Norwegian University of
Science and Technology

A Study of Applied Passive TLS Analysis

Anders Sefjord Torbjørnsen

15-12-2018

Master's Thesis
Master of Science in Information Security
30 ECTS
Department of Information Security and Communication Technology
Norwegian University of Science and Technology,

Supervisor: Prof. Slobodan Petrovic
Co-Supervisor: Christoffer V. Hallstensen

Preface

This master thesis was carried out at the Norwegian University of Science and Technology at Gjøvik, during 2018. It is the final assignment of the master's program Information Security. The main subject of the thesis is discovering TLS encrypted malicious network traffic passively. This means that the privacy of users are kept intact, as the privacy of user would be somewhat broken if the network traffic was to be decrypted and then encrypted on the fly. While some research is published on this particular topic, it is not a well-researched topic. The thesis use an already existing, open source tool to extract the meaningful features of encrypted network traffic, and then an artificial neural network is applied to the dataset to classify the traffic as either benign or malicious. As the thesis moved forward, several subproblems occurred, such as obtaining an up-to-date dataset of benign traffic. As there was no existing good solution for publicly available datasets, a simple network traffic generator was made in order to create a baseline.

The idea of the master thesis was suggested by Christoffer V. Hallstensen at the section for digital security at NTNU, Gjøvik.

This thesis is intended for those that are interested in a combination of network security and machine learning, and perhaps especially for those that are interested in traffic from malware, encrypted with TLS.

15-12-2018

Anders Sefjord Torbjørnsen

Acknowledgment

I would like to thank my supervisor Slobodan Petrovic for being helpful and providing good ideas throughout this master thesis. A thank you is also directed to my co-supervisor, Christoffer V. Hallstensen for coming up with the topic of this master thesis, and taking the time and providing helpful insight in topics covered in this thesis.

A big thank you to fellow students for interesting discussions and different angles, especially (in no particular order) Jan Petter Berg Nilsen, Ingrid Larsen, Jørgen Ellingsen and Emil Ry. I would also like to thank Anders Granerud for taking the time to read through the thesis and provide constructive feedback.

A.S.T

Abstract

While the Internet is moving towards more and more encryption of the network traffic, it is also a trend that is picked up by authors of malware. The traditional way of detecting malicious traffic or malicious behaviour on the network is to use a signature-based network intrusion detection system. A signature-based system relies on reading the network traffic in plaintext in order to detect patterns, which it is not able to do when the traffic is encrypted. One work-around for this problem is to use SSL/TLS inspection.

Instead of breaking end users privacy by inspecting what they believe is encrypted communication, this thesis investigates the possibility of detecting malicious TLS encrypted network traffic passively. By taking a look at properties exchanged when the encrypted network communication channel is established, as well as the behaviour of the network traffic, the thesis uses these properties in a machine learning algorithm to classify network traffic as either benign or malicious. While the machine learning algorithm is easy to implement in a proof-of-concept, the lack of publicly, up-to-date datasets of benign, encrypted network traffic with TLS is almost non-existent. This leads to the creation of a TLS encrypted network traffic generator that creates a baseline for what is considered as benign TLS traffic. Malicious TLS traffic is collected from open sources, and run through a multilayer perceptron with backpropagation. Two experiments were carried out during this thesis; one leading to a correct classification rate of 83% using network behaviour. The other experiment looked at the ciphersuites found in the TLS handshake of the traffic, and had a correct classification rate of 80%.

Sammendrag

Ettersom bruk av internett stadig beveger seg mer mot bruk av kryptert nettverkstrafikk, så er dette også en trend som er plukket opp av de som lager skadevare. Den tradisjonelle måten å detektere skadelig nettverkstrafikk eller skadelig atferd er bruken av signaturbasert inntrengingsdeteksjon. Et signaturbasert system er avhengig av å lese nettverkstrafikken i klartekst for å oppdage mønster, noe som blir umulig å gjøre dersom nettverkstrafikken er kryptert. En metode for å omgå denne problemstillingen er å benytte SSL/TLS inspeksjon.

I stedet for å bryte personvernet til sluttbrukere ved å inspisere det sluttbrukere tror er kryptert kommunikasjon, så ser denne masteroppgaven på muligheten for å detektere skadelig TLS-kryptert nettverkstrafikk passivt. Ved å ta en titt på egenskaper utvekslet i det den krypterte kommunikasjonskanalen opprettes, i tillegg til atferden nettverkstraffiken har. Masteroppgaven bruker disse egeneskapene til å klassifisere trafikk som enten skadelig eller legitim. Ettersom en maskinlæringsalgoritme er enkelt å implementere i et proof-of-concept, så er det en mangel på et oppdatert offentlig datasett med legitim, TLS-kryptert nettverkstrafikk. Skadelig TLS-trafikk hentes fra åpne kilder, og kjøres gjennom en MLP-algoritme. To eskperimenter ble gjennomført i løpet av masteroppgaven; en ga en korrekt klassifiseringsprosent på 83% ved hjelp av atferd av nettversktrafikken. Det andre eksperimentet klassifiserte 80% korrekt basert ciphersuites som ble hentet ut fra TLS-håndtrykket fra trafikken.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Keywords

Encrypted network traffic, TLS, Neural networks, Intrusion detection, network traffic generation.

1.2 Topics Covered

Over the last couple of years, usage of encryption on websites have increased rapidly. Google's transparency report states that, as of 11th of March 2018, 92% of Googles services are delivered through HTTPS [1]. In January 2014, the percentage was at 50%. In February 2018, it was reported that roughly 40% from Alexa's top one million websites redirected to HTTPS, a number which was a little over 10% in January 2016 [2]. While the increased usage in encryption on the web benefits the end user with more privacy and security, it also poses a challenge to the ones monitoring corporate networks. Zscaler reports that they saw an increase of 30% in 6 months in 2017 on malware that uses TLS [3]. As traditional network-based intrusion detection systems relies on inspection of the network traffic in order to detect malware, other detection methods must be used. One way of solving this challenge is by using a technique called SSL/TLS-inspection. This is however a method that does not preserve the end-users privacy as well as perhaps being unfeasible in larger networks.

TLS relies on certificates, and more specifically certificate authors that validate certificates before they can be used. Letsencrypt [4], launched in 2016, enables free and automated signing of certificates, making it easier than ever before to obtain a valid TLS certificate. On one hand, this approach enables web sites to implement TLS without a cost, and implements encrypted network traffic to its services. On the other hand, this also enables people with bad intentions to obtain a free, valid TLS certificate. Between January 2016 and March 2017, Letsencrypt issued a little over 15 000 certificates that had the word "PayPal" in the certificate [5]. These certificates were among other things used in phishing, which would be harder to achieve had the certificate authority had to inspect the certificate

requests manually.

Traditional security monitoring of networks are using signatures to detect potential malicious network communication. Such signatures often rely on the capability to read the content of the network traffic, for example a string that is known to be part of malicious traffic. The signature-based approach of detecting malicious traffic will always be based on known, previously seen or expected network traffic behaviour. This means that if a signature is created to catch for instance a previously seen patterns, a slight change to this pattern may go unnoticed by the security monitoring system. Another approach that authors of malware may employ to circumvent the signature-based detection is to design their malware to communicate through encrypted communication channels. By deploying malware with TLS, the security monitoring systems will not be able to read the content, due to the encrypted traffic, and thus unable to alert on the malicious traffic.

While encrypted network traffic is currently bypassing signature-based intrusion detection systems, it exists methods that complement an IDS. Domain reputation lists and IP blacklists can help detecting such traffic if previously known-bad domains or IP addresses is reused. Host-based intrusion detection systems, such as antivirus solutions, are still a somewhat efficient countermeasure to malware, but encrypted network communication still makes it more difficult to detect malicious software based on the network behavior.

Network traffic generators that generate realistic, TLS encrypted network traffic is needed for experimenting with other solutions to malicious network traffic. One application for a network traffic generator is anomaly-based intrusion detection systems that are designed to learn what normal or good network behaviour is. The generator may create datasets that are shared amongst researchers and applied to various solutions so that they are able to compare solutions that utilize anomaly detection.

1.3 Problem Description

While there is nothing wrong with encryption of network traffic, it can be used for providing users privacy as well as hiding malware communication. When malware authors implement encrypted communication channels in malware, it

makes it harder for the intrusion detection system to detect. A costly solution to this is to deploy software and hardware that decrypts the encrypted traffic before it leaves the network before inspecting the data, and then reencrypt and send the data to its intended destination. This also breaks the privacy of the users, possibly making them believe they have established a secure communication channel when all of their network traffic is read in plaintext by an intrusion detection system.

This master thesis aims to develop an easy-to-use network traffic generator for encrypted network traffic, that also considers the context and the value of real services. The network traffic generator is capable of generating TLS traffic through simulated web browsing. By extracting unencrytped features and behaviour from the generated network traffic, a classifier using a multilayer perceptron with backpropagation is applied to test the generated network traffic against encrypted malware traffic. This is in essence a proof-of-concept anomaly detection network intrusion detection system for TLS encrypted network traffic. Because datasets on realistic network traffic are difficult to obtain, this thesis provides a method of creating a dataset of its own, enabling others to similarly create their own dataset, without the need of sharing possibly sensitive data that may be extracted from public datasets.

1.4 Justification, Motivation and Benefits

The motivation for writing a thesis on this particular topic is the potential of discovering malicious TLS encrypted network traffic, without SSL/TLS inspection. As well as being an interesting subject, the potential gain of this thesis is to gain insight in how to detect the malicious network traffic while at the same time preserving end users' privacy.A benefit of the thesis will be a methodology of how to both gather malicious TLS traffic from free and open sources, but also a network traffic generator to generate TLS traffic that may be applied by others.The thesis will look into, as well as use, standardized software and file formats, so that it may benefit others free of cost. Features of interest in TLS traffic will be examined, and machine learning will be applied to review whether the features are applicable or not.

1.5 Research Questions

This master thesis has two main questions, with two subquestions, where the main research question of this thesis is:

- How can encrypted malicious traffic be detected without decryption?
  - Which features are relevant in terms of detecting encrypted malicious traffic?
  - How to generate a realistic dataset for testing detection of malicious encrypted traffic?
- In which cases are passive traffic analysis better or worse than active traffic analysis?

1.6 Contribution

Since the usage of TLS increases, there is a need for passive TLS analysis to detect malicious TLS traffic used in malware and by attackers. The contribution of this thesis will be to provide insight in passive analysis of TLS encrypted communication, as well as looking into features and behaviour from network traffic that may be used to classify network traffic as malicious or benign. The proof-of-concept software and methodology developed in this thesis will become available. This may may also be of help to others doing experiments or researching TLS traffic. By creating a network generator, a baseline for testing intrusion detection system is created. This may be of help to others designing an anamoly detection system. The proof-of-concept will also potentially display that there is a possibility to detect malicious, encrypted network traffic without using TLS inspection.

Different software that may be used to generate a network traffic dataset is also examined, and this master thesis will give an insight into why these were not used in this thesis.

The thesis will also discuss advantages and disadvantages that arise with the usage of passive analysis, and compare it against the active TLS inspection. This may be of help to others that are considering a system that will analyze encrypted traffic, and help them make a decision.

1.7 Ethical and Legal Considerations

An ethical consideration that was done in this thesis was the use of real websites in the network traffic generator that was made. Since the collection of data to the network traffic generator was done in such a small scale (from one to two requests per site), it was considered to be okey as the websites selected for the generator usually have a high amount of traffic to their websites. However, the network traffic generator may be used to create a high volume of traffic to a supplied list of websites.

A way to circumvent the usage of real, live web sites would be to set up our own network with different services. This could either be done through a server with virtual machines, on multiple virtual private servers or on a software-defined network. However, this approach would not have the same realism as real network traffic to established websites.

Another consideration that has been done is the falsification of the user-agent in the aforementioned network traffic generator. This is done in order to generate network traffic that is as close as possible to "a real user". As websites are able to treat visitors differently, and are able to deny users' access to a website solely based on the user agent, this has been considered okey to perform as the traffic generator generates such a small amount of traffic.

1.8 Outline

This master thesis is structured into 7 chapters. Chapter 1 is the introduction, where the research questions, problem description and the contributions of the thesis are presented and discussed. Chapter 2 consists of the background topics for this thesis, which is created through a study of literature and research on the subject. In chapter 3, "Related Work", the state-of-the art within topics covered by these thesis is presented. Chapter 4 elaborates on the experimental method behind the thesis. Chapter 5 is the chapter where the results are presented. The 6th chapter contains the discussion, along with limitations and things that that were considered and later abandoned. The last chapter concludes the master

thesis, and presents ideas for future work.

