MASARYK
UNIVERSITY

FACULTY OF INFORMATICS

Detection of Malicious Network Traffic Behavior

Master's Thesis

BC. PAVEL NOVÁK

Brno, Spring 2022

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Detection of Malicious Network Traffic Behavior

Master's Thesis

BC. PAVEL NOVÁK

Advisor: Ing. Václav Oujezský, Ph.D.

Department of Computer Systems and Communications

Brno, Spring 2022


FACULTAS ARTIS INFORMATICÆ
UNIVERSITAS MASARYKIANA

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Bc. Pavel Novák

Advisor: Ing. Václav Oujezský, Ph.D.

Acknowledgements

I am very grateful to my supervisor, Ing. Václav Oujezský, Ph.D, for sharing his expertise and sincere and valuable guidance. I am also thankful to my family and everyone who supported me in my study.

Abstract

Encrypted traffic in today's networks is an indispensable attribute. It is a valuable tool to protect users' data against eavesdropping or modification. However, encryption also opened new opportunities for malware developers because it makes hiding malicious communication more effortless. This problem has become much more important nowadays because the portion of encrypted communication generated by the malware was around 45 % last year, and the number is increasing every year.

Encrypted traffic is much harder to analyze because of privacy issues and performance demands. This inevitably leads to massive research in this area and the development of techniques to classify encrypted traffic without decrypting it.

This thesis improves the JA3 fingerprinting method to detect unknown network traffic. The new method uses clustering of JA3 fingerprints to recognize malicious traffic.

Keywords

behavior analysis, cybersecurity, data analysis, JA3, malware analysis

Contents

List of Tables

List of Figures

Abbreviations

C2 Command and Control 3

CA Certificate Authority 11

CRL Certificate Revocation List 12

DBSCAN Density-based Spatial Clustering of Applications with Noise 29

DNS Domain Name System 18

DoH DNS over HTTPS 13

DPI Deep Packet Inspection 3

FTP File Transfer Protocol 6

HTTP HyperText Transfer Protocol 6

IDP Intrusion Detection and Prevention System 4

IP Internet Protocol 6

IPSec Internet Protocol Security 3, 6

MAC Message Authentication Code 8

OPTICS Ordering Points To Identify the Clustering Structure 29

PGP Pretty Good Privacy 3

PKI Public Key Infrastructure 11

RAT Remote Access Trojan 18

RSA Rivest – Shamir – Adleman Algorithm 6

RTT Round Trip Time 8

SMB Server Message Block 18

SMTP Simple Mail Transfer Protocol 6

SNI Server Name Identifier 13

SSH Secure Shell 3

SSL Secure Socket Layer 3, 6

TCP Transmission Control Protocol 6

TLS Transport Layer Security 3, 6

URI Uniform Resource Identifier 13

VPN Virtual Private Network 6

1 Introduction

Detection of malicious network traffic has been the subject of research for quite a long time. In the early phases of developing the Internet and networks, detecting undesirable traffic was relatively easy. The root cause of this easiness was that traffic was mainly unencrypted in those days. Networks were dedicated only to a few companies and academic institutions. Over time, however, the situation has changed due to the massive increase in users. Privacy has become an important issue, and communication have started to be encrypted before sending over the network. Encryption has become a double-edged weapon. On the one hand, it helps ordinary users protect their privacy, and without the use of encryption protocols, we cannot imagine a secure Internet today. On the other hand, it also helps hide malware-generated communication with Command and Control (C2) servers.

Over time, several encrypted communication protocols have been developed. Examples could be Secure Shell (SSH), Internet Protocol Security (IPSec), Pretty Good Privacy (PGP) and many others protocols. This thesis only deals with the Secure Socket Layer (SSL)\Transport Layer Security (TLS) protocol. The reason is that the vast majority of malware that uses encrypted communication on the Internet uses this protocol to communicate. Other protocols will not be considered.

The analysis of encrypted traffic becomes more and more topical. On average more than 95 % of traffic to services used by Google was encrypted, according to a Google Transparency Report [1]. In some countries, such as Belgium or India, it was almost 100 %. In the case of malware, the portion is not yet that high. According to the Sophos research last year, the share of encrypted communications generated by the malware was around 45 % [2]. Nevertheless, compared to the same research in 2020, this number has almost doubled, and this trend is likely to continue in upcoming years as well [3].

Thus, massive research has been going on for a long time in the field of encrypted traffic classification. Currently, there are three primary research directions [4]. The first is the Deep Packet Inspection (DPI) method, which decrypts and inspects each packet individually. Privacy issues and computational complexity severely limit this method, which prevents its use in large and public networks. The sec-

ond option is based on behavioural analysis. This method measures flow characteristics such as time between packets or the number of packets sent. Moore et al. proposed a large set of features suitable for behavior-based analysis [5]. Correct feature selection is critical to the successful use of this method. This method is less precise, but its main advantage is that it does not require knowledge of the underlying protocols. These two methods are out of the scope of this paper. The last option is fingerprinting method. This method uses information observable in the initial phase of an encrypted connection – the handshake. The crucial point is that all the information are unencrypted. This method is the basis of this thesis.

1.1 Scope and Goals

One method of fingerprinting TLS connections is the JA3 fingerprint method [6]. This fingerprint comprises the information contained in the TLS handshake and includes, for example, the TLS version or supported ciphersuits. This method is currently actively used for detection and is implemented in the Suricata Intrusion Detection and Prevention System (IDP) [7]. The considerable expansion of this method in the wild created a relatively large database of JA3 fingerprints assigned to specific malware families [8, 9]. However, this method has its drawbacks too. The main one is, that detection is based on a direct match between the JA3 fingerprint in the database and the unknown captured fingerprint. The direct match can be a problem because a JA3 fingerprint not in the database can not be recognized as malware.

The solution proposed, implemented and tested in this work is trying to solve the problem of unknown fingerprints by clustering them and comparing the cluster set structure with a known clean traffic. The idea behind is, that set of clusters containing malware traffic should have different structure caused by the malware clusters. The goal of this thesis is to provide the following major answers.

- **Accuracy and Usability** – Is it possible to recognize the malware traffic using the proposed method?
- **Efficacy** – Is this method sufficiently quick to be used in the large network?

1.2 Structure of the Thesis

The thesis has the following structure:

- Chapters 2 and 3 describe the usage of encrypted traffic in current networks. Chapter 3 focuses specifically on how malware uses encrypted traffic to hide its content.
- Chapter 4 describes methods currently used to classify encrypted traffic. The main focus here is on fingerprinting methods since they are the basis for this thesis.
- Chapter 5 describes the overall design of the proposed solution method. The clustering methods used are described here. Also, the way of measuring the distance between JA3 fingerprints is described in theoretical terms in this chapter.
- Chapter 6 deals with implementation details.
- Chapter 7 summarizes and describes the tests and results obtained.
- Chapters 8 and 9 summarize the results and the possibilities for further development and research in this area.

2 Network Security

Cryptography has been an important part of communication for a very long time. As a form of cryptography can be considered already hieroglyphics in ancient Egypt or transposition ciphers used in Greece. Cryptography has also played an essential role during wars. Perhaps the most famous example is the Enigma machine used during World War II. In short, protecting information has been vital to humankind since ancient times. Therefore, it is not so surprising that cryptography has infiltrated the modern form of communication – the Internet. The Internet itself originated in the 1960s as a communication network for the military and a few select universities. Shortly after that, the issue of protecting information transmitted over the Internet began to be addressed. The invention of asymmetric cryptography and protocols such as Diffie-Hellman or RSA in the 1970s contributed significantly to this. The real boom came in the 1990s with the opening of the Internet to the public, though. At the same time, the first versions of the application protocols known today as Secure Socket Layer (SSL), later Transport Layer Security (TLS), Virtual Private Network (VPN) or Internet Protocol Security (IPSec) were developed. Despite this, it took a relatively long time for secure traffic to become more massively embedded in today's Internet. As recently as 2014, the share of encrypted traffic was just under half. However, the situation has already changed, and unsecured traffic is the exception nowadays [10, 11].

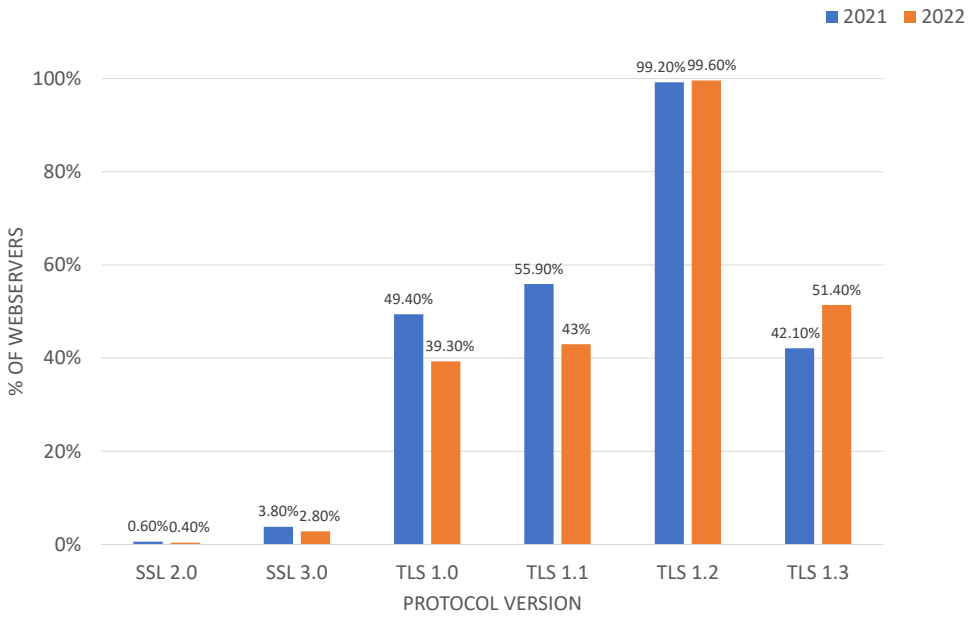
2.1 SSL/ TLS

Transport Layer Security is currently one of the most widely used protocols to encrypt much of the traffic on the Internet. Its predecessor was the SSL protocol, in development since 1995. SSL/TLS is a Layer 5 protocol of the TCP/IP model and provides encrypted data transfer for application protocols such as HyperText Transfer Protocol (HTTP), File Transfer Protocol (FTP), or Simple Mail Transfer Protocol (SMTP). SSL/TLS is also used in several Virtual Private Network (VPN) implementations [12]. The protocol provides authentication of communicating parties, confidentiality and integrity of messages. SSL has gradually evolved into three versions but has been already fully

replaced by TLS. TLS has four versions, but only version 1.2 and the latest 1.3 are commonly used. The currently mostly used TLS version 1.2 is vulnerable to many attacks like the Heartbleed Attack or the Triple Handshake Attack [13, 14].

This initiated development of the new TLS version 1.3, published in 2018. This version improved the security as well as speed in terms of handshake duration [15]. Figure 2.1 shows the evolution of support for each TLS version between 2021 and 2022.

Figure 2.1: Supported TLS Versions by Webservers (2021 vs 2022).



The dominant version is 1.2, supported by 99.6 % of all web servers in January 2022. There is also an evident loss of support for versions 1.0 and 1.1. This is due to the official deprecation of these versions in March 2021 [16]. In contrast, the latest version, 1.3, is supported in roughly half of the cases and increases [17].

The main goal of the TLS protocol is to provide three basic properties [18].

- **Authentication** – Authentication of the server is mandatory, and authentication of the client is optional. A server and a client can authenticate using either asymmetric or symmetric methods.

- **Confidentiality** – All data sent over the network is encrypted after the TLS connection is established and is thus visible only to the end devices. The actual data transfer is secured by a symmetric key established during the handshake.

- **Data Integrity** – Data cannot be modified by any means without being detected. After the handshake is complete, the peers exchange the Message Authentication Code (MAC) of the entire handshake, which prevents the modification of any part of the handshake. A message-digest protects all subsequently sent data as well.

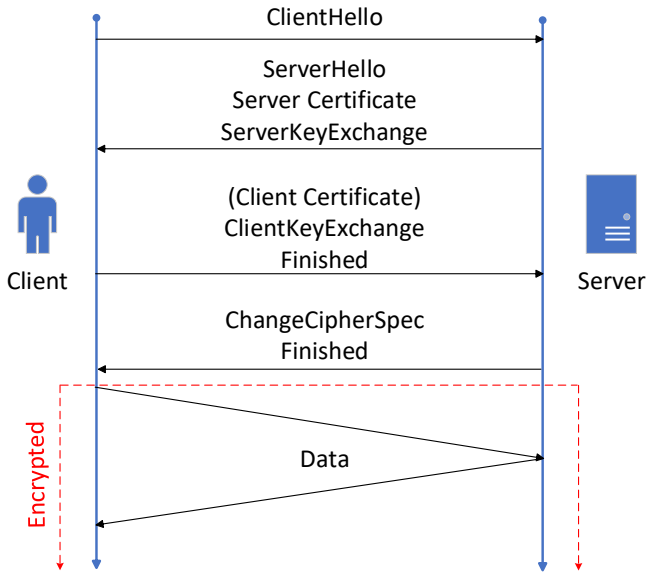
2.1.1 TLS Handshake

A TLS handshake is the initial part of a connection between two communicating parties. During this phase, the communication partners exchange connection parameters and establish a symmetric key. The handshake is the only part of a TLS connection that is not encrypted and is, therefore, the basis for fingerprinting-based detection mechanisms described in Chapter 4. Figure 2.2 shows a 2 Round Trip Time (RTT) TLS handshake used in version 1.2. The first round starts by sending the initial `ClientHello` message. Server responds with `ServerHello` message, certificate and server part of the key. The second round finishes the key establishment protocol and switches to encrypted communication.

