Imperial College London

Department of Computing

Detecting Malware in TLS Traffic

Project Report

Author:
Olivier Roques

Supervisor:
Sergio Maffeis
Co-Supervisor:
Marco Cova

Submitted in partial fulfillment of the requirements for the MSc degree in Computing Science / Security and Reliability of Imperial College London

September 2019

Abstract

The use of encryption on the Internet has spread rapidly these last years, a trend encouraged by the growing concerns about online privacy. TLS (*Transport Layer Security*), the standard protocol for packet encryption, is now implemented by every major websites to protect users' messages, transactions and credentials. However cybercriminals have started to incorporate TLS into their activities. An increasing number of malware leverage TLS encryption to hide their communications and to exfiltrate data to their command server, effectively bypassing traditional detection platforms.

The goal of this project is to design and implement an effective alternative to the unpractical method of decrypting TLS packets' payload before looking for signs of malware activity. This work presents a highly accurate supervised classifier that can detect malicious TLS flows in a company's network traffic based on a set of features related to TLS, certificates and flow metadata. The classifier was trained on curated datasets of benign and malware observations, which were extracted from capture files thanks to a set of tools specially developed for this purpose.

We detail in this report the complete development process, from data collection and feature extraction to model selection and performance analysis.

Acknowledgments

I would like to particularly thank Marco Cova and Sergio Maffeis, my project supervisors, for their valuable and continuous suggestions and for their constructive feedbacks on this project.

I would also like to thank the members of the Network Analysis Team at Lastline: Filippo Anarratone, Luukas Larinkoski, Maurizzio Abba, Alessandro Quaranta, Corrado Leita, who were always ready to provide me with ideas and answers to the various issues I encountered.

Finally I must thank all employees of Lastline's I did not mention for their keenness to help. It was a pleasure to work and learn in such an environment.

Contents

Chapter 1

Introduction

In the space of 20 years, Internet has become an integral part of our lives. We communicate with friends and colleagues, do shopping, send money and entertain ourselves online. Guaranteeing that our online activities and the sensitive information we exchange on a daily basis remain private has become a matter of vital importance. This concern resulted in the creation of the SSL (*Secure Scoket Layer*) protocol in 1996 then replaced by the first version of the TLS (*Transport Layer Security*) protocol in 1999 [3] which both provide encryption for the application layer.

Today most of the Internet traffic is encrypted with TLS. Gartner has estimated that by 2019, 80% of global web traffic would be encrypted [27]. Businesses, schools, governments and individuals all benefit from the privacy encryption provides and the usage of TLS will certainly continue to grow in the years to come.

However privacy does not guarantee security and malware authors have started to leverage TLS to hide their malicious activities. Malware have been seen to use TLS to communicate with their command server, either to receive instructions or to send back sensitive data collected on the infected machines. Cisco reports that the percentage of malware samples that used TLS in one way or another has risen from 2.21% in August 2015 to 21.44% in May 2017 [12]. We can expect that figure to grow, especially given how ubiquitous TLS has become and how cheap and easy it is nowadays to obtain valid TLS certificates.

Intrusion detection systems that decrypt network packets to apply traditional payload inspection techniques have proven to be inefficient and directly go against the purpose of TLS to keep user data confidential. In the light of all this, the goal of this project is to design and implement a detection system that avoids such shortcomings by using a machine-learning approach focusing on packets' metadata rather than on packets' contents, inspired by the work of a research team from Cisco [13, 15]. In addition, we are making available a curated set of capture files from malware that use TLS[1] and the tool to extract features from them.

We argue that malicious and benign flows do differ and that this separation is reflected in several parameters related to TLS, certificates or flow metadata. These differences

[1] https://tinyurl.com/tlsmalware

make possible the creation of a highly accurate classifier. The selected model, the random forest classifier, achieves an accuracy of 99.52% on never-seen before flows, and a true positive rate of up to 83.44% when limiting the number of false positives to 1 in 50000 benign TLS flows. The classifier was finally packaged and integrated into the intrusion detection platform of *Lastline*, the security company that supported this project.

This report is organized as follows: the second chapter provides a brief overview of the TLS protocol, explains how malware can take advantage of TLS and presents the current state of detection techniques against such malware. The third chapter details the implementation of the classifier, which includes the collection of data, the choice of relevant features and a comparison of different classification models. The fourth chapter presents the results of the system, its limitations and ways that it could be improved.

Chapter 2

Background

2.1 The TLS Protocol

The TLS Protocol is a cryptographic protocol whose primary goal is to "*provide privacy and data integrity between two communicating applications*" [3]. The first version of TLS was released in January 1999 to replace the now-deprecated SSL protocol. As of May 2019, the most widespread version of TLS is TLSv1.2 released in August 2008.

The TLS protocol sits below the application layer and on top of the transport layer, primarily TCP (TLS over UDP, DTLS, has been standardized independently [2]). TLS main use today is to encrypt HTTP traffic with which it forms the HTTPS protocol. According to Google, in the United States in April 2019 90% of all visited pages were loaded over HTTPS [32], a figure that keeps growing. However the usage of TLS is not limited to HTTP only: any application layer protocol can theoretically make use of TLS. For instance, the SMTP protocol and TLS together constitute SMTPS to protect emails.

A new version of TLS, TLSv1.3, has been released in August 2018 [5]. It introduces major changes and performance improvements over TLSv1.2. Browsers and webservers are slowly adopting this new version: major browsers such as Chrome, Firefox or Opera support it already and about 66% of all users would be able to use that version [55] if web servers would make the transition.

In the next sections, we present a quick description of the TLSv1.2 protocol, the main changes introduced by TLSv1.3 and a presentation of TLS certificates.

2.1.1 TLSv1.2

TLSv1.2 is currently the most used version of TLS: 95% of webservers support it [39]. It succeeded TLSv1.1 with the main goal of offering new alternatives to the insecure MD5 and SHA1 algorithms.

A TLS session is established in two round-trips of messages between a client and a server (figure 2.1) [4, 20]:

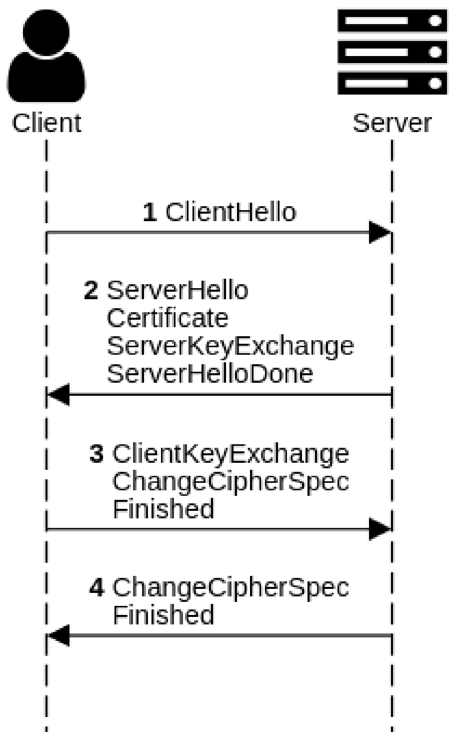


Figure 2.1: TLSv1.2 handshake

1. The client sends a `ClientHello` message. This message contains:
   - a list of ciphersuites supported by the client;
   - a list of compression methods;
   - a list of extensions that are used by the client to communicate additional information to the server (for instance to specify the destination hostname, the elliptic curves supported by the client etc.).
2. The server responds with several messages:
   (a) `ServerHello`: contains the cipher suite and the compression method selected by the server.
   (b) `Certificate`: contains a chain of TLS certificates proving the ownership of a public key.
   (c) `ServerKeyExchange`: contains information allowing the client to communicate a premaster secret (optional, useful when the server has no certificate or if its public key is for signing messages only).
   (d) `ServerHelloDone`: informs the client that the server is done sending messages in this first part of the handshake.
3. The client replies with:
   (a) `ClientKeyExchange`: contains information allowing the server to compute the final symmetric session key (a session key encrypted with the server's public key in most cases)
   (b) `ChangeCipherSpec`: informs the server that all subsequent messages will be encrypted with the session key.

   (c) `Finished`: informs the server that the TLS handshake was successful for the client.
4. Finally the server ends the negotiation with:
   (a) `ChangeCipherSpec`: informs the client that all subsequent messages will be encrypted with the session key.
   (b) `Finished`: informs the client that the TLS handshake was successful for the server.

All messages up until `ChangeCipherSpec` are sent in *clear text* since encryption is possible only when both parties share the symmetric key.

2.1.2 TLSv1.3

TLSv1.3 is the new version of TLS released in August 2018 destined to replace TLSv1.2. It has been designed for improved security and speed. The main changes are [5]:

- Obsolete ciphers and hashing algorithms have been removed: SHA1, MD5, DES...
- The number of messages needed for the handshake has been reduced.
- All messages after `ServerHello` are now encrypted.

The handshake only requires 3 sets of messages (figure 2.2) [5, 24]:

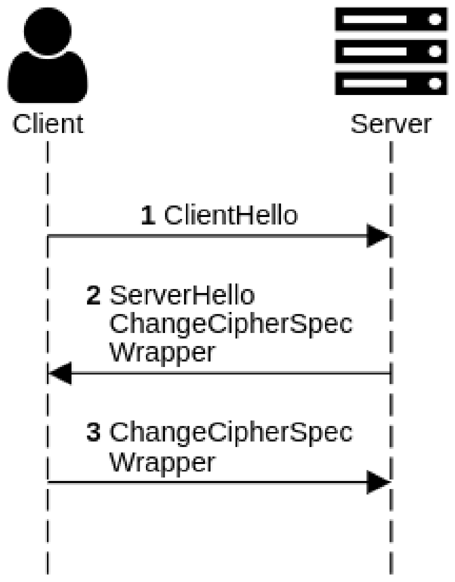


Figure 2.2: TLSv1.3 handshake

1. The `ClientHello` message is identical as the one in TLSv1.2 but also carries a list of public keys.
2. The server replies with several messages:

   (a) `ServerHello` contains the same information as in TLSv1.2 as well as a public key. From the client's public key and the server's public key, a new shared key is derived and is used only to encrypt the rest of the handshake.
   (b) `ChangeCipherSpec` has the same purpose as in TLSv1.2.
   (c) `Wrapper` encompasses server messages also present in TLSv1.2: `Finished`, `Certificate` etc.
3. The clients ends the handshake with:
   (a) `ChangeCipherSpec`, which has the same purpose as in TLSv1.2.
   (b) `Wrapper`, which encompasses client messages also present in TLSv1.2: `Finished`, optional messages...

It is important to note that all packets following the `ChangeCipherSpec` messages are encrypted, *including the server certificate*. This prevents eavesdroppers to identify the hostname associated with the server simply by looking at the certificate subject.

A new extension has also been added to TLSv1.3: *Encrypted Server Name Indication* (ESNI), which is the encrypted version of the SNI extension found in TLSv1.2, used by clients to indicate to which particular hostname they are attempting to connect to [31].

2.1.3 TLS Server Certificate

Server certificates are electronic documents binding a public key to a server, digitally signed by an entity called the *Certificate Authority*. Clients receiving a server certificate during a TLS handshake must verify that it is valid and that is has been signed by a trusted authority. If the certificate has been signed by an unknown organization, the client have to go up the chain of certificates until one from a trusted authority is found, which by construction validates all certificates below it in the chain. When the legitimacy of the server has finally been verified, the server's public key can be used to encrypt and share a symmetric session key.

TLS certificate generally follows the X.509 standard [6]. They contain several fields, some of them optional. The most notable ones are:

- **Issuer**: the entity that verified the legitimacy of the server and issued the certificate (a Certificate Authority in most cases).
- **Validity**: contains two sub-fields with the date from when the certificate is valid to the date it expires.
- **Subject**: the beneficiary of the certificate.
- **Subject Public Key Info**: contains two sub-fields indicating the server public key algorithm and the public key itself.
- **Extensions** (optional): contains several fields stating how the certificate should be used and additional information about the certificate.
- **Certificate Signature Algorithm** and **Certificate Signature Value**: the signature algorithm and the signature from the issuer of the certificate body.
- **Fingerprint** (not an actual part of the certificate): the hash of the entire

certificate (generally SHA256 or SHA1).

However certificates don't all have the same validation level and browsers display different symbols for each of them: Chrome's padlocks are shown figure 2.3. Validation

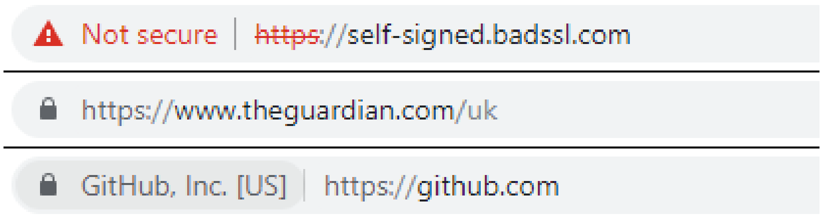

Not secure
~~https~~://self-signed.badssl.com
https://www.theguardian.com/uk
GitHub, Inc. [US]
https://github.com

Figure 2.3: Certificate validity levels: SS, DV and OV/EV

levels can be grouped into four categories:

- **Self-Signed** (SS). This is the lowest level of validation, since it means that the certificate has been backed up by the certificate recipient itself. Usually self-signed certificates are used by Certificate Authorities to share their public key, and browsers are configured to trust these particular certificates (called *root certificates*). From entities other than CAs, such certificates provide no proof of key ownership at all and browsers usually display warnings for them.
- **Domain Validation** (DV). the CA has only checked the connection between a public key and a domain name with no further identity checks.
- **Organization Validation** (OV). The CA has checked the domain name and some information about the organization before issuing the certificate.
- **Extended Validation** (EV). the "strongest" certificate, where the CA has conducted a thorough investigation on the organization in addition to verifying the domain name.

2.2 Malware' Use of TLS

The ways to take advantage TLS are numerous and we present here some of the techniques malware have been seen to use that involve TLS encryption. We deliberately do not discuss the exploitation of vulnerabilities in some implementations of TLS (*Heartbleed* [45] for instance) since such attacks may not use encryption themselves, which is the focus of this report.

