usefulness of individual features in predicting the target of a problem [42, p. 153]. The outcome of feature selection can be manually analyzed or included as an automatic step before the classification algorithm. Feature selection is a crucial task in the process of developing and understanding the results of a classification algorithm. In the next section is different methods of feature selection explained.

In classification, it is required to choose a suitable classifier for the problem, but based on a well-known theorem called *No Free Lunch Theorem*, there is no reason to favor one algorithm over another. The superiority of one algorithm is due to the problem and distribution of the investigated data [27]. Summarized, this theorem states the importance of focusing on the most important aspects; prior knowledge, data distribution, amount of training data, and the cost functions applied in the algorithms. This shows the importance of the feedback part of the method.

2.3.2 Feature Selection

The number of features used in learning samples has an impact on how well the data describes the object of the classification also has an impact on classification performance. *More is less* is an expression used in this field, and means that fewer features can provide better classification performance [46]. This fact is supported by a phenomenon called *Curse of Dimensionality*, which refers to problems classifying unseen observations from a limited number of training examples [46].

Therefore, the goal of feature selection is to choose the desired subset of features by optimizing some criterion, such as classification accuracy [42, p. 199]. The following two features selection methods are covered in this section:

Filtering

Filtering is the simplest and fastest method of the two in this section and utilizes a function to calculate the quality of a feature [42, p. 199]. Each feature is given a quality score, and it is necessary to define in advance how many features to keep or provide a cut-off value for the quality score.

Wrapper

The Wrapper method is slower than the filtering approach, as it utilizes machine learning algorithms combined with cross-validation [42, p. 199]. Different subsets of features are used in the classification algorithm until an optimum is found. Typically, greedy search algorithms are used for searching the space for subsets of features.

2.3.3 Validation

An essential part of the machine learning process is to validate the classification performance of a classifier and compare classifiers against each other. In this section, there are several performance evaluation metrics for binary classification listed and described.

Two different methods are commonly used to evaluate a classifier; splitting the dataset into two parts, for example, 70% training set of examples and 30% independent set of testing examples, or use a method called k-fold cross validation [42, p. 81]. Using the k-fold cross-validation, the dataset gets divided into k different folds. For each fold, use the fold for testing and use the remaining k-1

folds for training. This process is repeated in k iterations, where each fold is used for testing. Ten is a commonly chosen number for k. Then after ten iterations, the results are averaged to provide one estimate [42, p. 83]. In this context, results are different performance metrics for classification.

Methods to estimate the performance of the classifier are essential to be able to answer the research questions. Typical metrics for a binary classification problem are accuracy, confusion matrix, sensitivity, specificity, ROC curve, recall, and precision. First, to understand these metrics, it is necessary to understand the numbers used in the calculation.

- **True positive (TP)** - the number of correcly classified as positive
- **True negative (TN)** - the number of correcly classified as negative
- **False positive (FP)** - the number of falsely classified as positive
- **False negative (FN)** - the number of falsely classified as negative

Accuracy and Confusion matrix

The accuracy is defined as the value that describes the success of the classification problem [42, p. 70]. The following formula is used to calculate the accuracy:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Table 2: Example of confusion matrix

	Classified as		SUM
Correct class	Play golf	Dont't play golf	
Play golf	3 (TP)	2 (FN)	5 (POS)
Dont't play gol	1 (FP)	2 (TN)	3 (NEG)
SUM	4 (PP)	2 (PN)	8

Instead of looking at the overall performance, a confusion matrix is a method to visualize the performance of the classifier in a table. Table 2 is an example of the matrix, where the POS label represents the number of positive examples, and NEG represents the number of negative examples. PP shows how many examples were classified as positive and PN as negative.

Sensitivity, Specificity and ROC curve

The sensitivity, also called True Positive Rate (TTP) and Recall, metrics calculate the relative frequency of correctly classified positive examples, and specificity, also called True Negative Rate (TNR), calculates relative frequency of negative classified examples. The following formulas are used to calculate these scores:

$$Sensitivity = \frac{TP}{TP + FN} \qquad Specificity = \frac{TN}{TN + FP}$$

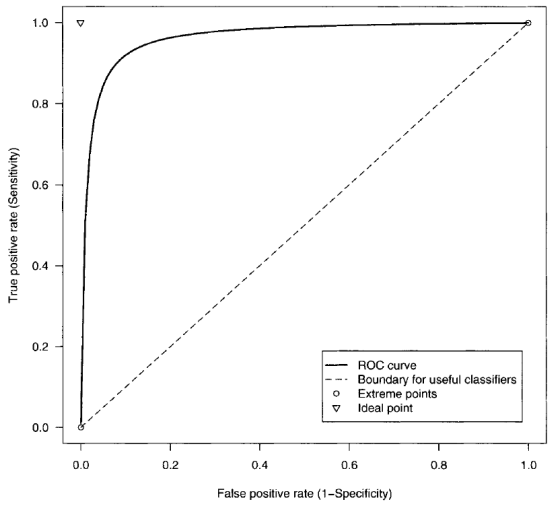


Figure 5: Example of ROC curve [42, p. 77]

Finding an optimal classification algorithm can be performed by looking at the relationship between sensitivity and specificity. Using a ROC curve can be used to display this relationship between these two metrics. Figure 5 shows an example of a ROC curve, and the goal of this analysis is to increase the space under the line. The space under this line is reflected with a metric called Area Under Curve (AUC).

Recall, Precision and F-measure

The recall measures are equal to sensitivity, as described above. The precision metric estimates the portion of correctly classified examples that were classified as positive [42], where F-measure calculates the harmonic average of recall and precision. The following formula shows the calculations of F-measure and precision:

$$Precision = \frac{TP}{TP + FP} \qquad F\text{-}measure = \frac{2 * Recall * Precision}{Recall + Precision}$$

Important to know about these three measures is that they do not account for true negatives. However, in IDSs the true negatives are not as relevant compared to true positives, false positives, and false negatives. The number of true negatives contains how many times, e.g., a network flow classified as non-malicious when it was a normal network flow. This number should be high compared to the other metrics in a corporate network since most of the traffic is regular user traffic.

2.3.4 Challenges

In this section, several challenges in machine learning and classification are encountered on; *Section 2.3.1 - Classification (p. 9)* included the *No Free Lunch Theorem* , *Section 2.3.2 - Feature Selection (p. 11)* looked into the *Curse of Dimensionality*.

Other challenges that are important to consider in the field of machine learning are overfitting, underfitting, concept drift, and adversarial examples. The two last concepts are not studied in this thesis, but are included in Section 8 as topics that can be further explored.

Overfitting and Underfitting

Overfitting is a term used to describe the behavior of a classifier that achieves high classification accuracy on the training data but performs poor on the testing data. This problem can occur when the number of parameters is increased in favor of reducing the bias. However, then the variance is often high. Bias is error originating from the learning algorithm, and variance is the error arising from the learning data [42]. With overfitting, the classifier does not learn the true nature of the data and fails to generate a generalized model of the data.

Underfitting occurs when the model is too generalized, and the classification performance is therefore low [42].

A solution to these problems is to divide the dataset into a training and a testing dataset and utilizer methods such as K-fold, described in Section 2.3.3.

Concept drift

The natural behavior of data changes over time, as well as network traffic. New applications are introduced into the network, or new types of malware are created. Due to a concept called concept drift, it is important to consider these changes. This refers to a non-stationary learning problem over time [80]. One of the problems when dealing with concept drift is to differentiate between concept drift and noise in the data. Some learners might overreact to noise and interpret it as concept drift. It is important that learners combine robustness to noise and sensitivity to concept drift [78].

Literature defines two types of occurrence for concept drift: a sudden change and a gradual change over time. For example, sudden changes can occur in situations when network protocol is upgraded, such as a major version of the TLS library. Instead, gradual changes can be due to malware evolution, where malware authors have minor improvements in their code. The cause of concept drift is not always related to a change of the target concept, but instead the underlying data distribution. The need to change the classification model due to the data distribution is called in literature virtual concept drift [78].

A possible solution to counter concept drift is to use online learning instead of batch learning or also re-training of the models if it cannot be done as online. While batch learning creates a trained model, online learning creates a learnable model on fly and updates the model as new instances are processed [78].

Adversarial examples

In the field of using machine learning in a IDS, it is essential that the system itself is resistant against attacks. If not, adversaries can develop crafted methods that can evade detection and result in false negatives, the intrusion not being detected. In machine learning, it has been identified that several machine learning models are vulnerable adversarial examples. This vulnerability is exploited by examples which are slightly different from correctly classified examples, that get misclassified [33]. This problem exposes fundamental blind spots in the training data. One of the problems can be classification models learn how to classify the testing set with high accuracy, but do not learn the true nature of the data.

3 Related Work

In the previous chapter, the topics relevant to this thesis were presented. These topics build the essential knowledge to understand the next chapter, related work. In this chapter, related research is highlighted; fingerprinting of TLS clients, and how to detect malware in TLS encrypted traffic by using classification methods.

3.1 TLS client fingerprinting

The purpose of TLS is to provide a secure channel between two communicating peers by ensuring that the server is always authenticated, that the data is only visible for the communicating peers, and that the data is not altered without being detected by the protocol [67]. However, some information needs to be transmitted in clear text, as it is related to encryption options, which need to be agreed on before the encryption session starts.

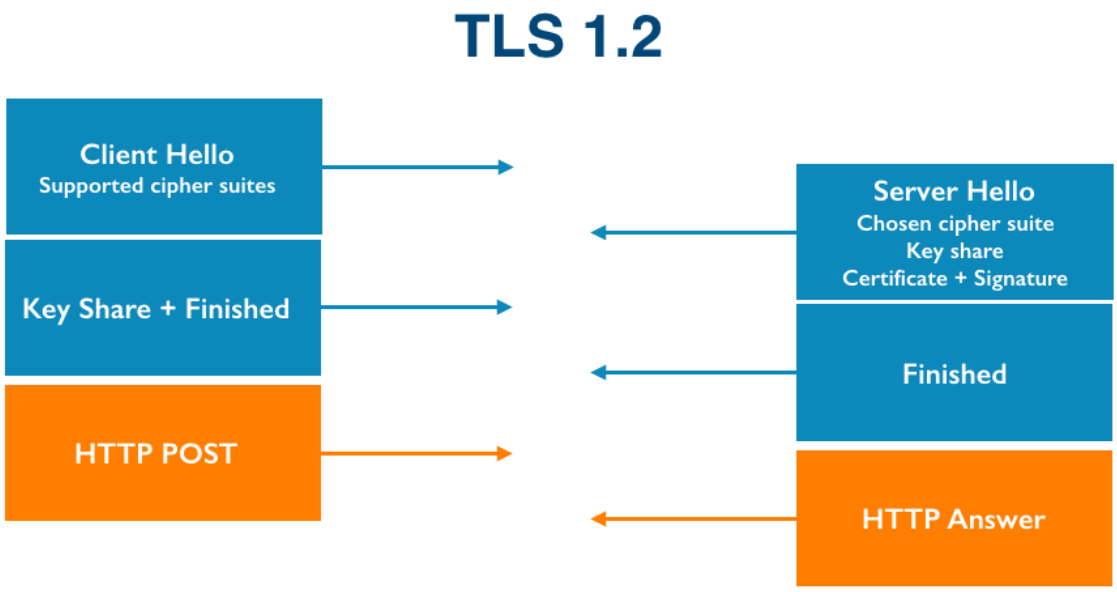


Figure 6: Sequence diagram of TLS 1.2 [23]

Figure 6 shows that the client initiated the first packet with a client hello. This packet includes the supported client configurations. Then the server sends the chosen options and certificate information. Followed, is the key-share and then the encrypted session can be started. In this figure, the

blue color represents information sent in cleartext and orange color represent encrypted data.

The main idea of TLS client fingerprinting is to fingerprint which TLS client application that is being used. Different TLS clients use different options of the TLS protocol, and it is therefore possible to fingerprint the TLS client application. These settings are possible to observe as cleartext information in the network traffic, and are transferred in the TLS client hello packet, which is the first packet when establishing a new TLS connection [25].

Listed below, are the most relevant TLS1.2 options in the hello packet. This is also the most supported version of TLS, even though the latest version is TLS1.3 [43]. Most of the same information can be found in TLS1.3, but fields like compression are now listed as legacy and are not used in this version [67]. Also, in Figure 6 it is possible to see that the certifcate is transmitted in cleartext, but in TLS1.3 is the certificate also encrypted.

Cipher Suite is the client's list of supported cryptographic options. The first option is the preferred one.

Compression Method is the client's list of preferred compressions algorithms. This list is sorted based on preference.

Client version is the preferred TLS version for the client. This should be the latest supported version.

Extensions is the field used by the client to request extended functionality by the servers. This field includes an extension type, which is used to specify which extension is used, and extension data, the information to the specific extension.

In one of the early studies of TLS client fingerprinting, the cipher suites were correlated with the HTTP User Agent and used to identify the possible client applications [37]. From this study, it was possible to identify the client applications based on this TLS option. However, it was shown in this study that a single client fingerprint could match to several HTTP user agents, and therefore several client applications [37]. Lesson learned from this study is that there exist distinguishable information in the TLS hello packet, but a single option contains too little information to generate a unique fingerprint.