2 Background

2.1 TLS Encrypted Network Traffic

As Internet browsing became more and more popular, encryption of the network traffic became a need in order to securely communicate with services such as banks and online shopping websites. Netscape created the first version of Secure Sockets Layer (SSL), although not publicly released [6]. SSL v2 [7] was released in 1995, which implemented MD5 instead of the old CRC method implemented in version 1. Microsoft later released an improved version of SSL 2.0 called Private Communication Techonlogy (PCT) in 1995 [8]. The draft of SSL 3.0 was released in 1996, and TLS 1.0 was proposed as an upgrade in 1999. TLS 1.1 was defined in 2006. TLS version 1.2 was released in 2008, and the newest version is TLS 1.3, released ten years after TLS 1.2, in August 2018 [9]. TLS v1.3 has implemented these changes:

- Removed deprecated ciphersuites.
- Added new and better ciphersuites.
- Added a feature called 0 round-trip time.
- Enforcing encryption on all messages sent after the ServerHello message.
- Increased the speed of the handshake.

Transport Layer Security operates on the 7th layer of the OSI model, the application layer. It is stated RFC for TLS v1.2 that:

> *The primary goal of the TLS protocol is to provide privacy and data integrity between two communicating applications*" *[10]*.

In essence, this means that TLS is supposed to prevent eavesdropping of network traffic, so that a client and a host may send encrypted messages to each other that an eavesdropper is unable to read. Data integrity is needed to prove that the data is sent is valid. HTTPS uses TLS, and is often implemented in browsers

to ensure integrity and confidentiality for clients and servers. The standard port used for HTTPS is port 443.

Private-public key cryptography is used in TLS, where certificates in the form of X.509 is used. This format of certificate is specified in RFC 5280 [11]. X.509 Certificates have to be digitally signed in order to be valid, either by the server that it is using it itself (self-signed) or by a trusted authority (Certificate Authority), CA.

The following happens when a new session is established in TLS v1.2: First, the client sends a "Client hello" message. This message includes the supported ciphersuites of the client (algorithms for encryption), supported compression methods by the client, and extensions supported. The server will then respond with a "Server hello" message if the cipher suites that is proposed by the client is acceptable to the server. This message includes among other things the ciphersuite the server wants to use, and must be one of the cipher suites that the client initially sent. It also includes a compression method, this must also be one of the initial presented compression methods presented by the client. After the "Server hello" message is sent, the server sends a certificate message. This contains the certificate of the server, as well as the certificate of the certificate author (CA). The last certificate in this chain will be from a root CA, and is self-signed. As the client should have a record of all the signed root CAs, it should check its validity. The key exchange protocol is next, and this is where the client and server agrees upon a secret key. This uses the previously agreed upon cipher suite to generate the keys. The server sends its part of the key to the client with the "Server key exchange" message, and then sends a "Server hello done" message to tell the client that the server is finished. The client sends its key through the "Client key exchange" message, followed by the "Change cipher spec" message. This indicates that the handshake is finished, and the rest of the communication is to be continued using the agreed upon keys and encryption scheme. The last messages that is sent in the handshake is the "Encrypted handshake message". This contains a hash of all the previously messages sent. Table 1 summarizes the unencrypted messages that is sent between client and server.

Table 1: Summary of Unencrypted Messages in the TLS 1.2 Handshake

Message	**From**	**Information in message**
ClientHello	Client	Supported cipher suites Supported compression methods Supported extensions Random bytes Session ID Version of TLS
Server hello	Server	Selected cipher suite Selected compression method Selected extensions Random bytes Session ID Version of TLS
Certificate	Server	Certificate chain of servers certificate
Server key exchange	Server	Various cryptographic information
Server hello done	Server	Finished sending information
Client key exchange	Client	Various cryptographic information
Change cipher spec	Client	Compressed value to indicate that future messages will be encrypted

The TLS handshake initiates an encrypted session. That means that after a couple of messages back and forth (the handshake), the client and the server agrees on an encryption setup that only the client and the host are able to decrypt and read, thus preventing eavesdropping and at the same time enabling data integrity. Figure 1 shows the messages that is sent in the TLS handshake.

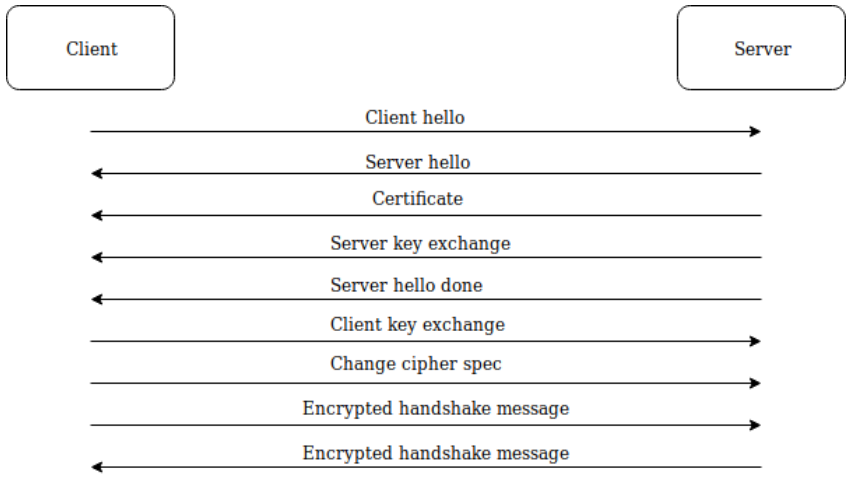

Figure 1: TLS 1.2 handshake

After the handshake is finished, encrypted data can be sent back and forth between the client and the server.

TLS v1.3 has some minor modifications in the handshake that version 1.2 does not have

- The Server Name Indication (SNI) is now mandatory
- The certificate sent from the server is now encrypted
- Allows resumption of recent established sessions

2.2 Network-based Intrusion Detection Systems

Network-based intrusion detection systems are designed to alert when an intrusion occurs, by passively monitoring the traffic that flows in a network. There are two existing broad classes of IDSs today; Network-based IDS (NIDS) and host-based IDS (HIDS). A host-based intrusion detection system will be installed on each individual computer in a network (endpoints), and monitor a combination of disk activities, memory activities as well as network activity performed by the computer. Typically, this is presented as antivirus software, which also includes intrusion prevention system (IPS) to take actions on the alerts created by the built-in IDS. This means that the software is capable of detecting through the HIDS, and prevent it using the intrusion prevention module, by for instance

putting files in quarantine.

One of the reasons for having a network-based intrusion detection system is to catch malicious or uncommon behaviour on the network. Intrusion detection systems can be separated into two classes, based on their detection method; signature-based or anomaly-based [12].

The components in an IDS is typically a preprocessor, a detection algorithm and an alert filter. Figure 2 presents the flow of the data in an Network based IDS.

The pre-processor of the IDS is used to among other things decode the net-

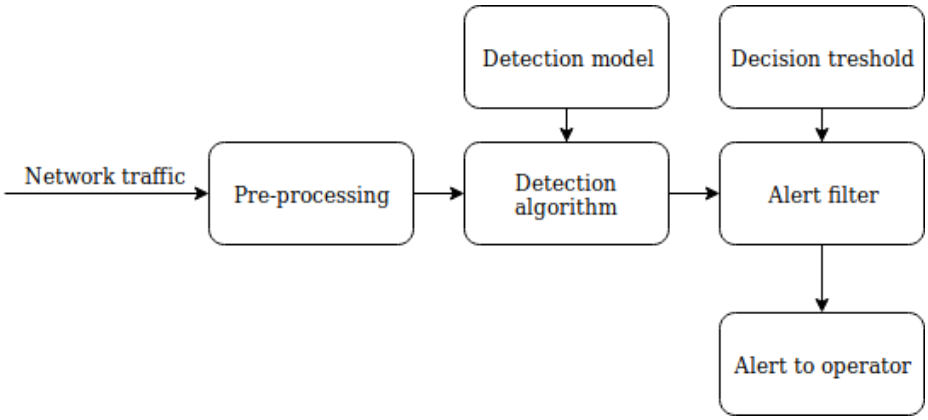

Figure 2: Flow of IDS

work data and detecting the underlying protocol that is being used. The data is then sent to the detection algorithm. Here the algorithm will either use pattern matching (signature-based) or compare the traffic to what is normal/abnormal (anomaly based). The detection algorithm send its result to the alert filter, which uses a decision threshold in order to decide if the event shall be notified to an operator that may decide to investigate the incident further.

Signature-based intrusion detection systems relies on signatures written of known attacks. This means that in theory, the IDS is able to detect all of the previ-

ous known attacks, but incapable of detecting a new attack, or even a slightly changed previous attack. Metamorphic malware employs such a technique that it is changed slightly each time it is run [13]. Metamorphic traffic may also be designed, so that a signature-based NIDS is not capable of detecting it.

A typical signature for network-based IDSs is simply definitions of what the IDS is looking for. Rules may be created in such a manner that tells the IDS that it should send an alert if the pattern that is defined in the rule matches. A single rule could be so easy as the following pseudo-code:

"*Create an Alert if source IP equals destination IP*"

The rule could also be so complex that it is defined how many bytes into the packet the pattern must match, with regular expressions, ascii and hexadecimal values.

Anomaly-based intrusion detection systems are based on defining "normal" traffic. An example would be to run the IDS in an environment for a couple of days to "learn" what kind of traffic that is normal in this particular network. During the learning period, the IDS will not trigger alerts, but rather try to build a level of understanding of how the network traffic is supposed to look like or behave. A problem with deploying the IDS in a production environment would be if an attacker has already compromised a machine in the network. The anomaly detection will thus have a false interpretation of how the network traffic should behave, and the network traffic may go unnoticed later. After the training period is over, the anomaly-based IDS is activated, and will generate alerts based on what the IDS defines as abnormal or unusual network traffic. This method is however error-prone, as it has a high rate of false positives because of the complexity of network traffic. New devices or protocols added to the network may potentially trigger a lot of alerts because the IDS has never seen it before and thus label it as an anomaly. While an anomaly-based IDS in theory are able to detect slight changes in network traffic that a signature-based IDS are not able to, the amount of false positive alerts such an IDS potentially are creating may be really high [14].

An IDS based on anomaly detection requires a baseline, or ground truth, of what normal traffic is. There is however few publicly available data sets available that is up to date. As Sommer and Paxson [15] states;

> *A challenging problem in the evaluation of anomaly detection methods is the lack of test data with ground truth, due to the limited availability of such data.*

One concern in making a real dataset public, derived from a network in production, is revelation of sensitive data that may be extracted from the dataset. Anonymizing the data set before public release has been proposed, but with larger data sets, the anonymization is not that easy: Coull et al. [16] showed that it is possible to deanonymize a network data set, and extract information such as subnets and real IP addresses even though the dataset was supposed to be anonymized. Manual confirmation of anonymization of the dataset is in most cases unfeasible, as inspecting each network packet in a large data set is very time consuming.

One way of generating a labeled dataset of ground truth is by manually inspecting the data and label it by expert knowledge. This is however difficult to carry out, as the amount of data often is overwhelming. Since one of the most common data sets, the KDD99 dataset [17] is greatly outdated, there is a need for an improved data set in order to test various IDS solutions and implementations up against each other.

Since there is a need for evaluating performance in an intrusion detection system, several metrics may be applied in order to test the performance of a specific IDS against another. It does however not exist a set of agreed upon or standardized metrics. An IDS can classify an event as either true or negative, but in theory, there are four possible outcomes when an IDS classifies an event:

- True positive
- True negative
- False positive
- False negative

A true positive is when an alarm is correctly triggered, and there has been an incident. A true negative is when an alarm is not triggered, and there has not been

an incident. A false positive is when an alarm is triggered, but should not have been, because there has not been an incident. A false negative is when an alarm is not triggered, but an incident has occurred. From these four different types, it is obvious that the most wanted outcome of an alarm would be true positives and true negatives. While the worst kind of event probably is a false negative, because the operator will not receive an alarm of an incident, the amount of false positives may also be disrupting, depending on the amount of alarms.

These four types of events may also be represented in a confusion matrix, as seen in table 2:

Table 2: Simple Confusion Matrix

	P	N
P	True positive	False positive
N	False negative	True negative

2.3 Malicious Network Traffic

By first defining malware, we establish the context around the definition of malicious traffic. In the National Information Assurance (IA) Glossary, the Committee on National Security Systems defines malicious code as:

> *Software or firmware intended to perform an unauthorized process that will have adverse impact on the confidentiality, integrity, or availability of an information system.* [18]

As there exists multiple reason for creating such software, here are a few:

- Damage reputation
- Financial gain
- Stealing information/intelligence

- Showcase skill
- Political reasons

As some of the more advanced types of malware is not only a hit-and-run, or created as a quick information stealer or credential harvester, some types are professional made by probably state-actors or large organizations.

More sophisticated forms of malware may incorporate so-called command and control infrastructure. The "Mitre ATT&CK Matrix for Enterprise" has an entire column dedicated to this technique [19]. This is one technique that also is used for botnets, where one or more "masters" controls multiple "slaves" or "zombies" that can be used for anything the master of the botnet would like, for instance DDoS attacks. Mirai [20] from 2016 is a well-known botnet created for "Internet of Things" devices, and later used for DDoS attacks.