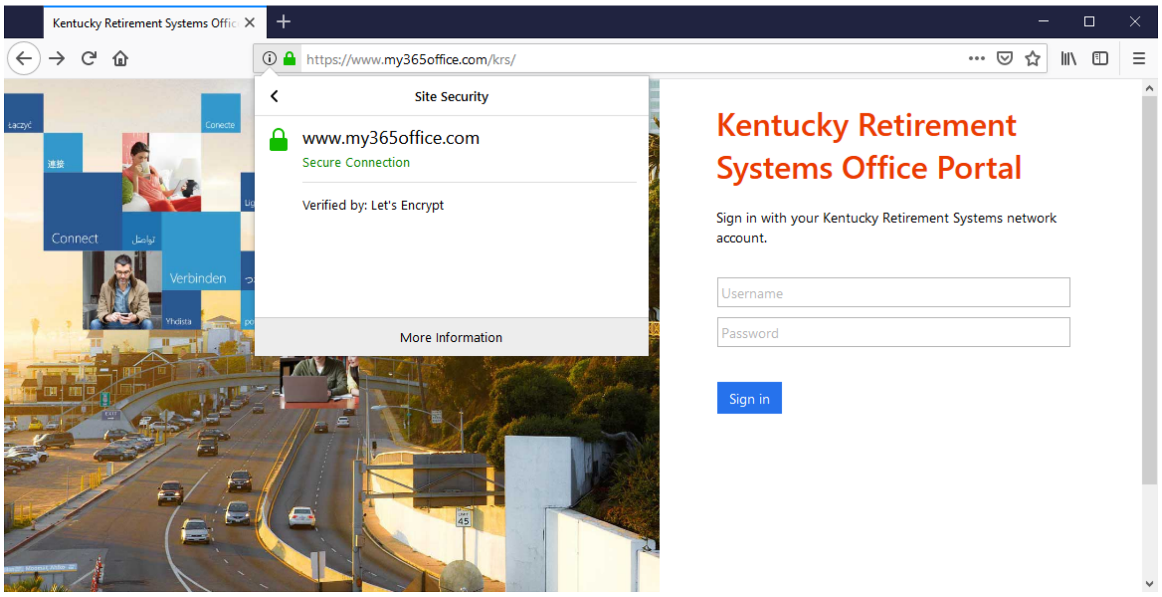
2.2.1 TLS-based Threats

Payload deposit TLS may be used to hide infection of a machine. For instance, an employee could visit a malicious website using TLS and a drive-by download malware would install itself without the user being aware of anything. Since all communications with the malicious webserver are encrypted, the malware would evade basic payload inspection set up by the employee's company.

Data exfiltration Encryption can be used to hide exfiltration of sensitive data: passwords, cookies, company data etc. A simple POST request encapsulated by TLS to port 443 (an usual destination port) of the attacker's server could leak precious information while not being blocked by internal firewalls and without raising suspicions. A new covert channel has also been recently discovered using certificates [50]: attackers use certificate fields such as `SubjectKeyIdentifier` to embed data, both for payload deposit to be used at a later time of to exfiltrate data via the sending of client's certificate to malicious remote servers.

Command and Control Malware authors are more and more often leveraging TLS to obfuscate the fetching of C&C commands by infected machines, for the same reasons as above: encryption allows to bypass the inspection of payload and TLS traffic to usual ports are often overlooked.

Phishing All means to deceive users and steal sensitive information are worth exploring for phishing authors and padlocks displayed by Chrome or Firefox are no exceptions as figure 2.4 shows. According to a survey from November 2018 published by PhishLabs, 80% of users believe that a green padlock means that the website is safe or can be trusted, and the fact that 49% of all phishing websites are seemingly using HTTPS can be interpreted as an exploitation of this misunderstanding [37].



Kentucky Retirement Systems Office Portal

Sign in with your Kentucky Retirement Systems network account.

Figure 2.4: A pretty convincing phishing website, bearing a green padlock in Firefox

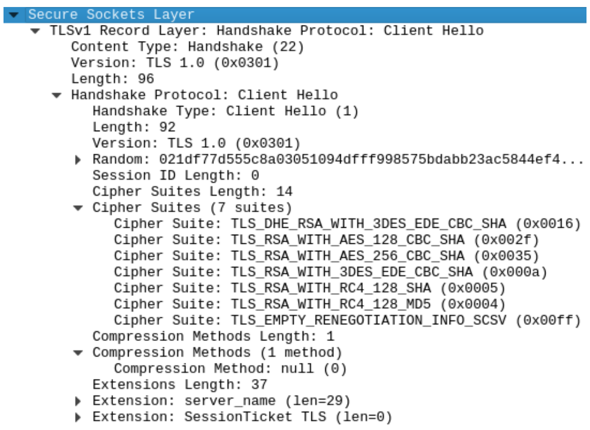
2.2.2 Known TLS-based Malware

There are many known malware that have been reported to use TLS or SSL. Banking trojans such as Trickbot, Emotet, Dyre make use of TLS to communicate data back to their master server. Ransomware too have been using TLS to infect machines and transfer information, such as Troldesh, Jigsaw, Locky or Petya. Understanding what makes malware’ use of TLS unique relatively to normal traffic is a key step to the creation of an efficient classification system, so here we present briefly the behaviors exhibited by two well-known malware, *TorrentLocker* and *Dridex*.

TorrentLocker

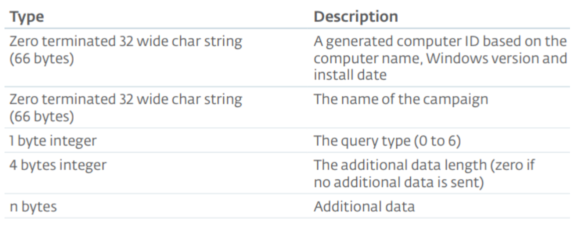
TorrentLocker is a ransomware first observed in February 2014 that encrypts a victim’s disk using AES. It then instructs the target to pay an amount of around 500$ to unlock the machine. It was generally distributed via localized spam campaigns and drive-by downloads exploiting Flash and Internet Explorer vulnerabilities [41].

TorrentLocker relies on TLS to communicate with its C&C server via POST requests. Figure 2.5a shows the TLS parameters send by the infected host. It is quite different from usual browser handshakes: we can note the low number of extensions and ciphersuites offered and the obsolete version of TLS used (TLSv1.2 was already out for 6 years at that time). Figure 2.5b and 2.5c show the actual structure and type of encrypted messages sent to the C&C server.



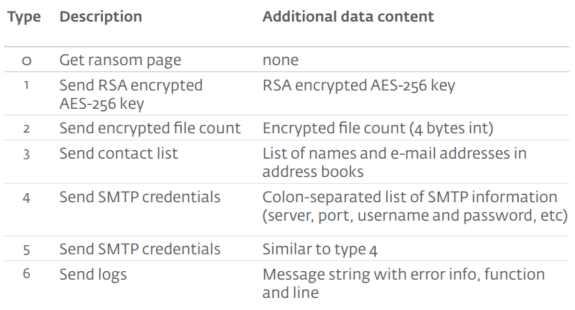
```
▾ Secure Sockets Layer
  ▾ TLSv1 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 96
    ▾ Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 92
        Version: TLS 1.0 (0x0301)
      ▸ Random: 021df77d555c8a03051094dfff998575bdabb23ac5844ef4...
        Session ID Length: 0
        Cipher Suites Length: 14
      ▾ Cipher Suites (7 suites)
          Cipher Suite: TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (0x0016)
          Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
          Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
          Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
          Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
          Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
          Cipher Suite: TLS_EMPTY_RENEGOTIATION_INFO_SCSV (0x00ff)
        Compression Methods Length: 1
      ▾ Compression Methods (1 method)
          Compression Method: null (0)
        Extensions Length: 37
      ▸ Extension: server_name (len=29)
      ▸ Extension: SessionTicket TLS (len=0)
```

(a) `ClientHello` content



Type	Description
Zero terminated 32 wide char string (66 bytes)	A generated computer ID based on the computer name, Windows version and install date
Zero terminated 32 wide char string (66 bytes)	The name of the campaign
1 byte integer	The query type (0 to 6)
4 bytes integer	The additional data length (zero if no additional data is sent)
n bytes	Additional data

(b) Structure of a query



Type	Description	Additional data content
0	Get ransom page	none
1	Send RSA encrypted AES-256 key	RSA encrypted AES-256 key
2	Send encrypted file count	Encrypted file count (4 bytes int)
3	Send contact list	List of names and e-mail addresses in address books
4	Send SMTP credentials	Colon-separated list of SMTP information (server, port, username and password, etc)
5	Send SMTP credentials	Similar to type 4
6	Send logs	Message string with error info, function and line

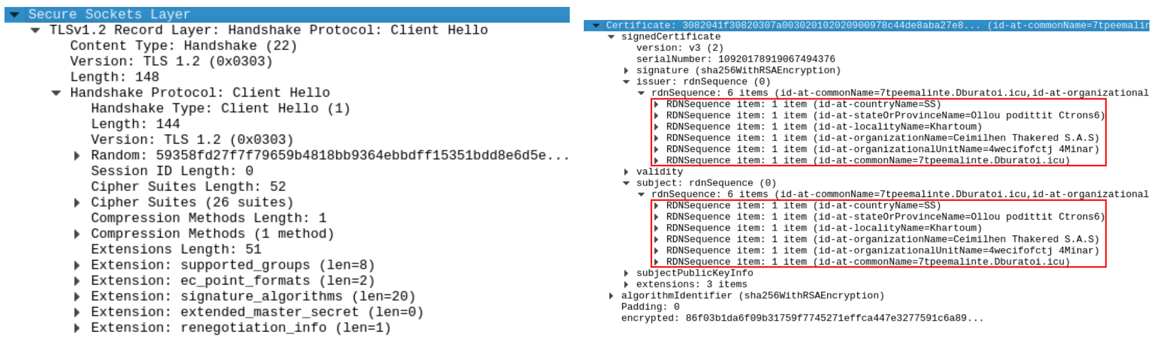
(c) The different query types

Figure 2.5: TorrentLocker

Dridex

Dridex is a financial trojan first seen in 2014 that targeted individuals in order to steal their banking credentials. It became in 2015 one of the most active banking trojan. It spread via spam campaigns sending out a Word document containing a malicious macro. That macro downloaded and installed Dridex on the victim's machine, turning it into a bot and stealing credentials [47].

Dridex uses TLS at different stages. The malware is made of different modules that can be enabled and disabled at will. A loader module would first download the main module via TLS, which is responsible for most of Dridex's features. All subsequent communications to the C&C server would also be encrypted by TLS. Figure 2.6a shows a `ClientHello` packet of a handshake with the C&C server. Nothing seems particularly abnormal: unlike TorrentLocker, the TLS version is recent and there are several extensions and ciphersuites offered by the client. But when looking at the certificate sent by the server figure 2.6b, we see that it is a self-signed certificate, which is unusual enough to maybe deserve further investigations.



```
▾ Secure Sockets Layer
  ▾ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 148
    ▾ Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 144
        Version: TLS 1.2 (0x0303)
      ▸ Random: 59358fd27f7f79659b4818bb9364ebbdff15351bdd8e6d5e...
        Session ID Length: 0
        Cipher Suites Length: 52
      ▸ Cipher Suites (26 suites)
        Compression Methods Length: 1
      ▸ Compression Methods (1 method)
        Extensions Length: 51
      ▸ Extension: supported_groups (len=8)
      ▸ Extension: ec_point_formats (len=2)
      ▸ Extension: signature_algorithms (len=20)
      ▸ Extension: extended_master_secret (len=0)
      ▸ Extension: renegotiation_info (len=1)
```

(a) `ClientHello` content

```
▾ Certificate: 3082041f30820307a003020102020900978c44de8aba27e8... (id-at-commonName=7tpeemalin
  ▾ signedCertificate
      version: v3 (2)
      serialNumber: 10920178919067494376
    ▸ signature (sha256WithRSAEncryption)
    ▾ issuer: rdnSequence (0)
      ▾ rdnSequence: 6 items (id-at-commonName=7tpeemalinte.Dburatoi.icu,id-at-organizational
        ▸ RDNSequence item: 1 item (id-at-countryName=SS)
        ▸ RDNSequence item: 1 item (id-at-stateOrProvinceName=Ollou podittit Ctrons6)
        ▸ RDNSequence item: 1 item (id-at-localityName=Khartoum)
        ▸ RDNSequence item: 1 item (id-at-organizationName=Ceimilhen Thakered S.A.S)
        ▸ RDNSequence item: 1 item (id-at-organizationalUnitName=4wecifofctj 4Minar)
        ▸ RDNSequence item: 1 item (id-at-commonName=7tpeemalinte.Dburatoi.icu)
    ▸ validity
    ▾ subject: rdnSequence (0)
      ▾ rdnSequence: 6 items (id-at-commonName=7tpeemalinte.Dburatoi.icu,id-at-organizational
        ▸ RDNSequence item: 1 item (id-at-countryName=SS)
        ▸ RDNSequence item: 1 item (id-at-stateOrProvinceName=Ollou podittit Ctrons6)
        ▸ RDNSequence item: 1 item (id-at-localityName=Khartoum)
        ▸ RDNSequence item: 1 item (id-at-organizationName=Ceimilhen Thakered S.A.S)
        ▸ RDNSequence item: 1 item (id-at-organizationalUnitName=4wecifofctj 4Minar)
        ▸ RDNSequence item: 1 item (id-at-commonName=7tpeemalinte.Dburatoi.icu)
    ▸ subjectPublicKeyInfo
    ▸ extensions: 3 items
  ▸ algorithmIdentifier (sha256WithRSAEncryption)
    Padding: 0
    encrypted: 86f03b1da6f09b31759f7745271effca447e3277591c6a89...
```

(b) Server certificate

Figure 2.6: Dridex

2.3 Overview of TLS Anomaly Detection

Organizations and companies are becoming increasingly aware of the threats these new types of malware represent. A study by Ponemon Institute from May 2016 reports that 68% of the 1023 companies surveyed have expressed concerns that encrypted malware communications would allow them to bypass their network's protection. Among the respondents, at least 33% have already been targeted by an encrypted attack and 54% expect these attacks to become more frequent in the coming years [33].