By using several options in the TLS hello packet, it gets a more distinct fingerprint. This has resulted in a fingerprint standard, called JA3, that combines the following options in the hello packet; SSLVersion (Client version), Cipher (Cipher Suite), SSLExtension (Extensions), EllipticCurve, and EllipticCurvePointFormat [9]. The fields related to Elliptic Curve are extensions to the hello packet and should be used if the client proposes to use Elliptic Curve Cryptography [59]. The Elliptic Curve field indicates the set of elliptic curves that are supported by the client. EllipticCurvePointFormat indicates a set of point formats that the client can parse.

Using these five fields, it is possible to generate a MD5 fingerprint.

These fields are difficult to interpret by looking at them but can be converted to meaningful information. Converting the SSLVersion number to hex, it is 0x301. By following the TLS speci-

```
Fields -> SSLVersion,Cipher,SSLExtension,EllipticCurve,EllipticCurvePointFormat
Values -> 769,47-53-5-10-49161-49162-49171-49172-50-56-19-4,0-10-11,23-24-25,0
MD5    -> ada70206e40642a3e4461f35503241d5
```

Figure 7: JA3 scheme

fication, the first number is the major version and the second is the minor version. This number represents TLS version TLS1.0, as this version is a minor modification in the SSL3.0 protocol [26].

This standard has been proven to be useful in malware detection, and lists of JA3 fingerprints are shared as IoC [11]. JA3 implementations are found in industry-leading technology, such as Suricata, Zeek (Bro), and Moloch [1, 10, 8].

These research fields show that there exists cleartext information in the TLS communication that can be used to differentiate between normal users and malware traffic. The five fields used to generate the JA3 fingerprint, have shown to include valuable information and should be useful to use as a feature in supervised learning. This information is also, to a certain degree resilient to tampering compared to the common HTTP "fingerprint", User Agent. User Agent is a text string that can be manipulated easily, but the TLS fingerprint options are used by the protocol to establish a connection, and tampering might lead to protocol errors.

3.2 Machine learning

In recent years there has been an increase in research on using computational methods to detect malware in encrypted traffic. The research of using supervised learning in this domain has shown that it is possible to provide results with > 99% accuracy [13, 14, 15, 76, 77]. These results are based on binary classification, and the following algorithms have shown to be best suited; l1-logistic regression, random forest ensemble, CART, and gradient boosted trees. Even with good classification accuracy, it is important to ensure that the number of false positives is below a suitable threshold. This threshold can be defined as the number of false positives that can be manually analyzed by a security analyst. Also worth mentioning, is that false positives generated by computational methods are more difficult to validate since models are less interoperable than patterns from classic IDS.

3.2.1 Features

In terms of this research, all features are derived from the network traffic, and a source is a type of network log that contains data that can be pre-processed to features. Using appropriate types of features have a much higher impact than using different classifiers [14].

Network flow

A network flow is the same as Session Data, described in Section 2.1.2, and contains summarized network information. McGrew and Anderson [50] included as a part of the experiments to test only flow-based features to classify malware in encrypted traffic. These features consisted of inbound and outbound bytes, inbound and outbound packets, source and destination port, and the duration

of the flow. These features are information that can be extracted with network devices that support NetFlow or IPFiX. In their research, these features were extracted with their open-source tool called Joy. The result of using only flow-based features was 95.68% accuracy, but with 0.01 False Discovery Rate (FDR) they achieved 0.05% and was the worst test case of their experiments.

Lokoč et al. [47] extracted similar information from HTTPS proxy logs, but with some differences; the number of bytes sent from the client to the server, number of bytes received by the client from the server, the duration of the connection in milliseconds, and the time in seconds elapsed between start of the current and previous request by the same client. Also from HTTPS proxy logs, Prasse et al. [65] extracted flow-based features and generated the following features: one-hot encoding of the port value, duration, time gap from the proceeding packet, and the number of sent and receives bytes. In their research, domain-name related features were also included. They conclude that flow-based features performed best in combination with other features and not alone.

Cleartext information in TLS

Features extracted from TLS traffic are taken from the hello packet (explained in detailed in Section 3.1), the server response, or the certificate. In [13], they used several features on client and server side. On the client side, they used properties like the list of offered cipher suites, advertised extensions, and the client's public key length. On the server side, they used properties like selected cipher suites, supported extensions, number of certificates, number of SAN names, validate in days, and if the certificate is self-signed. These property values combined result in 198 features.

Several studies looked into using features derived from the TLS certificate. A part of Střasák [77] study, they experimented with only using certificate features to detect malware traffic. This study concluded that it was difficult to detect malware traffic by only relying on certificate information, and the best results were 76.42% accuracy.

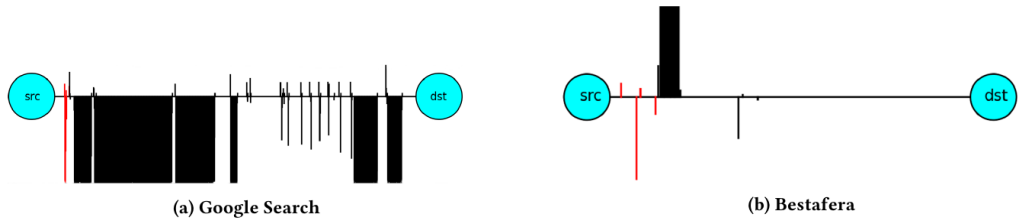
As a part of the TLS handshake, a domain name is also often communicated in cleartext - called SNI. SNI is an extension used to specify which server to reach if there are several HTTPS websites behind a single IP address. Prasse et al. [64] extracted features from the observed domain name in the Server Name Indication (SNI). These features included the ratio of vowel changes, the maximum occurrence ratio of individual characters for the domain and subdomain, the maximum length of substring without vowels, presence of non-base-64 characters, the ratio of non-letter characters. In addition, the domain name string was decomposed into overlapping substring as 2-gram features.

In Section 3.1 it is mention that certificate information is not accessible in TLS1.3. In the same version, a draft of encrypting the SNI in the TLS traffic is proposed [68]. This feature is already supported by Cloudflare [22], a company that delivers CDN along with other services. If this functionality is practiced, it will not be possible to extract features from the SNI.

Network Packets

Packet features are features that are extracted from each packet in a flow. In this section, two different use cases are explained, looking into the content of the packets and looking into the metadata of the packets.

Anderson and McGrew [13] used the byte distribution of packets payload as a feature. Byte distribution uses a 256-byte length array used to keep track of all the bytes in the payload. The feature has 256-byte distribution probabilities. The purpose of this method is to extract information that described protocol usage, use of encryption protocol or compression.



(a) Google Search (b) Bestafera

Figure 8: Comparison of TLS packet lengths and inter-arrival times for a Google search and data exfiltration of the Bestafare malware [14]

The second method is to look into the metadata of the TCP packets. The core principle of features based on this data is to collect TCP packets and extract size and inter-arrival times. Figure 8 shows the principle of capturing this information, as it is easy to see the differences between traffic towards Google search and traffic from the Bestafare malware. In this figure, the upward axis is data sent from the client towards the server, and the downwards axis is data transmitted from the server towards the client. In the Google search traffic, more data is sent to the client than the Bestafare malware. In the malware traffic, is one peak of traffic sent from the client and, according to this research, this peak is related to data exfiltration.

In the research of McGrew and Anderson [50], the authors captured only the first 50 packets of each flow, and they did not include retransmissions and payload with zero-length. The feature representation is accomplished with a Markov chain. Classification with this feature alone provided an accuracy of 93.84% but drastically decreased to 0.31% with a 0.01% false discovery rate.

Stergiopoulos et al. [76] expressed that they could to detect encrypted malware features with fewer features than the study of [15]. The following features were used: packet size, payload size, payload ratio, ratio to the previous packet, time differences between packets. In their experiments specific to detect encrypted malware traffic, they used K-nearest neighbors (KNN) and Classification And Regression Tree (CART). Both classifiers presented good results in terms of accuracy, 99.63%, and 99.63%.

Use of side channel features in TCP seems to have a considerable advantage in the field of detecting malware in network traffic. In the first research, it is shown that the feature used alone is not usable, but in the next study, this is not the case. They did not use this feature type in the same way and not on the same dataset, and the results are not comparable.

Cleartext protocols

In Anderson and McGrew [13]'s research, HTTP requests were collected for 5 minutes for each TLS flow to collect all of the observed HTTP headers. The headers were represented in a feature vector, and if any of the headers were observed with a value, the feature was set to 1. Seven different features were used; Content-Type, User-Agent, Accept-Language, Server, and Code.

The same research used also features extracted from the Domain Name Server (DNS) traffic. These features consist of binary vectors for common suffixes and Time To Live (TTL) values, the number of numerical and non-alphanumeric characters, the number of IP addresses returned in the DNS response, and if the domain name was listed in Alexa list. This list includes the most visited websites [12]. The authors concluded that DNS features provide valuable, discriminatory information that gives additional context to increased viability and develops highly accurate machine learning classifiers.

In spite of the discussion above, there are some practical reasons to exclude DNS features. Currently, there exist two specifications which ensure encryption of DNS traffic, DNS over HTTPS (DoH) and DNS over TLS (DoT). DoT implements TLS tunneling of DNS traffic over port 853 [36], while DoH tunnels DNS traffic as HTTPS traffic. Use of DoH is observed as normal HTTPS traffic.

4 New methodology for detecting malware in TLS traffic

The purpose of this chapter is to discuss and describe the new methodology used in this thesis to answer the research questions. The structure of this chapter is a top-down approach, where the choice of features and dataset is first presented, followed by feature extraction and pre-processing Classification and validation methods are discussed last. Quantitative research is selected for this study and involves that our results must be able to measure and apply statistical methods [44]. This type affects the methods described in this chapter and have a high impact on the justification of a method.

Following is a brief overview of the steps in the methodology, used to conduct the experiments:

1. Identify resilient network features
2. Collect raw network traffic from public sources
3. Generate network logs from the traffic
4. Correlate logs and generate correlated flow examples
5. Identify the ground truth of the data with data exploration
6. Apply feature selection to identify the most important features.
7. Apply machine learning algorithm
8. Validate results

4.1 Resilient network features

Before we dive into the method of normalizing network traffic and extract features, it is necessary to determine which features are going to be extracted. The goal of the first research question was to identify resilient network features to detect malware in TLS traffic. Therefore, this section focuses on defining resilient network features based on the background and related work chapter.

In this thesis, a potential IDS continuously analyze network traffic for malware events, and it is therefore important that the system can detect threats if the actors use evasion techniques or change their tactics. The change of tactics can be a result of the actors knowing the system analyzing encrypted traffic, and the system should therefore be resilient against evasion attacks that make malware traffic look similar to normal traffic.

In 2.2.3, it is described how threat actors use evasion techniques in TLS based C2 traffic. These techniques make it challenging to use static information such as SNI, certificate, HTTP, and DNS data since threat actors try to blend their traffic into normal traffic. Also, Střasák [77] concludes that it is difficult to detect malware traffic solely based on certificate features.

By using only these static features, it would be possible to craft packets identical with normal user traffic. The static features are therefore not used in this thesis. Instead, we only rely on features that describe the behavior of TLS encrypted traffic. Behavior-based features are difficult to modify

from a threat actor's perspective, and the behavior-based features can be related to the *tool* level in the *Pyramid of Pain* [20].

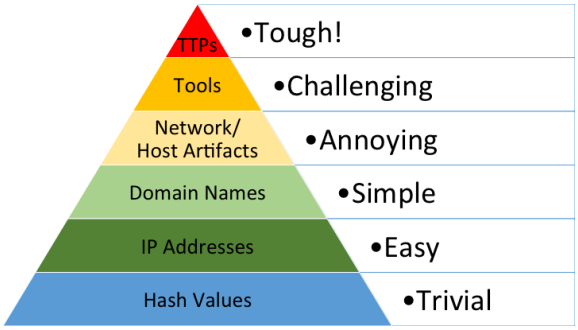


Figure 9: The Pyramid Of Pain [20]

The *The Pyramid of Pain* is the diagram in Figure 9, the idea is to show the relationship between indicators and the defender's value of detecting them. The different levels in the Pyramid symbolize indicators, where the indicators in the bottom are easy to detect and do not cause much pain towards the adversary since they are easy to modify. IP addresses and Domain Name are easy for an adversary to modify. Since they are not causing severe pain change, we have not included features extracted from the IP address or Domain Name. We decide to choose features to be in the tool level and will therefore cause pain for an adversary to modify its malware.

In this thesis, the possibility to evade detection with adversarial examples is not studied. Because the chosen features are more challenging to modify, adversarial examples are more difficult to craft and evade detection.

Accordingly, it is chosen to look into features extracted from the JA3 strings, flow-based features, and features extracted from individual network packets. All of them contains information about the behavior of a TLS network flow and have shown great value in existing research. In the next section, we look into the method of acquiring raw network traffic, and the technique used to extract these features from the traffic.

4.2 Data collection

The source of data in this research is raw network capture stored in network trace files. Collecting these trace files can be achieved using multiple methods, and three different methods have been analyzed and discussed. Before the different methods are enumerated, relevant criteria for the traffic are listed below:

- Network traffic must be publicly available.
- A variety of normal and malware traffic must be contained in the traffic.
- Network traffic must represent the real world.

The first capturing method is to collect network traffic by intercepting normal and malware

traffic [15]. A common method to collect the malware traffic is to intercept the traffic from a malware sandbox. If the malware samples are public, it should not be any problem to release the capture. With full control of this environment, it is possible to collect traffic from a huge amount of data and have a variety of different malware families. Malware sandbox is also a common system to use in enterprise networks, and should therefore be close to representing the real world. One can argue that live malware on compromised systems is more realistic and more likely to be executed as intended as malware authors tend to implement checks for malware analyst tools, or if the malware is executed in a virtual machine. However, collecting samples from compromised systems is a method that is difficult to use due to the complexity of labeling the samples correctly. It is also difficult to collect enough samples. Therefore, malware sandbox is the preferred method.