One of the techniques used in the Att&ck framework for command and control, "Commonly used ports" is based on "hiding in plain sight" by using standard ports for the command and control infrastructure. While some protocols, like HTTP and DNS, send traffic in plain text, making it easier to use a signature-based IDS, the usage of TLS on a standard port blends more into the environment. As stated by Gardiner, Cova and Nagaraja in [21], C2 channels have been found to not necessarily consist of a physical machine stored in a location, it may also consist of blog posts, forums and comments in HTML on some web servers. Also under the exfiltration technique of the Att&ck framework, there is a technique called "Exfiltration Over Command and Control Channel" which is seen in for instance Rokrat [22]. Rokrat also took advantage of "commonly used ports" in the Att&ck framework by using Twitter accounts for the command and control infrastructure.

Measures used to detect malicious network traffic is typically SSL/TLS decryption, reputation of IP addresses/domains, host-based IDS/IPS, network-based anomaly detection, network-based signature detection methods or preferably a combination of these.

2.4 Network Traffic Generation

As research sometimes depends on network traffic that is as close as possible to "real" network traffic, there exists multiple forms of network traffic generators. Because networks are complex and relies on several different protocols and architectures it does not exist an all-in-one solution that satisfies everyone's need.

Three different words that sometimes are interchanged in network traffic is simulation, emulation and generation. Network traffic *simulation* enables a simulation of a network, and it sends simulated data between the components in the simulated network. Network emulation allows for real, physical devices to be connected to a simulated network [23]. Network traffic generation is network traffic that is generated in order to test specific applications. A model of network simulation, network emulation and network generation can be seen in figure 3, figure 4 and figure 5.

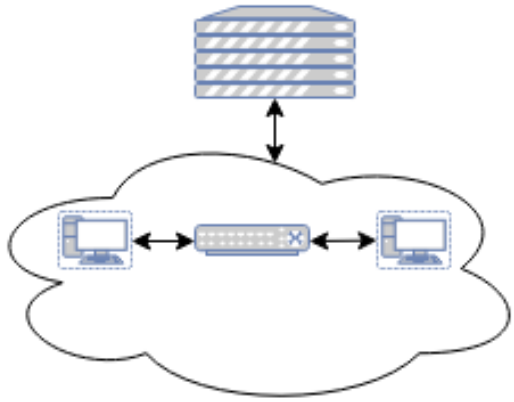

Figure 3: Network Traffic Simulation

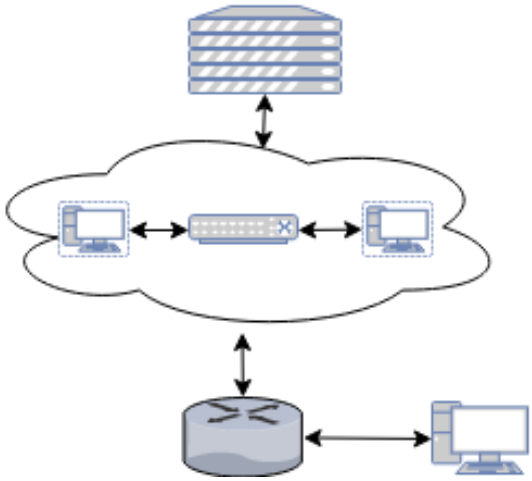

Figure 4: Network Traffic Emulation

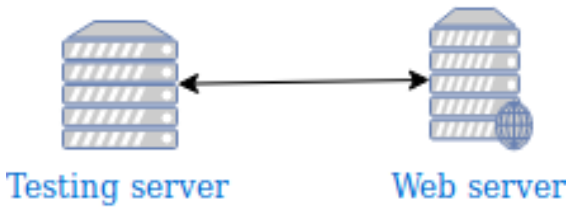

Figure 5: Network Traffic Generation

As stated earlier, network traffic generation is often done in order to test specific application. This may be everything from a web server (as in figure 5) to a an API. On the application side, many network traffic generators are done in order to test how for instance web servers handle large amount of network traffic. Others are made to test lower layers of the OSI model, such as that it tests network components' ability to handle large amounts of traffic going through it.

Software-based network generators have the advantages over hardware-based network generators that it is portable; it can be installed on multiple instances and in some cases also connect to other nodes to generate traffic [24]. However, a hardware-based network traffic generator often consists of specialized hardware

that perhaps are better in terms of amount of traffic it can generate, but more expensive due to the cost of hardware.

2.5 Machine Learning

As a subset of artificial intelligence, machine learning has a wide range of application [25], and is used for predicting outcomes given input. The word machine is defined by Merriam-Webster as "a mechanically, electrically, or electronically operated device for performing a task" [26]. Learning is defined as "knowledge or skill acquired by instruction or study" [27]. Put together, they form what is known as a popular used term for using computational power in order to see or detect a pattern, often implanted by various sorts of algorithms. Already in 1959, Arthur L. Samuel stated that

"*Enough work has been done by verify the fact that a computer can be programmed so that it will learn to play a better game of checkers than can be played by the person who wrote the program*" [28].

There are multiple types of machine learning that exist:

1. Supervised
2. Unsupervised
3. Semi-supervised
4. Reinforcement learning

When utilizing supervised learning, the algorithm is fed with attributes or features that describe a phenomenon, as well as a categorization or a class, also known as a label. After the algorithm has been fed with attributes and the labels, new data is given to the algorithm without labels. The machine learning algorithm will then attempt to classify the newly provided data into a class or a category that it saw in the training data.

Unsupervised learning may used when one do not know the categorization or class of the attributes. Instead, the machine learning algorithm is fed with attributes without a class, and told to classify the attributes into a specified number of classes. This essentially asks the algorithm to look after a pattern in the data, and classify the instances based on the patterns that is found.

Machine learning usually needs a lot of computing powers, and in some instances, it is not computationally worth it to use all of the features in the learning. Instead, only a fraction of the the attributes are used, in a way that compromises the classification of the machine learning with the computational power it needs to complete the task satisfying. More features used in machine learning usually means more complexity. By only selecting the most important features of a dataset to classify correctly, the complexity lowers while the correctly classified instances are still held up to a high standard. This is called feature selection, and is in essence the selection of a subset of the initial features that perhaps gives a slightly less accurate result, but still computes a good enough answer to the machine learning problem.

Artificial neural networks, or just neural networks, is a framework that attempts to mimic the way a human brain works. The topology of a neural network consists of an input layer (data), zero or more hidden layers, and output layer (classification). In each of the layers there are nodes. Each node has a connection to another node, and if all the nodes are connected to a node in the next layer, the neural network is considered a feed-forward neural network. Figure 6 visualizes a feed-forward network with 4 input nodes, 3 nodes in the hidden layer and 2 output nodes. The coloured circle represents bias.

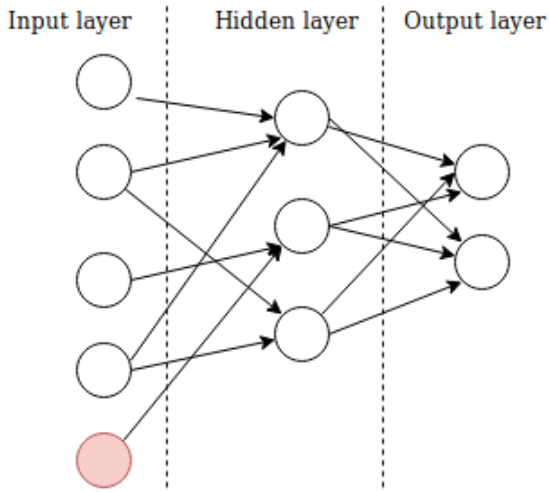

Figure 6: Simple Feed-Forward Neural Network

Biases are not connected to the previous layer, as the nodes are, and is added so that the network have better freedom to find the most optimal solution. Each of the links between nodes consists of weights, that are continuously changed during the training of a neural network to find the most appropriate weight.

One of the the neural networks that has been of great benefit is called multilayer perceptron [29]. It consists of more than zero hidden layers, and is considered being a feed-forward neural network.

3 Related Work

3.1 Analysis of Encrypted Network Traffic

Anderson and McGrew [30] did an experiment on passive analysis on encrypted traffic, using a Support Vector Machine to classify traffic as either malicious or benign. They used Joy [31] that parsed the data set they created into JSON formatted data. They classified on different compositions, and tried with for instance HTTP, DNS and TLS, as well as only TLS when they were looking for malicious activity in their machine learning. This research helped Cisco develop their ETA (Encrypted Traffic Analytics) [32]. One of the features used in Cisco's ETA for determining whether network traffic is malicious or benign is SPLT (Sequence of Packet Lengths and Times)[33]. The SPLT feature contains the size in bytes (length) of the payload in each packet of a flow, as well as what they call inter-arrival, or inter-packet time, which is the time between the sending of packets in a flow. This is done by checking the time between the first and the second packet, the second and the third packet and so on until all the interrarival times are calculated. They illustrate the practical meaning of SPLT in their comparison between a search on Google and the Betafera Malware, as shown in figure 7.

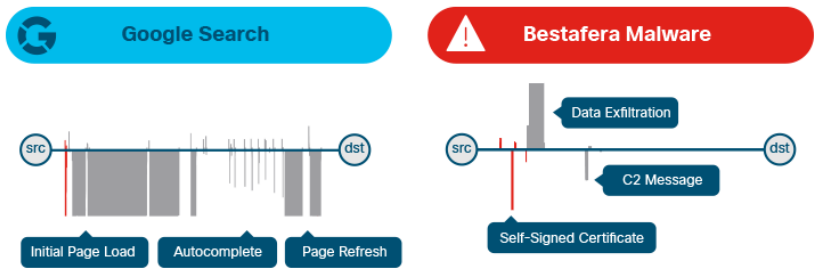

Figure 7: Comparison of a Google Search and Malware. [34]

In the figure, the horizontal lines that go above the vertical line represent bytes sent from the client to the server. The horizontal line that goes below the vertical

line illustrates bytes sent from the server to the client. The vertical line represents time. SPLT is especially interesting for detecting uncommon patterns or behaviour in network traffic, and it also, in theory, does not care about what type of encryption the network traffic is using.

Anderson and McGrew [35] used the the following features in a machine learning experiment where Joy [31] was used for feature extraction:

1. Netflow data
2. SPLT
3. Byte distribution
4. TLS information found in the initial handshake

The network-focused, versatile software package "Joy" is described as:

> *"A package for capturing and analyzing network flow data and intraflow data, for network research, forensics, and security monitoring."* [31]

As understood by the description, Joy has a wide range of application, and it is both able to run either live captures or analysis of previously captured network traffic in pcap-format. Joy outputs the information in JSON-format, which makes it easy to read or parse. The analysis of packet capture has the possibility to extract information regarding a wide range of features from a packet capture:

- Netflow
- SPLT
- Unencrypted features from the TLS handshake
- Byte distribution
- Entropy of data fields
- DNS information
- SSH information
- DHCP information
- HTTP information

The software package also includes another tool called Sletuh. Sleuth reads JSON formatted output generated by Joy and has the feature to do SQL-like statements

on packet captures, with clauses such as "SELECT", "WHERE", and "GROUPBY" for easy filtering of data.

Anderson and McGrew [30] discovered that in their data set, roughly 70% of the malicious traffic used self-signed certificates. Lets Encrypt [4] makes it easy for anyone to create legitimate certificates to a domain, and then signing them automatically by Lets Encrypt. Lets Encrypt works like a regular CA, but the service of signing the certificates by Lets Encrypt is provided for free, making it easier for good or worse to get a signed certificate from a legitimate CA.

Classification of malware spreading over Skype (and thus utilizing TLS) used among other things the byte frequency distribution[36]. As Anderson and McGrew also did in 2016 [35], the packet flows are stitched together to a bi-directional flow based on source address, destination address, source port, destination port and the type of transport layer protocol used in the traffic. Then, the Kullback-Leibler divergence is used. According to Korczyński et al.[36], the encrypted data in the TLS/SSL sessions seemed to be equally distributed. The features that were used to classify the traffic were the following:

- Frequency of bytes
- A hash based on the three first bytes in each packet
- A hashed value of the byte offset
- Reoccuring bytes in the first 4 bytes of a packet
- The hashed value of the four first packets' 16 bytes
- Four first directions of packets
- First packet in direction
- Distribution of packet size
- Packet size distribution based on direction
- Bytes that occur more than one time
- The first packet from the server

Their experiment resolved around capturing traffic of the Skype worm known as "Skipi"[37], and compared it to legitimate Skype usage.

In the experiment done by Bekerman et al. [38], they wanted to find unknown malware based on the network traffic produced by the malware. They gathered their dataset by using a sandbox with samples, collected samples from VirusTotal,

as well as generated benign network traffic in a lab over 10 days. They labeled their data using Snort and Suricata with updated rulesets, and the TLS specific features they looked at was server name, SSL/TLS version and the date the certificate expired. Other features was also used in their experiment, such as what ranking the domain had on Alexa, destination ports used, inter-arrival times and geolocation of the destination IP.