Given the rapid adoption of TLS since May 2016, which went from supporting 55% of the web traffic in the USA to 90% three years after, in May 2019 [32], we can extrapolate that malware have also followed this trend and we can better understand the urgency to address these threats as efficiently as possible. We present in this section an overview of the existing solutions to detect such malware, with their advantages and shortcomings.

2.3.1 Traditional Inspection Platform

A decryption platform is the most common way to deal with TLS communications in an enterprise setting. It consists of an appliance that intercepts encrypted traffic, decrypts it on the fly and inspects contents before forwarding the packets inside and outside the network. That is the method implemented by 38% of companies from the Ponemon's survey [33].

However it has several drawbacks:

- This technique is responsible for a large drop in network performance due to the constant interception and decryption of packets, up to a 92% drop in performance in average and a 672% increase in latency. 61% of companies claim that it is one of the main reasons they are turning away from inspection platforms [33].
- It may not be compliant with today's privacy standards in some countries and might even violate employees' rights. Sensitive information such as employees' banking credentials and emails have to be specially taken care of to avoid exposing their contents in clear text, which adds another layer of complexity.
- The platform must keep track of all TLS certificates and keys needed to decrypt packets, again increasing the overall complexity of such a system.

Even if this technique allows companies to benefit from all existing detection methods based on payload inspection, the burden of management and the high costs on performance make this solution not very practical.

2.3.2 Certificate Analysis

The analysis of certificates sent by web servers can also help detect phishing websites or anomalous connections. Certificates are sent in clear text during the handshake (see section 2.1.1), so traditional signature-based detection techniques can apply here. Alerts can be raised when detecting self-signed server certificates or abnormal strings in the fields of client certificates (uncommon but can be requested by web servers), which may be an indicator of data exfiltration [50].

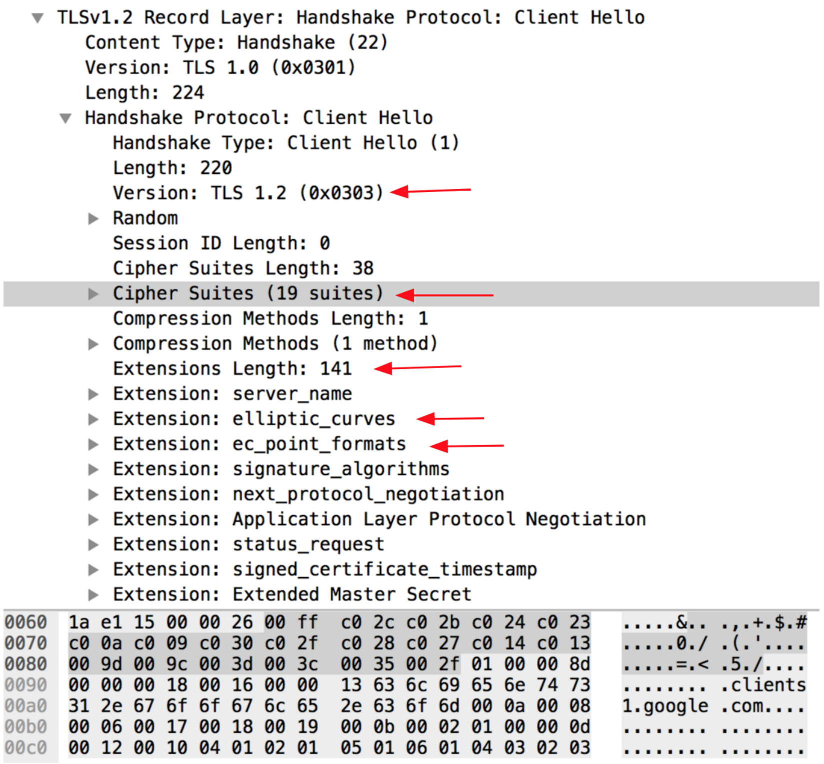
However the easiest way of dealing with certificates remains to maintain a blacklist of certificates known to be from malicious servers and check all incoming certificates against that list. Suricata [30], a popular open-source intrusion detection system, implements such blacklists. Public blacklists are also available online, for instance on SSLBL [7]. This method suffers from the drawbacks common to all blacklist-based methods, which is the burden of management: blacklists have to be updated regularly by trusted people to be useful and mistakes resulting in legitimate websites being blocked can be source of a lot of frustration for users.

2.3.3 TLS Fingerprinting with JA3

The initial `ClientHello` message, first introduced section 2.1.1 is composed of several fields all sent in clear text (see figure 2.7). These fields have values that either depend on the underlying TLS library used to initiate the connection or values that are directly set by the application, for optimization purposes. For instance, browsers tend to offer heavier and more secure ciphersuites while mobile applications favor faster and lighter algorithms. In any case, these values are unique to the application and remain fixed across sessions.

JA3 [52] is a project from Salesforce to fingerprint TLS sessions that leverages this property. It extracts five values from the `ClientHello` message: SSL version, list of ciphersuites, length of TLS extensions, elliptic curve groups and elliptic curve point formats, all shown figure 2.7. Values are then concatenated and the resulting string is hashed with MD5 to obtain the final JA3 fingerprint. An example is shown below:

```
JA3 string format:
TLSVersion,Cipher,TLSExtension,EllipticCurve,EllipticCurvePointFormat
Example of a JA3 string:
769,47-53-5-10-49161-49162-49171-49172-50-56-19-4,0-10-11,23-24-25,0
Resulting JA3 fingerprint:
ada70206e40642a3e4461f35503241d5
```



```
▼ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 224
  ▼ Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 220
      Version: TLS 1.2 (0x0303)
    ▶ Random
      Session ID Length: 0
      Cipher Suites Length: 38
    ▶ Cipher Suites (19 suites)
      Compression Methods Length: 1
    ▶ Compression Methods (1 method)
      Extensions Length: 141
    ▶ Extension: server_name
    ▶ Extension: elliptic_curves
    ▶ Extension: ec_point_formats
    ▶ Extension: signature_algorithms
    ▶ Extension: next_protocol_negotiation
    ▶ Extension: Application Layer Protocol Negotiation
    ▶ Extension: status_request
    ▶ Extension: signed_certificate_timestamp
    ▶ Extension: Extended Master Secret
0060  1a e1 15 00 00 26 00 ff  c0 2c c0 2b c0 24 c0 23   .....&.. .,.+.$.#
0070  c0 0a c0 09 c0 30 c0 2f  c0 28 c0 27 c0 14 c0 13   .....0./ .(.'....
0080  00 9d 00 9c 00 3d 00 3c  00 35 00 2f 01 00 00 8d   .....=.< .5./....
0090  00 00 00 18 00 16 00 00  13 63 6c 69 65 6e 74 73   ........ .clients
00a0  31 2e 67 6f 6f 67 6c 65  2e 63 6f 6d 00 0a 00 08   1.google .com....
00b0  00 06 00 17 00 18 00 19  00 0b 00 02 01 00 00 0d   ........ ........
00c0  00 12 00 10 04 01 02 01  05 01 06 01 04 03 02 03   ........ ........
```

Figure 2.7: The fields used to create the JA3 fingerprint (taken from [11])

JA3 can be used to identify the application that initiated a TLS session by checking databases of known JA3 fingerprints (the original repository offer such lists [52]).

This feature makes JA3 particularly interesting for threat detection. Malware often use custom parameters when communicating via TLS to their C&C server, which results in an unique JA3 fingerprint. Knowing that, blacklists of malware' JA3 hashes can be compiled [7]. Another great benefit of JA3 is that since it extracts information from the transport layer only, it remains unaffected by traditional evasion methods targeting the internet layer: Domain Generation Algorithm, change of IP or even the use of legitimate websites such as Twitter as C&C servers become irrelevant.

However client applications sometimes use the same common libraries or OS sockets which results in a shared JA3 hash. An extension of JA3, JA3S [52], uses fields from the `ServerHello` message. Since that message depends on values from `ClientHello`, it cannot be used to fingerprint TLS *servers*. But an application communicating with one server will always receive the same JA3S from it. This is useful for example when a malware uses a common library to connect to a single C&C server: the JA3 is indistinguishable from a legitimate application using the same library. But the way the C&C server responds to this malware is unique compared to the response of a normal webserver. Therefore the combination of JA3 + JA3S is able to detect more accurately malicious communications.

JA3 is not perfect. It suffers from the same shortcomings of `user-agents` in HTTP: nothing prevents an attacker to modify `ClientHello`'s values to imitate legitimate applications, and already such evasion techniques have been seen in the wild [8]. The choice of MD5 in JA3 is also debatable, first because MD5 is now obsolete and second because a fuzzy hashing algorithm would have made more sense [11]: it would have allowed the detection of very similar `ClientHello` messages, likely coming from the same application, based on their JA3 fingerprint only. Finally, JA3 blacklists suffer from the same drawbacks common to all blacklists in general, already described in the previous section. Despite all this, JA3 remains a useful piece of metadata that, especially when combined with other methods, is certainly very relevant to malware detection.

2.3.4 Machine Learning Techniques

Anomaly Detection and Machine Learning

The goal of anomaly detection systems is to find behaviors that deviate from a baseline, the "*normal*" (and preferably benign) behavior. But first, a model of what normal behavior looks like must be created. Statistical analysis is a natural solution to such problems. There are two kinds of approaches here:

- **Supervised learning**. In this setting, we have access to a set of n observations with p features for each one. We also have the corresponding n responses which can be used to train and validate models. The goal is to predict responses of previously unseen observations.
- **Unsupervised learning**. In this setting, we only have the set of observations with their features. The objective is not to predict a response but to discover interesting relationships between observations, if they can be classified into

subgroups for instance.

This report will focus on *supervised learning*, because this area is more developed and understood than unsupervised learning [36]. Supervised learning also suits better our initial goal which can be described as a classification problem for which labelled sets can be created or fetched online. Finally, this approach allows to get better insights on the results obtained, that in return helps analysts understand what exactly distinguish malware traffic from normal traffic [53].

Supervised machine-learning methods have been applied extensively to anomaly detection [38, 46, 56]. However not much literature exists on TLS anomaly detection, probably because the use of TLS was not so prevalent a few years back. Security companies do advertise their use of machine-learning algorithms against abuse of TLS [22], but are understandably reluctant to go into details. The main public resources on this subject come from a Cisco team who published several papers on encrypted malware detection [12, 13, 14, 15, 44]. We present a summary of their results here.

Characteristics of TLS malware

What exactly distinguish encrypted malware communications from legitimate ones? Based on a sample of around 26000 malicious TLS flows from 18 different malware families and benign TLS flows collected in an enterprise setting, the Cisco team reported these findings:

- From the infected client's `ClientHello` packet (figure 2.8, taken from [15]):
  - Malware families offer a completely different set of ciphersuites than normal clients. Furthermore these ciphersuites are often weak or obsolete. In comparison, almost all benign applications offer the same set of ciphersuites.
  - Malware rarely offer more than one extension where enterprise clients advertise up to 9 extensions.
  - The client's public key length varies between malware and normal clients.

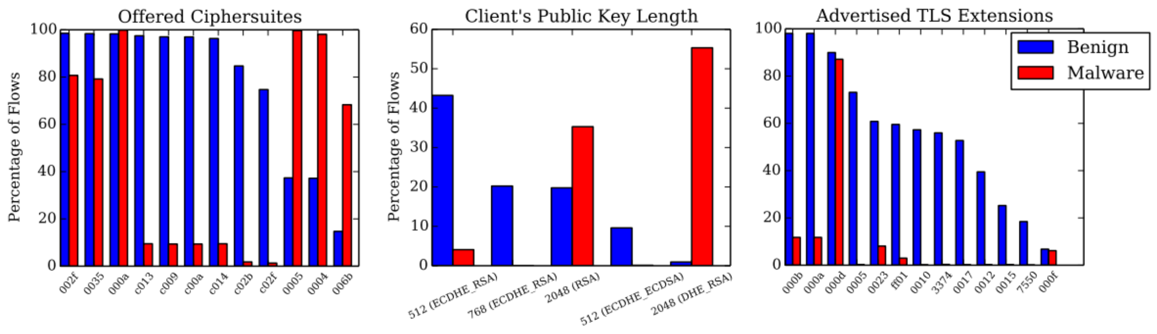


Figure 2.8: Differences between normal and infected hosts [15]

- From the malicious server's `ServerHello` and `Certificate` packets (figure 2.9, taken from [15]):

- Servers queried by malware select uncommon ciphersuites, as expected given the restricted size of ciphersuites offered in the first place.
- The same observation applies for extensions chosen by servers.
- Certificates' period of validity is also discriminant: some specific periods were used more often than others.
- The number of `subjectAltName` (SAN) entries in certificates, which allows a certificate to cover several domain names, differs quite a lot.
- The percentage of malicious servers sending a self-signed certificate is an order of magnitude higher than for normal servers.