The second method is to manually execute malware, capture its network traffic as well as the traffic that compounds to surfing legitimate websites, which can be captured and used to create mixed packet captures. A considerable advantage of this method is the control of the dataset. However, that method may be time-consuming and challenging in representing real-world traffic.

The third method that could be used is network simulation. The purpose of this method is to develop a program that can simulate both normal and malware traffic. With this method, the size of the dataset is not a concern as it would be possible to generate new traffic and perform live tests. One of the drawbacks is that malware simulations are challenging to develop as most malware families are different, and their behavior is different. It is difficult to capture diversity in malware and normal traffic. To our knowledge, there is no open-source packet generator that can generate TLS encrypted normal and malware C2 traffic.

The fourth method is to capture normal traffic from outgoing connections from a corporate DMZ. This method would represent the normal traffic best, as this is the same traffic that is going to be used in a production scenario. But for academic research, it would be difficult to release the dataset and have repeatable experiments with the same dataset.

The fifth option, the one used in this research, is to use already captured network traffic that is released publicly. Using this method ensures the reproducibility of experiments. A drawback is that the files are too big for manual inspection, and we need to assume that it includes false positives. Without control of the creation of these files, we cannot be guaranteed that the traffic is what it claims to be without doing time-consuming checks. This challenge can be solved to some extent by manually analyzing the traffic on a bigger scale and look for similarities, normal traffic such as Windows API calls, or outliers, possible malware traffic, etc. The traffic is also encrypted, so it is difficult to verify the ground truth of the traffic.

The public PCAPs were downloaded from four different sources, and the primary purpose of using various sources is to be able to create a dataset with diversity. Using different sources can help to get a realistic scenario, where malware traffic used to build the classifier is not similar to the malware traffic that is observed. Different legitimate traffic is also mixed in to identify the differences between a classifier built with normal traffic from the same source and a classifier built from a different source.

In the following bullet points, the different PCAPs sources are listed and briefly described.

Malware Capture Facility Project is a sister project to the Stratosphere IPS project, which is an open source project for a behavior-based intrusion detection system. The sister project captures malware and normal traffic analyzed by the IPS [40]. This project captures malware traffic over a longer period, and in several cases over days. In this thesis' experiments, there were used both malware and normal traffic from this source. The normal traffic was used in similar research in detecting malware in HTTPS traffic [30]. The captures are dated between 2013 and 2018. The legitimate traffic consists of 14 different captures with a total size of 6.6GB, and the malware traffic consists of 262 different captures with a total size of 27GB. All of the malware samples have a related MD5 hash, and by utilizing VirusTotal is it possible to identify the likely malware family of the traffic. However, we should assume that the malware traffic consists of small parts of normal traffic as well. Possible normal traffic in malware dataset is automatic requests performed by the underlying Operating System.

Malware Traffic Analysis is a blog that publishes information and analysis of malicious network captures and includes the network capture of the traffic that is analyzed in the blog post [7]. The network captures listed on this blog are related to ongoing campaigns and threats. The captures used in this thesis' experiments are dated between 2018 and 2019 and consist of 368 different captures with a total size of 2GB.

Hybrid analysis is a web service offering free malware sandbox analysis [4]. This function requires a vetting process, but it did not take long to obtain it for an academic purpose. 1713 different malware captures are used, with a total size of 2.8GB. This set includes only malware samples, and it is the most realistic method used in the process of building the classifier.

Intrusion Detection Evaluation Dataset (CICIDS2017) is an IDS dataset from Canadian Institute for Cybersecurity [74]. This dataset is created with the purpose of testing and validating IDS systems, and includes both benign and common attacks that reassemble real-world data. Benign traffic is created by profiling normal human interactions to generate captures that are similar to real-world traffic. In this work, only the benign traffic from this source was used and got merged into one network capture with a size of 11GB.

4.2.1 Network data normalization

The collected data from the previous section is in a PCAP format, and the data need to be normalized into information that is useful for machine learning algorithms. Two different tools are used for the normalization; Suricata and a custom tool called Metadata Collector developed for this thesis to extract information from individual TCP packets.

The general information extracted from these tools is bidirectional network flows, cleartext TLS information from the hello packet, and the payload size of TCP packets in a flow. These three fields are correlated to a single flow containing all information. The aggregation is performed on the IP and port pairs. In the following three sections, the usage of these tools and correlation are explained in more detail.

4.2.2 Suricata

Both bidirectional flows and cleartext TLS information are possible to extract natively in the version of Suricata used for this thesis, Suricata-4.1.0-Dev. The field that includes the cleartext information is called JA3 and is explained more in-depth in Section 3.1. To ensure that this functionality is enabled in Suricata, the JA3 option must be enabled in Suricata's config, see Figure 10

```
app-layer:
  protocols:
    tls:
      enabled: yes
      detection-ports:
        dp: 443
      # Generate JA3 fingerprint from client hello
      ja3-fingerprints: yes
```

Figure 10: Enabling JA3 logging in Suricata

The output of this value consists of two different fields, JA3 hash, and JA3 string. The hash value is used in data exploration, but not in machine learning algorithms. The string value is further normalized by splitting the different values on the comma.

The functionality to extract bidirectional flows is enabled by default in this version of Suricata, and the following fields are extracted with this functionality.

- **pkts_toserver**: the number of packets sent to the server.
- **pkts_toclient**: the number of packets sent to the client.
- **bytes_toserver**: the number of bytes sent to the server.
- **bytes_toclient**: the number of bytes sent to the client.
- **age**: the duration of the flow.

4.2.3 Metadata Collector

This tool was developed to be able to capture the payload size of the TCP packets in TCP flows. The tool extracts the payload similar to the SPLT method of Anderson et al. [15] but captures the packets based on flows and not TCP sessions. The tool is developed using the programming language Golang, combined with a well-used network library called Gopacket [3]. This library is among others used by Packetbeat, a popular open source network analytics tool written by Elastic [28].

The tool is developed to filter out TCP retransmissions and packets with zero payloads, and not include these packets in the output. Features that the tool can extract are; the payload size (in bytes) and the time differences between packets (in milliseconds) in network flows. These are the same features used in the research by Anderson et al. [15] and have turned out to be essential features in detecting malware in TLS traffic. However, in this thesis, the time differences between packets are excluded due to incorrect timestamps in a significant portion of the flows in the collected dataset. It is likely that these timestamps are wrong as a result of how the traffic was captured.

Features are collected by intercepting the first 50 packets (using default settings). An important part of a tool is that it is capable of identifying TCP retransmissions. Retransmissions occur when the TCP acknowledgments are delayed or not sent [63]. This problem can occur when the destination of a TCP packet is unavailable due to network issues, or the destination is not reachable. In most cases, this type of traffic is not related to malware behavior, and should not be included as a network feature for the machine learning algorithm. This tool detects retransmissions by comparing the TCP sequence number and acknowledgment number with the previous packet in the flow. If the numbers are equal to each other, the packet gets dropped.

The source code for the Metadata Collector is found in Appendix B.

4.2.4 Correlating network logs

To be able to include features from both Suricata (JA3 and flow) and the Metadata Collector, it was necessary to merge information from both sources. The correlation was accomplished by correlating the IP addresses and port pair in both logs. This correlation results in a merged log that includes all information for a single flow event. In Appendix A, is an example of a correlated malware flow.

4.3 Pre-processing

The pre-processing part of this research is packet features modeled as in a first-order Markov chain, features extracted from the JA3 string. Further, we describe the method used to normalize all features and provide correct labels. Finally, records with missing values are removed, and malware samples with a high number of flows generated are reduced in favor of a more balanced dataset.

Packet payload in first-order Markov chain

The features extracted from the packet payload are modeled with a first-order Markov chain. Generally speaking, a Markov chain is a process where an outcome of experiments can affect the outcome of the next experiment. A Markov Chain consists of a set of states and moves sequentially from one state to another. This move action is called a *step*, and moving through all the states, a transition matrix is created [34, p. 405-407].

For this feature, ten different states were used and each state represents 150 bytes. Then, it is necessary to estimate the transition probabilities with the collected TCP payloads. These are the same values used by Anderson et al. [15], and we assume that the MTU is 1500 bytes (150*10).

The following process is performed twice, packets to the client, and packets to the server:

1. Create a 10*10 matrix.
2. Find the position of the correct packet size in the matrix, divide the packet size by 150 and check if that is less than the matrix size. Choose the lowest number. Then do the same on the next packet, and choose both integers as the index in the matrix. Perform all these steps on all packets.
3. Then calculate empirical transition probabilities.

To ensure similar results as Anderson et al. [15]'s research, our implementation of the first-order Markov chain, presented in Appendix C is based on their implementation. This implementation is

a part of the repository of Joy, their open-source network collecting and analyzing tool for their research [5].

TLS

The TLS features from the JA3 string are first normalized by splitting each of the values into their columns. The splitting format and naming convention is shown in Figure 7 in Section 3.1. The SSL version field was extracted as a single numerical feature. Further processing of remaining fields was necessary as they consisted of categorical values. Three different approaches were considered: one-hot-encoding, hashing encoding, and counting the number of values in each field. The two last methods were used in this thesis, but all three of them are explained.

One-hot-encoding, also called dummy encoding, takes all categories of a feature and creates new binary features of each category. If category value is present in the learning example, a one is stored in the feature, and if not, a zero is stored. An example of this method is found in Figure 4, where Figure 3 is the original feature column.

Table 4: Example one-hot-encoding

Malware family	Emotet	Wannacry	Trickbot
Emotet	1	0	0
Wannacry	0	1	0
Trickbot	0	0	1

This method was not used because of its limitations by including new data and the fact that it does also lead to a high number of dimensions.

The hashing encoding method was one of the chosen methods due to the possibility to reduce feature space to a fixed size, and remove the problem of unobserved data not being classified.

```
hashing_list = [0] * 100
for category in categories:
    category_hash = md5(category)
    hashing_list[int(category_hash) % 100] += 1
```

Listing 1: Simplified python code for feature hashing

In Listing 1, we present a simplified version of the algorithm used to perform feature hashing. One thing to notice is that two different categories can be stored in the same index. Such collision appears since the hash value is reduced to a fixed value. However, in the research on SPAM filtering, these collisions rarely affect the classification results [17].

In this research, the fixed number of features set to 100. This value is based on the reduction percentage by one hot encoding, and *Curse of Dimensionality*, a challenge in machine learning, which is further explained in Section 2.3.2. Summarizing all unique categorial features in TLS, it is in total 310 different values. Reducing this number to 100, it reduced 2/3 of the feature space and

reduced the risk of *Curse of Dimensionality*. In the experiments, this value seems not to affect the classification performance.

A possible method to identify the appropriate number of features could be to perform cross-validation with a classification algorithm, and modify the feature size iteratively, and use the count that shows the highest performance, e.g., accuracy.

The last chosen method was counting the number of values in each category field. Each category resulted in new features with their counts.

Scaling

Scales all feature to a value between 0 and 1. Scaling performed with the following formula: $X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$

The reason to perform feature scaling is that several machine learning algorithms do not work correctly without normalization of the value. This problem exists in the classifiers that calculates the distance between two data points, such as Euclidean distance [42, p. 260]. One algorithm that utilizes this metric is KNN.

Also, algorithms that use gradient decent have benefits of normalization as they converge faster with normalized values [39].

Labeling

All samples were given a label of 0 or 1, representing benign or malware traffic respectively. These labels were assigned based on the metadata of the malware capture.

Family labeling

Identifying a possible malware family name was performed to get a deeper understanding of the ground truth of the dataset. The family label is only used in data exploration to provide an understanding of the malware family distribution. If all Anti-Virus engines followed the proposed Caro method described in Section 2.2.1, the malware family name could be extracted from their detection. Since most of the vendors are not using this scheme, a different method was used to identify the possible name.

Sebastián et al. [73] proposed a method and a tool, called AVclass, for automatically labeling the most likely malware family name based on a set of labels from Anti Virus (AV) engines. Their method can be described in two phases: preparation and labeling. In the first phase, generic tokens and an alias list are generated. These tokens can be, e.g., the architecture (win32), class name (trojan). The alias list is created to provide an of between known family aliases to one specific family. The purpose is to mitigate the problem of having the same malware with different family names.

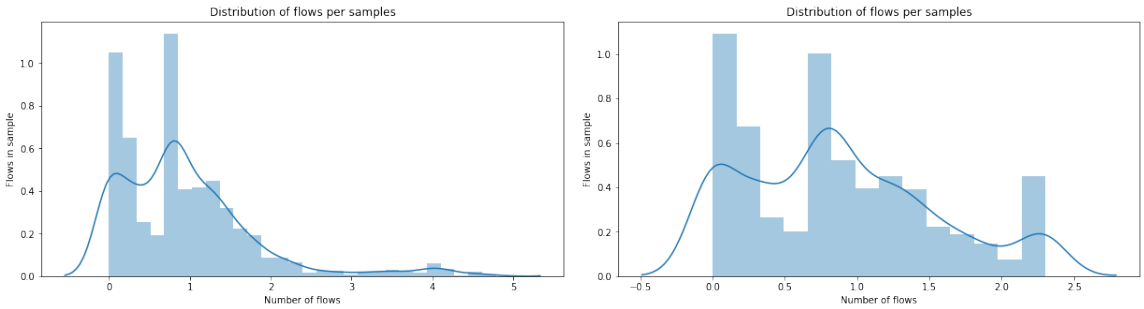
In the second phase, the actual labeling is performed, which includes normalizing each of the input samples and outputs a rank list of the most likely malware family. The normalization process removes duplicates and filters the labels from the engines. The generated alias list is used to replace alias with the family name, further unique names of in list removed. Then the family name is selected based on plurality vote.