TLS version 1.3 brings, among other things, two faster handshake mechanisms. The 0-RTT and 1-RTT handshake does not require as many messages to be exchanged to establish a connection. For this reason, the establishment of a secured channel is faster. 0-RTT handshake even allows sending data on the first flight ("early data"). The prerequisite for using this method is sharing a common pre-shared key between peers, for example, from the previous session. Figure 2.3 shows the 1-RTT handshake used in version 1.3. The messages from the 2-RTT handshake are preserved, but just one round is needed to exchange them.

The important message regarding TLS connection fingerprinting methods is the initial `ClientHello` and `ServerHello` message. These messages contain the parameters listed below. Sufficient diversity of

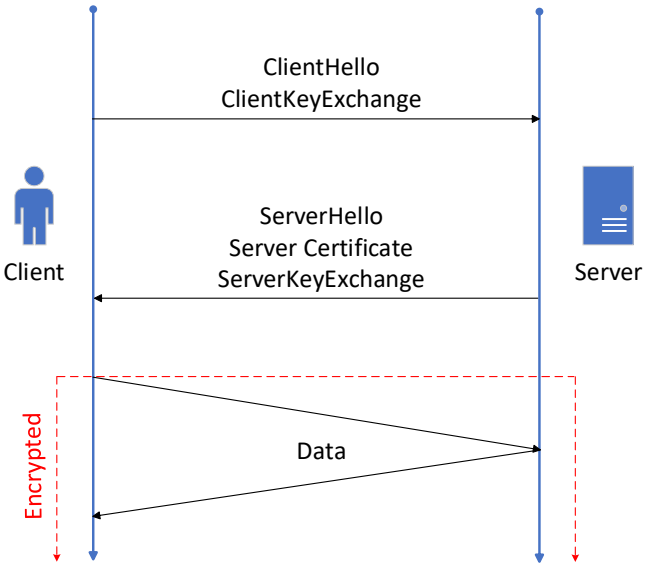
Figure 2.2: TLS 1.2 Handshake.



these parameters is an essential prerequisite for fingerprinting-based methods.

- **ClientHello Message**
  - **TLS Version** – 1.1, 1.2 or 1.3.
  - **Client Random Number** – Can be used to generate a symmetric session key.
  - **Session ID** – It can speed up the handshake process by reference to the algorithms and keys agreed upon in the previous session.
  - **Cipher Suites List** – A list of supported ciphers ordered by preference.
  - **Supported Compression Methods** – List of supported compression methods.
  - **List of Extensions** – Additional attributes, e.g. Server Name or the signature algorithm.

Figure 2.3: TLS 1.3 Handshake.



  - **List of Public Keys** – A server may use them during the key exchange process.

- **ServerHello Message**
  - **Negotiated TLS Version** – Usually the highest offered by a client.
  - **Chosen Cipher Suit** – Selected cipher suit.
  - **Session ID** – Newly created ID of the session.
  - **Compression Method** – Selected compression method.
  - **Server Random Number** – The second part of the symmetric shared key.

2.1.2 TLS Certificate

Another valuable piece of information that can be observed during a handshake is a certificate verifying the identity of a server or client. The

certificate is always sent by the server during the handshake. Thanks to this, it is possible to deduce more about the nature of communication. There exist typical features, like self-signed certificates and a strange issuer or subject names, in the case of malware-related traffic (characteristics of malware encrypted communication are described in Chapter 3). The TLS protocol uses the X.509 Public Key Infrastructure (PKI) standard. Figure 2.4 shows the format of an X.509 certificate [19].

Figure 2.4: TLS Certificate Structure.

```
Certificate:
        signedCertificate
            version:
            serialNumber:
            signature (sha256WithRSAEncryption)
                Algorithm Id:
            issuer: rdnSequence (0)
            validity
                notBefore: utcTime (0)
                notAfter: utcTime (0)
            subject: rdnSequence (0)
            subjectPublicKeyInfo
                algorithm (rsaEncryption)
                subjectPublicKey:
                    modulus:
                    publicExponent:
            extensions (optional):
```

- **Version** – Certificate version number (v1, v2 or v3).
- **Serial Number** – Unique ID for every certificate issued by the same Certificate Authority (CA).
- **Signature Algorithm ID** – The algorithm used to sign the certificate.
- **Issuer** – Identification of the certificate issuer. Typically one of the trusted CA, but this field may contain any value.
- **Validity** – Limits the duration of the certificate. Standard is one year, typically no more than 13 months [20].
- **Subject** – Identification of the owner of the certificate.

- **Subject Public Key Info** – Public key of the subject.
- **Extensions** – Optional extensions like Certificate Policies or Certificate Revocation List (CRL) Distribution Points.

The certificate is either self signed or digitally signed by the issuing CA. The information contained in the certificate is practically usable only in the case of TLS version 1.2 and lower. Handshake in version 1.3 is entirely encrypted except for the initial `Hello` messages, i.e. the certificate is encrypted as well. This poses a severe challenge to fingerprinting the certificate. However, this could be solved by proactively querying the server for the certificate for further inspection.

2.1.3 Differences Between TLS 1.2 and TLS 1.3

The introduction of the new TLS 1.3 standard brought a shift in both performance and security compared to the previous version. Of course, there are many changes, and only the most important ones are listed below.

- **Stronger Ciphersuits** – There has been a significant improvement in security in the new version. This is due to the deprecation of old and not very secure ciphersuits and hash algorithms such as SHA-1, RC4, DES or 3DES. This leads to a significant increase in security as it forces users to use more secure ciphersuits and prevents so-called downgrade attacks.
- **Perfect Forward Secrecy** – To further enhance security, all supported cryptographic protocols in version 1.3 have the Perfect Forward Secrecy property. This property makes it impossible for an attacker to decrypt intercepted messages if he later obtains the key used to encrypt them.
- **Encrypted Handshake** – All handshake messages except the `Client Hello` and `ServerHello` messages are now encrypted. The newly introduced `EncryptedExtensions` message allows various extensions previously sent in the clear text as the part of `ServerHello`.
- **Faster Handshakes** – TLS 1.3 brings a speed acceleration in the form of 1-RTT and 0-RTT handshakes. Since there is no need

to exchange so much information when establishing a secure connection, the whole handshake is much faster [21, 18].

2.2 Application Protocols Using TLS/SSL

TLS protocol is universal. It allows to secure any communication on the application layer and is therefore widely used. The following are examples of the well-known L7 protocols using TLS nowadays.

- **HTTPS** – Probably the most widespread use of TLS today. TLS secures HTTP communication using the well-known port 443. It also provides authentication of the web server being visited. All data exchange, including visited Uniform Resource Identifier (URI) or form data is also encrypted. Because of the ability to host multiple URIs on a single server, Server Name Identifier (SNI) information has been added to the HTTPS handshake to identify the requested resource [22].
- **FTPS** – FTP extension. It provides both control message transfer and data transfer and server authentication. The implicit version uses ports 990 and 989. Users can also switch to the secure mode during the connection and thus use standard ports 20 and 21 [23].
- **DNS over HTTPS (DoH)** – A new protocol used to secure the DNS service. The goal is to prevent modification and spoofing of DNS messages. This protocol is relatively new. It started to be tested in 2018 and therefore its adoption is not yet high. In 2020, it started to be offered as a default option for Firefox users [24].
- **VPN** – Some VPN implementations use TLS under the hood. One of these is the open-source software OpenVPN. It uses OpenSSL libraries to provide encrypted and authenticated traffic between end devices [25].

3 Malware and Encrypted Traffic

Malware developers have not been too far behind in using encrypted communications. The advent of SSL/TLS, which is supported by default on all commercially available devices, makes this protocol an ideal tool to use. Nevertheless, malware developers have a slightly different motivation to use encrypted connections than other developers. While in benign software, the motivation is to protect user data, in the case of malware, the primary motivation is to harden traffic analysis by firewalls and IDP systems. Encrypted traffic is usually considered clean because an inspection of such traffic requires deploying a full proxy, and so this solution is usable only in companies. The main reasons why malware communicates over the network are listed below.

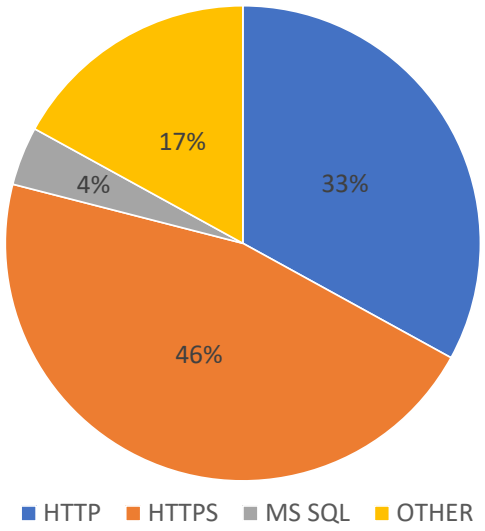
- **Downloaders** – The first reason is downloading additional malware. This is a typical example of a downloader malware, which infects a computer and then downloads the final payload, like ransomware, from a Command and Control (C2) server.
- **Data Exfiltration** – The second reason is the exfiltration of stolen data. The typical example here is spyware that sends data such as screenshots or captured keyboard logs to C2 servers.
- **Command and Control** – The last option is malware communicating with the C2 server to receive instructions. Examples are botnet malware like Mirai or Zeus.
- **Phishing** – Phishing is an attempt to mimic a legitimate website's appearance and extract information such as a user's credit card number or password. TLS is often used here as well to spoof a secure connection. According to research by PhishLabs, 80 % of users believe that the green lock symbol in the browser means that the page is safe or can be trusted, and the fact that 49 % of all phishing websites are seemingly using HTTPS can be interpreted as exploitation of this misunderstanding [26].

In all cases, encrypted traffic provides a significant advantage and dramatically improves the chance that the communication will not be detected using traditional signature based methods [27].

3.1 Current State

The adoption of encrypted communication by malware is slightly behind the current state in the case of benign software. While the total amount of encrypted traffic in legit traffic is over 90 %, it was only slightly less than 50 % in the case of malware in 2021. Nevertheless, compared to 2020, this number is more than double the increase, and this trend can be expected in the coming years as well [2, 3].

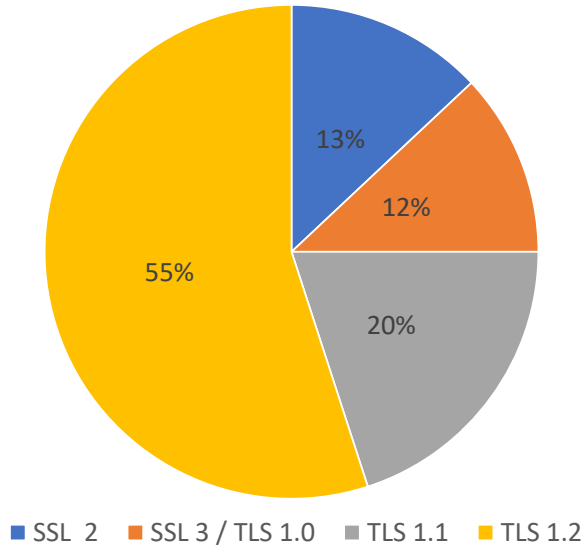
Figure 3.1: Malware C2 Communication Protocols.



Malware developers, unlike others, use encrypted traffic mainly to hide data content. For this reason, malware is less likely to use best-practice protocols, and these deviations from best practices can also be used to classify and detect unsolicited traffic. One example of this trend is using older versions of the TLS protocol or weaker ciphersuits. Figure 3.2 shows the ratio of TLS versions used by the malware. TLS 1.2 has the majority, and no traffic was captured using the latest version, TLS 1.3. This does not mean that no malware is using this version, but its share is so small that it was not captured in real traffic.

The TLS certificate handling is also a typical feature of malware. More than 60 % of malware samples use self-signed certificate, which is not very common in legit traffic. Of the certificates issued by legitimate CAs, the most common are those from Let's Encrypt since it

Figure 3.2: TLS Versions Used by Malware.



issues certificates for free, in an automated fashion, with no significant oversight on the entity requesting it, with the caveat that the certificate is only valid for 90 days. Therefore, the certificate needs to be renewed after 90 days. Table 3.1 shows the names of the top 10 malware certificate issuers. The abnormality is usually the name of the malware, default or nonsense issuer name [28, 29].

3.2 Examples of TLS-based Malware

This section presents selected malware strains and their usage of the encrypted network traffic to hide malicious communication with the C2 servers.