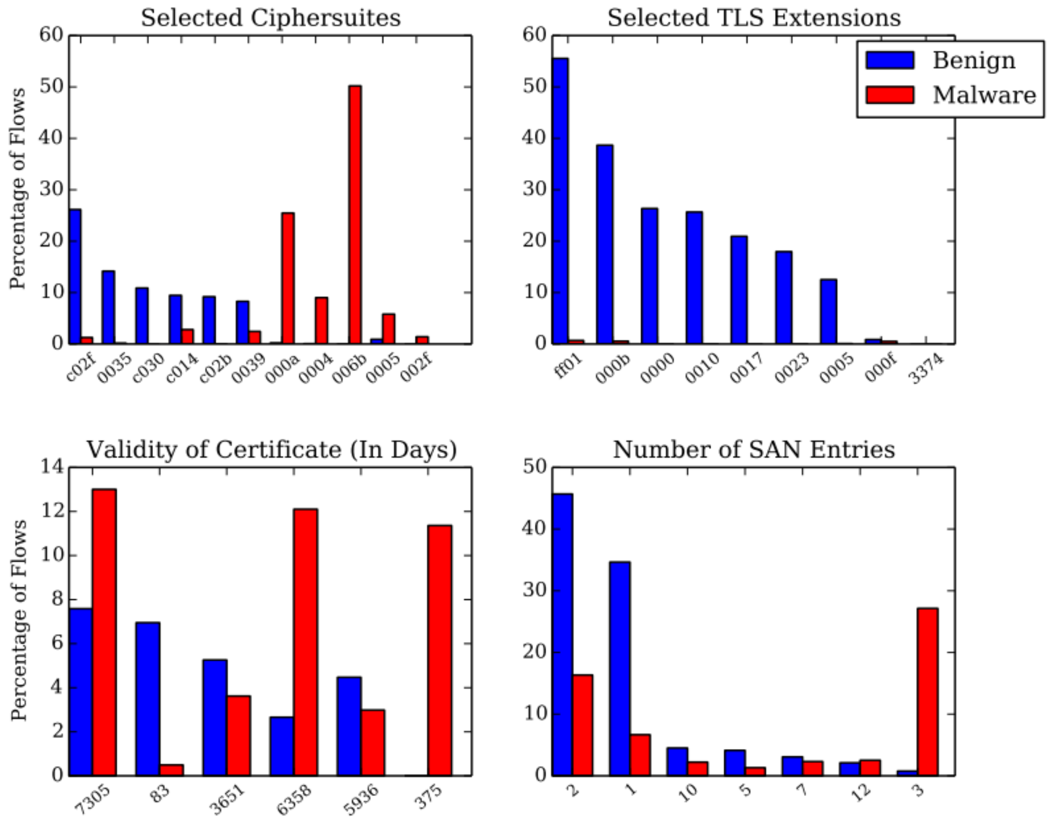


Figure 2.9: Differences between malicious and normal servers [15]

- Finally packet lengths and inter-arrival times are behavioral features that may be very different between benign and malicious sessions as shown figure 2.10 (taken from [12]). Sizes of packets sent from the client to the server are represented with upward lines, sizes of packets from the server to the client are represented with downward lines and the horizontal axis represents time.

Data Features

The choice of relevant features is arguably one of the most important step in the creation of an efficient model suitable to anomaly detection. Ideally these features should

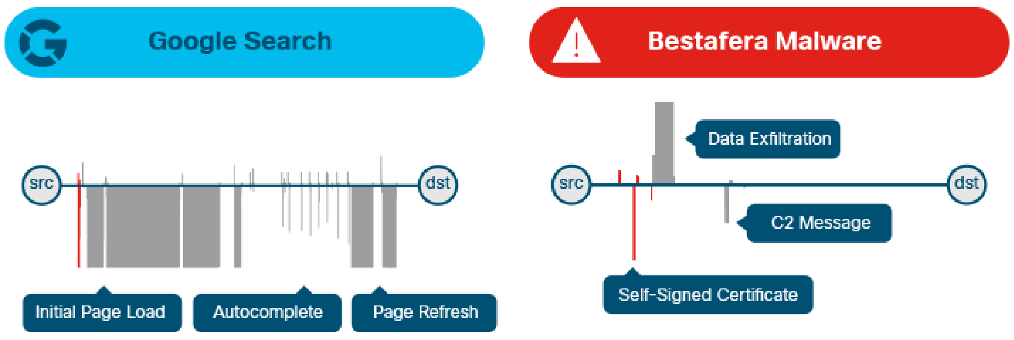


Figure 2.10: Packet lengths and inter-arrival time of benign and malware sessions [12]

have the following properties to maximize detection while minimizing management and storage costs [44]:

- **Compact**: the number of bits needed to store one observation (composed of multiple features) must be significantly smaller than the original flow size to ensure that a large number of observations can be stored and re-used later for training.
- **Informative**: the features must contain relevant information to the initial problem and should be as independent as possible from one another.
- **Economical**: collecting flows and extracting features should not require too much computing power and time.

Based on the previous observations and a study of TLS parameters [44], the features selected by Cisco are presented table 2.1. They can be divided into 3 categories:

1. **Flow Metadata**: features not related to TLS that are observable in any NetFlow.
2. **Distributions**: features that cannot be directly extracted from packets but are the results of some kind of frequency analysis on a flow's packets.
   - *Sequence of packet lengths*. Packet lengths are placed into bins, *e.g.* packet of size in range $[\![0\,,150]\!]$ will go into the first bin etc. A matrix A is then constructed where each entry $A[i,j]$ counts the number of transitions between bin i and j. The rows of A are finally normalized to ensure a proper Markov chain and entries of A are used as features (explanation from [15]).
   - *Sequence of packet size*. Same as for packet lengths but with packets' inter-arrival times.
   - *Byte distribution*. A length-256 array that keeps count of each byte value encountered in the payload of packets.
3. **TLS Metadata**: features that are extracted from TLS handshake packets (`ClientHello`, `ServerHello`, `Certificate`).

Category	Feature	Type
Flow Metadata	Source port	Integer
	Destination port	Integer
	Number of inbound bytes	Integer
	Number of outbound bytes	Integer
	Number of inbound packets	Integer
	Number of outbound packets	Integer
	Duration of the flow	Integer
Distribution	Sequence of packet lengths	Stochastic matrix
	Sequence of packet times	Stochastic matrix
	Byte distribution	Length-256 Array
TLS Metadata	List of ciphersuites	Binary vector
	List of TLS extensions	Binary vector
	Client's public key length	Integer
	Selected cipher suite	Integer
	Selected extensions	Binary vector
	Number of SAN	Integer
	Validity (in days)	Integer
	Certificate self-signed or not	Boolean

Table 2.1: Cisco's features for TLS malware detection

Cisco's Results

Two models were tested by the Cisco team:

- **Logistic regression** computes the probability that an observation belongs to a specific class. It is a highly interpretable algorithm that provides the weights associated to each feature.
- **Random forest** returns the averaged output of multiple decision trees. Each decision tree has been built with a bit of randomness to decrease correlation between them. Random forest performs especially well on non-linear observations and are also highly interpretable. This is the algorithm that performed the best according to a comparison of different models [14].

The models were trained on two large labelled sets of flows: one collected in an enterprise setting, considered benign, the other composed of malicious flows the Cisco team have gathered internally. Finally 10-fold cross validation was performed to estimate the test error. The accuracy when the classifier is only allowed one false positive for every 10000 benign flows has also been computed. The summary of the results is reported table 2.2.

Algorithm	**Total Accuracy**	0.01% **FP**	$< 0.01\%$ **FP**
Logistic Regression	99.6%	87.4%	$\sim 86.4\%$
Random forest	99.9%	–	$\sim 86.8\%$

Table 2.2: Accuracy for different algorithms and false positive rates [14, 15]

2.4 Legal and Ethical Considerations

The goal of this project is to implement a detection system targeting malware hidden in encrypted traffic, ideally to be used in an enterprise setting. Traditional techniques involve intercepting and decrypting all packets coming from outside or inside the enterprise, which means that employees cannot expect any privacy in their network activities. Someone having access to such a system could see in clear text all passwords, bank details, private messages sent by any employee. The detection system presented here avoid such privacy breaches by never actually decrypting packets: it only looks at packet metadata such as ports, flow duration etc. The only potentially sensitive piece of information the system currently has access to would be the domain name an employee is connecting to. But we argue that in an enterprise setting, people already expect that their web history is visible to their employer. Furthermore while active, the detector never stores the domain name: the detector uses it only to filter out legitimate flows and discard it afterward.

The classification model requires a dataset composed of benign traffic to be trained on. That "good" dataset was collected by sniffing traffic regularly for approximately three weeks from Lastline's offices, with the permission of the office manager. The resulting capture files have been saved in an internal storage space to re-train the classifier if needed, but they are not available to the public and to all employees. The "bad" dataset, composed of malware traffic, is mainly composed of flows publicly available on the Internet.

Eventually the classifier has been integrated into Lastline's internal network analysis system (Llanta), which is a proprietary software. To do so, an additional module has been developed that processes and forwards flows received by Llanta to the trained model. This module is not presented in this report since it depends on many Lastline's libraries and cannot function without them. However the classifier itself can be demonstrated on a testing dataset without requiring any Lastline's software, which is how the results presented section 4.2 have been obtained.

Apart from these three points, there are no other legal or ethical considerations related to the project to report. The scripts all use open-source software and there are no apparent scenarios where they could be used to do harm.

Chapter 3

Design and Implementation

The aim of this project is to design and implement an efficient classifier capable of labelling TLS flows as either benign or malicious. As stressed by [53], having a good understanding of the repartition and origins of the available data is a key step towards the interpretation of results. With that in mind, the chapter is divided into three sections that go through the development process. The first section presents where and how the data was collected and pre-processed. The second section provides insights on how exactly malware and benign TLS flows differ, what features were selected based on these observations and how robust they are against tampering. The third and last section looks into the selected classification models and their implementation.

The project was supervised by Lastline, a cyber security company with offices in North America and London. The classifier was developed in Lastline's London office with the intent of being integrated into their main product, an intrusion detection platform.

3.1 Data Collection

Numerous and relevant observations provide the basis of a supervised machine-learning model. Since the goal of the present work is to implement a supervised classifier that would eventually be able to distinguish between *benign* and *malicious* TLS flows, two datasets containing each a large number of samples from both classes are needed. We call here *TLS flow* a set of packets representative of a TLS session, which combine a successful TLS handshake and subsequent encrypted application data packets. This section presents how the data was collected and where and when the samples come from.

3.1.1 Sources

In total, 16275 malware flows and 28136 benign flows were collected from May to June 2019. The repartition of sources are shown figure 3.1.

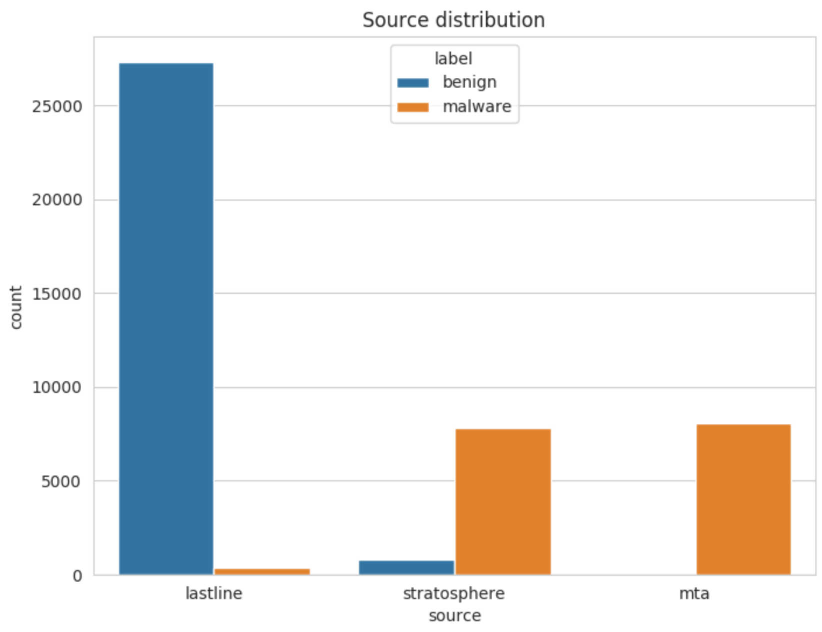


Figure 3.1: Source of TLS flows

Benign Dataset

The capture of benign TLS flows was straightforward: traffic from Lastline's offices was captured over a period of approximately three weeks. Two offices were monitored, the London office and the Redwood City office in California. There are about 15 employees at London's office, all software developers, and they all use either Linux or MacOS as operating system with Chrome or Firefox as browser. They mainly visit internal or IT related websites. The Redwood City office is primarily composed of business teams which account for around 30 employees, and they use a mix of MacOS and Windows machines. The captures can largely be considered benign, because the shared Internet connection is protected with Lastline's main product, an intrusion detection system covering several surface attacks, and because all employees are are regularly sensitized on online threats given the company's main activity.

It should be noted that the benign traffic collected comes from an enterprise setting, more specifically from a medium-sized and IT-oriented office, which may differ a lot from traffic generated in an university, at home or from a company with a different activity. The resulting classifier would therefore suits to similar network traffic, but may not perform well in these other settings. New benign traffic should be captured for each new context to improve the classifier's capabilities. Finally, to introduce a bit more traffic coming from Windows machines, benign captures were also downloaded from Stratosphere IPS [35]. It represents 821 flows (3% of the total).

Malware Dataset

A lot of resources dedicated to machine-learning projects are available online [21, 23, 42]. However most of them focus on clear text protocols such as HTTP or DNS. In addition, the distinction between benign and malware flows in these captures is often blurry, or the authors only provide `.csv` files and not original capture files which we need to take advantage of the full range of TLS parameters.

Therefore only individual capture files have been selected where malware have been placed in an isolated sandbox before having their traffic captured, which theoretically ensures that only malicious flows are considered. Samples made of pre-infection TLS traffic exclusively, for instance users visiting phishing website that have HTTPS enabled, were ignored and only capture files that mentioned encrypted post-infection traffic were kept. The malware dataset has been made public on Google Drive[1]. It was built using three main sources:

malware-traffic-analysis.net [25] All capture files from January 2014 up until June 2019 that contained TLS flows were collected. This represents 8080 flows (after the filtering process explained in the next chapter) across 311 capture files.

Stratosphere IPS [35] Stratosphere IPS provides several long-term captures of malware that use TLS to communicate. There are fewer captures available here than on *malware-traffic-analysis*, but because traffic is collected over a longer period of time, a lot more flows are present in these captures: 348041 flows from only 31 captures. To avoid an excessive imbalance between benign and malicious flows and also to avoid an over-representation of some malware families such as Dridex, only the first 300 flows of each capture were kept, which amounted to 7848 flows in total.

Lastline Lastline saves very short capture files of malware their network sensors detect. All captures from known TLS malware seen in a two weeks period have been kept, from April 24, 2019 to May 7, 2019. This represents 347 flows across 101 files: a small number in comparison to the other two sources, but they have the advantages of being very recent and representative of actual threats companies encounter on a weekly basis.

3.1.2 Date Distribution

All benign flows are either from 2019 (Lastline's traffic) or 2017 (captures from Stratosphere IPS). Malware flows are spread over 4 years from 2016 to 2019. The full repartition is presented figure 3.2.