This tool was used together with VirusTotal's API to extract AVs detections. Virus-Total is a public website where malicious files can be submitted, and over 70 AVs scans the file for malicious artificats[1].

4.3.1 Removing missing values and balancing of the dataset

The steps performed in pre-processing in the experiments results in 751,271 malware flows and 156,109 normal flows. In both flow sets there are TLS features with missing values, and 1623 flows were removed from the dataset due to this problem. Identifying why these values were missing is out of the scope of this thesis, but it could be due to missing packets in the raw network capture. Since only 0.18% samples were removed, it was not considered as a problem.

With data analysis of the dataset, we identified considerable differences in the number of flows generated by each sample. To reduce the risk of training the classifier to a specific malware family, we had to remove some flows to balance the data set. In Figure 11 a the number of flows per malware sample is visualized. The number of flows is scaled to log10 to make it human readable. The goal of the balancing is to remove the tail in the graph since the tail represents malware samples with a high number of generated flows.



(a) Before balancing

(b) After balancing

Figure 11: Before and after balancing number of flows generated by malware

A simple method used to balance the dataset was to reduce the number of flows per sample to a maximum of 200 flows. Malware with more than 200 flows was reduced by picking 200 random flows from the sample. The result of this process is displayed in Figure 11 b.

In the new graph, the tail is removed.

4.4 Classification algorithms

The choice of classifiers in this research is based on earlier research and using simple models. We choose multiple different algorithms to test, and this is due to the *No Free Lunch* principle, there is no superior algorithm. The machine learning framework scikit-learn is used to provide most of the classification algorithms [62]. In addition to this framework, a second library is used to provide an

[1] https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works

implementation of gradient boosted trees. Both frameworks utilize a similar API in Python, which makes it easy to validate several algorithms.

The following classifiers were used:

Decision tree is a tree-based classification algorithm that consists of nodes that represent features, edges represent a subset of feature values, and terminal leaves correspond to target labels [42, p. 227]. An important factor in this method is to choose the best feature. Frequently used algorithms are information gain, information gain ratio, Gini-index, and RelieF [42, p. 228]. Default settings from the Scikit-learn framework were chosen, where the following list contains the most relevant options:

- Gini-index is chosen as the function to calculate the quality of the split.
- The max depth of the tree is chosen to be until all leaves are pure or contain less than two samples.

Random Forest is a classification algorithm that consists of several decision trees. It is recommended that the number of generated trees should be at least 100 [42, p. 95]. One setting was changed from the default ones in the Scikit-learn framework. The most relevant are described in the list below:

- The Gini-index is used in the experiments, which is the same as the one used in the Decision Tree.
- The number of generated trees are chosen to be 100. The default setting in the framework was ten, but based on recommendations from the literature, it was chosen to be 100.

K nearest neighbors (KNN) is a distance-based learning algorithm, and predictions are made by calculating the distance from an object towards K subset of similar objects. The K value is usually set to an odd number, and this value can help to average the results of the classification if there is noise in the data [42, p. 260]. The defaults settings from the Scikit-learn framework were chosen, where the k value is set to 5. This value follows the recommendations from literature, where the k value should be an odd number.

Logistic regression is a classification algorithm that is used to predict the likelihood of the outcome based on the features [42, p. 266]. For this classifier, one option was modified from the default settings in Scikit-learn. The following settings were used:

- L2 is the norm used for penalization.
- Saga is the optimization solver used for this algorithm. This algorithm is one of the optimization algorithms recommended to use for large datasets. Defazio et al. [24] states that this optimization algorithm has a theoretical better convergence rate than SAG, another algorithm that could be used by the classifier.

XGboost (eXtreme Gradient Boosting is a machine learning library that has implemented gradient boosted trees [21]. Gradient boosting is based on the theory of gradient boosting trees [29]. In this library, all default settings were used, and the most important are:

- Tree booster uses the maximum tree depth of six.
- The numbers of generated trees are 100.

Multi-layer Perceptron is a learning algorithm from the neural network field. A multi-layered perceptron is one or more hidden layers of neurons [42, p. 302]. A neuron is a simple processing unit that uses an activation function to calculate the weighted sum of its inputs, and transform the obtained sum with an output function into two possible output values [42, p. 277-278]. A common activation function is the following linear activation function: $A(X_j) = \sum_i W_{ij}X_i + C_j$ [42, p. 289]. For the output function, a logistic sigmoid function it is usually used: $f(X) = \frac{1}{1+e^{-X}}$ [42, p. 290]. The default options from Scikit-learn were used for this classifier, and the most important are:

- The number of hidden layers were 100.
- Activation function for the hidden layers were *relu*. This function is calculated with the following formula: $f(X) = max(0, X)$.
- The solver for weight optimization was *adam*.

4.5 Model evaluation

Three different metrics were selected to validate the performance of the classifier: accuracy, recall, and precision. These metrics are explained in Section 2.3.

Accuracy is one of the most common metrics to use and is included in this research to provide comparable results. However, we do not see this as the most valuable metric because it is possible to achieve high accuracy rate with a too high number of false positives. Axelsson [18] looked into the phenomenon of base-rate fallacy in intrusion detection, and to achieve high detection rate, which means that when it is an intrusion it is also an alarm, it is necessary to have a low number of false positives. He states that an IDS should have less than 1/100 000 false positive alarms to reach its expectation.

Based on this fact, we have chosen the precision as the most valuable performance metric, as it takes into account the number of false positives on the true positives. With a high precision score, it is a high probability that an alert from a IDS based on this research, is a true positve [71]. It is therefore expected that a classifier with a high number of false positives will have a low precision score. Since it is also important to look into true negatives, intrusions that were not alerted, we have included the recall metrics.

To ensure reliable results, a k-fold variant was chosen. The variant is called stratified k-fold, and ensures the same percentage of targets in each class, with the k value ten, as this the most frequently used value [42, p. 83].

Other types of metrics that are important to consider, but not accounted for in this study are:

- Building time of the model.
- Time to predict new observations.
- Memory consumption.

4.6 Feature subsets

Several feature subsets were tested in the experiment. The purpose of this was to identify which features are best to classify malware in encrypted traffic and if it is possible to achieve the same results with a minimal subset. To identify a minimal subset, two different methods were; information gain and Correlation-Based Feature selection (CFS).

Information gain, also called mutual information, is defined as the amount of information obtained from a given feature for determining the target label [42, p. 157]. The following formula is used to calculate the information gain: $Gain(F) = H_T - H_{T|F}$. This formula calculates the target entropy and subtracts it with the conditional target entropy given the value of feature *F*.

CFS is a filtering method which evaluates the correlation of a subset of the features with a heuristic evaluation function [16]. This method selects features with high correlation towards the target and uncorrelated to each other. Irrelevant features are removed as they have low correlation towards the target, as well as reduce the number of features as they are highly correlated with one of the other features. Andrew. Hall [16] found that, in several cases, CFS gave comparable results to Wrapper methods, and on small dataset outperformed the method. The wrapper method is covered in Section 2.3.2 but is, in general, slower than filtering approaches.

The following list consists of the different feature subsets used in the experiments:

- Only TLS based features.
- Only Flow based features.
- Only Packet based features.
- TLS and Flow based features.
- TLS and Packet based features.
- Flow and Packet based features.
- Full dataset; TLS, Flow, and Packet based features.
- Top ten features extracted with information gain.
- All features determined with CFS.

5 Experimental setup and classification results

In this chapter, the experimental setup is explained, both physical and logical. To understand the ground truth of the dataset, we have applied data exploration methods. This consist both of understanding of statistical representations of the data and different visualizations. Then, the actual results of the experiments are listed. These results are both from feature selection and classification.

5.1 Experimental environment

In this section, the environment for the experiments is explained, with a description of the data flow from raw network traffic to samples consumed by classification methods. To provide an understanding of the logical environment, we have created a data flow diagram. The diagram is explained in Figure 12. In this diagram, it is possible to see how Suricata and Metadata Collector process raw network packets on sources listed in 4.2. Logs from both sources are correlated with the method described in 4.2.4 and displayed in the diagram. All correlated logs are stored as JSON objects in a file, and an example of this log format is displayed in Appendix A.

The next step is pre-processing as defined in Section 4.3. This step is the final step before performing feature selection, then classification.

5.1.1 Physical environment

The computer resources used in this thesis are listed below. The server was running as a virtual machine in NTNU's OpenStack solution. An essential requirement in this setup is to have big enough disk size to have the full raw dataset stored on disk. Further, it is essential to have enough memory to process the dataset, and enough CPU capabilities to perform classification within a reasonable time.

The following resources were provided in the primary environment for this thesis:

- Virtual Server
- 16 Core vCPU
- 32 GB memory
- 600 GB disk
- Cent OS 7.6.1810

The feature selection was performed on a personal laptop with Weka installed. The following list includes the available resources on this laptop:

- Lenovo ThinkPad yoga 2
- 4 Core CPU Intel i5-4300U
- 8 GB memory
- 500 GB disk

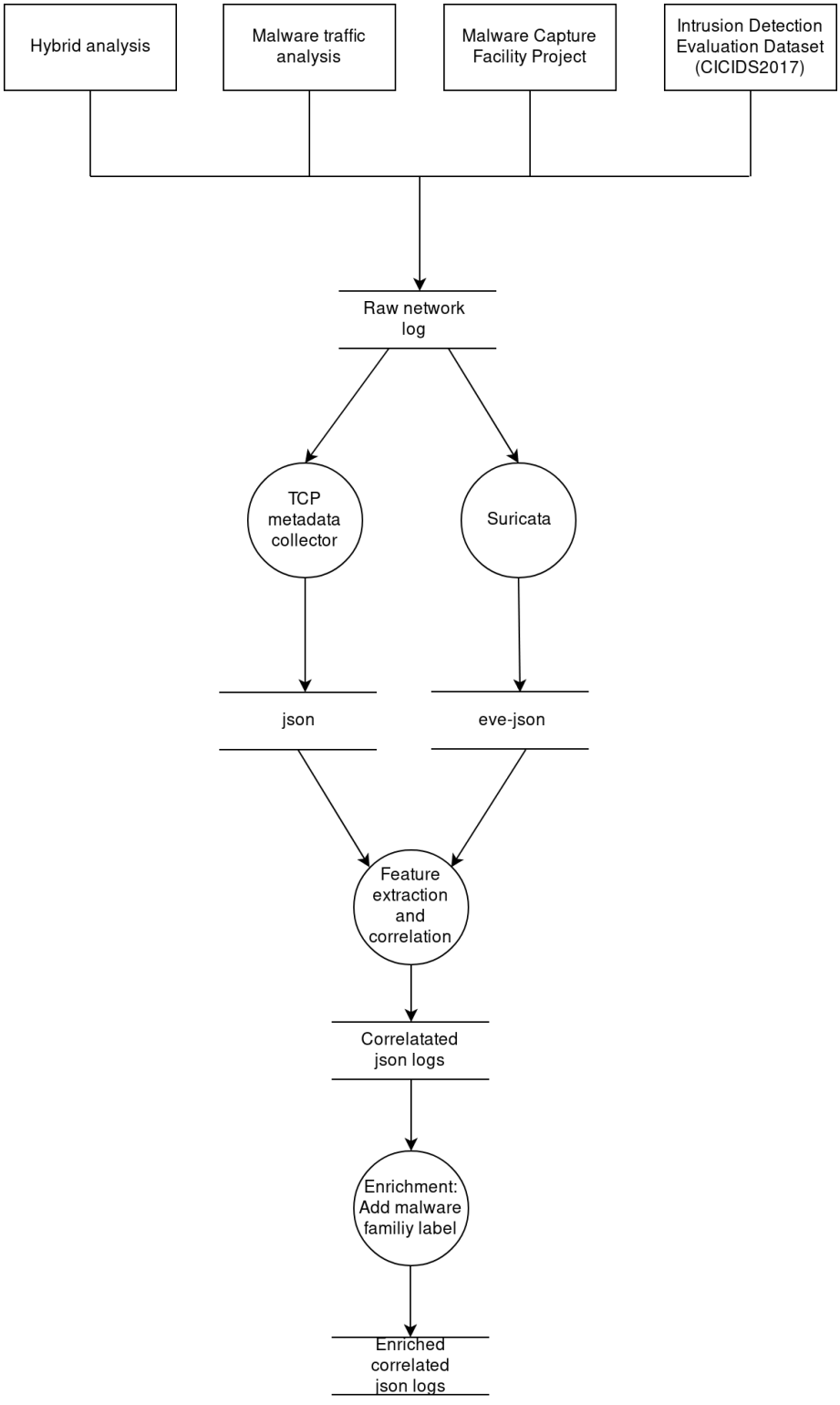


Figure 12: Data flow of experiment

- Arch Linux with kernel 5.0.13

5.1.2 Logical environment

In the list below, the main applications are given, with version number, used in the experiments.

- Python 3.6
- scikit-learn 0.20.2
- pandas 0.22.0
- xgboost 0.82
- Weka 3.8.1
- Suricata 4.1.0-Dev

5.2 Data exploration

In the dataset that was used for the experiments, the total numbers of normal flows were 156,098 and 40062 malware flows. The choice of using an unbalanced dataset is due to the nature of malware traffic in corporate networks, where it is naturally more normal traffic than malware traffic.