Velan et al. [39] generalized multiple protocols used for encryption of network traffic, such as IPsec, TLS and SSH into two stages. The first stage contained the initialization of encrypted network traffic, where unencrypted data were sent between client and server unencrypted in order to establish the encrypted session.This stage is divided into two parts, the handshake phase and the authentication phase. The second stage is entered after the client and server has established a common ground, and agreed upon an encryption scheme, and the data transferred is encrypted. Data that can be read from the first stage is split into two main categories; the connection and handshake, and the identifiers that is used in the earlier mentioned authentication phase. One point of interest for the paper are the possibility of fingerprinting clients based on ciphersuites offered in the first stage of their model. This is from the handshake phase. The X.509 certificate is mentioned from the authentication phase because of the possibility to discover whether the certificate is valid, regardless of what actually happened when the certificate was used. They specifically mentioned that Server Name Indication (SNI) may be derived from stage two, where the data that is sent is encrypted.

Fingerprinting of clients was further looked at by Husák et al. [40]. They took the fingerprinting a step further, by creating a dictionary of TLS fingerprints and matched it against user-agents. They used two different methods for creating the dictionary; the first one was host-based and relied on users to visit their website. If they did, the researchers were able to read the user agent in plaintext, and they also logged the ciphersuites that were offered in the Client Hello message in the TLS handshake. The second method they used looked into the source address of the network connections. If they had captured the ciphersuites of a HTTPS session, and also found a HTTP session from the source address within a certain time limit.

FingerPrinTLS [41] is a tool that is able to fingerprint packet captures or live traffic to specific browser versions. Currently, the newest fingerprint is from Firefox 52, which was released in March 2017. Sleuth, from the package Joy [31] also has the ability to fingerprint packet captures based on the ciphersuites. Although it contains a low number of fingerprints, the format of manually adding fingerprints is made easy by having the list of fingerprints in JSON. The newest fingerprint in Sleuth is Firefox 58, which was released in January 2018. The tool called Pytls [42] offers functionality that runs a webserver on an arbitrary port, and prints out all of the ciphersuites that is offered by the client that connects to it.

Proxies are in some cases used to inspect the traffic that moves in and out of the network. The proxy works like a man-in-the-middle attack. By terminating the connection before it leaves the internal network, and then opening up a new connection from the proxy to the destination, the inspection is possible. This is an active approach to TLS analysis, and enables traditional intrusion detection systems to inspect the payloads of the packets in order to determine if the traffic is malicious or benign. The open source Squid Project has a feature called "SslBump Peek and Splice" [43], which enables implementation of a proxy that is able to decrypt, inspect and reencrypt TLS traffic. However, this approach is more visible to the end user, as the certificate of the proxy has to be accepted by the user, unless the certificate is deployed on an endpoint during installation. O'Neill et al. [44] scanned for presence of TLS proxies on 2.8 million connections, and discovered that around 1/250 of the connections was behind TLS proxies, most likely to be used for TLS inspection. They also discovered that in the certificate of those that had a TLS proxy, the issuer name was set to the name of organizations such as Bitdefender.

Deepend research [45] holds a spreadsheet of of some traffic patterns seen in malware, HTTP method used, user agent, referring URL and headers. The list also contain links to the packet capture of some of the malware traffic seen, while the link to the executable is gone. The list was last updated in 2015. SSL Blacklist [46] is a site that monitors certificates that are related to malicious activity. They provide a CSV formatted list with labels of what the certificate is related

to doing, for instance command and control infrastructure. Hybrid-Analysis [47] allows for searching for specific ports used in samples submitted. Using the API, it is possible to craft even more fine-grained searches, such as searching only for traffic over port 443 that has been labeled as malicious by the sandbox. PacketTotal [48] allows for submission and download of packet captures from their public web site. PacketTotal utilizes Bro and Suricata under the hood, and it is possible to search for packet captures based on protocol, verdict of the network traffic, ports used and underlying protocols. For each packet capture that is analyzed, rules that are triggered are displayed, and a lot of information may be retrieved in the analysis. However, there does not exist an API for the site, which makes it cumbersome to download packet captures in bulk. Malware Traffic Analysis [49] is a blog that specializes in analysis of malware traffic. There is usually posted more than 15 entries a month, and the library contains both malware and packet captures ranging from 2013 to 2018, and it has over 1 500 blog posts in total. Usually a quick walk through of the malware is done in the blogposts, but the lack of tagging of the network traffic can make it difficult to find specific patterns in the traffic. The posts are grouped into names of the malware, so that one can easily find many samples of a particular malware if that is of interest.

3.2 Intrusion Detection Systems

Bro Network Security Monitor is an open source, scriptable network monitoring application capable of being implemented as both an IDS and/or an IPS [50]. It is included in included in the SecurityOnion distribution [51]. In his experiment with machine learning on HTTPS traffic, František Střasák used Bro in order to create features, in which later was used to classify traffic as either malicious or benign [52]. By replaying already captured malware traffic and benign traffic through Bro, he was able to use Bro's engine to extract features. Bro was also used in Holz et al. [53], in order to measure TLS certificates. In Lee et al. [54], Bro was used to create signatures to detect botnet activity. When the signature that was written matched, the following features where recorded to create a profile of the traffic:

- count of UDP packets
- count of HTTP packets
- count of SMTP packets
- count of unique IP addressess seen

Stratosphere IPS [55] is an open source network intrusion detection system, running on Linux, Windows and MacOS. It utilizes machine learning to detect malicious traffic, and relies on another program for the collection of network traffic. Straosphere utilizes free and public data sets for the training of their machine learning algorithm in what they call behavioral models. This is a set features extracted from the network traffic that describes the behaviour of connections. It can for instance extract information whether the network traffic is sent in intervals, such that one may see in beacon traffic from command and control infrastructure.

Cisco has a commercial product called Encrypted Threat Analysis (ETA) [32]. This product transfers a copy of the network traffic that is seen in a network to Cisco, where machine learning is applied to classify the traffic. The results are then sent back, so that one may receive an alert if the machine learning classifier from Cisco has labeled a flow malicious.

Snort [56], maintained by Cisco, is an open source IDS that has the capability to log TLS traffic by using what they call "The SSL Dynamic Preprocessor (SSLPP)". This feature enables deployment of Snort and then specifies how far into the TLS handshake that traffic is supposed to be logged. However, the FAQ of SSLPP states that "Encrypted traffic should be ignored by Snort for both performance reasons and to reduce false positives." [57].

3.3 Network Traffic Generation

Garcia and Rigaki [58] used generative adversarial networks (GANs) to create bengin network traffic that is supposed to behave like real-world network traffic by using reccurrent neural network. They modified the open source version of a remote access Trojan called "flu" to generate the malicious traffic, in this case over HTTP rather than HTTPS. However, the network traffic they generated in this paper was unencrypted HTTP traffic.

For some of the encrypted network traffic used in [52], three days were used to surf Alexa top 1000 websites to generate network traffic to the experiment. That meant that accounts were created in order to use services such as Facebook, Google and Twitter. The datasets created, CTU-20 to CTU-32 are available

at [59], [60], [61], [62], [63], [64], [65], [66], [67], [68], [69], [70] and [71]. Anderson and McGrew [72] captured the network traffic of an enterprise over 12 months in order to use the traffic as benign network traffic in their experiments of classifying network traffic as malicious or benign. Later they ran their assumed benign enterprise network traffic dataset through an up-to-date IP-based blacklist maintained by Cisco Talos, in order to remove malicious network flow from their dataset, as they assumed their dataset contained malicious network traffic.

In UniLoG (Unified Load Generation Tool) [73], a prototype of a network generation was made based on a custom, minimal version of Firefox with customized header fields, "user agent" and "accept". In order to test whether the generation of traffic resembled a real browser, the performance was measured against usage of a real browser. Later in the project, a new prototype based on on the WinInet API was created. This prototype had the possibility to execute modern languages used on web sites such as JavaScript and CSS. Later, a "pool of websites" was used in the testing. This consisted of the Alexa top 1 million. The network traffic generator that was built also supported TLS.

Pupy [74] is a RAT that can be used to generate malicious traffic, as it contains capabilities of exfiltration of data. It runs entirely in memory and by being modular, it is up to the user of Pupy to specify which protocols to use for the exfiltration of data. Pupy already has a module for exfiltration of data over TLS. Iperf [75] is software that can be used to generate network in a client/server architecture. It is created both for testing bandwitdth and packet loss, but may also be applied in order to create network traffic at the transport layer and below, which means no TLS support.

For packet-level network traffic generators, Scapy [76] is widely used for crafting, sending and receiving packets. It provides an easy to use framework for creating all kinds of network packets. It currently has support for TLS up to 1.2. Trafgen is a part of the netsniff-ng [77] toolsuite. It reads definitions of what packets to be sent from a text file, and then sends them out on the network.

For generating network traffic without interaction with real websites, virtualization of software using hypervisors in tier 1 or tier 2 enables users to deploy

virtual machines that is interconnected through virtual network cards, and thus can be used to generate network traffic between them, running arbitrary software for the generation. Docker [78] has the ability to easily deploy and use software without much configuration, and may set up a virtual network between its containers. Nflow-generator [79] creates dummy data and sends it, where the primary purpose is to test NetFlow log collection. It has the capability to send HTTPS, SSH and IMAPS in a docker container, which again generates network traffic. Mininet [80] allows users to easily deploy virtual networks and virtual hosts in a software-defined networking environment. It has the capability to run real applications on the virtual hosts, enabling network traffic to be sent back and forth. Containernet [81][82] is software that "dockerizes" Mininet, and easily deploys several nodes that can communicate with each other.

TRex is a network traffic generator that has the ability to create network traffic from layer 4 (transport layer) to layer 7 (application layer) [83]. It is able to use previously recorded packet captures and use the content (payload) of these as a template for the traffic generation. Further, it allows for customizing of round trip time, time between packets sent and duration of the generation. TRex then sends the packets on its virtual network card to another network card, rewriting the source and destination address and applying the configuration earlier set. TRex supports HTTPS, by using a HTTPS packet capture as a template.

Warp17 is a network traffic generator that "currently focuses on L5-L7 application traffic..." [84], but has the ability to generate traffic from layer 1 (physical layer) to layer 7. A virtual network has to be built using command line arguments, where there has to be specified IP ranges to be used, gateways, mulitcast and ports to be used. A wide variety of settings may applied to the generator such as latency, and it even has a built-in webserver that can be configured to respond to HTTP traffic. Warp17 currently do not support HTTPS.

Erlacher and Dressler [85] made a network traffic generator they named GENESIDS, based on Snort. GENESIDS is able to generate malicious traffic based on Snort rules, so that one may use a Snort rule to generate malicious network traffic, or Snort syntax to define malicious network traffic. After a rule is selected or written, GENESIDS uses TRex to generate the network traffic, and it may then

send the network traffic to an IDS to see if the IDS detects the malicious network traffic.

3.4 Machine Learning

Anderson and McGrew [30] compared the usage of 1-logistic regression and SVM with Gaussian kernel in their work of classifying encrypted data, but no significant differences were found, apart from the fact that the training of the SVM was more costly in terms of computational power. The Server Name Indication (SNI), an optional feature of TLS, was used to gather information about the domain name and IP address when it was in use. SNI was however only present in roughly 27% of the malicious traffic. The metadata in ClientKeyExchange and clientHello in TLS was used to determine specific libraries used in TLS implementation. In their data set, they discovered that the "TLS_RSA_WITH_RC4_128_MD5" was the most offered by the malicious traffic, while "TLS_RSA_WITH_AES_128_CBC_SHA" was the most used in the benign traffic. Also discovered was the use of keys: the malicious traffic seemed to prefer 2048-bits RSA public keys. By putting the user-agent field together in the context of which library is in use, as well as features used from TLS on the server side. It consisted of which type of ciphersuites that was selected, which extensions that were supported, how long the validity of the certificate was, number of the domain names registered to the certificate subject alternate name, and if it was a self-signed certificate or not. The DNS data used in the experiment checked whether the domain was in one of the top 100/1000/100 00/100 000 in the Alexa list, which indicates visitors to the domain. Length of domain name used, along with the fully qualified domain name (FQDN) was also taken into consideration. From HTTP content fields, they used among other fields the content-type field and the user-agent field.

Inaccurate ground truth of TLS encrypted network traffic was looked into by McGrew and Anderson in [72]. They switched classes of a percentage in their dataset by between 1.5% and 5% because they presumed they had malware traffic that was labeled benign, and benign traffic labeled as malicious. After the labels were switched, they used different machine learning algorithms in order to see what impact it had on the results, and compared the results with same algorithms without relabed traffic. Their results indicated that both linear regres-

sion and multilayer perceptron had somewhat similar results given introduced noise, against the dataset without relabeling. SVM had the worst performance. The experiment they did also contained different features. One dataset had very few features, while another had a high number. The dataset with the highest number of features performed best when introducing malicous traffic to the benign class and vice versa.

An interesting dataset on botnet activity is created by Garcia et al. [86], however without encrypted traffic. The data set is labeled, hence it may be used on both supervised or unsupervised machine learning algorithms.

A survey of methods used for classification of encrypted network traffic has been created by Drasar et al. [39]. Different approaches on supervised machine learning is presented, such as using naive bayesian classification and support vector machine (SVM). In their work, they also compare different open-source tools for the classifications of encrypted network protocols.