Time can introduce bias, for instance if the time window between good and bad flows does not overlap. In this case, the classifier may learn to distinguish flows

[1]https://tinyurl.com/tlsmalware

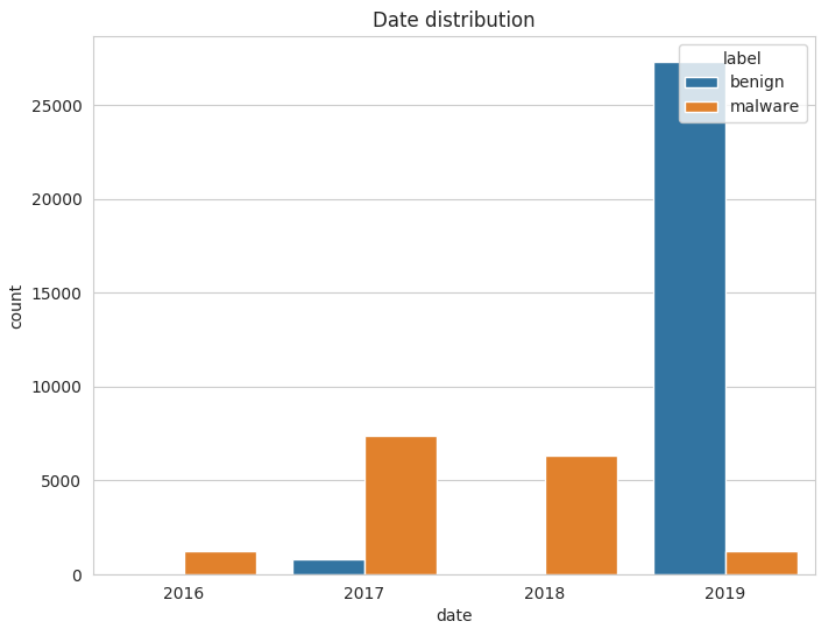


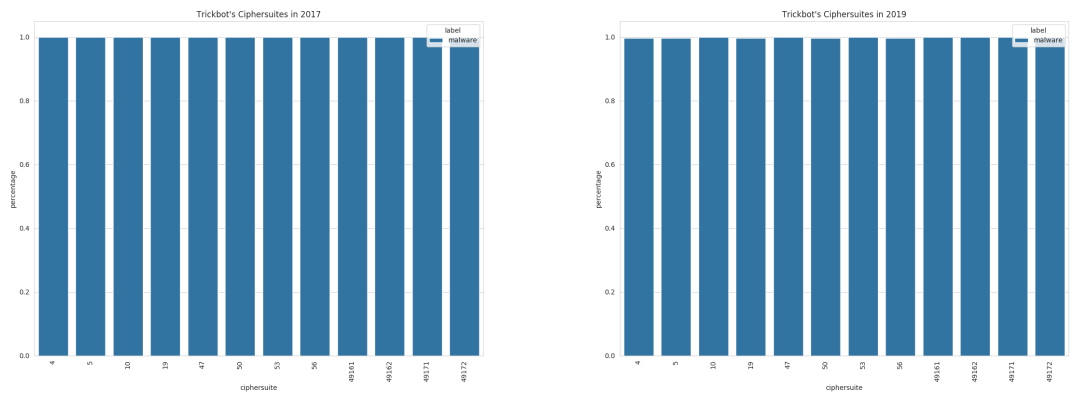
Figure 3.2: Number of flows by year

based on time rather than on the threat they pose [49]. Another type of bias appears when training the classifier on observations posterior to those present in the testing set. This leads to artificially improved accuracy as emphasized in [10] and does not guarantee that the classifier will be robust to time decay [49].

We tried to mitigate this in the datasets: first by collecting some benign flows from the same periods as malicious ones, second by also testing our classifier on very recent flows (see next chapter, section 4.1.1). Furthermore, time does not seem to affect much the malware dataset: in the past three years, malware families generally have not updated the ciphersuites or extensions they use. For instance figure 3.3 shows that Trickbot still uses the same twelve ciphersuites from 2017 in 2019. It is likely because from 2016 to 2019, there has not been many changes in the sets of ciphersuites made available by the TLS protocol. We can expect the classifier to still be relevant in the months to come.

3.1.3 Malware Families

The malware dataset contains 18502 flows shared among 40 distinct malware families (if we consider `unclassified` as a family on its own), shown figure 3.4. However this distribution should not be entirely trusted. The family label was assigned manually, based on the name of the downloaded capture file, without checking thoroughly if the characteristics exhibited in each capture indeed corresponded to the advertised name. For instance, `RIG` is an exploit kit that is often used to deliver trojan or ransomware. It does not make use of TLS in itself, but the malware it has delivered might be.



(a) 2017 **(b)** 2019

Figure 3.3: The identical set of ciphersuites offered by Trickbot in 2017 and 2019

However even if this list is not totally reliable, the scope of this project has been confined to the classification of TLS flows according to whether or not they are malicious and not according according to the malware family they may belong to.

Still, we recognise some families that are know to abuse TLS: `Dridex`, `Trickbot`, `Upatre`, `Zeus` (also known as `Zbot`) and `ZeuS Panda` (a variant of `Zeus`) are some of them. A full list of malware that leverage TLS which was used to select only relevant samples when building the datasets is present in the appendix A.

3.1.4 Data Extraction and Storage

The capture files in `.pcap` format were parsed into JSON files using Joy [43], an open-source packet analysis tool. The tool extracts TLS flows along with packet and byte distributions. Flows from the resulting JSON files are then filtered to remove unusable flows (more detailed in section 3.2.1). After that, the filtered files are parsed to extract TLS information used afterwards by the feature extraction tool, like the set of ciphersuites, extensions and elliptic curve groups seen across all files. This step serves to reduce the final number of features by only considering what has been found in the training datasets: for instance the TLS protocol offer 344 valid ciphersuites [34], but only 145 of them are actually present in the observations. The last step uses these sets and the filtered JSON files to extract or compute the features, and finally save them into `.csv` files. The whole pipeline is presented figure 3.5.

3.2 Feature Selection

This section presents how features were filtered, selected, and formatted. Distinct characteristics of malware and benign flows are also detailed here with their interpretation.

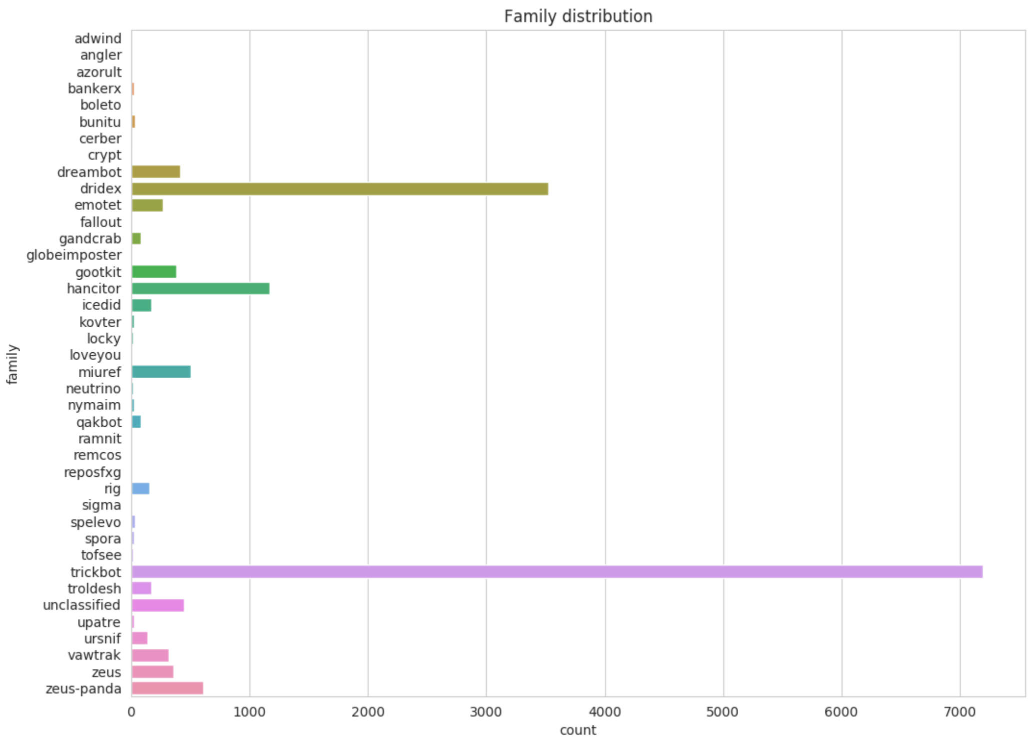


Figure 3.4: Number of flows by malware family

3.2.1 Flow Filtering

Before proceeding to feature extraction, TLS flows were first filtered based on three criteria. First, TLS flows with an incomplete handshake were discarded, as well as TLS flows that had less than three packets in each direction for the reason exposed in [57]: since we take into account the sequence of packet lengths and times, a minimal number of inbound and outbound packets must be defined.

Then are removed all flows that miss key features, such as server certificate or the list of offered ciphersuites. The mandatory features that all flows must possess have been kept as small as possible so as not to remove a too large number of flows from the datasets and for the classifier not to be too selective regarding the flows it can analyse.

Finally, since there are sometimes artifacts of benign flows in malware captures, we remove all flows present in malicious captures where a destination hostname is present that is mentioned in one of the three main top one million websites lists: Alexa[2], Cisco Umbrella[3] and Majestic[4]. This process, as well as the previous one, may

[2]https://www.alexa.com/topsites
[3]https://umbrella.cisco.com/
[4]https://majestic.com/reports/majestic-million

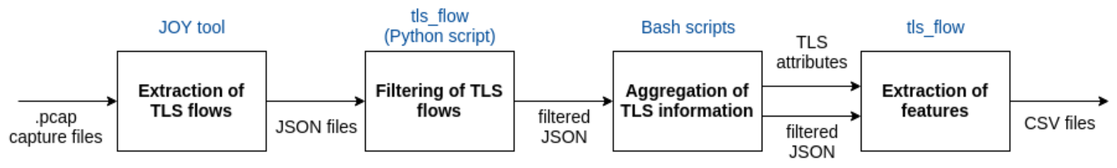


Figure 3.5: Feature extraction pipeline

remove bad flows: some malware have been seen to use social networks of Google services to receive their commands or send data. But keeping all such flows would skew the training process and result in a larger number of false positives.

All capture files are not equal in terms of number of packets and capture duration. To avoid having too many samples from a very small set of captures which could introduce bias in the classification, only the first 300 packets of each file were kept. This step ensures a better distribution of packets and of malware families across all available captures.

The filtering results are shown 3.1. The vast majority of removed flows comes from the absence of a mandatory feature, especially from the absence of a server certificate, and from the filtering based on top websites lists.

Source	Before Filtering	After Filtering	Reduction
`malware-traffic-analysis`	17442	8080	−53.7%
`malware_stratosphere`	507175	7848	−98.5%
`malware_lastline`	2998	347	−88.4%
`benign_lastline-london`	104728	15688	−85.0%
`benign_lastline-redwood`	68858	11627	−83.1%
`benign_stratosphere`	2427	821	−66.2%
Total	738408	44411	−94.0%

Table 3.1: Results of filtering

3.2.2 Differences between Malware and Benign Flows

The selection of features must be motivated by clear differences in the characteristics of malware and benign observations. If both datasets are very similar in the first place, trying to classify flows would most certainly fail. This section looks into the distinguishing features between the labelled flows and argues that a separation between the two datasets does exist.

Flow Metadata Differences

Malicious communications collected last longer in average than benign ones (figure 3.6a). This is due to the fact that Lastline's employees typically use TLS to load webpages and download small resources which results in short-lived TLS sessions, whereas malware may tend to send and receive larger amount of data (data exfiltration, fetching of additional malicious modules remotely...).

The figure 3.6b showing the difference in the average byte entropy is a bit more surprising. High entropy results from the use of strong encryption and where we could have expected malware flows to have less entropy due to maybe ignoring encryption of data packets, it is the contrary that happens. We speculate that this is correlated to the short flow duration of benign sessions: since the TLS handshake is not encrypted and account for a larger portion of the benign TLS session, it lowers the final average entropy.

Finally the destination port (from the client's point of view, therefore the server's listening port) is a good indicator of compromise (figure 3.6c). By default, legitimate servers listen on specific ports for TLS-related packets (see appendix B for a list of usual TLS ports), and apart from malware there is no reason for clients to initiate TLS sessions to other ports. There is also no real advantages for malware to use other ports than TLS ones since they are more prone to being blocked or detected, so it may be that authors just wanted to use encryption and did not care about which ports their server should use. The same goes for source ports figure 3.6d: by default operating systems assign source ports randomly in the range 49152–65535 [28]. Some malware do not request a source port from the OS and use a custom one, which make them stand out among normal connections.

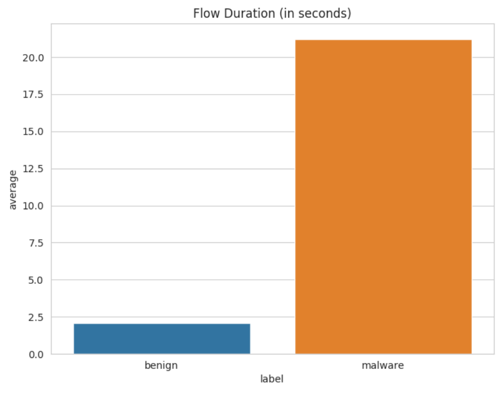
TLS Parameters Differences

The lists of offered ciphersuites and extensions by the client have discriminant power. Some ciphersuites are completely ignored by malware, others are disproportionally favored as shown in figure 3.8a. Appendix C.1 can be used to map codes to ciphersuite names. The most notable ones are:

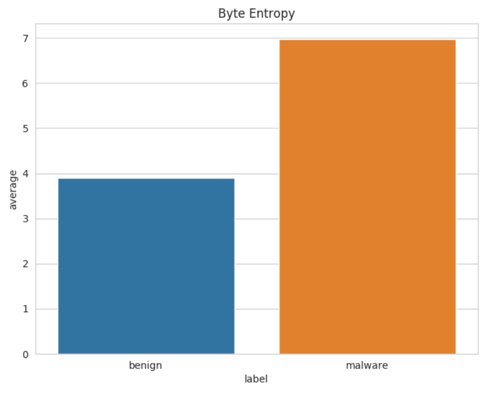
- `TLS_RSA_WITH_RC4_128_MD5` (4 or `0x0004`)
- `TLS_RSA_WITH_RC4_128_SHA` (5 or `0x0005`)
- `TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA` (19 or `0x0013`)
- `TLS_DHE_DSS_WITH_AES_128_CBC_SHA` (50 or `0x0032`)
- `TLS_DHE_DSS_WITH_AES_256_CBC_SHA` (56 or `0x0038`)

All five of them are considered weak or insecure according to [51]. The use of such ciphersuites may be attributed to a lack of awareness from malware authors of what would be strong choices, to the lack of mechanisms to remotely update malware with modern ciphersuites once they have been released, or to a lack of concern with the idea that the strength of ciphersuites does not matter as long as contents are encrypted. There have been evidences of the latter regarding the choice of an algorithm: some authors are only concerned about encryption and nothing else, like the ZeuS authors who explain their reasoning in their FAQ [58] shown figure 3.7.