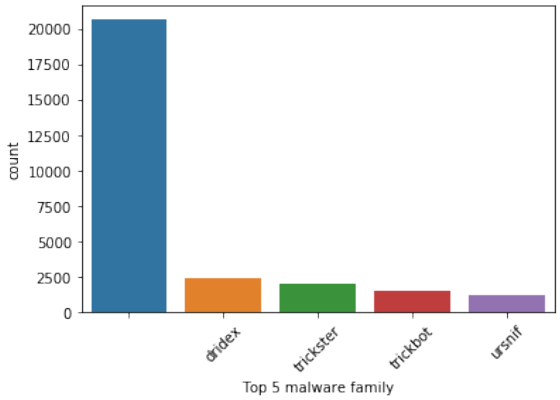
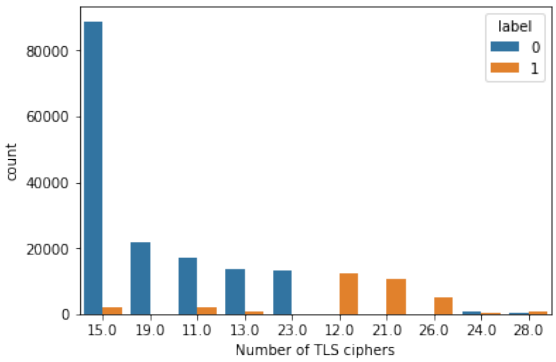
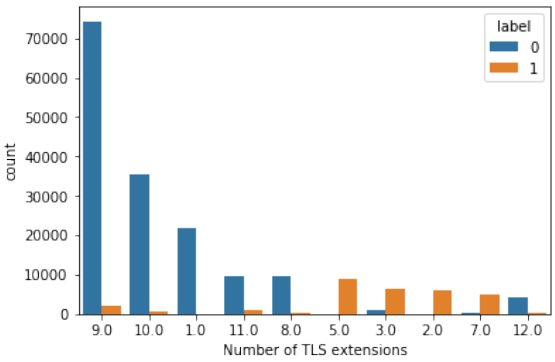


Figure 13: Top five malware families in the dataset

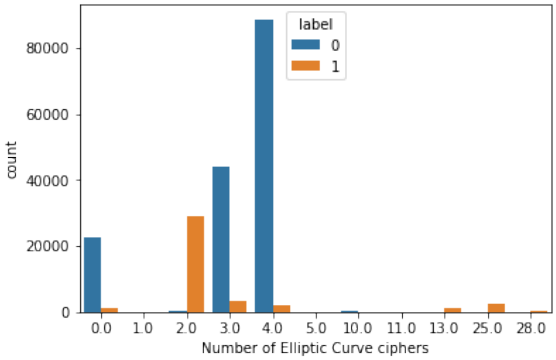
In total, there were 149 different malware families. The five most common are presented in Figure 13. The method of assigning malware families to the malware flows is described in Section 4.3. In the same figure, it is possible to see that the most common malware family does not have a name. The explanation of missing malware family is due to the malware not listed in the online resources, or the used method could not identify the possible malware family. Since this information is only used to understand the dataset, and not as a label in multinomial classification, this does not affect the results. However, if the purpose were to perform multinomial classification, it would require an in-depth analysis to identify why the chosen method was not able to identify any malware family for a given malware sample.



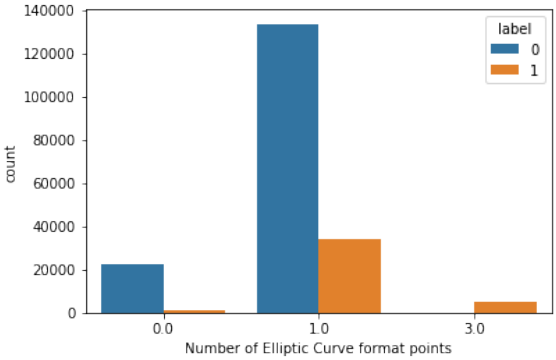
(a) Count of TLS ciphers



(b) Count of TLS extensions



(c) Count of TLS elliptic curves ciphers



(d) Count of TLS elliptic curve format points

5.2.1 TLS client fingerprints

In the malware subset, there are 270 unique TLS client fingerprints and 41 unique fingerprints. Based on this finding, it is more diversity in malware traffic than normal traffic. Further looking into metadata, the number of different values in a JA3 field, it is possible to observe additional differences. Looking at the differences in Figure 14a,14b,14c,14d, it is possible to manually make conclusion of benign and malware traffic. Label 1 refers to TLS values labeled as malware, and label 0 is TLS values labeled as normal.

5.2.2 Flows

In Table 5, the values for the different flow features are summarized as the mean and 75% percentile. Comparing benign and malware traffic, there are, in general, no significant differences between them. A conclusion is that malware, in general, sends fewer packets and data than normal traffic, but the duration of the flow is a bit longer. The mean and 75% percentile values for normal traffic, have no significant differences and are therefore less likely to have a lot of outliers. However, if these values for malware traffic are compared, it is possible to observe that the mean for malware traffic is much higher than the 75% percentile. A conclusion drawn from this observation is that

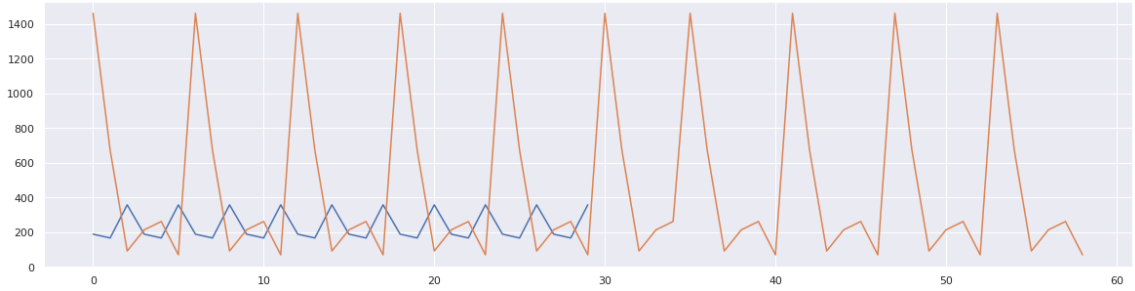


Figure 15: Payload size of 10 Cobalt Strike flows

the malware traffic includes several outliers, and it is more diversity in malware traffic than benign traffic in terms of flow attributes.

Table 5: Mean and 75% percentile of flow features

	malware		**normal**	
	mean	**75%**	**mean**	**75%**
age	115.81	69.00	53.50	67.00
bytes to server	8662.17	2451.00	3586.70	2992.00
bytes to client	70008.82	7541.00	40833.49	8319.00
packets to server	35.04	14.00	25.94	22.00
packets to client	64.00	16.00	41.33	22.00

5.2.3 Packets

In Figure 15, packet payload size from ten continuous flows from Cobalt Strike framework are visualized. The blue line represents packets sent to the client, and the yellow line present packets sent from the client to the server. A reason why there is a different number of packets in the directions may be due to occurrences of TCP retransmission; packets reach a timeout limit. Cobalt Strike is a penetration testing framework which includes an agent for post-exploitation and covert channels for emulating advanced adversaries and is observed to be used by several threat-groups [58]. As seen in this figure, there is communication that is transmitted in observable patterns, and based on analysis of these patterns, the communication is most likely related to beacons sent from an infected host towards the C2 server.

In Figure 16, the same amount of flows is collected for communication from a single client towards Facebook. This traffic is labeled as benign traffic, and the patterns are seen as more random and without the patterns displayed in the Cobalt Strike traffic example. Based on these observations, it is possible to conclude that it is possible to manually classify different types of traffic based on their packet behavior.

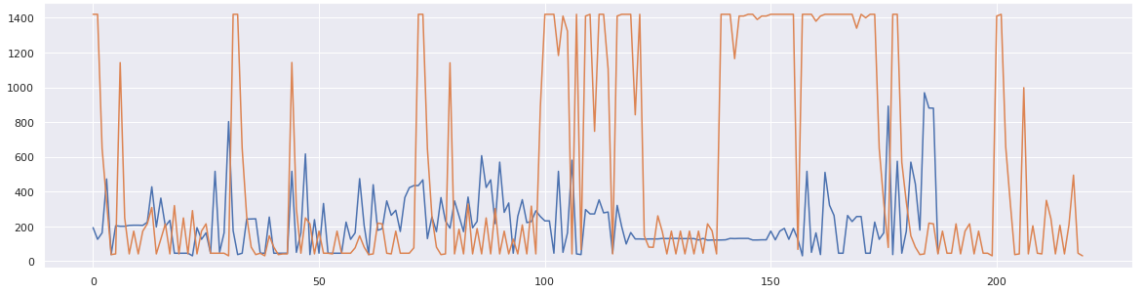


Figure 16: Payload size of 10 Facebook flows

Table 6: Feature selection: Results from information gain and CFS

	Information Gain		CFS results
	Merit	Feature	Feature
1	0.5749575	tls_Cipher_Count	tls_SSLExtension_Count
2	0.5646206	tls_SSLExtension_Count	packet_to_client_payload_size12
3	0.5183201	tls_EllipticCurve_Count	packet_to_client_payload_size60
4	0.4230407	tls_Hash_67	packet_to_client_payload_size82
5	0.2621329	flow_bytes_toclient	packet_to_client_payload_size44
6	0.2023395	flow_bytes_toserver	packet_to_client_payload_size60
7	0.156212	tls_Hash_58	tls_Hash_4
8	0.1520883	tls_Hash_11	tls_Hash_8
9	0.1520005	tls_Hash_37	tls_Hash_9
10	0.1435938	tls_Hash_69	tls_Hash_16
11	0.0984317	packet_to_client_payload_size94	tls_Hash_23
12	0.082506	flow_age	tls_Hash_32
13	0.0813237	tls_Hash_31	tls_Hash_35
14	0.0757281	tls_Hash_57	tls_Hash_38
15	0.0748446	tls_EllipticCurvePointFormat_Count	tls_Hash_48
16	0.0676145	packet_to_server_payload_size0	tls_Hash_49
17	0.062661	packet_to_server_payload_size30	tls_Hash_52
18	0.0608266	flow_pkts_toserver	tls_Hash_65
19	0.0601974	tls_Hash_50	tls_Hash_67
20	0.0598485	tls_Hash_90	tls_Hash_71
21			tls_Hash_75
22			tls_Hash_93
23			tls_Hash_94

5.3 Feature selection

According to Section 4.6, two different methods were applied on the full dataset; information gain and CFS. The feature selection was executed in Weka with default options. The goal of this process was to identify which features provide most value towards the target, to classify malware flows. The results of both methods were used as feature subsets in classification.

In Table 6 the top 20 features extracted from information gain are listed. The top ten features are colored in green, and those features are used as the feature subset.

The second used method, CFS, results in a different subset than information gain. In Figure 6, it is possible to see that *Packet*-based features have a higher relevance, and *Flow*-based features are not listed. Common in these methods is the relevance of *TLS*-based features. However, it is only the *tls_SSLExtension_Count* that is the same in both subsets, but in both subsets it is a high amount of these features.

5.4 Results

In line with Section 4.5, stratified 10-fold cross validations is used to ensure reliable results of the classification algorithms described in Section 4.4.

Table 7 contains all of the results from all of the defined feature subsets with all the chosen classifiers. In this table, *Acc* stand for accuracy, *Rec* for Recall, and *Prec* for precision. These metrics are explained in Section 2.3. The number of features in each subset is viewed inside the parenthesis. The best metric for each subset is colored in green, and the worst metrics are colored in red. In case of metrics with the same value, all the values are colored and marked with an asterisk.

The most interesting metrics are precision and recall, where precision is most weighted. Looking at all precision results in Figure 7, it is possible to see that the precision score is over 96% with the following feature subsets: (1)Flow and TLS, (2)Packet and TLS, (3) Flow, Packet and TLS, and (4) information gain with the best ten features. The common features between these subsets are TLS-based features and show the important value of including features extracted from the TLS hello packet. In terms of the classification algorithm, it is the Random Forest classifier that provides the best metrics. However, it does not outperform the other classifiers in a way to exclude them from consideration, such as XGboost and MLP.

Based on these results, we do not see any gain of using subsets generated with feature selection. Compared to the full feature set, they provide similar or worse performance in most cases.