3.2.1 Dridex

Dridex malware is a trojan specializing in stealing bank credentials [30]. It was developed back in 2014 and has undergone slight modifications since then, but it has been using TLS to hide communication with its C2 server since the beginning. Dridex has a modular architecture, meaning that the core of the malware itself is just a downloader that

Rank	CA Name
1	Let's Encrypt Authority X3
2	AsyncRAT Server
3	C=US, ST=Denial, L=Springfield, O=Dis
4	localhost
5	BitRAT
6	C=XX, L=Default City, O=Default Company Ltd
7	R3
8	example.com
9	C=AU, ST=Some-State, O=Internet Widgits Pty Ltd
10	*

Table 3.1: Top Issuing CAs [28].

downloads additional modules from the C2 server that have different functionality. The infection vector is often a malicious macro in Microsoft Word documents. These documents are usually distributed as an attachment of phishing campaigns via email. Once the user opens the document and runs the macro, Dridex contacts the C2 server and downloads the additional modules needed. Figures 3.3 and 3.4 show the TLS handshake of the Dridex malware with its C2 server [31]. The most important parts of the handshake are highlighted. The used protocol is SSL 3.0, which is considered obsolete today. Moreover, the offered ciphersuits are weak and do not match current standards. A considerable security violation is also the self-signed server certificate.

Figure 3.3: Dridex Client Hello Message.

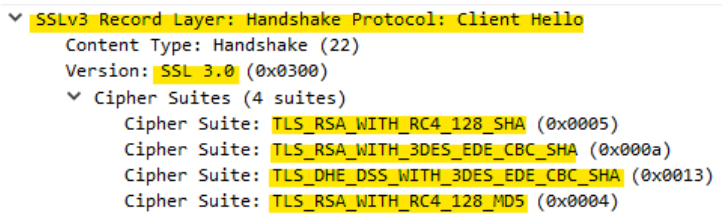
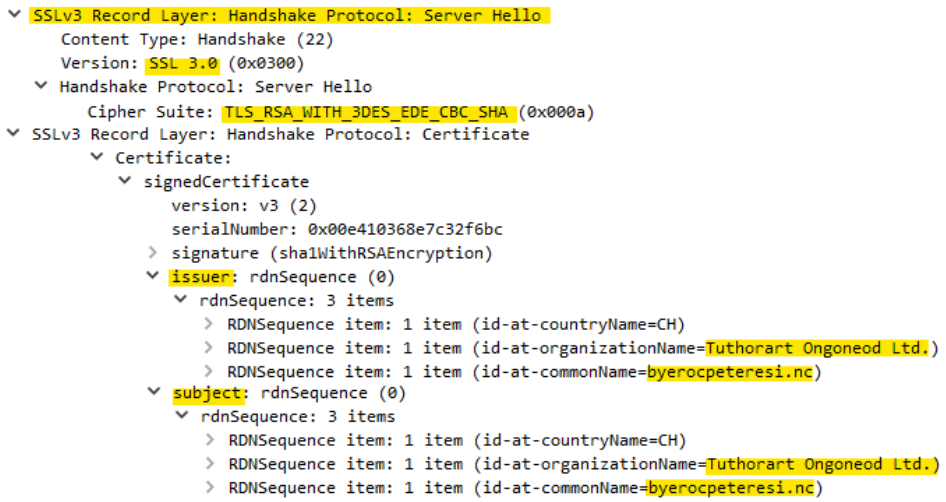


Figure 3.4: Dridex Server Hello Message.



3.2.2 Cobalt Strike

Cobalt Strike has initially been a paid tool used for penetration testing [32]. However, it is a Remote Access Trojan (RAT) malware at its core, so it is not surprising that it was soon misused for criminal purposes. Cobalt Strike allows communication with a C2 server using various protocols such as HTTP, HTTPS, Domain Name System (DNS) or Server Message Block (SMB). This mix of supported protocols makes the detection of this malware much more difficult. Also, the level of communication concealment is at a higher level than Dridex. Figures 3.5 and 3.6 show the TLS handshake of the Cobalt Strike [33]. The TLS version and the agreed encryption version already conform to standards. The traffic is suspicious based only on the supported ciphers and the name server certificate. Some of the ciphers offered by the client, such as RC4 in combination with MD5, are no longer considered secure and should not occur as part of the legit traffic. The name of the server certificate issuer called `mitmproxy`, is also suspicious.

Figure 3.5: Cobalt Strike Client Hello Message.

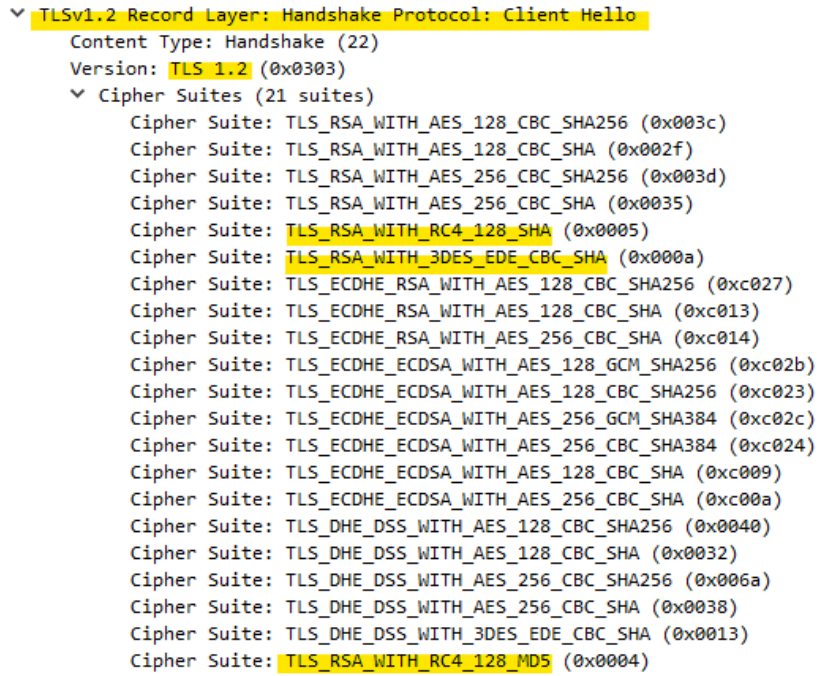
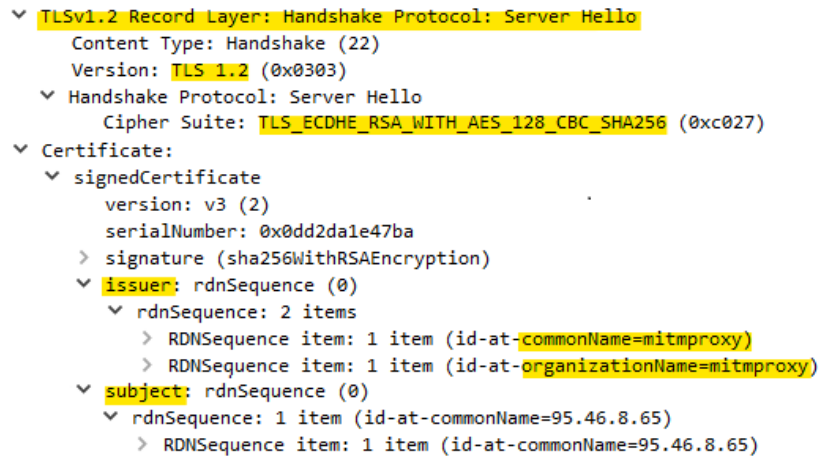


Figure 3.6: Cobalt Strike Server Hello Message.



4 Encrypted Traffic Classification Methods

The analysis of encrypted traffic has evolved in three basic directions. This chapter discusses and compares all three methods. Most attention is given to the method using fingerprints since this method is the basis of this thesis.

4.1 Deep Packet Inspection and Behavior-based Methods

The Deep Packet Inspection method is the most accurate but also the most challenging to implement. It is based on using a proxy or a firewall that decrypts all encrypted traffic, inspects it, and decides whether the traffic is benign or not. This method is effectively the Man-in-the-Middle attack, and as such, there are several problems with it. The first is the performance issue. Today's amount of encrypted traffic is so large that a considerable amount of computing power would be required to deploy this solution in a more extensive network.

Another problem is versatility. For end stations to communicate through the firewall and for the device to decrypt this communication, it is necessary to distribute firewall or proxy certificates as trusted to all end devices. Lastly, privacy is also an issue. The employer or network service provider usually does not have the legal ability to decrypt the communications of its employees or clients. The deployment of this solution usually needs to be covered by some exceptions or contract [4].

Behaviour-based or Feature-based methods is a set of methods that aim to avoid the need to decrypt traffic and even have knowledge of the protocol being used. However, the price for this solution is reduced accuracy. These methods use communication patterns to classify traffic. These patterns can be observed without the need to interfere with the traffic in any way or even decrypt it. This makes these methods far more applicable and can be deployed for arbitrary traffic capture. The patterns are calculated from traffic characteristics such as the total number of packets or bytes sent or the average packet interval time [5]. Then, it is possible to classify encrypted traffic based on the similarity of these patterns. However, there are two problems. The first is that even legitimate traffic may resemble malware-generated traffic in its

patterning and thus be misclassified. The second is the exact opposite. Malware can interact with its C2 server so well that it perfectly resembles normal traffic. This is especially the case when using covert channels such as Google Docs to deliver malware commands in the case of the Lampion banking trojan or covert channels like injecting malicious traffic to the DNS or misuse of the IP header fields such as `Traffic Class` [2, 34]. Such traffic using these cloaking methods is almost impossible to classify as malicious using the Behavior-based detection methods.

4.2 Fingerprinting Methods

Fingerprinting methods combine the two approaches introduced in Section 4.1. On the one hand, traffic inspection is based on the inspection of specific packets in the flow and thus should be more accurate than the Behavior-based approach. Nevertheless, on the other hand, only a few packets from each flow are examined, and these packets are not encrypted, eliminating the relatively expensive need to decrypt and inspect these packets. Yet, it is much faster and easier to implement compared to the DPI method. However, the amount of information obtained about each connection is not as large, and this method is not as accurate. The advantage over Feature-based methods is the ability to classify traffic immediately. In contrast, in the Feature-based method, the classification is made based on more extended term observation of the characteristics of each flow. The disadvantage is the need for knowledge of the structure of the investigated protocol, on which this method depends.

The information for fingerprinting SSL/TLS connections is obtained from the messages exchanged between the communicating parties at the beginning of the encrypted session, the so-called TLS handshake, described in Section 2.1.1. Since the secure connection is not yet established during the handshake, these messages are not encrypted and are thus sent in plaintext. Research in recent years has confirmed the differences between benign and malicious traffic in how communicating parties establish connections. An interesting observation is that malicious traffic tends to use weaker ciphers and TLS versions than benign traffic. An explanation could be that encrypted

traffic used by malware is because of camouflage the traffic, not because of protecting the traffic [35, 36]. So it does not matter on the used encryption algorithm so much. The specific methods listed in the next part differ only in what information they extract from the handshake and how they handle it. Another important feature of fingerprints is the way they handle the GREASE values. GREASE values are random values added to the TLS handshake and are defined in RFC 8701 [37]. The purpose of GREASE is to detect a bad implementation of the TLS protocol since the server or client should ignore unknown values during the handshake.

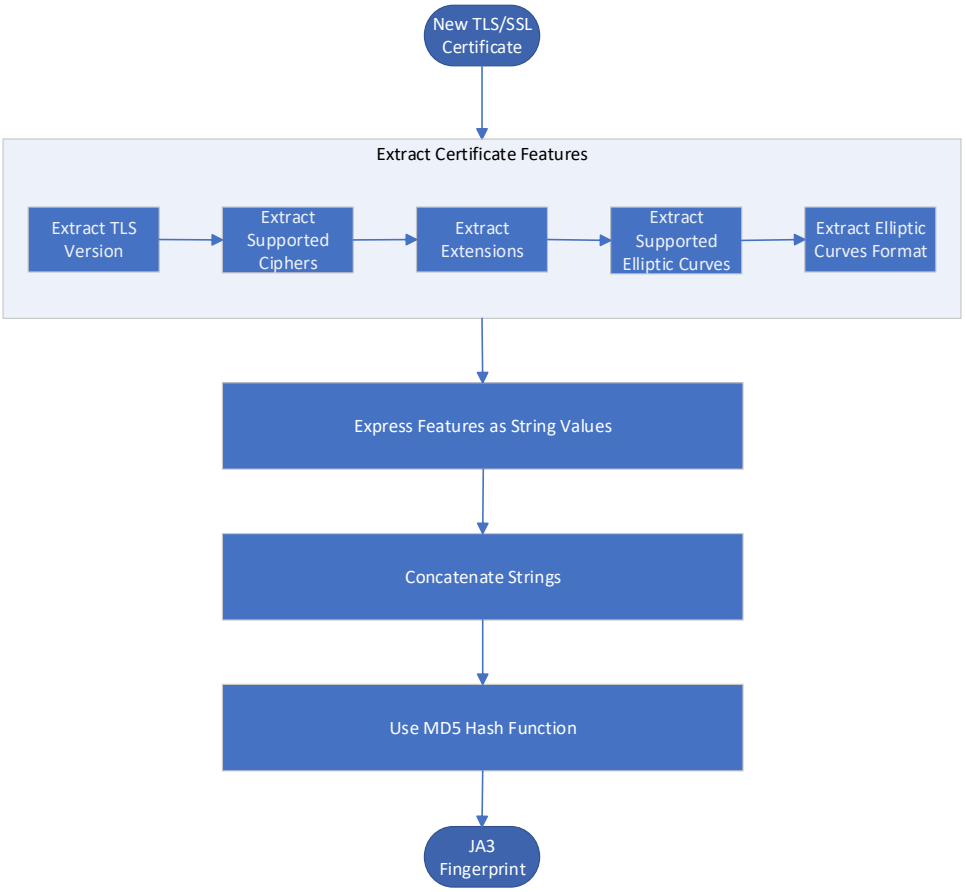
4.2.1 JA3, JA3s and JARM

JA3, the version of the JA3 for the server fingerprinting JA3s, and the active version of JA3 – JARM are three fingerprinting methods developed by John Althouse and his team in 2017 [6, 38, 39, 40]. The JA3 fingerprint is created from data obtained from the `ClientHello` Message when establishing an encrypted TLS connection. The specific values extracted from this message are the TLS version, Supported Ciphersuits, List of Extensions, Elliptic Curves, and Elliptic Curve Formats. Figure 4.1 shows the process of creating the JA3 fingerprint. Values from the specified fields are represented as strings and then hashed using the MD5 hash function. The MD5 hash is also the resulting JA3 fingerprint.