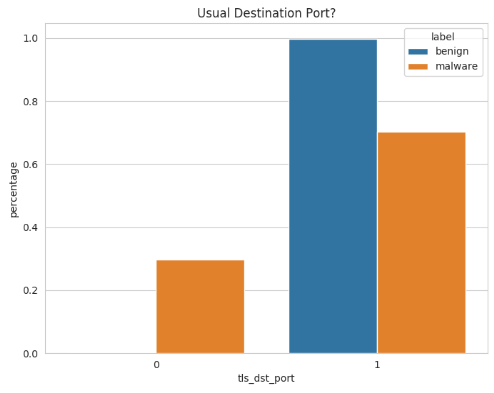
The same goes for extensions: malware tend to use a lot less extensions, with an average of 3 extensions offered by malware against 10 for the normal flows present in



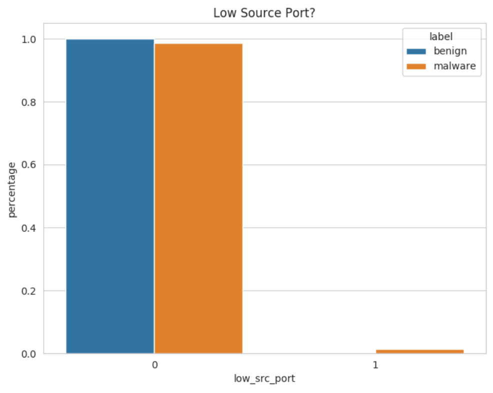
(a) Flow Duration



(b) Entropy



(c) Destination Port



(d) Source Port

Figure 3.6: Differences in flow metadata of benign and malicious flows

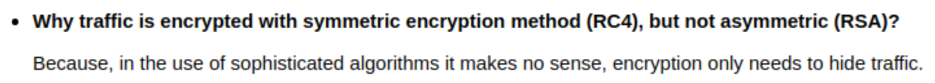
the datasets. The proportion of extensions offered by both classes is presented figure 3.8b (appendix C.2 maps codes to extensions). As for ciphersuites, some extensions are almost never offered by malware:

- `status_request` (5)
- `application_layer_protocol_negotiation` (16)
- `extended_master_secret` (23)
- `session_ticket` (35)

Normal hosts also favor shorter keys (figure 3.9d): they mainly use 512-bit key (which is used by the *ECDHE_RSA* key exchange algorithm) where malware also use 2048-bit key (*DHE_RSA* algorithm). This is in accordance with the findings of Cisco in [15].

Certificate Differences

Certificates contain a lot of parameters that are quite different for both types of observations. As expected, malware tend to rely a lot more on self-signed certificates (figure 3.9a). They are also more likely to use longer period of validity than legitimate


- **Why traffic is encrypted with symmetric encryption method (RC4), but not asymmetric (RSA)?**

  Because, in the use of sophisticated algorithms it makes no sense, encryption only needs to hide traffic.

Figure 3.7: An entry in the ZeuS malware FAQ regarding the choice of ciphersuites (translated from Russian)

certificates, which tend to favor shorter periods (figure 3.9b). Reasons behind short periods are to limit damages from the compromise of certificates and to push system administrators to automate the renewal process in order for HTTPS to become less cumbersome to manage [26].

The number of *Subject Alternative Names* (SAN), which indicates other domains for which the certificate is valid, stays relatively low for malicious flows. A possible explanation would be that legitimate certificates from reputable sources are expensive and companies often own several sub-domains, so there are incentives for them to put many alternative subjects in a single certificate.

3.2.3 Feature Selection

This section presents the features selected for classification based on the previous observations. Two different sets of features were extracted. The first one contains all types of features described in Cisco papers [15, 44] and presented section 2.3.4, plus a few others. The second set is a subset of the first one and contains only features present in Lastline's serialized representation of TLS flows. All features are shown table 3.2. *SPT* and *SPL* stand for *Sequence of Packet Lengths* and *Sequence of Packet Times* respectively.

The full set contains 417 features and the reduced one 208. However the size of these sets depends on the training set, since features that are absent from all training observations are removed. This is the case for ciphersuites, extensions, elliptic curve support groups (column *Dynamic* in table 3.2). These indicators are useless to the classifier and do not impact its performance anyway. When extracting features from testing sets, a special binary feature is set to 1 whenever a never-seen before (because absent from the training sets or removed) dynamic parameter is encountered in a flow.

Most of the features have already been described section 2.3.4. The new ones that are not self-explanatory are:

- **Ephemeral source port**. Source port is usually chosen randomly by the OS in the range 49152–65535 [28]. This feature accounts for source ports that are out of that range.
- **Usual TLS destination port**. Servers listen on specific ports for TLS connections, anything out of these few ports can be considered suspicious.
- **Elliptic Curve Groups**. `supported_groups` is a TLS extension that informs the server on the elliptic curves the client supports. This feature has been included because it is present in the list of TLS parameters used by JA3 to sign TLS

Feature	**Size**	**Dynamic?**	**In reduced set?**	**Type**
Ephemeral src port	1	No	Yes	Boolean
TLS dest port	1	No	Yes	Boolean
Nb of inbound bytes	1	No	No	Integer
Nb of outbound bytes	1	No	No	Integer
Nb of inbound packets	1	No	No	Integer
Nb of outbound packets	1	No	No	Integer
Flow duration	1	No	No	Integer
SPL	100	No	No	Stochastic matrix
SPT	100	No	No	Stochastic matrix
Byte dist mean	1	No	No	Float
Byte dist std	1	No	No	Float
Byte entropy	1	No	No	Float
Ciphersuites	146	Yes	Yes	Binary vector
Extensions	16	Yes	Yes	Binary vector
Nb of extensions	1	No	Yes	Integer
Supported Groups	36	Yes	Yes	Binary vector
Point Formats	4	No	Yes	Binary vector
Client’s key length	1	No	No	Integer
Certificate’s validity	1	No	Yes	Integer
Certificate’s nb of SAN	1	No	Yes	Integer
Self-signed certificate	1	No	Yes	Boolean
Total	417		208	

Table 3.2: Selected features

session [52], and therefore can be used to identify the client’s application.

- **Elliptic Curve Point Formats**. `point_formats` is another TLS extension that tells the server how elliptic curve points are represented (compact representation or not). It is also used by JA3.

The selected ciphersuite and extensions by the server have been ignored because they are strongly correlated to offered ciphersuites and extensions, since the former are always a subset of the latter.

3.2.4 Robustness of Features

The *Pyramid of Pain* [19] is an empirical method to evaluate the robustness of indicators of compromise (IoC): the higher the score for an IoC, the more difficult it should be for the attacker to find a way around it (figure 3.10). This concept can be applied to our selection of features. There are not all equals, and those who are the most useful to determine the threat level of a flow may also be trivial for attackers to modify as they wish. Following the Pyramid of Pain model, a score is given to each feature based on their resistance against tampering, from weak to robust:

- 1: little effort is required for any kind of attackers to evade the feature

- 2: not as trivial but a reasonably advanced adversary can bypass the indicator
- 3: consequent work is needed to evade the indicator, which is reserved for advanced and determined attackers

Table 3.3 below summarizes the analysis.

Flow Metadata By default on every major OS, source port is chosen randomly [28]. Destination port can be easily configured on the server side. Number of inbound and outbound bytes, inbound and outbound packets are not directly controlled by the attacker and depend on the data exchanged between the endpoints which makes it difficult for attackers to tamper with these parameters.

Distributions All distribution-related features are not directly modifiable but depend on the nature of the data exchanged itself, and are therefore resistant to spoofing. Entropy is a special case, it is also not controllable by the attacker but simply encrypting the payload generates high entropy indistinguishable from benign TLS flows.

TLS Parameters TLS libraries such as OpenSSL and its wrappers (like the `ssl` library for Python) allow the customization of ciphersuites and can also provide strong default lists. Attackers still need to be aware that ciphersuites can be indicators of compromise and that they should not use obsolete or rarely used ciphers. Mozilla keeps updated a list of recommended ciphersuites that are widely used in legitimate traffic [29].

Extensions are also customizable, however not all libraries offer the possibility to do so (`ssl` does not). It is arguably harder to change than ciphersuites but is still under the control of an attacker. The same goes for the number of extensions and elliptic curves related features.

Finally the client's key length depends on the key exchange algorithm offered by the client which can be easily modified.

Certificate Nowadays it has become very easy to obtain a certificate from a legitimate source, for instance with Let's Encrypt[5]. It solves the problem for malware authors of having to resort to self-signed certificates. Certificate's validity can be set by the subject or imposed by the issuer. In Let's Encrypt case, duration is fixed to 90 days by default which is commonplace.

Subject Alternative Name is a certificate extension that can be modified as one wishes during the certificate creation. Some providers also limit the number of SANs (100 for Let's Encrypt for instance).

[5]https://letsencrypt.org/

Feature	Score
Source and Destination ports	1
Bytes in and out	3
Packets in and out	3
Duration	3
Sequence of Packet Lengths	3
Sequence of Packet Times	3
Byte Distribution	3
Entropy	1
Ciphersuites	2
Extensions	2
Number of Extensions	2
Elliptic Curve Groups	2
Elliptic Curve Point Formats	2
Client's Key Length	1
Certificate's Validity	1
Certificate's Number of SAN	1
Self-Signed Certificate	1

Table 3.3: Robustness of features

3.3 Classification of TLS flows

3.3.1 Preprocessing

Raw observations have been processed to obtain more suitable inputs for the models. First, all two-level categorical variables were transformed into dummy variables where classes were represented by 0 or 1. This concerns all features of type *Boolean* of *Binary vector* in table 3.2. The observation label, *benign* of *malicious* was also converted in this way.

Then all columns with null variance, meaning columns in which all values were identical, have been removed since they do not carry any useful information for classification.

Finally, all non-dummy features were scaled in order to have zero mean and unit variance which is expected by some models (logistic regression is one of them) and helps the convergence of the gradient descent algorithm used by several of the models [9]. Unit variance also ensures that the fitting will not depend on the scale in which the samples were measured [36].

3.3.2 Models

Five classification models were tested. All models were implemented in Python 3.6 with `scikit-learn` [48], an open-source machine-learning library.

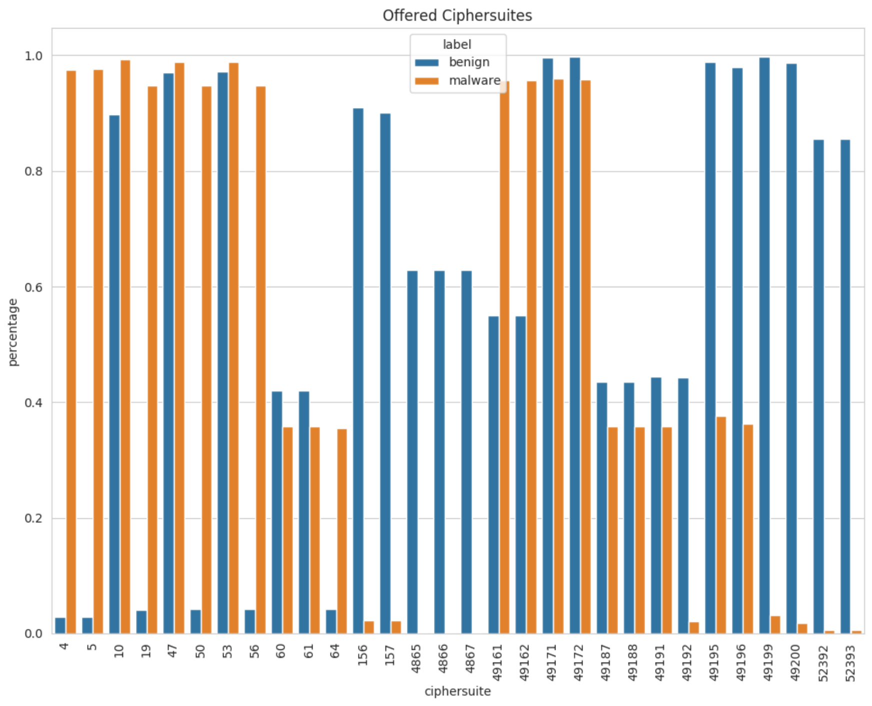
Logistic Regression This is a linear model which assumes a linear relationship between the features and the logarithm of the odds that a sample is malicious. It has the benefit of being highly interpretable and relatively fast to compute but its performance suffers if the linearity assumption does not hold.

Random Forest This model, first presented section 2.3.4, does not assume a linear relationship between the features and the label. It is also highly interpretable but takes longer to train because of the multiple intermediary decision trees to compute.