—	Decision Tree			KNN			LR			MLP			RF			XGBoost		
	Acc	Rec	Prec	Acc	Rec	Prec	Acc	Rec	Prec	Acc	Rec	Prec	Acc	Rec	Prec	Acc	Rec	Prec
flow(5)	91.09%	76.57%	79.35%	91.37%	71.39%	84.12%	79.59%	00.10%	63.85%	82.57%	19.31%	80.10%	93.84%	75.83%	92.73%	89.33%	55.33%	87.48%
packet(200)	94.14%	81.91%	88.46%	90.52%	81.53%	75.44%	91.40%	69.43%	85.30%	91.40%	69.43%	85.30%	95.74%	82.12%	96.36%	92.25%	67.54%	92.21%
tls(105)	98.89%	94.66%	99.89%	83.8%	94.57%	73.64%	98.78%	94.25%	99.79%	98.88%	94.65%	99.84%	98.93%	94.93%	99.83%	98.38%	92.35%	99.71%
flow, packet(205)	95.25%	86.09%	90.19%	94.92%	80.35%	93.85%	91.40%	69.44%	85.38%	95.32%	85.11%	91.37%	96.57%	84.19%	98.82%	93.28%	70.88%	94.88%
flow, tls(110)	99.29%	98.24%	98.28%	99.40%	97.99%	99.06%	98.78%	94.25%	99.79%	98.87%	94.63%	99.84%	99.63%	98.47%	99.71%	98.42%	92.48%	99.76%
packet, tls(305)	99.25%	97.73%	98.59%	98.40%*	94.46%	97.65%*	98.40%*	94.46%	97.65%*	99.39%	97.76%	99.26%	99.52%	97.93%	99.70%*	98.52%	93.02%	99.70%*
flow, packet, tls(310)	99.30%	97.93%	98.63%	98.75%	94.42%	99.40%	98.70%	94.39%	99.21%	99.40%	97.88%	99.15%	99.57%*	98.07%*	99.81%*	99.57%*	98.07%*	99.81%*
Info gain(10)	98.61%	96.29%	96.90%	98.89%	96.09%	98.43%	96.36%	83.90%	98.02%	97.95%	90.30%	99.65%	98.84%	95.01%	99.30%	98.09%	91.01%	99.59%
CFS(23)	98.41%	92.84%	99.33%	91.05%	93.30%	79.88%	97.66%	89.71%	98.69%	97.95%	90.71%	99.20%	98.42%	92.88%	99.34%	97.98%	90.72%	99.32%

Table 7: Classification results

6 Discussion

Anderson et al. [15] showed the capabilities of applying behavior-based information and cleartext metadata traffic in the task of classifying malware in encrypted traffic. Our study complements with their results and even shows that it is possible to achieve good results with a feature set only containing behavior based features. By using only behavior based features it introduces a chance of more false positives, but based on the expert knowledge presented in this thesis, is it more likely that the results give a better chance of detecting advanced malware. This research performed in this thesis answers our three research questions:

> Research question 1: What are resilient network features for malware detection in TLS encrypted traffic?

To identify resilient network features, we first needed to define this term in the context of the thesis. This term was defined in Section 4.1, and was identified as features describing the behavior of the TLS encrypted network flow. In this section, the features was compared against C2 evasion techniques described by Mitre, described in Section 2.2.3.

The evasion technique Commonly Used Port (T1043) do not affect our method since the port number is not a part of the feature set. Domain Fronting (T1172) is an evasion technique used to give a false representation of the real destination by using a legitimate SNI in TLS with a different HTTP host header. This evasion method is not going to affect our approach since we are not using features extracted from the SNI field. Also, the technique of using a legitimate Web Service (T1102) such as Twitter is not affected by our method, since we are not using features dependent on the destination service. However, we see problems with the detection of the last described technique by Mitre, Remote Access Tools (T1219). This technique is difficult to detect because of their legitimate usage and not a part of the malware dataset. Detection of the technique needs to be enforced by different controls such as end-point detection or policy based detection.

Three different sources of features were chosen: Network flow, individual TCP packets within a Flow, and information extracted from the TLS hello packet. We are confident that these sources provide valuable information on detection in malware in TLS encrypted traffic since similar features have shown success in related research described in Chapter 3.

The analyze of the classification result in Section 2.3.1 points out feature sets beside from flow-based features are resulting in overall good classification results. Random Forest is the only classifier with precision sore above 90%, and the lowest score is 63.83% generated with Logistic Regression. Decision tree results in a precision score of 79.35%, K-nearest neighbor is 84.12%, Multilayer Perceptron is 80.10%, and XGboost provides 87.48%. The packet-based feature provides similar results but in general, a higher precision score. Also, Random forest provides the best precision

score for packet-based features with 96.36%, and the worst score is provided K-nearest neighbors with 73.64%. These results can indicate that both feature sets include too little information to be used alone. In this research, TLS-based features are resulting in a precision score of over 99% with the following classifiers: Decision Tree, Logistic Regression, MLP, Random Forest, and XGBoost. Although the precision is over 99%, we do not think that this feature is enough alone. This high precision score can be due to overfitting since our normal dataset does not include enough variance. We think that this score would be lower with a bigger dataset with more variance of normal traffic.

> Research question 2: How can these features be extracted from the network traffic?

In this thesis, Suricata was chosen to extract features from flow-based and TLS-based features. Suricata implements this functionality with enabling JA3 logging as the only modification for its configuration.

Tools like Zeek (Bro) could also be an option and have proven to work in this type of research area [77]. In this research field, we do not think that one tool is superior over another, as long as they can extract similar data. However, practical considerations such as network throughput limitations need to be taken under consideration when used in production. Extracting packet-based features is limited in Suricata since this tool is flow-based. To overcome this issue, we developed a packet-based tool called Metadata Collector. This tool provides similar functionality as the tool developed as a part of Anderson et al. [15] research, called Joy. Joy is also able to log similar flow and TLS data as Suricata, but we choose to use Suricata due to its majority and its functionality to monitor high-speed networks.

> Research question 3: Which feature selection and classification algorithms provide the best classification performance?

In Section 5.4, the Random Forest classifier generally provides the best results for accuracy, recall, and precision. The best results are achieved with all feature set, and the Random Forest classifier results in an accuracy of 99.57%, recall value of 98.07%, and a precision score of 99.81%. It is possible to see that the Random Forest classifier is proving to be a suitable choice for this task, but there are minor changes compared to XGboost and MLP. A limitation of these results is that there are no tuning of the parameters in classifiers. Such tuning could increase the performance of several classifiers.

In terms of filtering based feature selection, we observe a minimal gain in performance. Instead, a wrapper approach looking into different feature sources provides more insight into the classification and which features that give the most gain into classification.

As we discussed in the first research question, the results can be too good due to the size of the dataset and too little variance in the normal traffic.

6.1 Limitations

- The size of the dataset can be viewed as one of the main limitations of this research. In contrast with Anderson et al. [15] research, where the dataset consisted of 1,500,005 normal

flows and 133,744 malware flows. However, when compared to other research articles in this field, the size of the dataset is sufficient for providing reliable results. The most realistic results would be provided with normal traffic collected from an internal network of a corporation, as described in Section 4.2. However, such an approach would limit the possibility for other researchers to repeat the experiments, as the dataset would contain sensitive information.

7 Conclusion

In this thesis, we have shown how it is possible to detect malware in TLS encrypted network traffic using behavior-based network features. It is possible to achieve good classification performance with common machine learning algorithms. Due to the *No free lunch principle*, six machine learning algorithms were validated to ensure reliable classification results. Of the six algorithms, the Random Forest classifier provides the best overall performance. However, the results are not outstanding compared to the other five algorithms. We were able to achieve an accuracy of 99.57% and precision of 99.81% using the Random Forest classifier with the full feature set. These are promising results for the problem of detecting malware in TLS encrypted networks.

The used dataset consists of real data from malware traffic and traffic from normal users. Using real-world data shows that there are possible to apply this method to a real-world scenario. We discussed in Section 6.1 some of the limitations of this research, which was the amount of data and its diversity of normal traffic. This problem can affect the real performance, but these results are not unrealistic when comparing to similar studies. Our results are comparable with Anderson et al. [15], and data exploration shows distinct differences between normal traffic and malware traffic.

Our experience in how advanced malware uses several evasion techniques has been valuable to identify the key features providing good performance using this method. We used that knowledge to reduce the number of different features used by Anderson et al. [15] to only features that describe the behavior of a TLS flow. Several species of advanced malware evade static features, but it is more difficult to obfuscate the behavior of the malware. There is no such thing as a silver bullet that detects all types of malware in TLS traffic, but our method increases the detection capabilities. As detection capabilities of malware in TLS encrypted traffic with machine learning, we expect to see an increase in adversarial examples to evade methods like this. It is, and will always be, an ongoing race between offensive and defensive security.

Our work is a contribution to the detection of C2 communication from malware in TLS encrypted traffic, which is a growing problem, as an increasing amount of malware utilizes this protocol. We have extended the existing research in this field by looking into how behavior-based features work in this field. This field is still in development, and there is a surge in new ideas which can contribute to better performance, as well as identify the problem with adversarial examples.

8 Future work

Based on our results from the experiments and the discussion, several subproblems and interesting research topics in this field it are raised. The proposed topics for future work should be interesting as individual topics and may contribute to improving this research area.

Classify malware families

The work of multinomial classification was slightly explored in the research of Anderson et al. [15], but based on our findings it requires more understanding of characteristics of malware families and which features that contribute most to differentiate malware with high accuracy. Based on our research, these features should be behavior-based. Another method could be developing more classifiers built with data from specific malware families. Both methods should be tested and validated, and the performance compared, to identify which one that is most suited for classification.

Analyze advanced malware behavior

In our study of malware, we identified that advanced malware utilized several methods to evade detection and blend their traffic with normal traffic. Therefore, we propose to perform an in-depth study in how the C2 communication functionality is implemented, and look into the behavior of the observed network traffic. Identifying unique artifacts in the traffic can be adapted by computational methods, such as the one presented in this thesis. Therefore, we propose to study how such malware operates. Looking into the behavior observed in the network traffic is valuable information that can be adapted in the method proposed in this thesis.

Concept drift

In our experiments, we did not look into how the performance of the classifier changed over time. We propose to look into how malware change over time by looking into the classification performance and which features that are most, and not, vulnerable to concept drift. Further, we suggest looking at how often it is required to retrain the classifier to ensure consistently high performance.

Adversarial examples

In this thesis, it was studied and discussed which features that are, to some degree, more resistant to attacks and obfuscation using this method. However, it has not been looked into how adversarial examples could be used to evade detection, and how such an attack can be avoided and which features are less prone to adversarial examples. For this reason, we propose to look into how adversarial examples can be crafted for these types of systems. Further, it is necessary to look into how the risk of adversarial examples can be reduced.

Bibliography

[1] The bro network security monitor. URL https://www.bro.org/. Accessed: 2018.12.14.

[2] A new virus naming convention. http://www.caro.org/articles/naming.html. Accessed: 2018.12.05.

[3] https://github.com/google/gopacket. https://github.com/google/gopacket. Accessed: 2018.12.12.

[4] Hybrid analysis. URL https://www.hybrid-analysis.com/. Accessed: 2019.02.25.

[5] cisco/joy. URL https://github.com/cisco/joy. Accessed: 2018.11.14.

[6] Let's encrypt stats. URL https://letsencrypt.org/stats/#percent-pageloads. Accessed: 2018.11.21.

[7] malware-traffic-analysis.net. URL https://www.malware-traffic-analysis.net/. Accessed: 2019.02.25.

[8] Moloch is a large scale, open source, full packet capturing, indexing, and database system. URL https://molo.ch/. Accessed: 2018.12.14.

[9] saleforce/ja3. https://github.com/salesforce/ja3. Accessed: 2018.11.25.

[10] Suricata open source ids / ips / nsm engine. https://suricata-ids.org/. Accessed: 2018.12.11.

[11] Abuse.ch. Ssl blacklist - ja3 fingerprints. URL https://sslbl.abuse.ch/ja3-fingerprints/. Accsed: 2019.03.07.

[12] Alexa (Amazon). The top 500 sites on the web. URL https://www.alexa.com/topsites/. Accessed: 2019.05.15.

[13] Blake Anderson and David McGrew. Identifying encrypted malware traffic with contextual flow data. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*, AISec '16, pages 35–46, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4573-6. doi: 10.1145/2996758.2996768. URL http://doi.acm.org/10.1145/2996758.2996768.

[14] Blake Anderson and David McGrew. Machine learning for encrypted malware traffic classification: Accounting for noisy labels and non-stationarity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 1723–1732, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4887-4. doi: 10.1145/3097983.3098163. URL http://doi.acm.org/10.1145/3097983.3098163.

[15] Blake Anderson, Subharthi Paul, and David McGrew. Deciphering malware's use of tls (without decryption). *Journal of Computer Virology and Hacking Techniques*, 14(3):195–211, Aug 2018. ISSN 2263-8733. doi: 10.1007/s11416-017-0306-6. URL https://doi.org/10.1007/s11416-017-0306-6.

[16] Mark Andrew. Hall. Correlation-based feature selection for machine learning. *Department of Computer Science*, 19, 06 2000.

[17] J Attenberg, Kilian Weinberger, Anirban Dasgupta, A Smola, and Martin Zinkevich. Collaborative email-spam filtering with the hashing trick. 01 2009.

[18] Stefan Axelsson. The base-rate fallacy and its implications for the difficulty of intrusion detection. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, CCS '99, pages 1–7, New York, NY, USA, 1999. ACM. ISBN 1-58113-148-8. doi: 10.1145/319709.319710. URL http://doi.acm.org/10.1145/319709.319710.

[19] Richard Bejtlich. *The Practice of Network Security Monitoring*. no starch press, 2013.

[20] David Bianco. The pyramid of pain. URL http://detect-respond.blogspot.com/2013/03/the-pyramid-of-pain.html. Accessed: 2019.03.10.

[21] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939785. URL http://doi.acm.org/10.1145/2939672.2939785.

[22] Alessandro Ghedini (Cloudflare). Encrypt it or lose it: how encrypted sni works, . URL https://blog.cloudflare.com/encrypted-sni/. Accessed: 2019.05.15.

[23] Dani Grant (Cloudflare). Tls 1.3 is going to save us all, and other reasons why iot is still insecure, . URL https://blog.cloudflare.com/why-iot-is-insecure/. Accessed: 2019.05.30.