The same principle is applied in the case of JA3s. The difference is that the data is retrieved from the `ServerHello` message. Since this message has a different structure compared to `ClientHello`, the information obtained from this message is also different. It contains only the TLS version, the chosen ciphersuit and the extensions. The rest of the JA3s fingerprint creation is identical to the JA3 fingerprint creation.

JARM is a tool based on active server fingerprinting. JARM repeatedly sends `ClientHello` messages to the server and monitors the server's responses. Then, it hashes selected attributes from the `ServerHello` messages together with encoded server responses into the resulting JARM fingerprint. These JARM fingerprints are unique enough to be used to identify malicious servers on the Internet.

Figure 4.1: JA3 Fingerprint Creation Process.



4.2.2 Mercury

Mercury is a relatively new tool developed by Balke Anderson of Cisco in 2020 [41]. The core principle of Mercury is the same as JA3, namely the analysis of `ClientHello` messages. However, unlike JA3 fingerprints, Mercury does not use a hashing function to create a fingerprint and contains much more detailed information about the `ClientHello` message. The main difference between JA3 and Mercury is the addition of the Extension enumeration, which is not included in the original JA3 fingerprint.

The entire format of the fingerprint is as follows.

```
(version)(cipher suites)((extension0)(extension1)...)
```

The resulting fingerprint consists of a hexadecimal representation of each field. In addition, GREASE values are preserved, and they are just normalized to `0x0a0a` form.

5 Design

Recognizing encrypted malicious traffic is an area that has been researched for many years and uses the methods described in Chapter 4. Each of these methods has its advantages and disadvantages. One of the main disadvantages of the fingerprint-based method is its low ability to detect unknown and emerging 0-days threats. Each fingerprint is compared against a database of known benign or malignant fingerprints. In the case of a match, such traffic is classified based on the application associated with the fingerprint. The problem arises when the fingerprint is not found in the database. This is where the abilities of the fingerprint-based methods end. The research in this paper attempts to address this limitation. Section 5.1 presents the overall motivation and description of the method. Sections 5.2 and 5.3 describe the theoretical background of the clustering problem. The last section 5.4 covers the cluster set comparison problem.

5.1 Fingerprints Cluster Comparison Method

The core of this work is to verify the ability to detect unsolicited encrypted traffic based on the (dis)similarity of JA3 fingerprints with the clean traffic and to analyze whether and what differences exist between the establishment of benign and malignant encrypted communication. As shown in Section 3.2, these communications exhibit some differences. However, the question remains whether these differences can be observed in general and the JA3 fingerprint of the malignant communication can be correctly detected with some probability based on its (dis)similarity to another, already known communication.

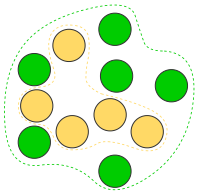
Therefore, the method proposed in this paper consists of two main parts. The first part uses the selected clustering methods to analyze and cluster JA3 fingerprints. The second part compares the distribution between clusters in the case of clean and malware traffic captures. The idea of this research is that if the cluster distribution for clean traffic is known, then any other captured traffic should be somewhat similar if it is also clean. Significant differences in cluster distribution can indicate the presence of a malware traffic. This research aims to verify

whether this detection method is applicable and which algorithm is best suited for this task. Accuracy and performance will be assessed.

5.1.1 Use Case

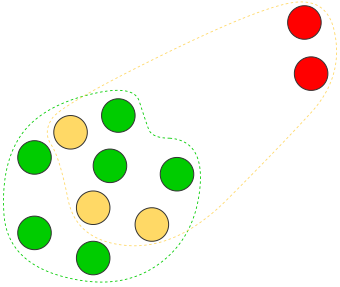
The illustration of the situation is shown in Figures 5.1 and 5.2. First, data that are expected for our application must be collected. The green dots represent the clusters of the JA3 fingerprint, which belong to the clean and expected communication. After that, the unknown traffic can be analyzed in the same way. Such traffic may contain a mix of clean and malware traffic. Orange dots in Figures below represents clean clusters, but it is not known for sure because they were captured from unknown traffic. The red dots represent clusters of malware. Therefore, if the captured traffic is also clean, it should look similar to the known clean traffic. This situation is depicted in Figure 5.1.

Figure 5.1: Clusters of Clean Traffic (Illustration).



However, if there is malware communication in the intercepted traffic, this traffic will probably deviate significantly from the known clean traffic, as shown in Figure 5.2.

Figure 5.2: Clusters of Mixed Traffic (Illustration).



The detection is based on comparing the structures of the set of JA3 fingerprint clusters with the known traffic. This method is even more universal, because it should be able to detect unexpected (and therefore malicious) behaviour.

5.2 Clustering

Clustering is the task of dividing input data into similar groups. The division is done in such a way that the points assigned to the same group are "more similar". Clustering can be divided into two basic groups [42].

- **Hard Clustering** – Each point is assigned to one group, i.e. for each group and each point, the given point belongs to the group or not.

- **Soft Clustering** – For each point and each group, a probability is assigned that the given point belongs to the group. Thus a point can belong to more groups with a certain probability.

There are a number of clustering algorithms. Their difference is mainly in the definition of the similarity between points. Each method thus behaves differently for different types of input data [43, 44].

- **Connectivity models** – Methods based on the observation that points which are close to each other in space are at the same time more similar. There are two approaches. The first starts with one large cluster, which is gradually divided into smaller ones. The second approach is exactly the opposite. Each point is initially in its own cluster, and gradually the clusters merge. These methods are also called hierarchical clustering. Its disadvantage is that it is not very effective on large datasets.

- **Centroid models** – The assignment of a point to a cluster depends on its distance from the cluster centre. The cluster centre is iteratively calculated until a local optimum is found. The disadvantage is the need to know the resulting number of clusters. Among the best-known representatives is the K-Means algorithm.

- **Distribution models** – The assignment to a cluster depends on the probability with which all data in the cluster are generated using the same distribution. An example is an Expectation-Maximization algorithm.
- **Density Models** – This method is based on searching for nodes with different point densities. The known algorithms are DBSCAN and OPTICS.

There are many specific methods, which differ in the number of parameters, the ability to scale for extensive input data and, of course, the metric used. From this, the most suitable use cases for each method emerge. The following subsections describe a few selected clustering algorithms used in this research.

5.2.1 CD-HIT

The CD-HIT method is a hierarchical clustering method used mainly in bioinformatics to cluster DNA sequences. Its great advantage is its simplicity and speed. The idea behind this algorithm is that two sequences can be in the same cluster if they share at least some of the features in the sequence. This minimum shared part is expressed as a percentage called similarity threshold. Thus, it is often sufficient to compare the lengths of two samples to determine whether two sequences can be in the same cluster. This avoids many unnecessary time-consuming comparisons, and in practice, this algorithm is very efficient. The whole algorithm works in two steps. In the first step, the input dataset is sorted by size. In the second step, the samples are processed in descending order. The processed sample is compared sequentially with the longest sample in the already formed clusters. If it exceeds the similarity threshold, it is assigned to the best matching cluster. Otherwise, a new cluster is created. All samples are processed in this way [45].

5.2.2 K-Medians

The K-Medians method is a clustering method that solves the partitioning of a dataset into k clusters. The core of each cluster is the so-called median string. The median string is the string out of all the possible

strings, which minimizes the sum of distances to all the strings of the set. However, the problem of finding such a string is NP-hard, and it is not possible to find it in a reasonable time. Thus, a suitable approximation of the string median is used in practice as the set median string of set T. The set median string of set T is always a point in set T and thus effectively reduces the size of the search space. Such a problem can be solved in polynomial time. The algorithm itself is greedy. Initially, k points are chosen randomly or according to some algorithm from the entire dataset. Then the remaining points are divided into clusters, and within these clusters, the set string medians are recomputed. The algorithm terminates when the process converges a local optimum [46].

5.2.3 OPTICS

Ordering Points To Identify the Clustering Structure (OPTICS) is a density-based clustering algorithm. It is a generalization of another well-known clustering algorithm – Density-based Spatial Clustering of Applications with Noise (DBSCAN). This algorithm arranges the samples so that the two closest samples are immediately after each other and creates a so-called reachability graph. This graph allows for dynamically changing the relative distance threshold, thus determining the resulting clusters. The resulting clusters are determined by splitting the sequence at points where the relative distance exceeds a given threshold (also called an epsilon value). "Cutting" the reachability plot at a single value produces DBSCAN like results; all points above the "cut" are classified as noise, and each time there is a break when reading from left to right signifies a new cluster. This approach has the advantage over the similar DBSCAN algorithm of identifying clusters with different densities [47, 48].

The average run-time of OPTICS is nearly the same as in the case of DBSCAN. The searching for neighbours mainly influences the complexity. In the worst case, the complexity is $\Theta(n^2)$, where n is the number of samples but can be lowered using different data structures [44, 47].

5.3 Metric Space

All clustering algorithms rely on measuring and quantifying the distance between two samples. The concept of metric space and metrics is used for this purpose. A metric ρ is a mapping from $\mathcal{M}^2 \to \mathbb{R}$, where $\mathcal{M}$ is any non-empty set. The metric must satisfy the fundamental axioms for arbitrary $x, y, z \in \mathcal{M}$.

- **Identity**:
$$\rho(x, y) = 0 \iff x = y \tag{5.1}$$
- **Symmetry**:
$$\rho(x, y) = \rho(y, x) \tag{5.2}$$
- **Triangle Inequality**:
$$\rho(x, z) \geq \rho(x, y) + \rho(y, z) \tag{5.3}$$

There are standard metrics for different sets $\mathcal{M}$.

- **Real Metrics**
  - **Euclidean Metric** – Probably the most famous metric that is very often used. It is defined over real sets and expresses the length of the line segment between two points. The distance d of 2 points p and q in n-dimensional space is defined as follows:
$$d(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \tag{5.4}$$
  - **Manhattan Metric** – Also called a Taxicab Metric. Distance d of two points p, q is defined as the absolute difference of their Cartesian coordinates. The distance is defined as:
$$d = \sum_{i=1}^{n} |p_i - q_i| \tag{5.5}$$

- **Discrete Metrics**
  - **Levenshtein Metric** – Also called edit distance. It is determined by the minimum number of operations that must be performed to change one string to another. Acceptable operations are deleting a character, adding a character, or replacing one character with another. Levenshtein distance lev between two strings a, b of length $|a|, |b|$ is given by:

$$lev(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ lev(tail(a), tail(b)) & \text{if } a[0] = b[0], \\ 1 + min \begin{cases} lev(tail(a), B) \\ lev(a, tail(b)) \\ lev(tail(a), tail(b)) \end{cases} & \text{otherwise.} \end{cases} \tag{5.6}$$

    where $a[0]$ resp. $b[0]$ stands for the first character of a resp. b and $tail$ is a function returning the string without the first character.

  - **Damerau-Levenshtein Metric** – It is a modified version of the Levenshtein algorithm. It admits the same operations and adds the possibility to swap two consecutive characters. It is possible to normalize this metric with respect to the length of the string. The result is normalized to the interval [0,1]. The Damerau–Levenshtein distance between two strings a and b is a function $d_{a,b}(i,j)$ whose value is a distance between an i-symbol prefix of string a a j-symbol prefix of b. The definition is as follows:

$$d_{a,b}(i,j) = min \begin{cases} 0 & \text{if } i = j = 0, \\ d_{a,b}(i, j-1) + 1 & \text{if } i > j, \\ d_{a,b}(i-1, j-1) + 1 & \text{if } i, j \neq 0, \\ d_{a,b}(i-2, j-2) + 1 & \text{if } i, j > 1 \text{ and } a_i = b_{j-1} \text{ and } a_{i-1} = b_j \end{cases} \tag{5.7}$$

5.3.1 Common Similarity

This metric is used specifically in the CD-HIT clustering algorithm. For two sets of discrete values s_1 and s_2, their similarity is defined as follows:

$$similarity = 1 - \frac{|s_1 - s_2|}{length\,of\,s_1} \tag{5.8}$$

where s_1 is the longer sequence or the greater set.

In the case of JA3 sequence clustering, this metric was used separately for each section of the JA3 fingerprint due to duplicate values with different meanings occurring in other sections of the JA3 fingerprint.