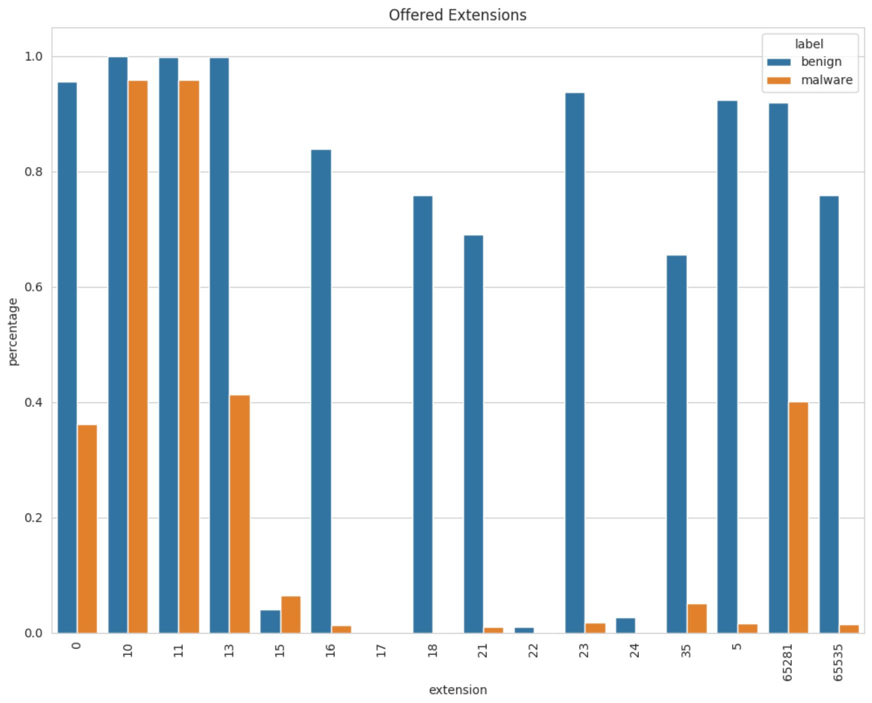
K-Nearest Neighbors This is another model that performs well on non-linear datasets. It labels a flow with the most present class of its K immediate neighbors. It is fast to compute but the parameter K must be chosen carefully and if both malware and benign datasets are very spatially close to each other, accuracy drops.

Linear Discriminant Analysis Another linear model that basically plugs estimates of means and variances for the distribution of flows in each class (two in our case, benign and malware) in another type of classifier, the *Bayes classifier* [36]. The Bayes classifier simply assigns to a flow the class for which the probability the flow is belonging to it is the highest. However LDA assumes that the probability distribution for each class is known in advance or at least accurately hypothesized, which is often not the case. It is not an easily interpretable model neither.

Linear Support Vector Classifier Linear SVC works by finding the best separating (linear) hyperplane with the largest possible margin between the two datasets. However if datasets cannot be clearly delimited by a linear hyperplane in the first place, accuracy suffers. This model is also not interpretable.

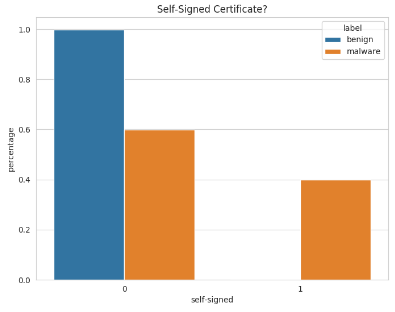


(a) Ciphersuites

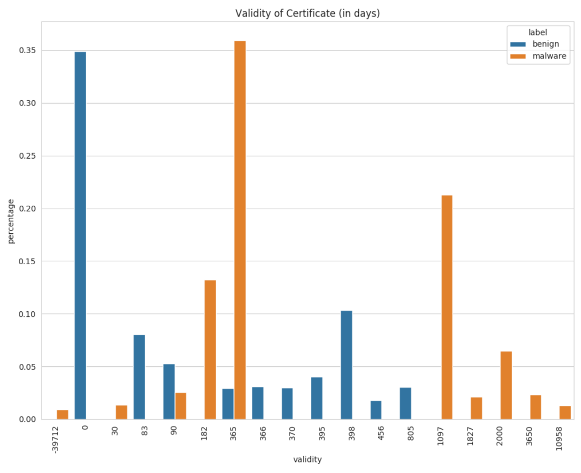


(b) Extensions

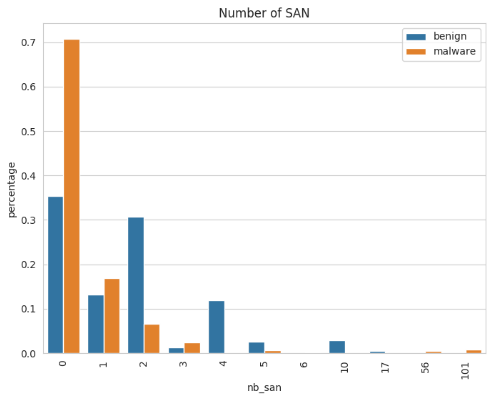
Figure 3.8: Differences in the set of ciphersuites and extensions offered by the client



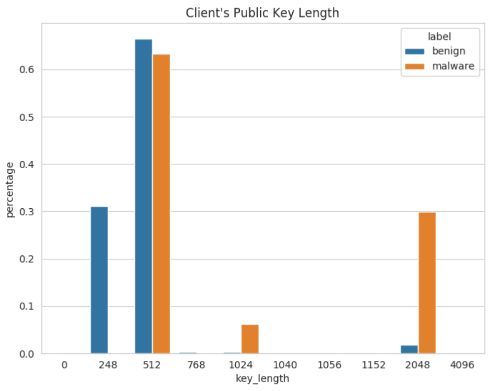
(a) Self-Signed



(b) Certificate's Validity



(c) Number of SAN



(d) Client's Key Length

Figure 3.9: Differences in the client's key length and in the certificate of malicious servers

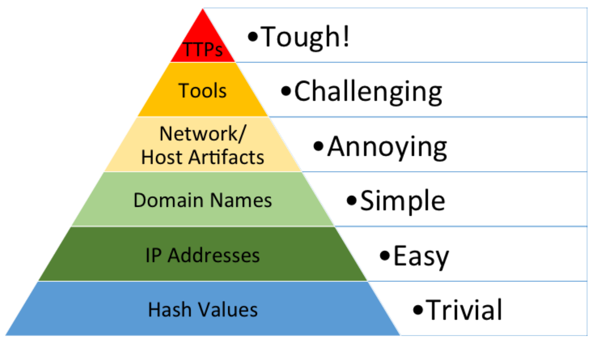


Figure 3.10: The Pyramid of Pain, with examples of IoCs

Chapter 4

Evaluation

This chapter presents the results obtained with the random forest classifier. It begins with details about the methodology used to validate the model, followed by the test results and a comparison of performances when one changes the model or the set of features. The chapter ends with a discussion about the limitations of the classifier and suggestions for future work that could improve the detector.

4.1 Methodology

4.1.1 Testing Datasets

Two sorts of tests were performed to evaluate models. The first tests were done using *Stratified K-Fold cross-validator*. With this method, the shuffled dataset comprised of all malware and benign flows is split into K folds. One of the fold is used as the testing set while the other $K - 1$ folds are used to train the model. The process is repeated K times, with a different fold being selected as the testing set each time. Finally the average metrics of the K runs is computed.

The K folds are not built randomly but are made by preserving the percentage of flows in each malware family (*benign* is considered as a family of its own). This stratification ensures that a fold does not inadvertently contain a disproportionate amount of flows from one malware family only, which could bias the model during training or skew the testing results if that fold is used for validation. The default value of K was fixed to 10.

The other validation method was to test the classifiers on a fresh dataset that was never used in the training process. To build the dataset, new samples were collected during July 2019. They come from the same sources described section 3.1.1. Again, network traffic from Lastline's London and Redwood office was collected for approximately two weeks to build the benign dataset. The malware samples come from capture files saved by Lastline where malware activity was detected during July 2019, and from https://www.malware-traffic-analysis.net/index.html for the month of July 2019.

The source distribution is shown figure 4.1. In total, the test dataset contains 21457 benign flows and 1473 malware flows. Samples of malware that use TLS are harder to find and represent 6.4% of all test flows. However a low amount is not a bad thing since the percentage of malware in the testing set should remain close to the true percentage of malware in the wild. This is to avoid spatial experimental bias in testing which result in inflated precision, as stressed out by the Tesseract paper [49], due to the fact that the number of false positives naturally decreases when reducing the number of benign flows.

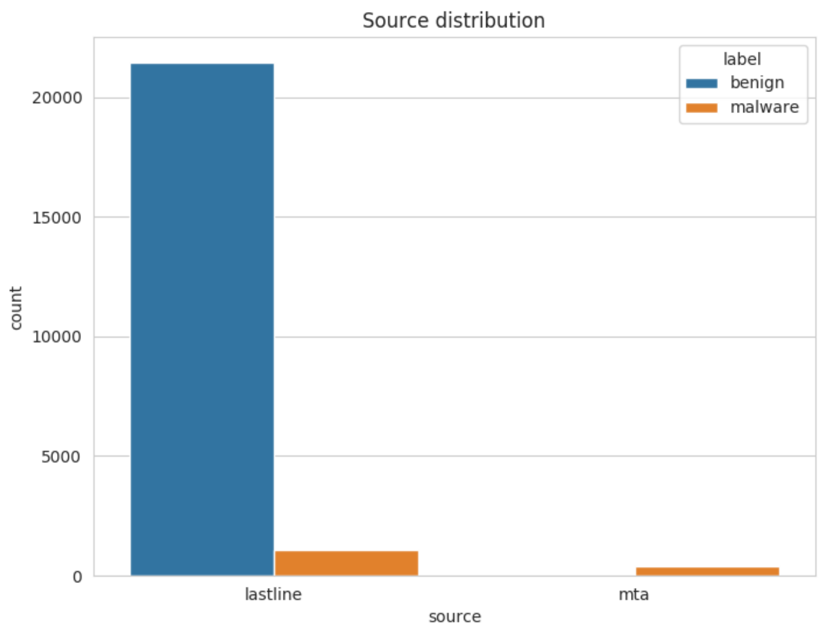


Figure 4.1: Source of flows used for validation

4.1.2 Testing Metrics

To evaluate the performances of a model, we use several metrics presented below.

Binary Classification In the rest of this report, *Positive* (P) refers to malicious flows and *Negative* (N) refers to benign flows. The terms *True* is employed when the classifier makes a correct prediction and *False* when it does not. Based on these definitions, table 4.1 presents the terms employed to evaluate the models. After running a classifier on the test datasets, all these metrics can be displayed using a *confusion matrix* which follows the same structure as table 4.1.

In addition, *False Positive Rate* (FPR) and *True Positive Rate* (TPR) are defined as follows:

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN}$$

		Predicted label	
		benign	**malware**
True label	**benign**	True Negative (TN)	False Positive (FP)
	malware	False Negative (FN)	True Positive (TP)

Table 4.1: Binary classification terms

$$TPR = \frac{TP}{P} = \frac{TP}{TP+FN}$$

FPR measures the proportion of benign flows mistakenly labelled as malicious. TPR measures the proportion of malware flows correctly labelled as such by the classifier.

Accuracy, Precision, Recall, F_1-score The main classification metric is the accuracy which gives the proportion of correctly labelled flows with respect to the total number of samples in the dataset. It is defined as:

$$Acc = \frac{TP+TN}{N} = \frac{TP+TN}{TP+TN+FP+FN}$$

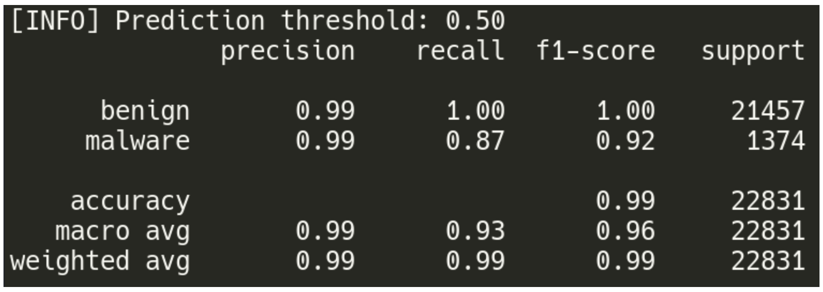
Other metrics that are returned by the classifiers developed for this project are:

- **Precision**, which gives the proportion of truly malicious flows among all flows labelled malicious by the classifier: precision $= \frac{TP}{TP+FP}$.
- **Recall**, which gives the proportion of malicious flows correctly found by the classifier among all truly malicious flows: recall $= \frac{TP}{TP+FN}$. Note that this is just another name for the *True Positive Rate*.
- **F$_1$-score** which returns the harmonic mean of the precision and recall:

$$\mathbf{F_1} = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2\,\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

The three scores listed above can also be measured for benign samples. For instance, the precision for legitimate flows is defined as the proportion of truly benign flows among all samples classified as benign. Running the classifier on a test dataset returns the metrics for the two classes and the average weighted by the number of samples in each class. An example of a test run is shown figure 4.2.