Shalaginov et. al [87] looked at DNS logs and developed a methodology for beaconing and detection of targeted attacks. Because the malware may beacon in irregular intervals, which means a threshold of a session time will have to be set in order to detect the beacon. Erquiaga et. al [88] took a look at what mechanisms the malware used when it was refused Internet connection. This is relevant in terms of failure functions the malware has, and the backup solution they discovered.

The Stratosphere IPS project [55] is a solution that leverages both a testing framework for machine learning [89], as well as utilizing this framework for an open source, network-based intrustion prevention system. The project has a sister project called Malware Capture Facility Project [90] which is both analyzing botnet behaviour as well as sharing the dataset. However, the dataset currently does not contain any large amounts of TLS traffic.

4 Methodology

A literature review had to be done before the experimental methodology was created. At first, the literature review was done with focus on getting an overview of what had previously been done in the field of encrypted network traffic analysis, as well as diving into the TLS protocol. Features that others had used was examined. This was supplied with a lot of practical testing of different software and solution, and getting an overview of the methodology that others had been using for network traffic generation of benign network traffic. Later, the focus shifted to examining TLS and malware traffic. Here, the search for already existing packet captures started. At the end, the methodology that was created is presented in the 4.1.

4.1 Experimental Design

The experimental design of the thesis is shown in figure 8. First, the network traffic generation is done, and explained in detail in 4.2. After this is done, malicious traffic is gathered, primarily from Hybrid Analysis. Feature extraction and feature selection is done in 4.4. A multilayer perceptron with backpropagation is then applied, and the results can be seen in chapter 5.

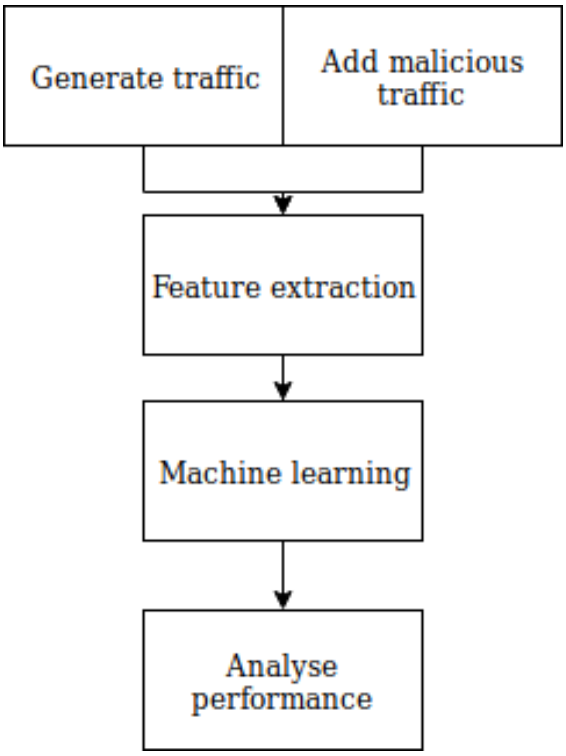

Figure 8: Flow of Experiment

4.2 Generation of Benign Traffic

In order to produce encrypted network traffic, several tools and methods have been assessed. None of the existing tools are able to produce encrypted network traffic that suits the needs of this thesis of real network traffic, hence a tool for creating such traffic was made. The tool consists of two modules, where each of the modules has its own purpose of generating a specific type of traffic. Figure 9 displays the generic flow in each of the network traffic generations done in this thesis.

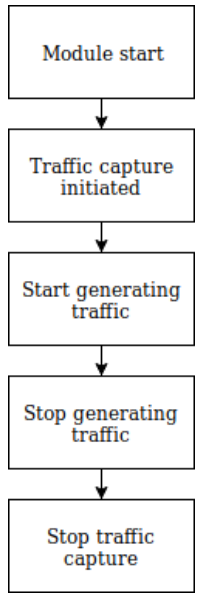

Figure 9: General Model

Tshark, a command line interface to Wireshark, is used for traffic capturing [91].

4.2.1 General HTTPS Traffic Generation

PhantomJS [92] and cURL [93] allows a headless browser to visit web sites. PhantomJS reads user defined JavaScript (server side), and is most commonly known for being a tool for testing of website interactions. cURL [93] however, is a more simple tool, which is used to visit websites and retrieve its content, and is among other things used for API interactions. cURL does not run JavaScript on websites, so the browsing is done both with PhantomJS (in order to execute JavaScript) and cURL. This allows generation of network traffic that is both based on simply fetching the site and the properties of the network traffic, as well as actually having a tool execute JavaScript and thus creating more realistic network traffic.

By using these tools, the experiment is using network traffic that may not actually behave like a person is browsing. On the other hand, it also enables the experiment to create network traffic that is somewhat close to malware, except

that the traffic is benign, and the generator was visiting renowned websites.

Figure 10 shows the flow of the network traffic generator for HTTPS browsing. A text file containing URLs are provided to the program, which must have one URL on each line without http:// or https:// prefix in front of it. The user agent is set to "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59.0) Gecko/20100101 Firefox/59.0" in order to not be neglected by web servers, and treated as a regular desktop browser. The generator visit the first link in the file containing URLs. If the link it visits do not have any links to other websites, it will move on to the next URL in the text file and visit it. If there are URLs, the program selects a random number between 1 and the number of links found on the website, and the number it generates will represent the selected URL. The program then generates a random number, either 1 or 0. If the number generated is 1, it will visit the previously selected link. After this is done, the generator moves to the next URL in the text file.

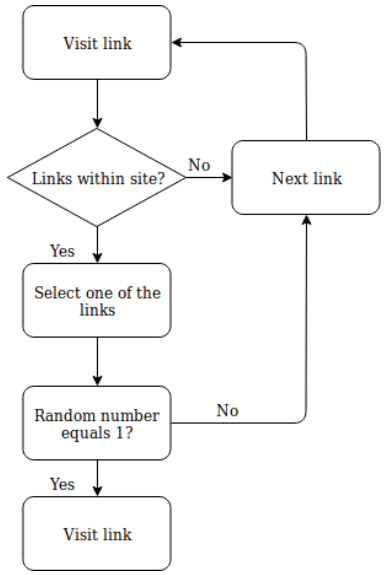

Figure 10: Flow of Generating HTTPS Traffic

While Alexa top 1 million [94] is a long list of links, not all of the websites may contain trustworthy content. To overcome this, a list of 60 URLSs of trustworthy sites are generated using expert knowledge. The content of the websites vary from newspapers to ordering pizza, and is a mix of Norwegian and English written websites. By visiting the websites and browsing them before using them in the experiment, they were confirmed to be running HTTPS and also reachable. The sites selected ranged from second on Alexa top 1 million, to 476 734th place. This ensures that both the initial web sites that the generator visits are benign, but also, in theory, ensures that all of the links on each of the sites consists of trustworthy content. The full list of URLs used can be found in appendix A.3.

Both the cURL and the PhantomJS experiment is run on a Raspberry Pi 2b [95], running on Raspbian Stretch 9.4, producing a pcap-file each.

4.2.2 HTTPS browsing

Since cURL and PhantomJS does not interact much with the websites while visiting them (other than establishing a connection and executing JavaScript), the Selenium IDE for Firefox was used on a subset of a small amount of website in order to simulate a real user. Selenium is a free and open source interaction testing framework for websites, and uses browsers such as Firefox or Chrome as the browser it runs website tests on. The Selenium IDE allows for recording of test, which is done for each of the websites visited. The tests of each website was set to last 2 minutes, and the following web sites was selected and tested using Selenium IDE:

- vg.no
- duckduckgo.com
- norwegian.no
- ebay.com
- twitch.tv
- mail.com

vg.no is a Norwegian newspaper that was used to browse the latest news. Duckduckgo is a search engine that was used to search for terms, and was also used to view pictures. Norwegian is an airline company, and was used to browse holiday destinations. Ebay is a site to sell and buy (almost) everything, and was used to browse different articles. Twitch is a streaming website for gamers that was used

to produce network traffic of streaming. Mail.com is an Internet e-mail provider, and was used to log into an e-mail account, reading the email and sending a mail.

By recording web-browsing on these websites, the traffic generated looks more like a user using the Internet, because there actually is a user behind the browsing. The recorded sessions were then replayed in order to capture the network traffic they all contain.

Since the replay functionality of Selenium IDE on live, ever-changing websites, it is difficult to replay the scripts without failure. Because of this, the procedure to capture the traffic is done at the same time as the recording of the browsing is done. The script that is generated by Seleniuem IDE is then saved, so that others may see what was done when the test was recorded, and possibly replay the tests so that they may generate somewhat similar traffic. These tests may be viewed at [96].

The network traffic that is generated is mostly TLS 1.2, but because the browser running the experiment supports 1.3 as well, some of the traffic is version 1.3. The packet capture generated produced by visiting eBay contained 281 packets with TLS v1.3, and 9271 packets using TLS v1.2. The rest of the packet captures only use TLS v1.2.

4.3 Malicious Traffic

In order to collect malicious TLS traffic, the following method was used:

1. Locate hash of malware that use TLS
2. Submit the hash to hybrid-analysis.com to see if exists there
3. Retrieve PCAP from hybrid-analysis

In order to locate malicious traffic, Hybrid-Analysis' API is used. The following parameters was used:

- verdict=5

- port=443

When the verdict parameter is set to 5, it means that the files that is searched for is labeled as "malicious" in Hybrid-Analysis' database. Below is a list of verdicts that can be specified when searching for malware on Hybrid-Analysis:

- 1. Available
- 2. Whitelisted
- 3. No verdict
- 4. Suspicious
- 5. Malicious

Port 443 specifies the port that the malware must communicate to, and usually would imply that the traffic that is used is HTTPS. The search used can be found in appendix A.1

The search returns a JSON formatted list that is saved to disk, with 196 unique entries at the time the search was conducted. However, of the 196 entries, 157 are URLs. Some of these are among other things phishing sites, malicious redirects and browser hijacking, and they were excluded. The rest of the types ("type_short" in the returned JSON) that is used is found in table 3.

Table 3: Number of Different Files and Classes

Type	#
64-bits exe	3
doc	1
exe	19
.NET exe	3

In appendix A.2, the verdict, sha256 and the type of each hash is found.

In order to make sure that the packet captures of these files actually contain TLS traffic, they are inspected for TLS traffic. Packet captures were downloaded using a script that read the SHA256 sum from a file, and then downloaded from Hybrid-Analysis. Because of restraints on the API, this had to be done in batches of 5 downloads each hour.

Since the amount of network traffic found in some of these packet captures, the largest packet captures are removed from the experiment to even the amount between malicious and benign traffic.

4.4 Feature Extraction and Selection

Feature extraction is done using "Joy" and the accompanying "model.py" Python program [31]. Joy is used with these options:

- bidir=1 (Bi-directional)
- tls=1
- type=1 (SPLT)

The feature extraction ("model.py") takes two directories as input; one directory with malicious traffic in PCAP or PCAPNG format, and one directory with benign traffic in PCAP or PCAPNG format. The features are extracted, and written to a CSV-file where the last row is the label indicating if the traffic is malicious or benign. A quick summary of the features are found in table 4.

Table 4: Features Extracted

Option/feature	**# of features**	**Information**
–ssl	187	Offered cipher suites, extensions
–meta	7	Ports, addresses, packets in/out, bytes in/out
–lengths	100	Size of packets
–times	100	Time between packets
Total	**394**	

394 features is a quite large number considering the network traffic that is generated in this experiment. Instead of using all of the features, two separate experiments are conducted: One with only the TLS features, and one with the SPLT feature (–length and –times).

The malicious network traffic that collected from Hybrid Analysis most probably contained some benign traffic, as the server name extension in some of the traffic was enabled and had server names such as facebook.com, google.com and bing.com. As these instances are put in the malicious category, some malicious network traffic (without a SNI indicating that it is benign) is labeled as benign

network traffic. Approximately 1% (27 flows) of the network traffic that is fed to the machine learning algorithm as benign traffic, is actually malicious.
This was previously done in [72] to counter the inaccurate ground truth they had, because they knew they had malware traffic classified as benign traffic, and vice versa. In that experiment, they purposely switched labels on between 1.5% and 5% of their dataset.

Since the amount of malicious flows of all the hashes found appendix A.2 is much higher than in the benign dataset, the dataset is reduced. Since the SPLT dataset consists of 3335 benign flows, 6670 malicious flows (the double of benign) is randomly selected from all of the malicious flows. These are used in the SPLT experiment.For the TLS experiment, there was 2723 benign flows. The 6670 malicious flows selected in the SPLT dataset are trimmed down to 5446 flows so that the malicious traffic in the TLS experiment is double the size of the benign traffic.

4.5 Machine Learning

The machine learning software Weka [97] is used with the "MultillayerPerceptron" function with 10-fold cross validation. The following options are set:

- Epochs = 500
- Momentum = 0.2
- Learning rate = 0.3
- Batch size = 100
- Hidden layers = 1
- Input layer nodes = 100
- Hidden layer nodes = 51
- Output layer nodes = 2
- Number of decimals taken into consideration = 2
- Seed = 0
- Cross validation = 10

5 Results

5.1 Experiment with SPLT features

A total of 200 features was used in this experiment, all related to packet length and times. The dataset consisted of 3335 attributes that was labeled as benign, and 6670 samples that was labeled malicious. Options –lengths and –times are given to model.py to extract the SPLT. Table 5 shows the confusion matrix for this experiment.