5.4 Classification

The analysis and classification of the set of clusters is the second part of the whole process. The method is based on the idea that the JA3 fingerprint cluster space will be different from the clean traffic in the case of malware traffic. Since there is no metric space to place these clusters, this problem must be viewed slightly differently. Therefore, the solution to this problem was not to compare cluster sets directly but to use the properties of the set of clusters to compare them. Three properties were selected. The first is the average distance of the clusters within the set. The second property is the maximum distance of two clusters within a set. The last metric is the maximum dissimilarity (in the sense of the function used in the CD-HIT algorithm described in Section 5.3.1). Neither of these properties guarantees the correctness of the results, but their combination can achieve reasonably good accuracy. In all cases, the normalized Damerau-Levenshtein distance is used to measure the distance between clusters. The distance is measured in four different modes. Cluster representatives distance, cluster medians distance or the closest or farthest cluster points distance.

6 Implementation

This section describes the implemented tool.

Section 6.1 describes the overall design of the tool. Section 6.2 describes the first module responsible for loading and parsing data. Section 6.3 describes the second module responsible for clustering JA3 fingerprints. Section 6.4 describes the last module responsible for classifying the clusters and comparing the traffic.

6.1 High-Level Design

The whole tool, whose high-level design is shown in Figure 6.1, consists of 3 main engines. The first one provides loading, parsing and processing of PCAP files. This engine uses a slightly modified version of the official JA3 parser [38]. The output of this engine is a structure storing individual JA3 fingerprints in text form and their associated destination IP addresses. This part can be replaced in the future by another data source that preserves the defined interface.

The second part is the clustering engine. This part is responsible for clustering the JA3 fingerprints supplied from the previous engine. To do this, it uses one of the implemented clustering algorithms. An overview of the algorithms implemented in this work is given in Section 5.2, and a more technical description is given in Section 6.3.

The last part is the classification engine. The classification engine is responsible for detecting samples and clusters that may originate from malware communication. It should be emphasized that the result of the whole process is not a sample classification. Just as the mere presence of a known malware JA3 fingerprint does not automatically imply the presence of malware, in this case, the engine only aims to highlight possible suspicious samples. A more detailed description is in Section 6.4.

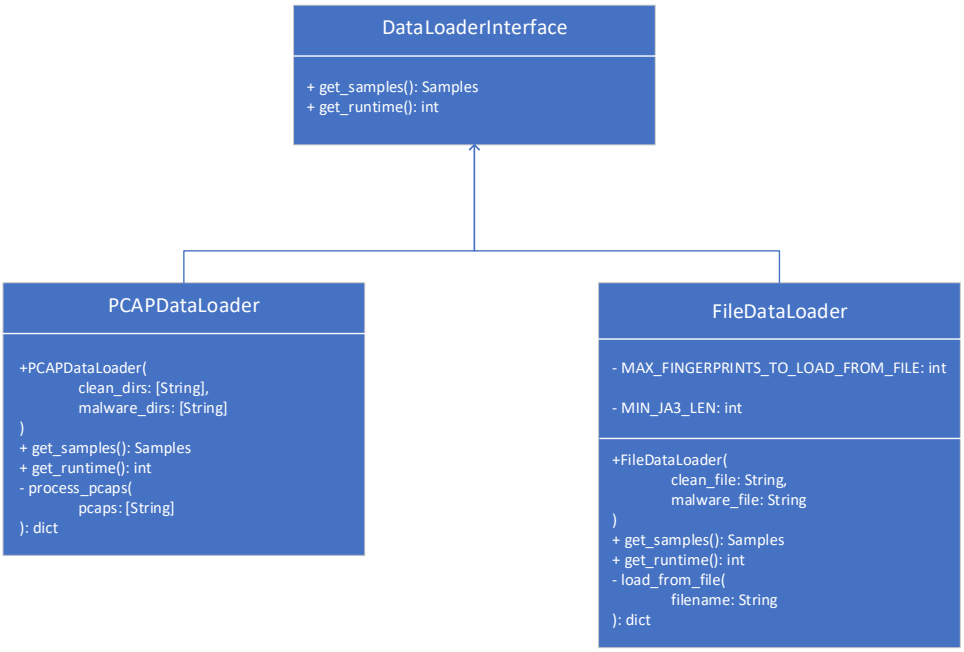
6.2 Data Loader Engine

The first part module is responsible for processing the data into a form accepted by the following engines. In this thesis, data from the

Figure 6.1: Design Overview.

PCAP file or precomputed JA3 prints were processed (to save time during testing). Therefore only the `FileDataLoader` and `PCAPDataLoader` classes were implemented. Both of them implement the DataLoader Interface. Thanks to this, other data sources could be added later. Figure 6.2 shows the UML Class Diagram of the Data Loader Engine.

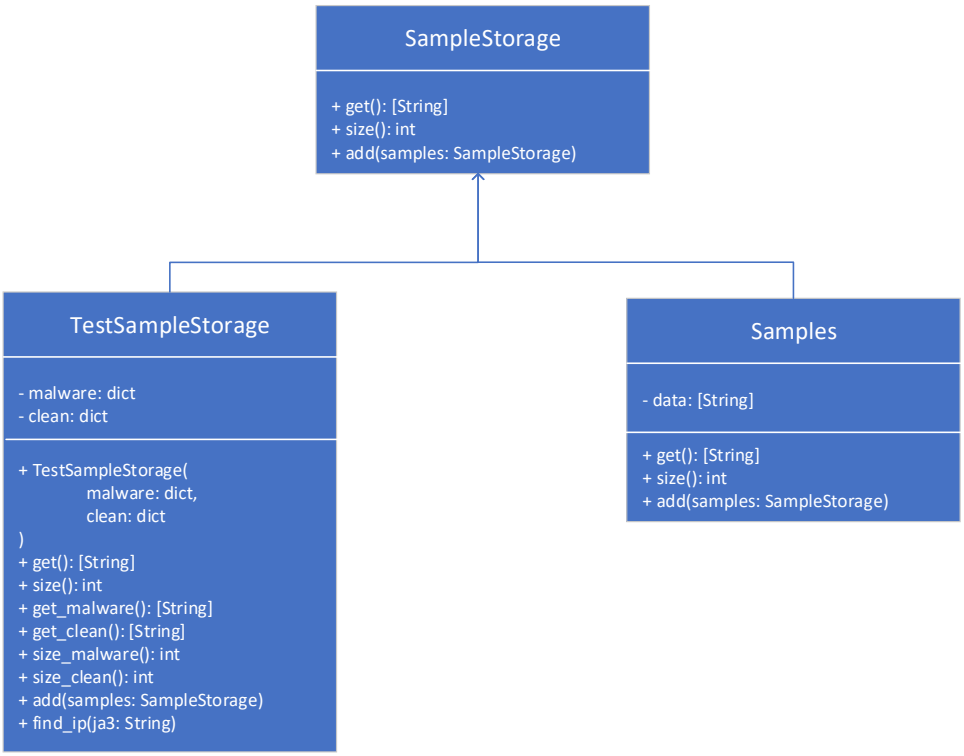
Figure 6.2: DataLoader Classes.



The `DataLoaderInterface` defines two methods. The first one is the `get_runtime` method. This method is used for performance monitoring and returns the data loading operation time. The second defined method is `get_samples`. This method returns an instance of the class implementing the `SampleStorage` interface. This is responsible for

storing the data. The `TestSampleStorage` class, which separates malware and clean samples, is used for testing purposes. However, in general, the `Samples` class can be used.

Figure 6.3: SampleStorage Classes.

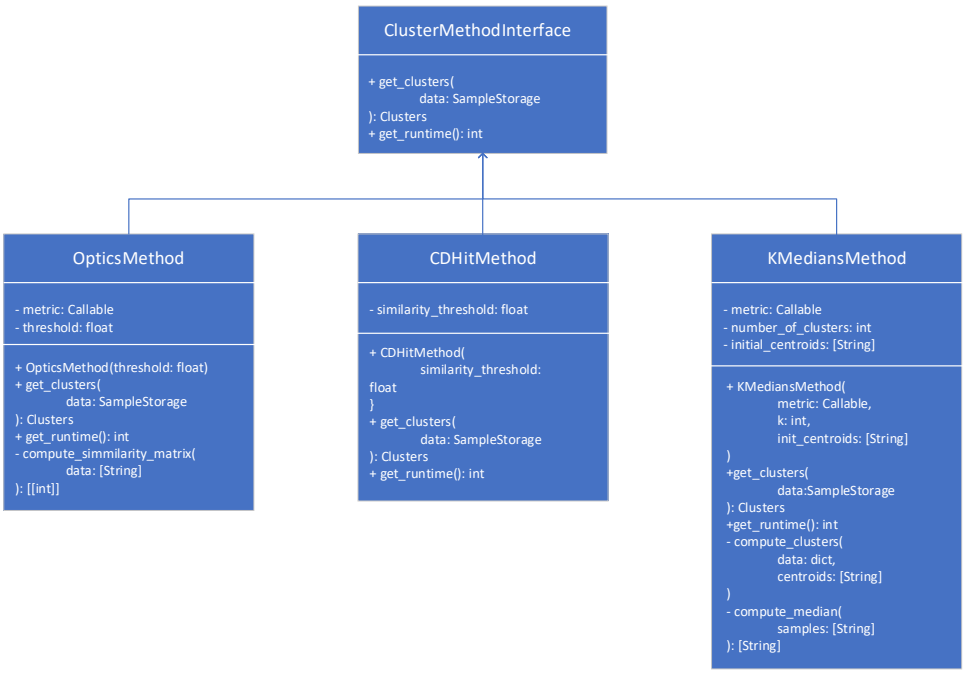


6.3 Clustering Engine

The second module of the tool ensures the clustering of JA3 fingerprints. The data is passed as an instance of a class implementing the `SampleStorage` interface. As part of this research, three types of clustering algorithms described in Section 5.2 were implemented. The design of these classes is shown in Figure 6.4. Each of the three clustering algorithms has its class implementing the common `Cluster Method Interface`.

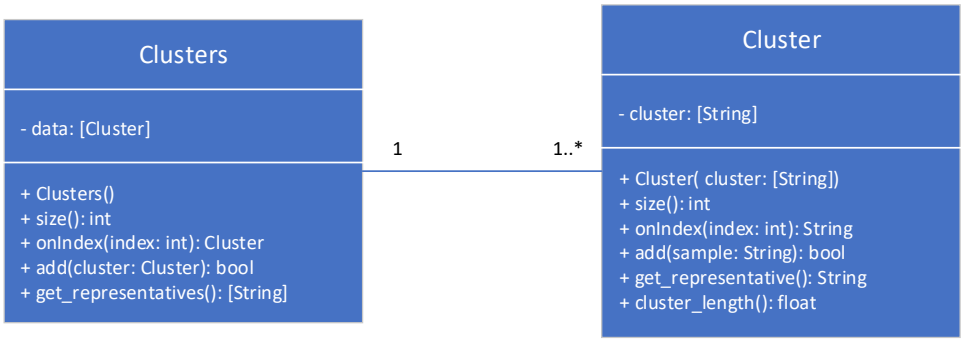
Two classes are used to store cluster data. The `Cluster` class is used to store data about one cluster. It also provides methods for cluster

Figure 6.4: ClusterMethod Classes.



analysis that are used in the classification engine. The second defined class, the `Clusters` class, is a wrapper for keeping all clusters together and for working with clusters in bulk. The UML Class Diagram is shown in Figure 6.5.

Figure 6.5: Cluster Data Storage Classes.



6.3.1 Implementation of the CD-HIT

The CD-HIT algorithm is used mainly in biomedicine for clustering protein sequences, and its description is in Section 5.2. Algorithm 1 shows the pseudocode of the CD-Hit algorithm. This algorithm takes two parameters as input. The *fingerprints* parameter represents the input sequences. The second parameter, *threshold*, specifies the percentage of elements two sequences must have in common to be in one cluster. The output of this function is the fingerprints grouped into clusters.

Algorithm 1: CD-HIT Clustering Algorithm.

```
   Input fingerprints – JA3 fingerprints
   Input threshold – A number between 0 and 1
   Output Clusters – Clustered fingerprints
 1: procedure CD-HIT
 2:     fingerprints ← sorted fingerprints
 3:     clusters ← new Clusters()
 4:     for each fingerprint in fingerprints do
 5:         max_similarity ← −∞
 6:         closest_cluster ← NULL
 7:         for each cluster in clusters do
 8:             representative ← representative of cluster
 9:             if fingerprint ≥ threshold ∗ representative then
10:                 similarity ← get_similarity(representative, fingerprint)
11:                 if similarity ≥ max_similarity then
12:                     max_similarity ← similarity
13:                     closest_cluster ← cluster
14:         if max_similarity > −∞ then
15:             closest_cluster.add(fingerprint)
16:         else
17:             new_cluster ← new Cluster()
18:             new_cluster.add(fingerprint)
19:             clusters.add(new_cluster)
20:     return clusters
```

At the beginning of this algorithm, the input sequences are sorted by size in descending order (line 2). On lines 4 – 13, the percentage of common elements with the representative of the already formed

clusters is calculated for each fingerprint. The cluster representative is the first assigned (and therefore the longest) sequence in the cluster. If this value exceeds the value given by the threshold parameter for at least one cluster, then the fingerprint is assigned to the cluster with the highest similarity (lines 14 – 15). Otherwise, a new cluster is created, and the fingerprint is the new representative of the cluster.

The acceleration of this algorithm happens on line 9. If the length of the fingerprint being compared is less than a threshold multiple of the length of the cluster representative, it is clear that the fingerprint does not belong to the cluster. In this way, many measurements are eliminated, making this algorithm very efficient.