[24] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. *Advances in Neural Information Processing Systems*, 2, 07 2014.

[25] T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.2. RFC 5246, RFC Editor, August 2008. URL http://www.rfc-editor.org/rfc/rfc5246.txt. http://www.rfc-editor.org/rfc/rfc5246.txt.

[26] Tim Dierks and Christopher Allen. The tls protocol version 1.0. RFC 2246, RFC Editor, January 1999. URL http://www.rfc-editor.org/rfc/rfc2246.txt. http://www.rfc-editor.org/rfc/rfc2246.txt.

[27] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley, New York, 2 edition, 2001. ISBN 978-0-471-05669-0.

[28] Elastic. Packetbeat: Lightweight shipper for network data. URL `https://www.elastic.co/products/beats/packetbeat`. Accessed: 2019.05.16.

[29] Jerome Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 10 2001. doi: 10.2307/2699986.

[30] S. García, M. Grill, J. Stiborek, and A. Zunino. An empirical comparison of botnet detection methods. *Computers & Security*, 45:100 – 123, 2014. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2014.05.011. URL `http://www.sciencedirect.com/science/article/pii/S0167404814000923`.

[31] Joseph Gardiner, Marco Cova, and Shishir Nagaraja. Command & control: Understanding, denying and detecting. *CoRR*, abs/1408.1136, 2014. URL `http://arxiv.org/abs/1408.1136`.

[32] Gigamon. To tap or span? Technical report.

[33] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[34] Charles M. Grinstead and J. Laurie Snell. *Introduction to Probability*. AMS, 2003. URL `http://www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/book.html`.

[35] Christoffer Hallstensen. Multisensor fusion for intrusion detection and situational awareness. 2017.

[36] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. Hoffman. Specification for dns over transport layer security (tls). RFC 7858, RFC Editor, May 2016.

[37] M. Husák, M. Cermák, T. Jirsík, and P. Celeda. Network-based https client identification using ssl/tls fingerprinting. In *2015 10th International Conference on Availability, Reliability and Security*, pages 389–396, Aug 2015. doi: 10.1109/ARES.2015.35.

[38] FIREEYE THREAT INTELLIGENCE. Hammertoss: Stealthy tactics define a russian cyber threat group. Technical report.

[39] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456. JMLR.org, 2015. URL `http://dl.acm.org/citation.cfm?id=3045118.3045167`.

[40] Stratosphere IPS. Stratosphere ips protecting the civil society through high quality research. URL https://www.stratosphereips.org/. Accessed: 2019.02.25.

[41] A. K. Jain and R. P. W. Duin and. Statistical pattern recognition: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37, Jan 2000. ISSN 0162-8828. doi: 10.1109/34.824819.

[42] Igor Kononenko and Matjaz Kukar. *Machine Learning And Data Mining*. Woodhead Publishing, 2007.

[43] Qualys SSL Labs. Ssl pulse. URL https://www.ssllabs.com/ssl-pulse/. Accessed: 2019.03.08.

[44] Paul D. Leedy and Jeanne Ellis Ormrod. *Practical Research Planning and Design*. Pearson, 2015.

[45] Michael Ligh, Steven Adair, Blake Hartstein, and Matthew Richard. *Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code*. Wiley Publishing, 2010. ISBN 0470613033, 9780470613030.

[46] Huan Liu and Hiroshi Motoda. *Computational Methods of Feature Selection (Chapman & Hall/Crc Data Mining and Knowledge Discovery Series)*. Chapman & Hall/CRC, 2007. ISBN 1584888784.

[47] Jakub Lokoč, Jan Kohout, Přemysl Čech, Tomáš Skopal, and Tomáš Pevný. k-nn classification of malware in https traffic using the metric space approach. In Michael Chau, G. Alan Wang, and Hsinchun Chen, editors, *Intelligence and Security Informatics*, pages 131–145, Cham, 2016. Springer International Publishing. ISBN 978-3-319-31863-9.

[48] Eric M Hutchins, Michael J Cloppert, and Rohan M Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1, 01 2011.

[49] Arna Magnúsardóttir. Malware is moving heavily to https. https://www.cyren.com/blog/articles/over-one-third-of-malware-uses-https, June 2017. Accessed: 2018.10.19.

[50] D. McGrew and B. Anderson. Enhanced telemetry for encrypted threat analytics. In *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, pages 1–6, Nov 2016. doi: 10.1109/ICNP.2016.7785325.

[51] Microsoft. Malware names. https://docs.microsoft.com/en-us/windows/security/threat-protection/intelligence/malware-naming. Accessed: 2018.12.05.

[52] Mitre. Command and control. https://attack.mitre.org/tactics/TA0011/, . Accessed: 2019.05.02.

[53] Mitre. Commonly used port, . URL https://attack.mitre.org/techniques/T1043/. Accessed: 2019.05.06.

[54] Mitre. Standard application layer protocol, . URL https://attack.mitre.org/techniques/T1071/. Accessed: 2019.05.06.

[55] Mitre. Domain fronting, . URL https://attack.mitre.org/techniques/T1172/. Accessed: 2019.05.14.

[56] Mitre. Remote access tools, . URL https://attack.mitre.org/techniques/T1219/. Accessed: 2019.05.14.

[57] Mitre. Web service, . URL https://attack.mitre.org/techniques/T1102/. Accessed: 2019.05.14.

[58] Mitre and Josh Abraham. Cobalt strike. URL https://attack.mitre.org/software/S0154/. Accessed: 2019.05.15.

[59] Y. Nir, S. Josefsson, and M. Pegourie-Gonnard. Elliptic curve cryptography (ecc) cipher suites for transport layer security (tls) versions 1.2 and earlier. RFC 8422, RFC Editor, August 2018.

[60] Digit Oktavianto and Iqbal Muhardianto. *Cuckoo Malware Analysis*. Packt Publishing, 2013. ISBN 1782169237, 9781782169239.

[61] Vern Paxson. Bro: A system for detecting network intruders in real-time. *Comput. Netw.*, 31(23-24):2435–2463, December 1999. ISSN 1389-1286. doi: 10.1016/S1389-1286(99)00112-7. URL http://dx.doi.org/10.1016/S1389-1286(99)00112-7.

[62] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[63] Jon Postel. Transmission control protocol. STD 7, RFC Editor, September 1981. URL http://www.rfc-editor.org/rfc/rfc793.txt. http://www.rfc-editor.org/rfc/rfc793.txt.

[64] P. Prasse, L. Machlica, T. Pevný, J. Havelka, and T. Scheffer. Malware detection by analysing network traffic with neural networks. In *2017 IEEE Security and Privacy Workshops (SPW)*, pages 205–210, May 2017. doi: 10.1109/SPW.2017.8.

[65] Paul Prasse, Gerrit Gruben, Lukas Machlika, Tomás Pevný, Michal Sofka, and Tobias Scheffer. Malware detection by https traffic analysis. 2017.

[66] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, Mar 1986. ISSN 1573-0565. doi: 10.1007/BF00116251. URL https://doi.org/10.1007/BF00116251.

[67] Eric Rescorla. The transport layer security (tls) protocol version 1.3@techreportRFC8422, author = Y. Nir and S. Josefsson and M. Pegourie-Gonnard, title = Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier, howpublished = Internet Requests for Comments, type = RFC, number = 8422, year = 2018, month = August, issn = 2070-1721, publisher = RFC Editor, institution = RFC Editor, . RFC 8446, RFC Editor, August 2018. URL https://www.rfc-editor.org/rfc/rfc8446.txt.

[68] Eric Rescorla, Kazuho Oku, Nick Sullivan, and Christopher Wood. Encrypted server name indication for tls 1.3. Internet-Draft draft-ietf-tls-esni-03, IETF Secretariat, March 2019. URL http://www.ietf.org/internet-drafts/draft-ietf-tls-esni-03.txt. http://www.ietf.org/internet-drafts/draft-ietf-tls-esni-03.txt.

[69] Martin Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Conference on System Administration*, LISA '99, pages 229–238, Berkeley, CA, USA, 1999. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1039834.1039864.

[70] Chris Sanders and Jason Smith. Chapter 6 - packet string data. In Chris Sanders and Jason Smith, editors, *Applied Network Security Monitoring*, pages 121 – 146. Syngress, Boston, 2014. ISBN 978-0-12-417208-1. doi: https://doi.org/10.1016/B978-0-12-417208-1.00006-4. URL http://www.sciencedirect.com/science/article/pii/B9780124172081000064.

[71] Chris Sanders and Jason Smith. Chapter 7 - detection mechanisms, indicators of compromise, and signatures. In Chris Sanders and Jason Smith, editors, *Applied Network Security Monitoring*, pages 149 – 173. Syngress, Boston, 2014. ISBN 978-0-12-417208-1. doi: https://doi.org/10.1016/B978-0-12-417208-1.00007-6. URL http://www.sciencedirect.com/science/article/pii/B9780124172081000076.

[72] Chris Sanders and Jason Smith. Chapter 4 - session data. In Chris Sanders and Jason Smith, editors, *Applied Network Security Monitoring*, pages 75 – 97. Syngress, Boston, 2014. ISBN 978-0-12-417208-1. doi: https://doi.org/10.1016/B978-0-12-417208-1.00005-2. URL http://www.sciencedirect.com/science/article/pii/B9780124172081000052.

[73] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. Avclass: A tool for massive malware labeling. In Fabian Monrose, Marc Dacier, Gregory Blanc, and Joaquin Garcia-Alfaro, editors, *Research in Attacks, Intrusions, and Defenses*, pages 230–253, Cham, 2016. Springer International Publishing. ISBN 978-3-319-45719-2.

[74] Iman Sharafaldin, Arash Habibi Lashkari, and Ali Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. pages 108–116, 01 2018. doi: 10.5220/0006639801080116.

[75] Michael Sikorski and Andrew Honing. *Practical Malware Analysis*. no starch press.

[76] George Stergiopoulos, Alexander Talavari, Evangelos Bitsikas, and Dimitris Gritzalis. Automatic detection of various malicious traffic using side channel features on tcp packets. In Javier Lopez, Jianying Zhou, and Miguel Soriano, editors, *Computer Security*, pages 346–362, Cham, 2018. Springer International Publishing. ISBN 978-3-319-99073-6.

[77] František Střasák. Detection of https malware traffic. Bachelor's thesis, Czech Technical University in Prague, May 2017. URL `https://dspace.cvut.cz/bitstream/handle/10467/68528/F3-BP-2017-Strasak-Frantisek-strasak_thesis_2017.pdf`.

[78] Alexey Tsymbal. The problem of concept drift : definitions and related work. 2004.

[79] I. You and K. Yim. Malware obfuscation techniques: A brief survey. In *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*, pages 297–300, Nov 2010. doi: 10.1109/BWCCA.2010.85.

[80] Indre Zliobaite. Learning under concept drift: an overview. *CoRR*, abs/1010.4784, 01 2010.

A Example of correlated malware flow

The JSON structure below shows an example of a correlated malware flow. Fields related to counting of TLS values are already pre-processed.


```
{
  "capture_date": "2018/01/03",
  "dest_ip": "95.46.8.65",
  "dest_port": 443,
  "family": "cobalt",
  "flow_age": 926,
  "flow_bytes_toclient": 3276,
  "flow_bytes_toserver": 1317,
  "flow_pkts_toclient": 9,
  "flow_pkts_toserver": 11,
  "label": "1",
  "packets_to_client": [
    {
      "payload_size": 1460,
    },
    {
      "payload_size": 672,
    },
    {
      "payload_size": 91,
    },
    {
      "payload_size": 213,
    },
    {
      "payload_size": 261,
    },
    {
      "payload_size": 69,
    }
  ],
  "packets_to_server": [
```

```
    {
      "payload_size": 188,
    },
    {
      "payload_size": 166,
    },
    {
      "payload_size": 357,
    }
  ],
  "pcap_filename": "CTU-Malware-Capture-Botnet-345-1.pcap",
  "source": "CTU",
  "src_ip": "192.168.1.121",
  "src_port": 49202,
  "timestamp": "1970-01-01T00:50:32.057061+0000",
  "tls_Cipher": "60-47-61-53-5-10-49191-49171-49172-49195-
       49187-49196-49188-49161-49162-64-50-106-56-19-4",
  "tls_Cipher_Count": 21,
  "tls_EllipticCurve": "23-24",
  "tls_EllipticCurvePointFormat": "0",
  "tls_EllipticCurvePointFormat_Count": 1,
  "tls_EllipticCurve_Count": 2,
  "tls_SSLExtension": "0-10-11-13",
  "tls_SSLExtension_Count": 4,
  "tls_SSLVersion": "771",
  "tls_ja3hash": "d7150af4514b868defb854db0f62a441"
}
```

B Code for Metadata Collector

The following sections include the source code of the Metadata Collector written in Golang. The program consists of three different files: main.go, community_id.go and packets.go.