Table 5: Confusion Matrix of SPLT Experiment

		Given label	
		Benign	Malicious
Predicted label	Benign	2186	1149
	Malicious	499	6171

Figure 6 shows details about the benign class from the machine learning. Figure 7 shows information about the class of malicious traffic.

Table 6: Benign SPLT Traffic Details

Benign traffic				
TP rate	**FP rate**	**Precision**	**Recall**	**ROC area**
0.655	0.075	0.814	0.655	0.873

Table 7: Malicious SPLT Traffic Details

Malicious traffic				
TP rate	**FP rate**	**Precision**	**Recall**	**ROC area**
0.925	0.345	0.843	0.925	0.873

The number of correctly classified instances is 8357, while the incorrectly classified instances is 1648, as seen in 7. This gives a percentage of 83.5282% correctly classified and 16.4718% incorrectly classified.

5.2 Experiment with TLS features

The TLS features extracted is a comprehensive list of possible ciphersuites offered, as well as extensions used in the network traffic. As cURL offers a number between 20 and 30 different ciphersuites when connecting to a server, and Firefox offers approximately the same amount, feature selection is done to reduce the amount of ciphersuites. PCA with Ranker is used to find the 100 best features for this experiment. A total of 187 features was trimmed down to 100 and used in this experiment, all related to the ciphersuites in the TLS handshake. The dataset consisted of 2723 attributes that was labeled as benign, and 5446 samples that was labeled malicious. Table 8 shows the confusion matrix for this experiment.

Table 8: Confusion Matrix of TLS Experiment

		Given label	
		Benign	Malicious
Predicted label	Benign	1721	1002
	Malicious	583	4863

Figure 9 shows details about the benign class from the machine learning. Figure 10 shows information about the class of malicious traffic.

Table 9: Benign TLS Traffic Details

Benign traffic				
TP rate	**FP rate**	**Precision**	**Recall**	**ROC area**
0.632	0.107	0.747	0.632	0.818

Table 10: Malicious Traffic Details

Malicious traffic				
TP rate	**FP rate**	**Precision**	**Recall**	**ROC area**
0.893	0.368	0.829	0.893	0.818

The number of correctly classified instances is 6584, while the incorrectly classified instances is 1585. This gives a percentage of 80.5974% correctly classified and 19.4026% incorrectly classified.

6 Discussion

The thesis has produced a methodology for generating benign network traffic, as well as utilizing it along with malicious network traffic to classify the network traffic as either benign or malicious. This was achieved doing two experiments, one that looked at the ciphersuites used, and one that looked at the behaviour of the packets with timing and packet sizes. By gathering a dataset as close as possible to real network traffic ensures that the experiment is more applicable than if the network traffic had been entirely simulated. However, the results showed that the machine learning classifier only managed to correctly classify 80% and 83% correctly. While it shows that the algorithm was able to see some patterns, the dataset may have contained malware that for instance used up to date ciphersuites, or malware that managed to blend in properly with the benign network traffic. As the ground truth of the dataset is not 100% accurate, the accuracy of the classifier may be slightly skewed upwards or downwards in reality. The problem with the inaccuracy of the dataset is of course that we cannot entirely trust the classification accuracy. The inaccurate ground truth was somewhat smoothed out by adding malicious network traffic and labelling it as benign.

The features used for the classification in this thesis was ciphersuites offered in the TLS handshake and the sequential packet lengths and times of the network traffic. Considering the complexity of the features, the dataset generated was not large enough to use all of these features together, instead two separate experiments were conducted and compared. Traditional methods, such as IP reputation and domain reputation is still relevant features in terms of detecting malicious encrypted network traffic, but the features used in this thesis focused solely on what could be derived directly from the network traffic capture without using external sources. This enables the possibility to detect security incidents that would have gone unnoticed based solely on reputation.

The ciphersuite features that was used could be used in a real time environment, even denying traffic that use specific ciphersuites. The analysis of SPLT however,

needs to be done sometime after the session is established so that enough information is gathered before the classification is done. As a larger dataset would allow for more features being used at the same time, both the ciphersuites, the extensions offered and the SPLT could be used together, which perhaps could lead to better accuracy when it comes to classification of the network traffic.

In order to generate a realistic dataset, real services have been used in this thesis. This enables the context that many of the existing network traffic generators are missing today, as they produce dummy data and traffic that does not aim to mimic specific usage of websites. Three different tools were used, cURL, PhantomJS and Selenium. cURL was used to simply generate the handshake and connect to websites for the certificates, PhantomJS was used to execute JavaScript on the websites, and finally Selenium were used in order to mimic different types of user behaviour on websites. By selecting broad categories when using Selenium, the traffic may also be generalized, i.e. the streaming on Twitch may also look like streaming on YouTube, Netflix and other websites. The ground truth of a dataset is important, and was taken into consideration when the traffic generator was made, hence the list of URLs was used.

The Selenium part of the methodology depends on users actually browsing the Internet, so that others may reproduce it by replaying the generated script. Even though the manual job of browsing is time consuming, the fact that the behaviour that a user does in the browser is transcribed into a script is big benefit. The script that is generated reveals what the user did in the browser, both typing on the keyboard and also clicking of the mouse. Unfortunately, it does not record time between pushed buttons, as would have been a great thing to have documented in cases where auto completion is proposed by a website.

In order for a regular, signature-based IDS to detect malicious network traffic, there exists a number of properties that it may look for. The destination IP from a network flow may be checked against a list of IPs that have been previously seen used in attacks. The same goes for domains, they may be checked against a list of bad reputation domains. As of now, the server name indication (SNI) extension of TLS 1.2 is unencrypted, and may therefore be viewed in plaintext by an IDS. However, SNI is moving towards being encrypted. There is an internet-draft [98]

for encryption of the SNI in TLS 1.3. CloudFlare [99] has already activated it in their networks, and Mozilla Firefox Nightly also has support for this feature [100]. The initial ciphersuites proposed by the client in the TLS handshake can also be utilized in an IDS. By first getting an understanding of what kinds of ciphersuites are used in the network the IDS is deployed in, rules may be set up to alert on outdated or suspicious cipher suites, or combinations of suggestions to ciphersuites that is unlikely to be used in legitimate traffic. In a real world setup, the intrusion detection systems are not only limited to the TLS traffic, but it also able to look at a combination of different protocols to gain a larger picture of what is happening.

Other features that have been seen used is the byte distribution of the content in each packet sent in a flow. This does however add an even higher complexity for the machine learning algorithm, as the proposed method of using it consists of 256 features. Each of the 256 features represent the count of bytes seen in the flow. Another feature, the initial data packet (also known as IDP) consists of all the bytes seen in the first data packet sent in a flow. These were not used in the experiments of this thesis due to the high complexity, which potentially could overfit.

6.1 Problems and Challenges

Several different approaches for the generation of network traffic was investigated and later abandoned. Software-defined networking was in the early phases looked into, especially Mininet [80]. Mininet builds on OpenFlow, and allows a fast and easy way of deploying new networks, with switches and hosts. The hosts can then be used to interact with one another, and it is possible to deploy a web server on one host, and use another host to connect to it - thus creating TLS traffic if HTTPS is implemented. The early phase looked into deploying INetSim [101] one a host, but was later abandoned due to the fact that it would lack the realistic touch one would get by using a real network, instead of running it in a virtual network. Docker was looked into at the same time as Mininet was looked into, and allows deployment of microservices/containers on a host, which can interact with other containers. Docker was also abandoned because it lacked the realistic context one would get by connecting to a real service to generate network traffic.

In terms of using an already established network traffic generator, this was also investigated during the thesis. However, all of the existing, open source tools that were located lacked either the capability to generate data higher than the transport layer or the ability to generate data that was not purely based on statistics. The closest generator found was TRex, which seems really promising for generating network traffic in a much larger scale.

Manually connecting to a web server was also investigated, where software like Scapy [76], Mausezahn of netsniff-ng [77] was investigated. These software applications give complete control of what kind of packets to send, but also adds complexity that is unnecessary, like manually performing the TCP handshake.

Collection of malware samples and/or malicious network traffic utilizing TLS has been thoroughly investigated. VirustTotal, hybrid-analysis, packetshare, malware-traffic-analysis, virusshare and packettotal was checked. The hashes from Freddi Barr-Smith's master thesis [102] were also looked into where the hashes had been categorized as using HTTPS, although lacking the information regarding which ports were used. Malware-traffic-analysis [49] provides packet captures and analysis of malware. Since there are a lot of packet captures there without proper labeling of the network features, these are not implemented in the master thesis.

Packettotal has a nice interface, but has however not yet implemented an API at the time of writing this thesis. Another problem with Packettotal is that the packet captures that are downloaded does not have any information regarding what malware produced the specific network traffic, other than the Snort and Suricata rules that potenitally was triggered.

VirusTotal [103] is a website that offers both malware and packet captures that is submitted by everyone. This site has an API that enables both search and mass-download of files. However, the ability to search for packet captures are almost non-existent. VirusTotal offers students free access to their paid services, and they are contacted with questions about a more fine-grained search on their website, along with an inquiry on retrieving a students license. The students license was granted, but there is no response whether they have a better API on retrieving

packet capture files. They do not hold information on whether their malware include encrypted network traffic, which means the job of discovering such activities will have to be done by running the malware, and capturing the network traffic the malware perhaps is sending. Because of this, VirusTotal is not looked into further.

NTNU Gjøvik holds a large malware repository. The owner of this repository was contacted, and it is discovered that they do not hold information of network behaviour. A large set of samples is retrieved from this repository, but none are used because of the lack of information regarding usage of network communication.

6.2 Limitations

One weakness of the reproducability of the configuration file in Selenium is that the test that is running must be tested against the exact same website that it was initially tested on. In several test cases done, the test failed only minutes after the recording of the session. This led to recording of the packet capture, as well as the Selenium test, was done at the same time. The recorded scripts for Selenium can be found at [96], however they are likely to fail as Selenium has does not record the waiting time between user clicks and behaviour. Instead, Selenium does everything as fast as possible, and thus removes the "human touch" of browsing when replaying the test. A sleep functionality may be added, which will slow down the tests, and also makes it configurable. By making the Selenium scripts available, the packet capture that was done during this master thesis may, in some way, be reverse-engineered, as the recipe for the packet capture lies in the Selenium script.

Only one browser was used to browse the Internet when generating network traffic. This results in some features being present in all of the browsing, such as ciphersuites offered by the browser. An improved way - obviously more time-consuming way would be to generate network traffic using several different versions of browsers, such as Google Chrome, Opera, Internet Explorer and Safari. Also, different releases of the browsers mentioned could be used, as they may produce different proposed ciphersuites in the TLS handshake. Due to to time

constraints, only one browser was selected. However, the SPLT features that was extracted from the manual browsing do not care what kind of browser or version of TLS that is in use, it only cares for size and timing of the packets. The TLS features would definately benefit from more diversity, as a broader range of ciphersuites investigated.

7 Conclusion

The thesis has verified that there is possible of using information from the TLS handshake and the behaviour of the network flow in a machine learning algorithm to distinguish malicious network traffic from benign. Although the results showed less accuracy than done by others, the process of generating a dataset consisting of accurate ground truth is done by visiting what is presumed to be legitimate websites, and thus producing benign TLS traffic. The methodology used for collecting the traffic also shows that it is worth thinking outside of the box when looking for software to for a specific purpose. Selenium was used for creating a recipe of how a user behaved on websites, and the automatically generated script that Selenium produced could be read by others to get an understanding of what was done on a specific website in each of the testing scenarios.

The thesis has investigated different tools and techniques for network traffic generation to be used in the experiment. When none of the existing tools suited what was felt needed for the thesis, a proof-of-concept network traffic generator was made, consisting of open source software. Malicious network traffic using TLS was found in open sources, where Hybrid Analysis provided the best way of obtaining the packet captures through their API.

The accuracy of the results, respectively 80% and 83% showed that there is still room for improvement in the classification. It also showed that the behavioural features (SPLT) was more decisive than the experiment with the ciphersuites.

7.1 Future Work

The experiments showed that there is possible to passively determine network traffic as either benign or malicious, but at the same time, a correct classification rate of between 80% and 83% is not good enough to be put into a production environment. As the experiment was divided into two separate experiments, it would be natural to combine the features in the two experiments into one experiment utilizing both SPLT, ciphersuites and extensions used. Because the dataset of benign network traffic contained a low number of flows compared to the amount of features that may be used, it would be a good idea to start with generating more benign network traffic so that the experiment can be done in larger scale. TRex, as mentioned earlier in the thesis, could also be of great benefit for future work in this research area. As TRex replays packet capture and may modify timestamps, it could be of great help to generate a large amount of data from a small amount of packet captures.