6.3.2 Implementation of the K-Medians

The K-Medians algorithm for strings is an iterative clustering algorithm, and its detailed description is in Section 5.2. Finding the median for a given set of strings is an NP-hard problem, so an approximation called the set median problem is used in the implementation. This approximation admits as the median for a given group of strings only strings from the given group. This significantly reduces the search space. However, it also lowers the accuracy. Algorithm 2 shows the pseudocode of the implemented K-Medians algorithm for strings.

Algorithm 2: K-Medians Clustering Algorithm.

```
Input fingerprints – JA3 fingerprints
Input k – A number of clusters
Output Clusters – Clustered fingerprints
 1: procedure K-Medians
 2:     centroids ← k random samples from fingerprints
 3:     changing ← TRUE
 4:     clusters ← assign data to clusters given by centroids
 5:     while changing do
 6:         new_centroids ← []
 7:         for each cluster in clusters do
 8:             new_median ← compute median for cluster
 9:             new_centroids ← new_centroids + new_median
10:         new_clusters ← assign data to clusters given by new_centroids
11:         if new_clusters ≠ clusters then
```

```
12:             clusters ← new_clusters
13:         else
14:             changing ← FALSE
15:     return clusters
```

At the beginning of the algorithm on line 2, random centroids are selected from the input data. Then, on line 4, the data is assigned to a cluster based on the closest centroid. The loop on lines 5 – 14 is executed iteratively until a local optimum is found. On line 8, a new centroid is calculated for each cluster, and the data is reassigned to the clusters based on the new centroids. This loop is repeated until a local optimum is found.

6.3.3 Implementation of the OPTICS

The OPTICS algorithm was implemented using the python library `scikit`, which implements this algorithm. The input of this algorithm is data and a metric function or table of pre-calculated measurements. The pseudocode implementation of this function is shown in Algorithm 3.

Algorithm 3: OPTICS Clustering Algorithm.

```
Input fingerprints – JA3 fingerprints
Input min_samples – The number of samples in a neighborhood ( See 5.2)
Input max_eps – The maximum distance between two samples ( See 5.2)
Input threshold – Used for separating clusters ( See 5.2)
Output Clusters – Clustered fingerprints
1: procedure OPTICS
2:     similarity_matrix ← distance between every pair JA3 fingerprints
3:     optics ← OPTICS(min_samples, max_eps)
4:     optics.fit(similarity_matrix)
5:     clusters ← separate clusters based on the threshold from optics
6:     return clusters
```

The first step of this algorithm is to calculate the similarity matrix. This matrix contains the distance between all pairs of prints. The OPTICS algorithm implementation from the `scikit` library processes this matrix using the *max_eps* and *min_samples* parameters, whose meaning is described in chapter 5.2 [44].

6.3.4 Implemented Metrics

This thesis implements three clustering algorithms OPTICS, K-Medians for string, and CD-HIT. Each of these algorithms works on a slightly different principle. OPTICS and K-Medians algorithms use the Damerau-Levenshtein distance to measure the distance of two fingerprints. This metric is described in section 5.3. A normalized version of this distance is used here with respect to the length of the individual JA3 sections of the fingerprint. The normalized Damerau-Levenshtein distance is calculated for each section of the JA3 fingerprint separately. The values from all sections of the JA3 fingerprint are then summed together. The code snippet used to calculate the normalized Damerau-Levenshtein distance between two fingerprints is shown in Listing 6.1.

Listing 6.1: Normalized Distance Computation.

```
1  @staticmethod
2  def normalized_distance(fingerprint1: str, fingerprint2: str) -> float:
3      weights = []
4      f1 = JA3Methods.Parser.parse_fingerprint(fingerprint1)
5      f2 = JA3Methods.Parser.parse_fingerprint(fingerprint2)
6
7      for part in range(JA3Methods.JA3_PARTS):
8          weights.append(1/max(len(f1[part]), len(f2[part])))
9      return JA3Methods.Metrics.__weighted_distance(fingerprint1,
10     fingerprint2, weights=weights)
```

In the first step, JA3 fingerprints are parsed on lines 4 and 5. Then, the normalized distance is calculated as the weighted distance. On line 8, weights are set for each part of the JA3 fingerprint separately. They are calculated as 1 divided by the length of the JA3 part.

On line 9, the result is returned as the weighted distance of two fingerprints. The weighted distance is computed in two steps. In the first step, the Damerau-Levenshtein distance between individual parts of the JA3 fingerprint is computed. Then the results are multiplied by weights and summed together.

The last-mentioned algorithm called CD-Hit uses a slightly different method to define the similarity of two fingerprints. This function

is implemented as a `common_similarity` function. It compares two fingerprints and checks if the overlap in all features of the JA3 fingerprint is above the threshold. The code snippet showing the implementation of this function is shown on Listing 6.2.

Listing 6.2: Common Similarity Computation for the CD-HIT.

```
1  @staticmethod
2  def common_similarity(fingerprint1: str, fingerprint2: str) -> float:
3      if JA3Methods.Metrics.length(fingerprint1) <
4      JA3Methods.Metrics.length(fingerprint2):
5          fingerprint1, fingerprint2 = fingerprint2, fingerprint1
6
7      f1 = JA3Methods.Parser.parse_fingerprint(fingerprint1)
8      f2 = JA3Methods.Parser.parse_fingerprint(fingerprint2)
9      common = 0
10
11     for part in range(JA3Methods.JA3_PARTS):
12         for value_f1 in f1[part]:
13             for value_f2 in f2[part]:
14                 if value_f1 == value_f2:
15                     common += 1
16     return common/JA3Methods.Metrics.length(fingerprint1)
```

Two fingerprints are swapped on lines 3 – 5, so the longer one is always *fingerprint1*. After that, the fingerprints are parsed on lines 7 – 8. In the loop on lines 11 – 15, the method calculates the number of common values. The result is then divided by the length of the fingerprint, giving the resulting percentage of the common values.

6.4 Classification Engine

The clustering engine is the last module that classifies intercepted traffic and determines whether or not it contains malware. The description of this method is in Section 5.1. However, unlike the illustrations in this section, the problem that arises in implementation is how to define the area of the cluster. The nature of this problem makes it impossible

for us to use traditional cluster comparison methods, such as the hull comparison method [49]. Only the distance between the two clusters can be measured but not the direction.

This problem was solved by comparing the average distances between clusters. Since the assumption is that malware samples from the site are significantly different, such traffic will likely have a greater average distance between clusters.

Three methods were implemented to measure the distance between clusters. The first method defined the distance between two clusters as the (normalized Damerau-Levenshtein) distance of their representative. This method was used exclusively for clusters created by the CD-HIT algorithm, which defines a cluster representative as the longest sequence that occurs in it. The other two methods defined the distance between clusters as the distance of the nearest or outermost sequences, respectively. The last method defined the distance of the clusters as the distance of their medians.

Algorithm 4: Classification.

Input *clean_clusters* – Known clean clusters
Input *unknown_clusters* – Clusters of the unknown traffic
Output $Difference$ – Difference between average distances of clusters

1: **procedure** Classification
2: *clean_average_distance* ← average distance of *clean_clusters*
3: *unknown_average_distance* ← average distance of *unknown_clusters*
4: **return** $\frac{clean\_average\_distance - unknown\_average\_distance}{clean\_average\_distance}$

Therefore, the result is a relative average difference between clusters of clean and unknown traffic. Then it depends only on a limit, from which the traffic is considered suspicious.

7 Testing and Results

This chapter describes the tests and their results. Section 7.1 describes the data used for the testing. Section 7.2 describes the results regarding the time performance. The last Section 7.3 presents the results of the testing regarding the ability to detect malicious network traffic behavior.

7.1 Dataset

This research used publicly available sources for testing purposes. Data sources were not from the production network for several reasons. Firstly, there was no need to anonymize the data, which would have been necessary when using data from a production network. Based on this data, it was also straightforward to distinguish between malware-generated traffic and clean traffic because it was possible to analyze PCAP files directly, or the source indicated the nature of the traffic. The following paragraphs describe the datasets used in this research.

The dataset consists of PCAP files available from public databases, specifically the Stratosphere IPS and Malware Traffic Analysis projects [50, 51]. The Stratosphere project offers PCAP files of clean, mixed and malware traffic. The disadvantage is that many samples are pretty old, especially the malware samples, thus do not contain encrypted traffic so much. On the other hand, it offers a large number of clean and mixed PCAP files that have been used for testing. PCAP files from the Malware Traffic Analysis project were used as malware samples.

Four malware families were used for testing. Specifically, these were PCAP files of the Dridex, Emotet, QakBot and IcedID families. All captures were from the Malware Traffic Analysis project [51].

7.2 Performance

The performance of each clustering algorithm is crucial for practical applications. Since clustering is only one part of the whole process and can take place over different data sizes, the algorithm must be fast. The three implemented algorithms were compared in processing time

Table 7.1: Runtime of Clustering Algorithms comparison between Number of Samples (NoS) and Used Algorithm (Algo).

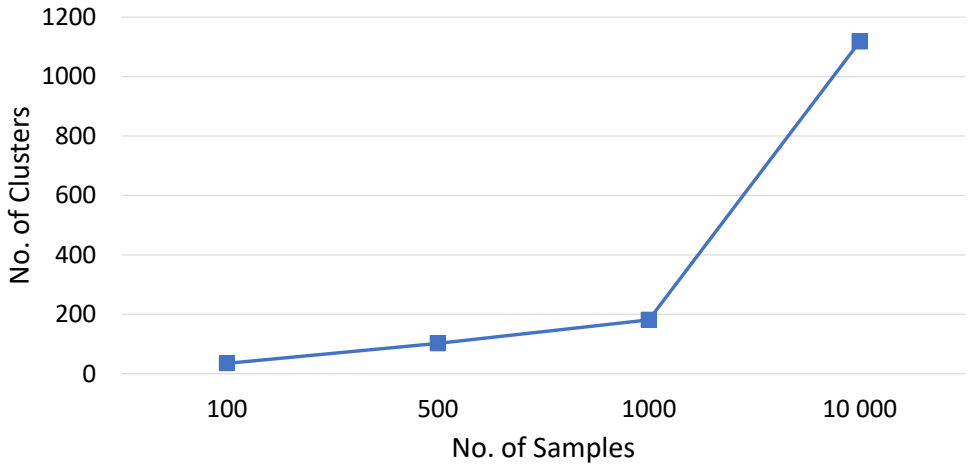
	Algorithm			
	X	CD-Hit	OPTICS	K-Medians
#JA3	100	0.06 sec	12.11 sec	90.6 sec
	500	0.98 sec	4.83 min	29.02 min
	1000	3.53 sec	19.31 min	-
	10 000	4.31 min	-	-

for different data sizes. The CD-Hit algorithm proved to be the best in the practical tests. This result is expected since both the OPTICS algorithm and the K-Medians algorithm have a quadratic complexity concerning the number of sequences compared. The K-Medians problem is even NP-Hard, so the approximation described in Section 5.2 was performed here. However, even with this approximation, the complexity is still quadratic. The CD Hit algorithm achieves a considerable speedup because many distance measurements between pairs need not be made at all since the difference in their lengths is too large. The results of the measurements for different Numbers of Samples (NoS) and various algorithms are given in Table 7.1. It is evident that the CD-Hit algorithm was able to cluster even 10 000 samples in a reasonable time, while the K-Medians algorithm was practically unusable even for quite a small amount of data. For this reason, only the CD-Hit algorithm is used as a clustering algorithm in the next part of the thesis.

The CD-HIT algorithm produced more and more clusters as the number of samples increased. Together with the increasing average cluster size, this fact caused the classification engine to slow down. Graph 7.1 shows the number of clusters created for different input data sizes with a similarity threshold set to 80 %.

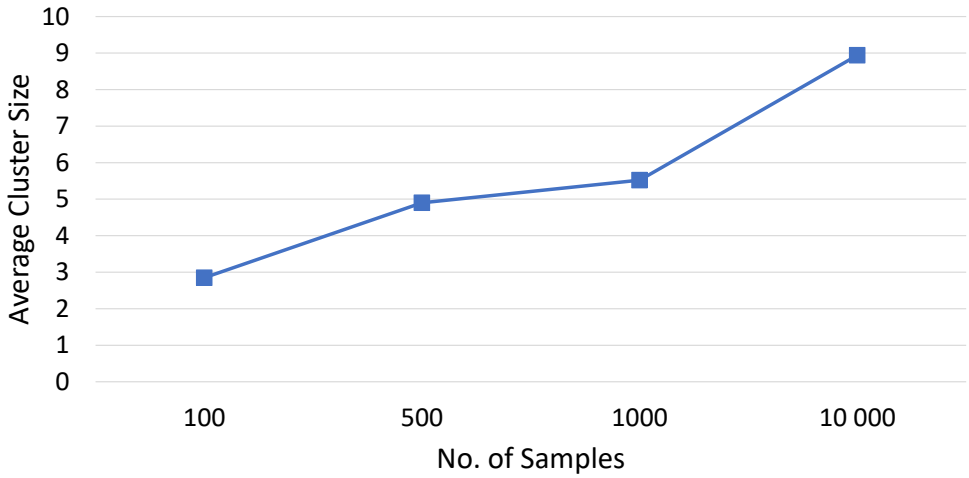
Graph 7.2 shows the average cluster size with the same setup. Both values increase with an increasing number of input sequences, and for an input sequence length of 10 000, the CD-HIT algorithm generated

Figure 7.1: Number of Clusters.