B.1 main.go

This file is the main file, which consists of the main logic for the program.

```
// main.go
package main

import (
    "flag"
    "log"
    "encoding/json"
    "os"
    "time"
    "net"
    "./community_id"
    . "./packets"
    "github.com/google/gopacket"
    "github.com/google/gopacket/layers"
    "github.com/google/gopacket/examples/util"
    "github.com/google/gopacket/pcap"
    "os/signal"
    "syscall"
)



func isRetrassmission(packets *[]Packet, packet Packet) bool {
    packet_len := len(*packets)
    if packet_len == 0 {
        return false
    }
    prevPacket := (*packets)[packet_len-1]
    prevSeq := prevPacket.SeqNumber
    prevAck := prevPacket.AckNumber
    if prevSeq == packet.SeqNumber && prevAck == packet.AckNumber {
        log.Println("Detected retransmission")
```

```
        return true
    } else {
        return false
    }
}

func writeAllSessions(s Sessions) {
    for _, session := range s {
        sessionJson, err := json.Marshal(session)
        if err != nil {
            panic(err)
        }
        writeOutput(sessionJson, *outputFilename)
    }
}


func writeOutput(content []byte, filename string) {
    f, err := os.OpenFile(filename, os.O_APPEND|os.O_WRONLY|os.O_CREATE, 0600)
    if err != nil {
        panic(err)
    }

    defer f.Close()

    if _, err = f.WriteString(string(content)+"\n"); err != nil {
        panic(err)
    }
}


var iface = flag.String("i", "eth0", "Interface to get packets from")
var fname = flag.String("r", "", "Filename to read from, overrides -i")
var snaplen = flag.Int("s", 1600, "SnapLen for pcap packet capture")
var filter = flag.String("f", "tcp", "BPF filter for pcap")
var logAllPackets = flag.Bool("v", false, "Logs every packet in great detail")
var outputFilename = flag.String("l", "packets.json", "Filename of output file")
var packetCount = flag.Int("c", 50, "Packet count to store")


type Sessions map[string]*Session

func main() {

    var sessions = make(Sessions)
    defer util.Run()()
```

```
var handle *pcap.Handle
var err error

c := make(chan os.Signal)
signal.Notify(c, os.Interrupt, syscall.SIGTERM)
go func() {
    <-c
    writeAllSessions(sessions)
    os.Exit(1)
}()

// Set up pcap packet capture
if *fname != "" {
    log.Printf("Reading from pcap dump %q", *fname)
    handle, err = pcap.OpenOffline(*fname)
} else {
    log.Printf("Starting capture on interface %q", *iface)
    handle, err = pcap.OpenLive(*iface, int32(*snaplen), true,
     ↪  pcap.BlockForever)
}
if err != nil {
    log.Fatal(err)
}

if err := handle.SetBPFFilter(*filter); err != nil {
    log.Fatal(err)
}

// Set up assembly
log.Println("reading in packets")
// Read in packets, pass to assembler.
packetSource := gopacket.NewPacketSource(handle, handle.LinkType())
packets := packetSource.Packets()
//counter := 0
for {
    select {
    case packet := <-packets:
        // A nil packet indicates the end of a pcap file.
        if packet == nil {
            writeAllSessions(sessions)
            return
        }
        if *logAllPackets {
            log.Println(packet)
        }
```

```
if packet.NetworkLayer() == nil || packet.TransportLayer() == nil ||
 ↪  packet.TransportLayer().LayerType() != layers.LayerTypeTCP {
    log.Println("Unusable packet")
    continue
}
var SrcIP net.IP
var DstIP net.IP
var PacketSize uint64
if ipv4Layer := packet.Layer(layers.LayerTypeIPv4); ipv4Layer != nil
 ↪  {
    ipv4 := ipv4Layer.(*layers.IPv4)
    SrcIP = ipv4.SrcIP
    DstIP = ipv4.DstIP
    PacketSize = uint64(ipv4.Length)
} else if ipv6Layer := packet.Layer(layers.LayerTypeIPv6); ipv6Layer
 ↪  != nil {
    ipv6 := ipv6Layer.(*layers.IPv6)
    SrcIP = ipv6.SrcIP
    DstIP = ipv6.DstIP
    PacketSize = uint64(ipv6.Length)
} else {
    panic("Not ipv4 or ipv6!")
}
tcp := packet.TransportLayer().(*layers.TCP)
timestamp := packet.Metadata().Timestamp
s := Session{Timestamp: timestamp, DestIP: DstIP, SourceIP: SrcIP,
 ↪  DestPort: uint16(tcp.DstPort), SourcePort: uint16(tcp.SrcPort)}
hash := community_id.Generate(s, 32)
if _, ok := sessions[hash]; ! ok {
    sessions[hash] = &s
}
var pkt Packet
var tmpPkts *[]Packet
if tcp.DstPort < tcp.SrcPort {
    tmpPkts = &sessions[hash].PacketsToServer
} else {
    tmpPkts = &sessions[hash].PacketsToClient
}
if len(*tmpPkts) < *packetCount {
    pkt.PayloadSize = uint64(len(tcp.Payload))
    pkt.SeqNumber = tcp.Seq
    pkt.AckNumber = tcp.Ack
    if pkt.PayloadSize != 0 && !isRetrassmission(tmpPkts, pkt) {
        pkt.PacketSize = PacketSize
        pkt.PayloadRatio = float64(pkt.PayloadSize) /
         ↪  float64(pkt.PacketSize)
```

```
                    pkt.Time = timestamp.UnixNano() / int64(time.Millisecond)
                    if len(*tmpPkts) != 0 {
                        prevPacket := (*tmpPkts)[len(*tmpPkts)-1]
                        pkt.TimediffPrevPkt = pkt.Time - prevPacket.Time
                        pkt.RatioPrevPkt = float64(pkt.PacketSize) /
                        ↪  float64(prevPacket.PacketSize)
                    }
                    *tmpPkts = append(*tmpPkts, pkt)
                }
            }

        }
    }
}
```

B.2 community_id/community_id.go

This file implements the flow identifier. This identifier is used to correlated network packets in the same network flow.

```
// community_id/community_id.go
// Implementation of community flow id
// https://github.com/corelight/community-id-spec/blob/master/community-id.py


package community_id

import (
    "../packets"
    "fmt"
    "crypto/sha1"
    "net"
    "encoding/binary"
    "encoding/base64"
    "bytes"
)
type ipv4 struct {
    seed uint16
    src_addr net.IP
    dst_addr net.IP
    proto uint8
    pad0 uint8
    sp  uint16
    dp  uint16
}
```

```
func ip2int(ip net.IP) uint32 {
    if len(ip) == 16 {
        return binary.BigEndian.Uint32(ip[12:16])
    }
    return binary.BigEndian.Uint32(ip)
}

func int2bytes(i_int interface{}) []byte {
    buf := new(bytes.Buffer)
    err := binary.Write(buf, binary.LittleEndian, i_int)
    if err != nil {
        fmt.Println("binary.Write failed:", err)
    }
    return buf.Bytes()
}

func Generate(pkt packets.Session, seed uint16) string {
    var hashstate = sha1.New()
    var r ipv4
    r.seed = seed
    if ip2int(pkt.SourceIP) < ip2int(pkt.DestIP) || (ip2int(pkt.SourceIP) ==
    ↪  ip2int(pkt.DestIP) && pkt.SourcePort < pkt.DestPort) {
        r.src_addr = pkt.SourceIP
        r.dst_addr = pkt.DestIP
        r.sp = pkt.SourcePort
        r.dp = pkt.DestPort
    } else {
        r.src_addr = pkt.DestIP
        r.dst_addr = pkt.SourceIP
        r.sp = pkt.DestPort
        r.dp = pkt.SourcePort
    }
    r.proto = 0x1
    r.pad0 = 0
    hashstate.Write(int2bytes(r.seed))
    hashstate.Write(r.src_addr)
    hashstate.Write(r.dst_addr)
    hashstate.Write(int2bytes(r.proto))
    hashstate.Write(int2bytes(r.pad0))
    hashstate.Write(int2bytes(r.sp))
    hashstate.Write(int2bytes(r.dp))
    return base64.StdEncoding.EncodeToString(hashstate.Sum(nil))
}
```

B.3 packets/packets.go

This file implements the structure used to hold information about network flows. This structure also includes the fields exported to the JSON file.

```
// packets/packets.go
package packets


import (
    "net"
    "time"
)
type Packet struct {
    PacketSize uint64 `json:"packet_size"`
    PayloadSize uint64 `json:"payload_size"`
    PayloadRatio float64 `json:"payload_ratio"`
    RatioPrevPkt float64 `json:"ratio_prev_pkt"`
    Time int64 `json:"-"`
    SeqNumber uint32 `json:"-"`
    AckNumber uint32 `json:"-"`
    TimediffPrevPkt int64 `json:"time_diff_prev_pkt"`
}

type Session struct {
    Timestamp time.Time `json:"timestamp"`
    SourceIP net.IP `json:"src_ip"`
    DestIP net.IP `json:"dest_ip"`
    SourcePort uint16 `json:"src_port"`
    DestPort uint16 `json:"dest_port"`
    PacketsToClient []Packet `json:"packets_toclient"`
    PacketsToServer []Packet `json:"packets_toserver"`
}
```

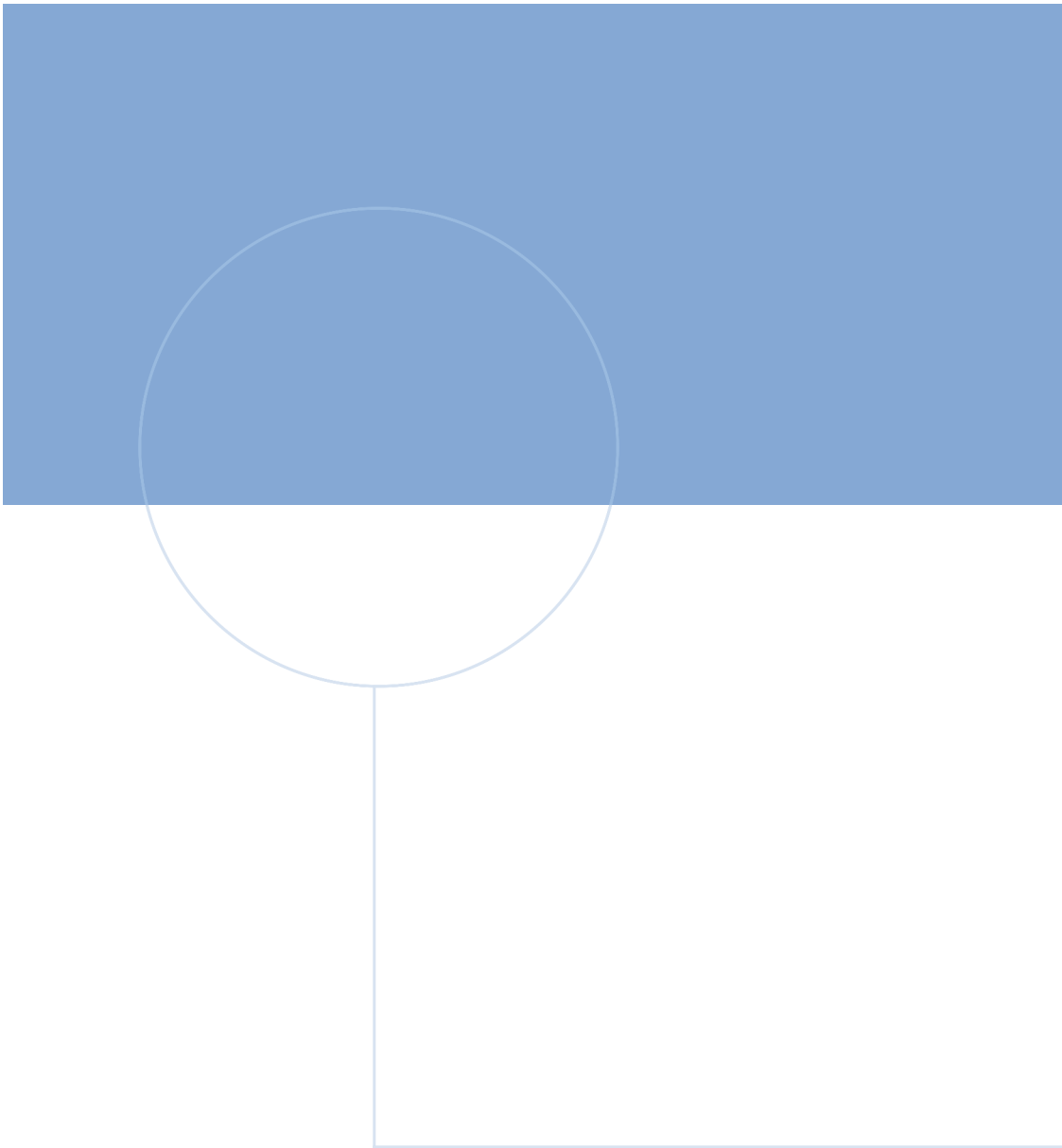
C Implementation of first-order markov chain for packet payload

The following python code includes the implementation of the first-order Markov chain used in this thesis used to model a sequence of network packets' payload.

```
def getFlowPacketPayload(packet_sizes):
        data = []

        numRows = 10
        binSize = 150.0
        transMat = np.zeros((numRows,numRows))

        if len(packet_sizes) == 1:
            curPacketSize = min(int(packet_sizes[0]/binSize),numRows-1)
            transMat[curPacketSize,curPacketSize] = 1
            return list(transMat.flatten())
        # get raw transition counts
        for i in range(1,len(packet_sizes)):
            prevPacketSize = min(int(packet_sizes[i-1]/binSize),numRows-1)
            curPacketSize = min(int(packet_sizes[i]/binSize),numRows-1)
            try:
                transMat[prevPacketSize,curPacketSize] += 1
            except:
                print(packet_sizes)
        # get empirical transition probabilities
        for i in range(numRows):
            if float(np.sum(transMat[i:i+1])) != 0:
                transMat[i:i+1] = transMat[i:i+1]/float(np.sum(transMat[i:i+1]))
        return list(transMat.flatten())
```



NTNU
Norwegian University of
Science and Technology