The inaccurate ground truth found in the malicious network traffic could also be further looked into. This experiment "polluted" the benign class with a small amount of malicious traffic, but further steps to ensure that the malicious network traffic is in fact malicious may be taken. One possible way to remove wrongly labeled malicious traffic is to run the dataset through an IP blacklist and a domain reputation list, so that traffic that is isn't found in the blacklist/domain reputation list could be removed. Another way of better controlling the ground truth of the malicious network traffic, is by using software such as GENESIDS. As it has the possibility to generate malicious network traffic based on Snort rules, it could be really useful it encapsulated the network traffic with TLS for encrypted traffic analysis.

More additions to the benign network traffic generator could also be further looked into. As online chatting services, such as XMPP, may use TLS (and thus be used in a C2 setting), generating such benign network traffic could be advantageous. Also, scenarios such as VoIP could be further looked into and possibly implemented. The Selenium tests written in the thesis could also benefit from being further enhanced, so that they are more robust and possibly work better in scenarios for replaying the traffic. Another thing to further investigate is to aggregate data from other protocols that is commonly used in a network connection,

such as DNS.

Bibliography

[1] 2017. Transparency report - https encryption on the web. https://transparencyreport.google.com/https/overview. Online; Accessed 25-March-2018.

[2] Helme, S. 2018. Alexa Top 1 Million Analysis - February 2018. Online; Accessed 25-March-2018. URL: https://scotthelme.co.uk/alexa-top-1-million-analysis-february-2018/.

[3] Desai, D. 2018. February 2018 Zscaler SSL Threat Report. Online; Accessed 25-March-2018. URL: https://www.zscaler.com/blogs/research/february-2018-zscaler-ssl-threat-report.

[4] 2017. Free ssl/tls certificates. https://letsencrypt.org/. Online; Accessed 01-December-2017.

[5] PayPal Phishing Certificates Far More Prevalent Than Previously Thought. https://www.thesslstore.com/blog/lets-encrypt-phishing/. Online; Accessed: 2018-09-22.

[6] Oppliger, R. 2009. *SSL and TLS: Theory and Practice*. Artech House, Inc., Norwood, MA, USA.

[7] Hickman, K. E. The SSL Protocol. Expired Internet-Draft, April 1995. URL: https://tools.ietf.org/html/draft-hickman-netscape-ssl-00.

[8] Thomas, S. A. 2000. *SSL TLS Essentials: Securing the Web*. Wiley.

[9] Rescorla, E. The Transport Layer Security (TLS) Protocol Version 1.3. Proposed standard, August 2018. URL: https://tools.ietf.org/html/rfc8446.

[10] T. Dierks, E. R. The Transport Layer Security (TLS) Protocol Version 1.2. Proposed standard, August 2008. URL: https://tools.ietf.org/html/rfc5246.

[11] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., & Polk, W. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Proposed standard, May 2008. URL: https://tools.ietf.org/html/rfc5280.

[12] Di Crescenzo, G., Ghosh, A., & Talpade, R. 2005. Towards a theory of intrusion detection. In *Proceedings of the 10th European Conference on Research in Computer Security*, ESORICS'05, 267–286, Berlin, Heidelberg. Springer-Verlag. URL: http://dx.doi.org/10.1007/11555827_16, doi:10.1007/11555827_16.

[13] Choudhary, S. & Vidyarthi, M. D. 2015. A simple method for detection of metamorphic malware using dynamic analysis and text mining. *Procedia Computer Science*, 54, 265 – 270. Eleventh International Conference on Communication Networks, ICCN 2015, August 21-23, 2015, Bangalore, India Eleventh International Conference on Data Mining and Warehousing, ICDMW 2015, August 21-23, 2015, Bangalore, India Eleventh International Conference on Image and Signal Processing, ICISP 2015, August 21-23, 2015, Bangalore, India. URL: http://www.sciencedirect.com/science/article/pii/S1877050915013551, doi:https://doi.org/10.1016/j.procs.2015.06.031.

[14] Wang, J., Yang, L., Wu, J., & Abawajy, J. H. May 2017. Clustering analysis for malicious network traffic. In *2017 IEEE International Conference on Communications (ICC)*, 1–6. doi:10.1109/ICC.2017.7997375.

[15] Sommer, R. & Paxson, V. May 2010. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE Symposium on Security and Privacy*, 305–316. doi:10.1109/SP.2010.25.

[16] Coull, S. E., Wright, C. V., Monrose, F., Collins, M. P., & Reiter, M. K. 2007. Playing devil's advocate: Inferring sensitive information from anonymized network traces. In *in Proceedings of the Network and Distributed System Security Symposium*, 35–47.

[17] Bay, S. D., Kibler, D. F., Pazzani, M. J., & Smyth, P. 2000. The uci kdd archive of large data sets for data mining research and experimentation. *SIGKDD Explorations*, 2, 81.

[18] Committee on National Security Systems. 2010. *National Information Assurance (IA) Glossary*. Committee on National Security Systems. URL: https://www.cdse.edu/documents/toolkits-issm/cnssi4009.pdf.

[19] 2018. Mitre attck. https://attack.mitre.org/. Online; Accessed 10-November-2018.

[20] Sinanović, H. & Mrdovic, S. Sept 2017. Analysis of mirai malicious software. In *2017 25th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 1–5. doi:10.23919/SOFTCOM.2017.8115504.

[21] Gardiner, Joseph Cova, M. . N. S. 2014. Command control: Understanding, denying and detecting. https://arxiv.org/ftp/arxiv/papers/1408/1408.1136.pdf. Online; Accessed 10-November-2018.

[22] Mercer, W., Rascagneres, P., & Molyett, M. Introducing rokrat. https://blog.talosintelligence.com/2017/04/introducing-rokrat.html. Online; Accessed 23-November-2018.

[23] A Survey of Network Simulation Tools: Current Status and Future Developments. https://www.cse.wustl.edu/~jain/cse567-08/ftp/simtools/index.html. Accessed: 2018-11-03.

[24] Angrisani, L., Botta, A., Miele, G., & Vadursi, M. Oct 2013. An experimental characterization of the internal generation cycle of an open-source software traffic generator. In *2013 IEEE International Workshop on Measurements Networking (M N)*, 74–78. doi:10.1109/IWMN.2013.6663780.

[25] Kononenko, I. 2007. *Machine learning and data mining : introduction to principles and algorithms*. Horwood Publishing, Chichester, UK.

[26] 2018. "machine". https://www.merriam-webster.com. Online; Accessed 02-November-2018.

[27] 2018. "learning". https://www.merriam-webster.com. Online; Accessed 02-November-2018.

[28] Samuel, A. L. July 1959. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3), 210–229. doi:10.1147/rd.33.0210.

[29] Bishop, C. 2006. *Pattern recognition and machine learning*. Springer, New York.

[30] Anderson, B. & McGrew, D. 2016. Identifying encrypted malware traffic with contextual flow data. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*, AISec '16, 35–46, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/2996758.2996768, doi:10.1145/2996758.2996768.

[31] Anderson, B. & McGrew, D. 2016. Joy. https://github.com/davidmcgrew/joy. Online; accessed 19-November-2017.

[32] Encrypted traffic analytics (eta). https://www.cisco.com/c/en/us/solutions/enterprise-networks/enterprise-network-security/eta.html. Online; accessed 19-November-2017.

[33] Cisco. White paper, cisco public: Encrypted traffic analytics. https://www.cisco.com/c/dam/en/us/solutions/collateral/enterprise-networks/enterprise-network-security/nb-09-encrytd-traf-anlytcs-wp-cte-en.pdf. Online; Accessed 25-November-2018.

[34] Anderson, B. Detecting encrypted malware traffic (without decryption). https://blogs.cisco.com/security/detecting-encrypted-malware-traffic-without-decryption. Online; Accessed 23-November-2018.

[35] Anderson, B., Paul, S., & McGrew, D. Aug 2018. Deciphering malware's use of tls (without decryption). *Journal of Computer Virology and Hacking Techniques*, 14(3), 195–211. URL: https://doi.org/10.1007/s11416-017-0306-6, doi:10.1007/s11416-017-0306-6.

[36] Korczyński, M., Berger-Sabbatel, G., & Duda, A. 2013. Two methods for detecting malware. In *Multimedia Communications, Services and Security*, Dziech, A. & Czyżewski, A., eds, 95–106, Berlin, Heidelberg. Springer Berlin Heidelberg.

[37] IM-Worm:W32/Skipi.A. https://www.f-secure.com/v-descs/im-worm_w32_skipi_a.shtml. Accessed: 2018-10-11.

[38] Bekerman, D., Shapira, B., Rokach, L., & Bar, A. Sept 2015. Unknown malware detection using network traffic classification. In *2015 IEEE Conference on Communications and Network Security (CNS)*, 134–142. doi:10.1109/CNS.2015.7346821.

[39] Velan, P., Čermák, M., Čeleda, P., & Drašar, M. September 2015. A survey of methods for encrypted traffic classification and analysis. *Netw.*, 25(5), 355–374. URL: https://doi.org/10.1002/nem.1901, doi:10.1002/nem.1901.

[40] Husák, M., Čermák, M., Jirsík, T., & Čeleda, P. Feb 2016. Https traffic analysis and client identification using passive ssl/tls fingerprinting. *EURASIP Journal on Information Security*, 2016(1), 6. URL: https://doi.org/10.1186/s13635-016-0030-7, doi:10.1186/s13635-016-0030-7.

[41] Brotherston, L. Tls fingerprinting - smarter defending stealthier attacking. https://blog.squarelemon.com/tls-fingerprinting/. Online; Accessed 25-November-2018.

[42] Ltd, W. Westpointltd/pytls - a python tls library for penetration testers. https://github.com/WestpointLtd/pytls. Online; Accessed 10-December-2018.

[43] 2017. Features/sslpeekandsplice - squid web proxy plugin. https://wiki.squid-cache.org/Features/SslPeekAndSplice. Online; accessed 11-December-2017.

[44] O'Neill, M., Ruoti, S., Seamons, K., & Zappala, D. May 2017. Tls inspection: How often and who cares? *IEEE Internet Computing*, 21(3), 22–29. doi:10.1109/MIC.2017.58.

[45] Data, D. R. Malware traffic patterns. http://data.deependresearch.org/2015/01/malware-traffic-patterns-2015.html. Online; Accessed 26-November-2018.

[46] 2017. Ssl blacklist. https://sslbl.abuse.ch/. Online; Accessed 02-December-2017.

[47] Analysis, H. Readme.ssl. https://www.hybrid-analysis.com/. Online; Accessed 26-November-2018.

[48] PacketTotal - A free, online PCAP analysis engine. https://packettotal.com/. Online; Accessed: 1-March-2018.

[49] Malware-Traffic-Analysis-. http://malware-traffic-analysis.net/. Online; Accessed: 1-March-2018.

[50] Why Choose Bro? https://www.bro.org/why_choose_bro.pdf. Accessed: 2018-10-11.

[51] Bro - Security-Onion-Solution/security-onion Wiki. https://github.com/Security-Onion-Solutions/security-onion/wiki/Bro. Accessed: 2018-10-22.

[52] Detecting malware even when it is encrypted - Machine Learning for network HTTPS analysis. https://2018.bsidesbud.com/wp-content/uploads/2018/03/seba_garcia_frantisek_strasak.pdf. Online; Accessed: 2018-10-22.

[53] Holz, R., Braun, L., Kammenhuber, N., & Carle, G. 2011. The ssl landscape: A thorough analysis of the x.509 pki using active and passive measurements. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, 427–444, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/2068816.2068856, doi:10.1145/2068816.2068856.

[54] Lee, C., Chou, F., & Chen, C. M. Dec 2015. Automatically generating payload-based models for botnet detection. In *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, 1038–1044. doi:10.1109/SmartCity.2015.206.

[55] Straosphere IPS. https://www.stratosphereips.org/stratosphere-ips-suite//. Online; Accessed: 2018-10-09.

[56] Cisco. Snort - network intrusion detection prevention system. https://www.snort.org/. Online; Accessed 25-November-2018.

[57] Cisco. Readme.ssl. https://snort.org/faq/readme-ssl. Online; Accessed 25-November-2018.

[58] Rigaki, M. & Garcia, S. May 2018. Bringing a gan to a knife-fight: Adapting malware communication to avoid detection. In *2018 IEEE Security and Privacy Workshops (SPW)*, 70–75. doi:10.1109/SPW.2018.00019.

[59] Index of /publicDatasets/CTU-Normal-20. https://mcfp.felk.cvut.cz/publicDatasets/CTU-Normal-20/. Online; Accessed: 2018-10-22.

[60] Index of /publicDatasets/CTU-Normal-21. https://mcfp.felk.cvut.cz/publicDatasets/CTU-Normal-21/. Online; Accessed: 2018-10-22.

[61] Index of /publicDatasets/CTU-Normal-22. https://mcfp.felk.cvut.cz/publicDatasets/CTU-Normal-22/. Online; Accessed: 2018-10-22.

[62] Index of /publicDatasets/CTU-Normal-23. https://mcfp.felk.cvut.cz/publicDatasets/CTU-Normal-23/. Online; Accessed: 2018-10-22.