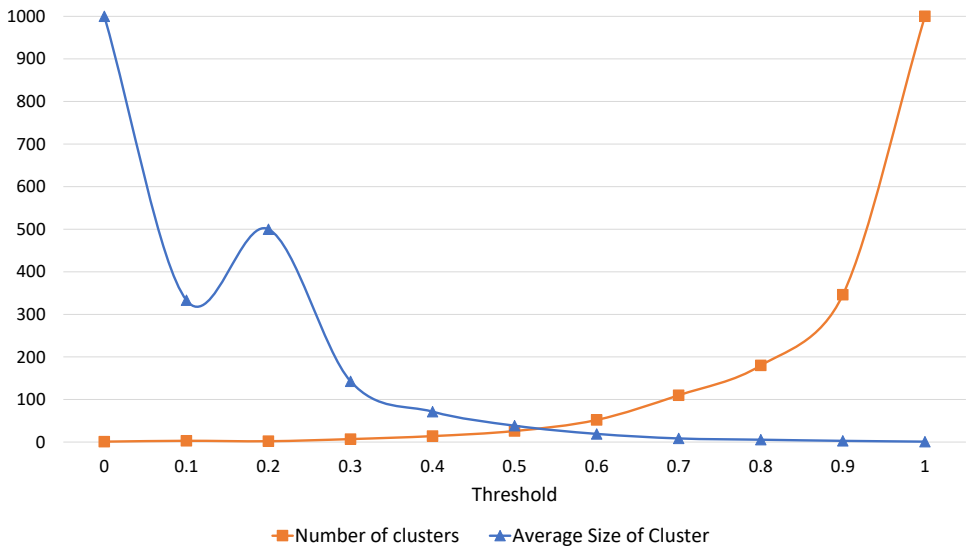
more than 1100 clusters. This fact is not surprising. However, it affects the performance of the classification engine.

Figure 7.2: Average Cluster Size.



The result of the clustering algorithm depends on the value of the threshold representing the percentage of parts that two fingerprints must have in common to be in one cluster. Figure 7.3 shows a graph of the dependence of the number of clusters created and their average size on the threshold.

Figure 7.3: Number of Cluster / Cluster Size Trade-off.



The second component that makes up the overall complexity of the algorithm is the complexity of calculating the average cluster distance. Here, of course, it depends on the measurement method used. The results for different input data sizes are shown in Graph 7.4. As in the case of the clustering time measurement, data with sizes ranging from 100 to 10 000 JA3 fingerprints were used.

7.3 Accuracy

The results and tests confirmed the expectations. In the first step, samples from the clean and mixed traffic were clustered.Then the three properties of these clusters were measured. Figure 7.5 shows the number of clusters for different datasets.

We can see that the number of clusters for mixed traffic containing malware was a bit higher. This is caused by the fact that the mixed traffic contained malware as well as clean traffic. Because the malware sample created separate clusters, the resulting number was a bit higher than in the case of clean traffic.

In the second step, three features of the set were measured. The first was the average distance of the clusters. The distance was mea-

Figure 7.4: Average Inter-Cluster Distance Computation Time.

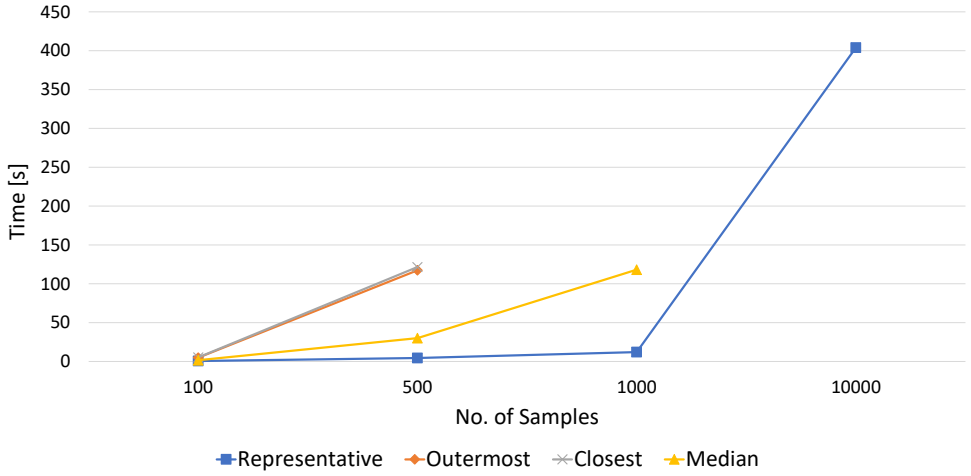
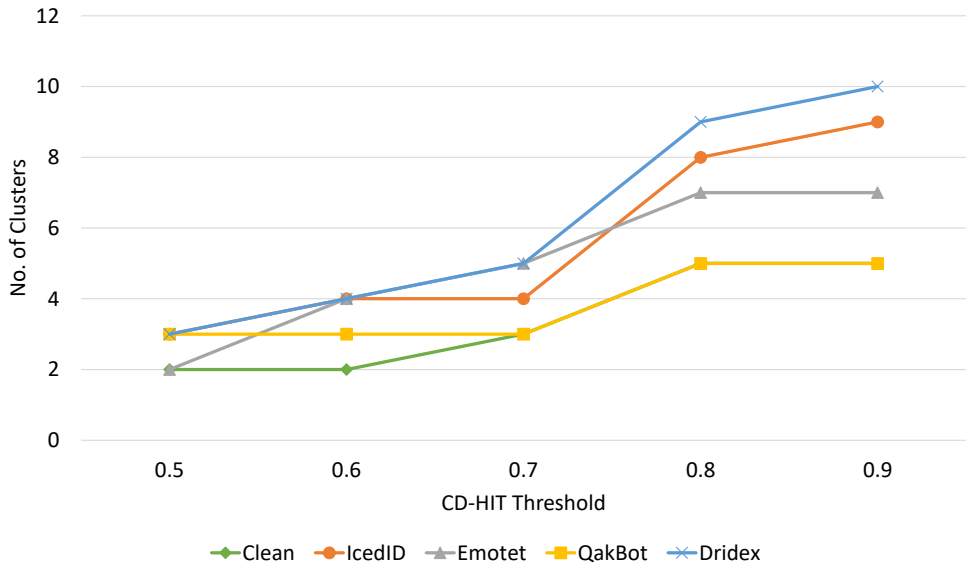


Figure 7.5: Number of Created Clusters.



sured by the normalized Damerau-Levenshtein method described in Section 6.3.4, and the distance of clusters is defined as the distance of the cluster's representatives. The measurement results are shown in Figure 7.6.

Figure 7.6: Average Inter-Cluster Distance.

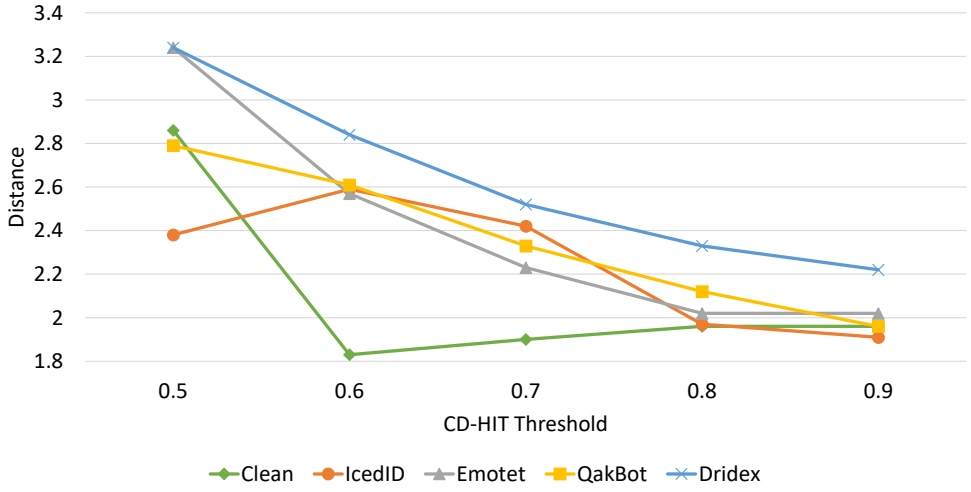


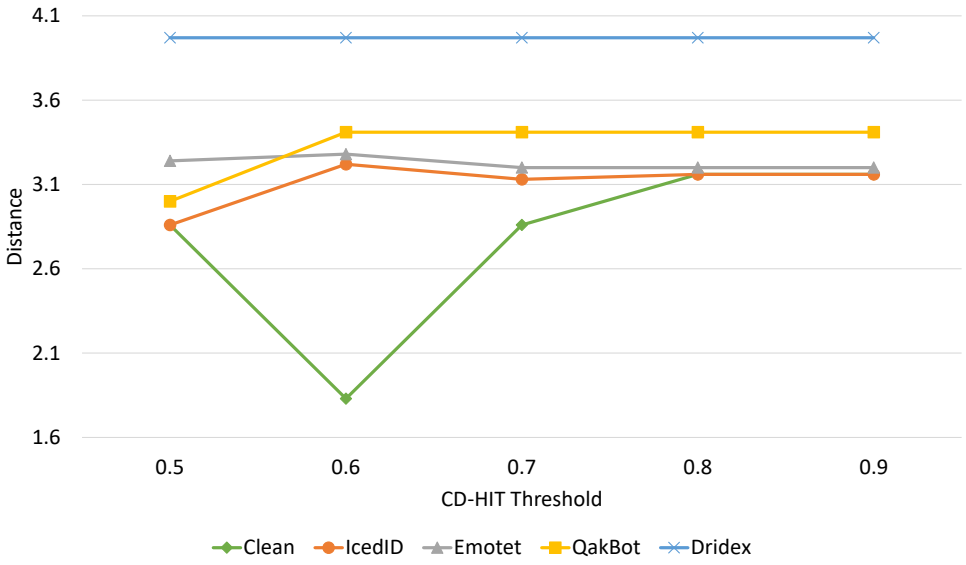
Figure 7.6 shows the average cluster distance for various types of traffic and different CD-HIT thresholds. In total, measurements were performed for each malware strain as well as for the clean traffic. In all cases, there is a visible difference between the average distance of the cluster in the case of clean and mixed traffic. The average distance between clusters is higher for mixed traffic in all four cases. It is confirmed that the mixed samples would have a larger average distance between clusters, even though the differences were less significant than expected. This may be because there are significantly more clusters with clean traffic than clusters with malware traffic, which distorts the results. A solution to this problem could be to weight the clusters in the measurement, with more distant clusters being given more weight. However, this has not been implemented in this work.

The second used method was to examine the distance of the two most distant clusters in the set. The measured values are shown in Figure 7.7.

This graph shows the distance of two outermost clusters in the set for different thresholds used by the CD-HIT algorithm. This test also confirmed the expectation that the distance of the two most distant clusters in mixed traffic would be greater than in clean traffic.

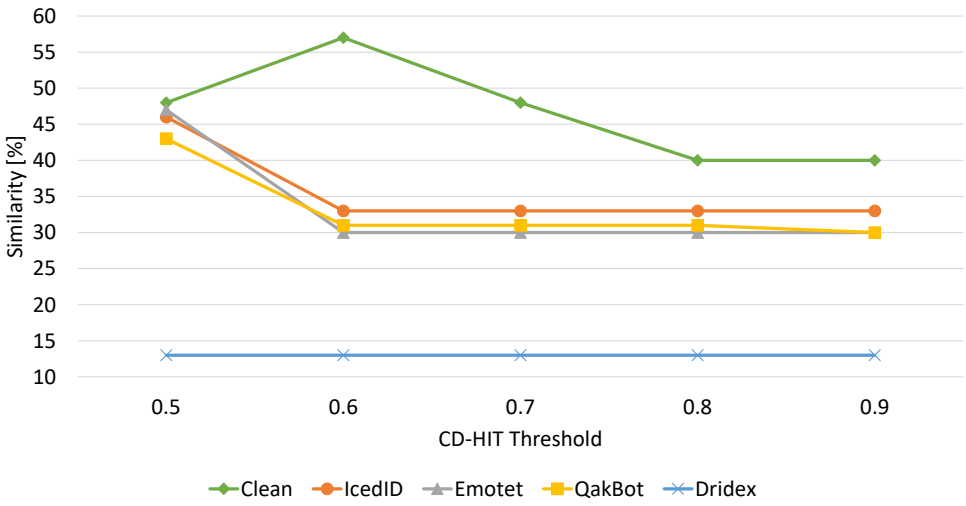
The last comparison method is based on the metrics used directly in the CD-HIT clustering algorithm. This metric represents the per-

Figure 7.7: Distance Between Outermost Clusters.



centage of matching elements of two sequences. If clean and mixed samples are compared, a rather significant difference can be seen. This difference is around 50 % in clean traffic, while in malware traffic, the two clusters differ by up to two-thirds of the values.

Figure 7.8: Minimal Similarity.



8 Conclusion

The importance of encrypted traffic on the Internet is growing, and it is almost inevitable that encrypted traffic will soon displace unencrypted traffic. However, what brings many benefits, such as ensuring data confidentiality or integrity, also brings the same benefits for the malware developers. Hiding traffic generated by malware becomes much easier and its detection more difficult.

However, the problem of detecting malware-generated traffic is not new and has been the subject of research for several years. The current methods are problematic because of their computational complexity, data confidentiality requirements or the detection based on known signatures.