[63] Index of /publicDatasets/CTU-Normal-24. https://mcfp.felk.cvut.cz/publicDatasets/CTU-Normal-24/. Online; Accessed: 2018-10-22.

[64] Index of /publicDatasets/CTU-Normal-25. https://mcfp.felk.cvut.cz/publicDatasets/CTU-Normal-25/. Online; Accessed: 2018-10-22.

[65] Index of /publicDatasets/CTU-Normal-26. https://mcfp.felk.cvut.cz/publicDatasets/CTU-Normal-26/. Online; Accessed: 2018-10-22.

[66] Index of /publicDatasets/CTU-Normal-27. https://mcfp.felk.cvut.cz/publicDatasets/CTU-Normal-27/. Online; Accessed: 2018-10-22.

[67] Index of /publicDatasets/CTU-Normal-28. https://mcfp.felk.cvut.cz/publicDatasets/CTU-Normal-28/. Online; Accessed: 2018-10-22.

[68] Index of /publicDatasets/CTU-Normal-29. https://mcfp.felk.cvut.cz/publicDatasets/CTU-Normal-29/. Online; Accessed: 2018-10-22.

[69] Index of /publicDatasets/CTU-Normal-30. https://mcfp.felk.cvut.cz/publicDatasets/CTU-Normal-30/. Online; Accessed: 2018-10-22.

[70] Index of /publicDatasets/CTU-Normal-31. https://mcfp.felk.cvut.cz/publicDatasets/CTU-Normal-31/. Online; Accessed: 2018-10-22.

[71] Index of /publicDatasets/CTU-Normal-32. https://mcfp.felk.cvut.cz/publicDatasets/CTU-Normal-32/. Online; Accessed: 2018-10-22.

[72] Anderson, B. & McGrew, D. 2017. 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, 1723–1732, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/3097983.3098163, doi:10.1145/3097983.3098163.

[73] Kolesnikov, A. 2017. *Load modelling and generation in IP-based networks : a unified approach and tool support*. Springer Vieweg, Wiesbaden, Germany.

[74] n1nj4sec/pupy. Pupy. https://arxiv.org/ftp/arxiv/papers/1408/1408.1136.pdf. Online; Accessed 23-November-2018.

[75] Jon Dugan, Seth Elliott, B. A. M. J. P. K. P. 2014–2018. Iperf3. https://github.com/esnet/iperf. Online; accessed 10-August-2018.

[76] Biondi, P. 2008–2018. Scapy. https://scapy.net. Online; accessed 10-August-2018.

[77] Borkmann, D. 2009–2018. netsniff-ng. http://netsniff-ng.org/. Online; accessed 10-August-2018.

[78] Merkel, D. March 2014. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239). URL: http://dl.acm.org/citation.cfm?id=2600239.2600241.

[79] Hidayat, A. 2015–2017. nflow-generator. https://github.com/nerdalert/nflow-generator/. Online; accessed 10-August-2018.

[80] Lantz, B., Heller, B., & McKeown, N. 2010. A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, 19:1–19:6, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/1868447.1868466, doi:10.1145/1868447.1868466.

[81] Peuster, M., Karl, H., & van Rossem, S. Nov 2016. Medicine: Rapid prototyping of production-ready network services in multi-pop environments. In *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 148–153. doi:10.1109/NFV-SDN.2016.7919490.

[82] Peuster, M. 2018. Containernet. https://containernet.github.io/. Online; accessed 10-September-2018.

[83] Cisco. Trex - realistic traffic generator. https://trex-tgn.cisco.com/. Online; Accessed 7-December-2018.

[84] Networks, J. Juniper/warp17 - the stateful traffic generator for layer 1 to layer 7. https://github.com/Juniper/warp17. Online; Accessed 7-December-2018.

[85] Erlacher, F. & Dressler, F. 2018. Testing ids using genesids: Realistic mixed traffic generation for ids evaluation. In *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*, SIGCOMM '18, 153–155, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/3234200.3234204, doi:10.1145/3234200.3234204.

[86] García, S., Grill, M., Stiborek, J., & Zunino, A. 2014. An empirical comparison of botnet detection methods. *Computers Security*, 45(Supplement C), 100 – 123. URL: http://www.sciencedirect.com/science/article/pii/S0167404814000923, doi:https://doi.org/10.1016/j.cose.2014.05.011.

[87] Shalaginov, A., Franke, K., & Huang, X. 04 2016. Malware beaconing detection by mining large-scale dns logs for targeted attack identification.

[88] Erquiaga, M., García, S., & Garcia Garino, C. 10 2017. Observer effect: How intercepting https traffic forces malware to change their behavior.

[89] StratosphereTestingFramework. https://github.com/stratosphereips/StratosphereTestingFramework/. Online; Accessed: 2018-10-09.

[90] Malware Capture Facility Project. https://mcfp.weebly.com/. Online; Accessed: 2018-10-09.

[91] Combs, G. 1998–2018. Wireshark. https://www.wireshark.org/. Online; Accessed: 1-March-2018.

[92] Hidayat, A. 2011–2018. PhantomJS. https://github.com/ariya/phantomjs/. Online; Accessed: 15-March-2018.

[93] Stenberg, D. 1997–2018. cURL. https://github.com/curl/curl. Online; Accessed: 15-March-2018.

[94] Keyword Research, Competitive Analysis, Website Ranking | Alexa. https://www.alexa.com. Online; Accessed: 1-May-2018.

[95] Raspberry Pi 2 Model B. https://www.raspberrypi.org/products/raspberry-pi-2-model-b/. Online; Accessed: 2018-05-01.

[96] Torbjørnsen, A. andetorb/mis4900 - software for my master thesis. https://github.com/andetorb/MIS4900. Online; Accessed 15-December-2018.

[97] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. 2009. The WEKA data mining software: an update. *SIGKDD Explorations*, 11(1), 10–18.

[98] Huitema, C. & Rescorla, E. SNI Encryption in TLS Through Tunneling. Expired Internet-Draft, March 2018. URL: https://tools.ietf.org/html/draft-ietf-tls-esni-02.

[99] Prince, M. Encrypting sni: Fixing one of the core internet bugs. https://blog.cloudflare.com/esni/. Online; Accessed 23-November-2018.

[100] Rescorla, E. Encrypted sni comes to firefox nightly. https://blog.mozilla.org/security/2018/10/18/encrypted-sni-comes-to-firefox-nightly/. Online; Accessed 23-November-2018.

[101] Thomas Hungenberg, M. E. 2007–2017. INetSim. http://www.inetsim.org/. Online; Accessed: 30-March-2018.

[102] Barr-Smith, F. Dynamic Behavioural Analysis of Malware via Network Forensics. Master's thesis, University of Oxford, England, 2018.

[103] VirusTotal - Free Online Virus, Malware and URL Scanner. https://www.virustotal.com/en/. Online; Accessed: 1-March-2018.

A Appendix

A.1 Malware Lookup - Hybrid-Analysis

Returns JSON formatted data.

```
curl -X POST \
    "https://www.hybrid-analysis.com/api/v2/search/terms?_timestamp=<timestamp>"\
    -H "accept: application/json" -H "user-agent: Falcon Sandbox" \
    -H "api-key: <API-key>" \
    -H "Content-Type: application/x-www-form-urlencoded" \
    -d "verdict=5&port=443"
```

A.2 Malware Data

Data from 26 malware samples from Hybrid-Analysis which uses TLS on port 443 and is labeled as malicious.

```
"verdict": "malicious",
"sha256": "d9651fa343f79f71baf3f0d6554eec10fbe82fea64dea52d6258f88a9e2acb61",
"type_short": "64-bit exe"

"verdict": "malicious",
"sha256": "5753fa05e5aa12e2304ad2e634c5efd3c04e29ef2536a8e6363598c7d360b1d7",
"type_short": "64-bit exe"

"verdict": "malicious",
"sha256": "6750eab2f9c583c272bd8011bc0c2f2b5bf3d947fac39d2150c493c7323d9b79",
"type_short": ".NET exe"

"verdict": "malicious",
"sha256": "10810ac39fa23e7e64330b95724cd649040729705b9fbeba03064fb81ab6346a",
"type_short": "exe"
```

"verdict": "malicious",
"sha256": "91bff092f7ce2d3b04aead6c553c8d6a13d4b6ae5aba1cc3cf7761e16efdbadc",
"type_short": "exe"

"verdict": "malicious",
"sha256": "c878b37b7236aa3a230b9e4b613dee0538182ff043944abcebadf78b08cfc426",
"type_short": "exe"

"verdict": "malicious",
"sha256": "3b0096d6798b1887cffa1288583e93f70e656270119087ceb2f832b69b89260a",
"type_short": "exe"

"verdict": "malicious",
"sha256": "2acd0d9ac5dca23aa232073aebc049ca54bfc8fd5dcae58a7c96cdc18fb7859e",
"type_short": "exe"

"verdict": "malicious",
"sha256": "19c17122fba29f87637d99e6a489dc89ba0baf33f23efe9a956c0cd6d3b0541e",
"type_short": "64-bit exe"

"verdict": "malicious",
"sha256": "001240304b1badd6d05e5d95fcc19316e11f8671ae9b003f8081de65f2e90405",
"type_short": "exe"

"verdict": "malicious",
"sha256": "08e91f45eae7126fddec2d92efd49b67ef3b82c59b0b8c037f675c00e72dec42",
"type_short": "exe"

"verdict": "malicious",
"sha256": "f36f9d655efb7eca37a8b4e0182a90fb2b6bafe6b9f9fa3fdc104405ad6f0d9e",
"type_short": "exe"

"verdict": "malicious",
"sha256": "da56618a8a5a0fb22bb12a90a63e8b9dd715b66f4ec7ba23eaed98d2b8cbddf0",
"type_short": "doc"

"verdict": "malicious",
"sha256": "dde36643ab431aba63538ab48d71c79964c9cdbfa60b1ed18a9ce4cccdd80b69",
"type_short": "doc"

"verdict": "malicious",
"sha256": "307c0d90e71f090fca4bc93dd242818f686188c5fb7bdeeab93d9f5a972c58b4",
"type_short": "exe"

"verdict": "malicious",
"sha256": "55fe5e4547d5d0c7e86a2e128b69dce5d9fad78f7781a8791bf03a2536c1511b",
"type_short": "exe"

"verdict": "malicious",
"sha256": "4c93a220ce61bc59280424ef8457951f9fa188de4ef7e7bd8a67ba3ed0d81ae3",
"type_short": "exe"

"verdict": "malicious",
"sha256": "3fb4b65cae358493a5d1d50be276fc863b7a41c2cc2c402cae57af437e075cca",
"type_short": ".NET exe"

"verdict": "malicious",
"sha256": "a9c2dc7b854b7de36380168ddee046c1433b9276c5e9fd1977773d2b0fe16935",
"type_short": "exe"

"verdict": "malicious",
"sha256": "5962f42dcb66ab283a9a9d407b3e90f3591c151e0d77afc5c1bca68e6befbfc6",
"type_short": "exe"

"verdict": "malicious",
"sha256": "9cf6dbcf73ae6bc8b01367bbc68ac9d518b96fe4fdbcfca84f6007e67488aff6",
"type_short": "exe"

"verdict": "malicious",
"sha256": "61216c38f1548b6184092869820343d2e5155fcf1986680e2e259c24239d92a6",

"type_short": ".NET exe"

"verdict": "malicious",
"sha256": "da70df51aa80414fcba9bf7322e44e8ea5ed6a3725f342cd05c733376c6f2121",
"type_short": "doc"

"verdict": "malicious",
"sha256": "4af17e81e9badf3d03572e808e0a881f6c61969157052903cd68962b9e084177",
"type_short": "exe"

"verdict": "malicious",
"sha256": "a8b49d5b0cb7e81d5e416bfe1f137d7e2e92087fd70abd905ac31211ee144d88",
"type_short": "exe"

"verdict": "malicious",
"sha256": "c17ac4abdec889f9bdcbf9c9a6144672da18ea5334fa84a34a4f2300f96610c2",
"type_short": "exe"

A.3 URLs Used for PhantomJS and cURL

vg.no
nrk.no
dagbladet.no
aftenposten.no
yr.no
digi.no
tu.no
uninett.no
ntnu.no
uio.no
hioa.no
samordnaopptak.no
altinn.no
skatteetaten.no

netflix.com
hbo.com
spotify.com
matprat.no
kolonial.no
peppes.no
dinside.no
finn.no
storm.no
norsk-tipping.no
lanekassen.no
snl.no
regjeringen.no
nsb.no
posten.no
vegvesen.no
ssb.no
dnb.no
lovdata.no
ffi.no
nbc.com
forskning.no
nba.com
paypal.com
ebay.com
nhl.com
usa.gov
amazon.com
cnn.com
peta.org
nato.int
marvel.com
ticketmaster.com
whatsapp.com
merriam-webster.com

youtube.com
vive.com/en
sas.no
norwegian.no
wikipedia.org
apple.com
twitch.tv
python.org
ryanair.com
duckduckgo.com
mozilla.org