One of the methods currently used is the JA3 fingerprinting method. Its main advantages are speed and simplicity, but detection is based on known signatures. This method is therefore not suitable for detecting new threats and malware strains.

The method proposed in this work tries to take advantage of the simplicity of the JA3 fingerprint and, at the same time, improve the ability to classify unknown traffic.

The method works in several steps. In the first step, JA3 fingerprints are collected first from traffic that is known to be clean and assumed to be legit for our system. Then the JA3 fingerprints from this traffic is grouped into clusters using a selected clustering algorithm and measure the properties of this set of clusters. Then the real unknown traffic data is collected and processed in the same way. If the traffic is legitimate, the properties of the set clusters from the unknown traffic should be approximately the same as the clean traffic. The measured properties include the average cluster distance, the maximum cluster distance or the minimum similarity between clusters.

This research implemented three clustering algorithms, namely the CD-HIT, OPTICS and K-Medians algorithms. These algorithms were tested in terms of performance. Here the CD-HIT algorithm clearly appears the best, which, also due to its simplicity, was used for the second part of the work.

In this part, the measurement of the properties of the set of clusters was implemented. Then the classification hypotheses were tested by

comparing the clean traffic and the mixed traffic with malware, namely containing the three Trojan horses strains.

The first hypothesis was that the average distance of a cluster in clean traffic should be lower than in mixed traffic. The second hypothesis was that the maximum distance between clusters should be more significant in the case of mixed traffic. The last assumption was that the minimum similarity between clusters should be smaller in the case of mixed traffic. All these hypotheses were confirmed, even though the differences in the average distances between clusters were not as significant as expected. This is very likely caused by the disproportion in the number of clusters with clean and malware traffic in the mixed data. Therefore, combining all three approaches seems to be a better option.

9 Discussion

This paper proposes a new method for detecting unsolicited traffic in the network based on the JA3 fingerprint. The method proposed and tested in this paper is entirely new, and at the moment, there is no publication on the same topic. For this reason, it is unfortunately not possible to compare the results independently.

This research have found a suitable clustering algorithm that is fast enough and produces good enough results in the tests. The CD-HIT algorithm, primarily used for clustering protein sequences in bioinformatics, proved to be suitable also for clustering JA3 fingerprint sequences. In addition, this algorithm also allows the simple influence of the results in terms of accuracy and computation complexity using a single parameter.

This research also confirmed the hypotheses about the different properties of mixed traffic and clean traffic and the possibility of using these differences to detect unsolicited traffic in a real network.

However, open and unresolved questions remain. The first is the lack of a multidimensional space for cluster comparison. This research have only worked with the ability to measure the distance between clusters but not the direction. The possibility of displaying the data in a space could increase the detection capabilities by considering the overlap of these spaces, for example.

Another way this work can be developed is by testing other clustering algorithms or measuring different properties of cluster groups than the three mentioned properties used in this work.

The last problem was the amount of test data. Data from publicly available databases were used for testing, and their quantity was not too large. However, obtaining a larger sample of data is quite problematic. The JA3 fingerprints themselves are not a problem, as they do not contain any personal data, and their acquisition is not complicated. However, verifying the accuracy of the method is problematic in this case because of results verification. It is necessary to decrypt the suspicious traffic and confirm whether it is unsolicited or not. The second way to get this data is to build your own network and simulate the whole process. However, the problem here is how to simulate the situation to reflect it in the real network.

A Attached files

ZIP archive containing the implementation of the tool used during the testing of the method.

Bibliography

1. *HTTPS encryption on the web — Google transparency report*. Google, 2021. Available also from: https://transparencyreport.google.com/https/overview.
2. GALLAGHER, Sean. Nearly half of malware now use TLS to conceal communications. *Sophos news*. 2021. Available also from: https://news.sophos.com/en-us/2021/04/21/nearly-half-of-malware-now-use-tls-to-conceal-communications/.
3. NAGY, Luca. Nearly a quarter of malware now communicates using TLS. *Sophos news*. 2020. Available also from: https://news.sophos.com/en-us/2020/02/18/nearly-a-quarter-of-malware-now-communicates-using-tls/.
4. VELAN, Petr; ČERMÁK, Milan; ČELEDA, Pavel; DRAŠAR, Martin. A survey of methods for encrypted traffic classification and analysis. *International Journal of Network Management*. 2015. Available also from: https://doi.org/10.1002/nem.1901.
5. MOORE, Andrew; ZUEV, Denis; CROGAN, Michael. Discriminators for Use in Flow-based Classification. *Queen Mary and Westfield College, Department of Computer Science*. 2005. ISSN 1470-5559. Available also from: https://qmro.qmul.ac.uk/xmlui/bitstream/handle/123456789/5050/RR-05-13.pdf.
6. ALTHOUSE, John. *TLS fingerprinting with JA3 and JA3S*. Salesforce Engineering, 2021. Available also from: https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967.
7. *Suricata Documentation*. 2017. Available also from: https://suricata.readthedocs.io/en/suricata-6.0.0/rules/ja3-keywords.html.
8. *Ja3 fingerprints Database*. 2022. Available also from: https://sslbl.abuse.ch/ja3-fingerprints/.
9. *JA3er*. 2022. Available also from: https://ja3er.com/.
10. PANDYA, Dwiti; THAKARE, BS; MADHEKAR, T; THAKKAR, S; RAM, K. Brief History of Encryption. *International Journal of Computer Applications*. 2015.

11. PRICHARD, Roger A. *Global Information Assurance Certification Paper*. 2002. Available also from: https://www.giac.org/paper/gsec/1555/history-encryption/102877.

12. GANCHEVA, Zlatina; SATTLER, Patrick; WÜSTRICH, Lars. TLS Fingerprinting Techniques. 2020. Available also from: https://doi.org/10.2313/NET-2020-04-1_04.

13. BHARGAVAN, Karthikeyan; LAVAUD, Antoine Delignat; FOURNET, Cédric; PIRONTI, Alfredo; STRUB, Pierre Yves. Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. 2014. Available also from: https://doi.org/10.1109/SP.2014.14.

14. *The heartbleed bug*. 2014. Available also from: https://heartbleed.com/.

15. DOWLING, Benjamin; FISCHLIN, Marc; GÜNTHER, Felix; STEBILA, Douglas. A cryptographic analysis of the TLS 1.3 handshake protocol. *Journal of Cryptology*. 2021. Available also from: https://doi.org/10.1007/s00145-021-09384-1.

16. MORIARTY, Kathleen; FARRELL, Stephen. Deprecating TLS 1.0 and TLS 1.1. *Internet Engineering Task Force, RFC*. 2021, vol. 8996. Available also from: https://datatracker.ietf.org/doc/html/rfc8996.

17. *SSL Pulse*. 2022. Available also from: https://www.ssllabs.com/ssl-pulse/.

18. RESCORLA, Eric et al. RFC 8446: The Transport Layer Security (TLS) protocol version 1.3. *Internet Engineering Task Force (IETF)*. 2018. Available also from: https://www.rfc-editor.org/rfc/rfc8446.txt.

19. MYERS, Michael; ADAMS, Carlisle; SOLO, Dave; KEMP, David. Internet X. 509 certificate request message format. *Request for Comments*. 1999, vol. 2511. Available also from: https://www.rfc-editor.org/rfc/pdfrfc/rfc2511.txt.pdf.

20. ALINA.BOLTON. *Maximum SSL/TLS certificate validity one year*. 2020. Available also from: https://www.globalsign.com/en/blog/maximum-ssltls-certificate-validity-now-one-year.

21. RESCORLA, Eric et al. RFC 5246 – The Transport Layer Security (TLS) protocol version 1.2. *The Internet Engineering Task Force (IETF)*. 2008. Available also from: https://datatracker.ietf.org/doc/html/rfc5246.

22. RESCORLA, Eric et al. *HTTP Over TLS*. RFC 2818, 2000. Available also from: https://www.hjp.at/doc/rfc/rfc2818.html.

23. FORD-HUTCHINSON, Paul et al. Securing FTP with TLS. *Request for Comments*. 2005, vol. 4217. Available also from: https://www.hjp.at/doc/rfc/rfc4217.html.

24. *Firefox DNS Over HTTPS*. 2022. Available also from: https://support.mozilla.org/en-US/kb/firefox-dns-over-https.

25. *OpenVPN Source Code*. 2022. Available also from: https://github.com/OpenVPN/openvpn3/.

26. CREBS, Brian. *Half of all phishing sites now have the Padlock*. 2018. Available also from: https://krebsonsecurity.com/2018/11/half-of-all-phishing-sites-now-have-the-padlock/.

27. *Encryption Is Now a Trojan Horse: Ignore It at Your Peril*. Fortinet, 2019. Available also from: https://www.fortinet.com/content/dam/fortinet/assets/white-papers/wp-Encrypt-Now-Trojan-Horse.pdf.

28. *SSL blacklist*. 2022. Available also from: https://sslbl.abuse.ch.

29. MOKBEL, Mohamad. *The state of SSL/TLS certificate usage in malware C&C Communication*. TrendMicro, 2021. Available also from: https://www.mfmokbel.com/wp-content/uploads/2021/09/ssl-tls-technical-brief.pdf.

30. *Dridex Malware*. CISA, 2019. Available also from: https://www.cisa.gov/uscert/ncas/alerts/aa19-339a.

31. *Dridex Malware Sample*. Stratosphere Project, 2016. Available also from: https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-153-1/.

32. *Cobalt strike: Adversary simulation and red team operations*. 2022. Available also from: https://www.cobaltstrike.com/.

33. *Cobalt Strike Malware Sample*. Stratosphere Project, 2018. Available also from: https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-345-1/.

34. CAVIGLIONE, Luca. Trends and Challenges in Network Covert Channels Countermeasures. *Applied Sciences*. 2021. Available also from: https://doi.org/%2010.3390/app11041641.

35. ANDERSON, Blake; PAUL, Subharthi; MCGREW, David A. Deciphering Malware's use of TLS (without Decryption). *CoRR*. 2016, vol. abs/1607.01639. Available from arXiv: 1607.01639.

36. *Zeus Source Code*. 2014. Available also from: https://github.com/Visgean/Zeus/.

37. BENJAMIN, David. *RFC 8701 Applying Generate Random Extensions And Sustain Extensibility (GREASE) to TLS Extensibility*. 2020. Tech. rep. Internet Engineering Task Force. Available also from: https://www.rfc-editor.org/rfc/rfc8701.pdf.

38. ALTHOUSE, John. *JA3 – A method for profiling SSL/TLS Clients*. 2019. Available also from: https://github.com/salesforce/ja3.

39. ALTHOUSE, John. *Easily identify malicious servers on the internet with Jarm*. Salesforce Engineering, 2021. Available also from: https://engineering.salesforce.com/easily-identify-malicious-servers-on-the-internet-with-jarm-e095edac525a.

40. ALTHOUSE, John. *JARM*. 2020. Available also from: https://github.com/salesforce/jarm.

41. DAVID MCGREW, Brandon Enright; ANDERSON., Blake. *Mercury: Fast TLS, TCP, and IP Fingerprinting*. 2020. Available also from: https://github.com/cisco/mercury.

42. *Clustering: Types of clustering: Clustering applications*. 2020. Available also from: https://www.analyticsvidhya.com/blog/2016/11/an-introduction-to-clustering-and-different-methods-of-clustering/#:~:text=Clustering%20is%20the%20task%20of,and%20assign%20them%20into%20clusters..

43. HUBERT, Lawrence; ARABIE, Phipps. Comparing partitions. *Journal of classification*. 1985. Available also from: https://link.springer.com/content/pdf/10.1007/BF01908075.pdf.

44. *Scikit – Clustering*. 2022. Available also from: https://scikit-learn.org/stable/modules/clustering.html#clustering.

45. LI, Weizhong; JAROSZEWSKI, L; GODZIK, Adam. Clustering of highly homologous sequences to reduce the size of large protein database. *Bioinformatics (Oxford, England)*. 2001. Available also from: https://www.researchgate.net/publication/12038873_Clustering_of_highly_homologous_sequences_to_reduce_the_size_of_large_protein_database.

46. MARTÍNEZ-HINAREJOS, Carlos-D; JUAN, Alfons; CASACUBERTA, Francisco. Generalized K-Medians Clustering for Strings. In: 2003, vol. 2652, pp. 502–509. Available from DOI: 10.1007/978-3-540-44871-6_59.

47. ANKERST, Mihael; BREUNIG, Markus M.; KRIEGEL, Hans-Peter; SANDER, Jörg. OPTICS: Ordering Points to Identify the Clustering Structure. In: *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, 1999. ISBN 1581130848. Available from DOI: 10.1145/304182.304187.

48. SCHUBERT, Erich; SANDER, Jörg; ESTER, Martin; KRIEGEL, Hans Peter; XU, Xiaowei. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. 2017, vol. 42, no. 3. ISSN 0362-5915. Available from DOI: 10.1145/3068335.

49. HERSHBERGER, J.; SHRIVASTAVA, Nisheeth; SURI, Subhash. Cluster Hull: A Technique for Summarizing Spatial Data Streams. In: 2006. Available from DOI: 10.1109/ICDE.2006.38.

50. STRATOSPHERE. *Stratosphere Laboratory Datasets*. 2015. Available also from: https://www.stratosphereips.org/datasets-overview.

51. *Malware Traffic Analysis*. 2022. Available also from: https://www.malware-traffic-analysis.net/.