ROC curve A *Receiver Operating Characteristic* curve is a plot which shows the relation between the FPR and TPR as the classification threshold varies. An example from scikit-learn's User Guide [48] is presented figure 4.3. The dotted blue line represents the ROC of a model that randomly classifies into one of the two classes. Ideally, the ROC curve should reach the top left corner where the false positive rate would be null and the true positive rate would reach its maximum for one specific threshold. Another benefit of this curve is the *Area Under the Curve*: computing the integral of the ROC curve returns the probability that a random malicious sample will have a higher classification score than a random benign one (if high scores indicate maliciousness) regardless of the threshold chosen [16].



```
[INFO] Prediction threshold: 0.50
              precision    recall  f1-score   support

      benign       0.99      1.00      1.00     21457
     malware       0.99      0.87      0.92      1374

    accuracy                           0.99     22831
   macro avg       0.99      0.93      0.96     22831
weighted avg       0.99      0.99      0.99     22831
```

Figure 4.2: An example of the results returned by the LDA classifier on the test dataset

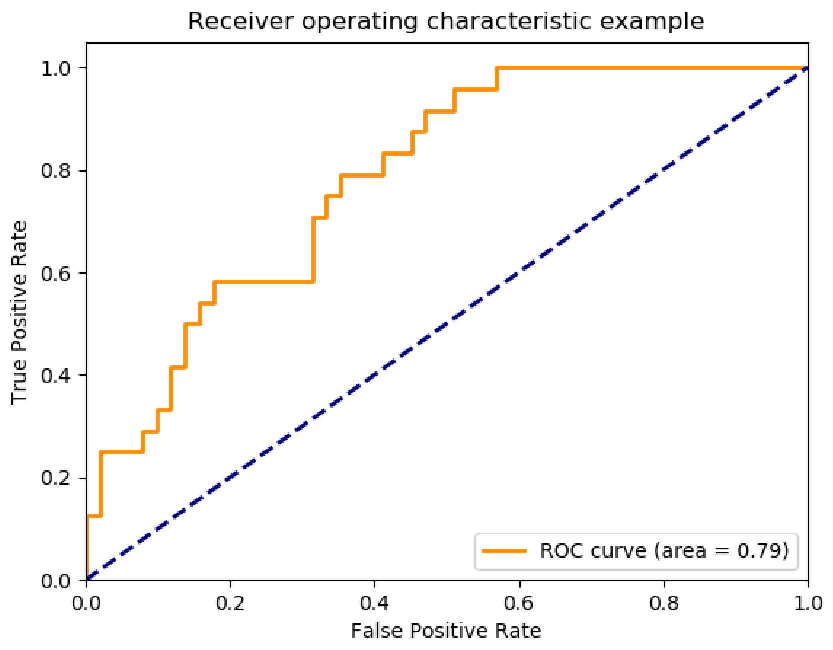


Figure 4.3: A ROC curve

4.2 Results

As mentioned section 3.3.2 and 3.2.3, five different models were tested and two different sets of features were built. We present here first the results for one model and one set: the random forest classifier on the reduced set of features.

The choice of the reduced set instead of the full set comes from the fact that ultimately this project will be merged into Lastline's intrusion detection platform and must be compatible with the features made available. The results obtained with the full set of features are exposed section 3.2.3.

The analysis of the random forest model over another one is motivated by a prior comparison between models presented section 4.2.4, where random forest classifier was generally the best performing one. It was also the model of choice in some of the work from Cisco [12, 14], whose results give us an idea of the performances that are reachable with this model and can be used as a benchmark against which to compare our own classifier.

4.2.1 Before Training

Data Visualization

One way to visualize multi-dimensional data is by performing *Principal Components Analysis*. *An Introduction to Statistical Learning* [36] defines PCA as a way to "*summarize a large set of correlated variables with a smaller number of representative variables that collectively explain most of the variability in the original set*". This technique can be used for data visualization by projecting the scaled training dataset onto two principal vectors, effectively reducing the dimensions from more than 200 to 2. Figure 4.4 shows the principal components analysis. The resulting projection is not interpretable but still shows a clear global separation between benign and malware samples. This is reassuring and legitimises the idea that the two classes are quite different.

Another interesting thing to note is that there are many more malware points indistinguishable from benign points than the opposite. We could expect the trained classifier to be mislead by this phenomenon and to return more false negatives than false positives.

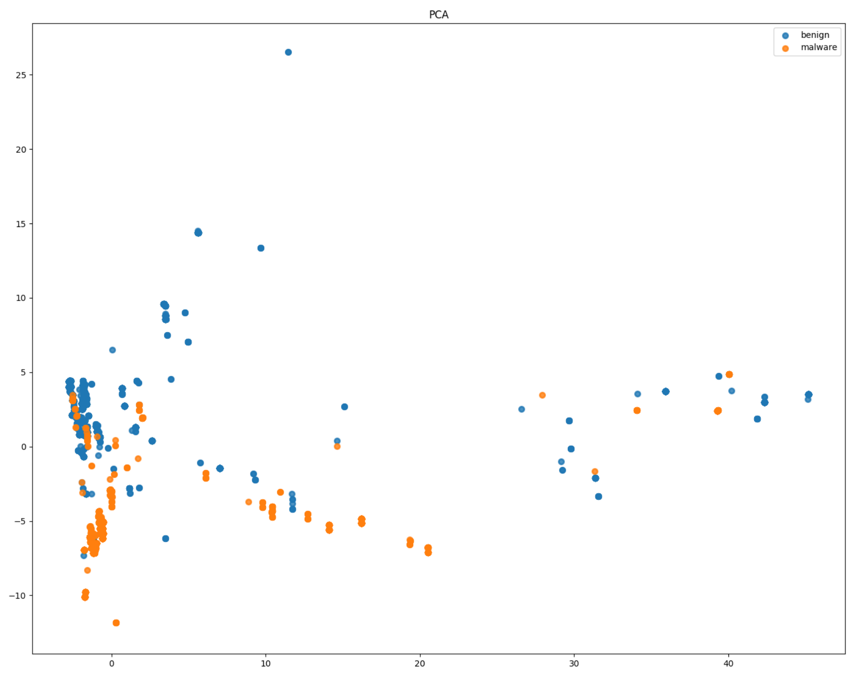


Figure 4.4: PCA of the training dataset

Choice of the Model Parameter

The random forest classifier is parameterized by N, the number of trees. The model basically builds N trees with a random subset of features to be used by each one then finally averages the predictions of all trees, which lowers the overall variance of the model. However this is not a critical parameter: a high number of trees results in more computation time but does not lead to overfitting. The number of trees only serves to smoothen the average output of trees, which in turn increases performances on never-seen before datasets [36].

Figure 4.5 plots the average accuracy of 100 runs of the random forest classifier on the testing dataset for different values of N. The graph is quite irregular but there is a clear tendency for the accuracy to rise alongside the parameter. Based on that plot, the number of trees was chosen to be 130 for the rest of the project, which offers a good balance between accuracy and computation time.

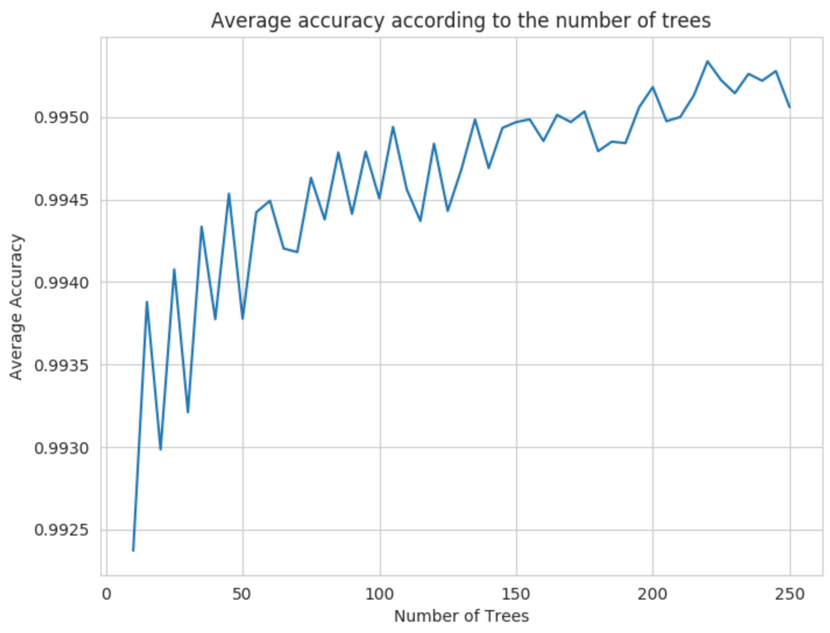
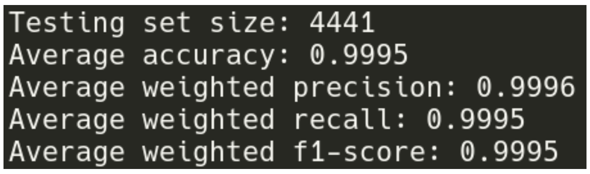


Figure 4.5: Accuracy with respect to the number of trees

4.2.2 Model Performance

Cross-Validation

The classifier capabilities were first measured via 10-fold cross validation, see 4.1.1. The results are displayed figure 4.6. We computed and kept the average *weighted* metrics because the two classes are not balanced. The results looks very good: a near 100% accuracy and almost all malware samples were found.



```
Testing set size: 4441
Average accuracy: 0.9995
Average weighted precision: 0.9996
Average weighted recall: 0.9995
Average weighted f1-score: 0.9995
```

Figure 4.6: Results of 10-fold cross validation

A likely explanation to such high performances is *temporal experimental bias* [10, 49]. This bias is caused by the nature of the data: TLS flows are subject to *concept drifting* meaning that they become obsolete over time. New malware are introduced, users' habits change etc. Therefore time does have an impact on the model, however K-fold cross validation ignores that and may select training flows posterior to testing flows, which results in a positively biased classifier.

In this first testing phase with cross-validation, the model assumes that flows are identically distributed in time. The classifier is then trained on folds that include flows both anterior and posterior to flows in the testing fold: the datasets are time-homogeneous. Therefore a test flow is more likely to be correctly labelled than in the case where the two datasets are time-heterogeneous: all training flows are anterior to test flows. This latter case is often what is available in practice, when the classifier is being fed newly received or sent data.

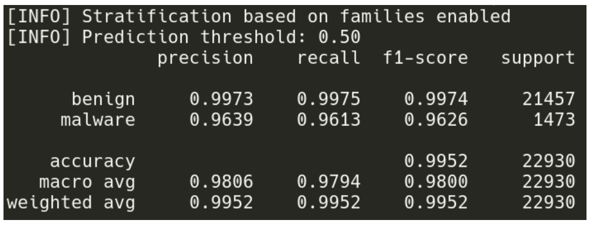
Fresh Testing Dataset

Figure 4.7 shows the resulting metrics as well as the corresponding confusion matrix when testing on the completely fresh dataset, whose construction has been detailed section 4.1.1. This dataset contains almost exclusively flows posterior to the ones in the training set.

The scores have slightly dropped from 10-fold cross-validation but are still high. The confusion matrices also confirm our prediction of section 4.2.1: the classifier generates more false negatives than false positives which affects the malware recall, 96.13% according to figure 4.7a. This is not surprising, since it is more likely that a malware would have a TLS configuration looking normal, for instance by using a standard TLS library with strong defaults, than for a normal client to tweak and use unusual TLS parameters that would be flagged as suspicious.

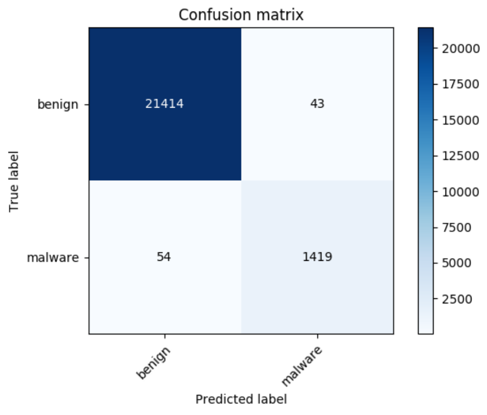
False Positive Threshold

High accuracy is nice to have, but a low number of false alarms is arguably a more important factor for the usability of an intrusion detection system. The two metrics are not correlated due to the base-rate fallacy [17]: the very high number of benign flows compared to the very low probability of infection may result in a large amount of false positives regardless of accuracy. This is undesirable since it can quickly become a management burden, to the point even legitimate alerts could be ignored.

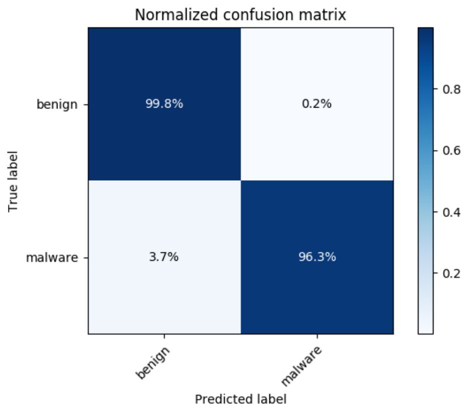


```
[INFO] Stratification based on families enabled
[INFO] Prediction threshold: 0.50
              precision    recall  f1-score   support

      benign     0.9973    0.9975    0.9974     21457
     malware     0.9639    0.9613    0.9626      1473

    accuracy                         0.9952     22930
   macro avg     0.9806    0.9794    0.9800     22930
weighted avg     0.9952    0.9952    0.9952     22930
```

(a) Scores



(b) Confusion matrix



(c) Normalized confusion matrix

Figure 4.7: Results on the fresh dataset

We consider a medium sized network of about 100 hosts. Given that Lastline's London office with a daily presence of around 10 employees generates about 5000 TLS flows per working day, we can extrapolate and say that a network of this size would generate 50000 TLS flows per day. In the following:

- I denotes the infection event "*a TLS flow comes from a malicious source*" and $\bar{I}$ is the complementary event.
- A and $\bar{A}$ relates to the raising or not of an alarm from the detector.

Therefore the key value we want to maximize is $P(I \mid A)$, the probability that a host in the network was indeed infected given that an alarm was raised. Using Bayes' theorem we get:

$$P(I \mid A) = \frac{P(A \mid I) \cdot P(I)}{P(A)} = \frac{P(A \mid I) \cdot P(I)}{P(A \mid I) \cdot P(I) + P(A \mid \bar{I}) \cdot P(\bar{I})}$$

- $P(A \mid I)$, the probability that an alarm is raised when an infection happens, is given by the true positive rate (TPR) of the classifier.
- $P(A \mid \bar{I})$, the probability that an alarm is wrongly raised, is given by the false positive rate FPR.
- $P(I)$ represents the proportion of malicious TLS flows over all generated flows. This figure is not known in advance but let's suppose that a host has 0.5% chance of being infected anytime during the day and that an infection generates 5 TLS flows per day. Therefore $P(I) = 100 \cdot \frac{5}{1000} \cdot \frac{5}{50000} = 5 \cdot 10^{-5}$, five flows in one hundred thousand would be malicious for that network.

- $P(\bar{I})$ is then derived from $P(I)$ by $P(\bar{I}) = 1 - P(I)$.

We can use the ROC curve of the random forest classifier to see how the FPR and TPR change when the threshold varies. The curve figure 4.8 was computed on the predictions returned by the model on the fresh dataset and by varying the threshold above which a sample is labelled as malicious.

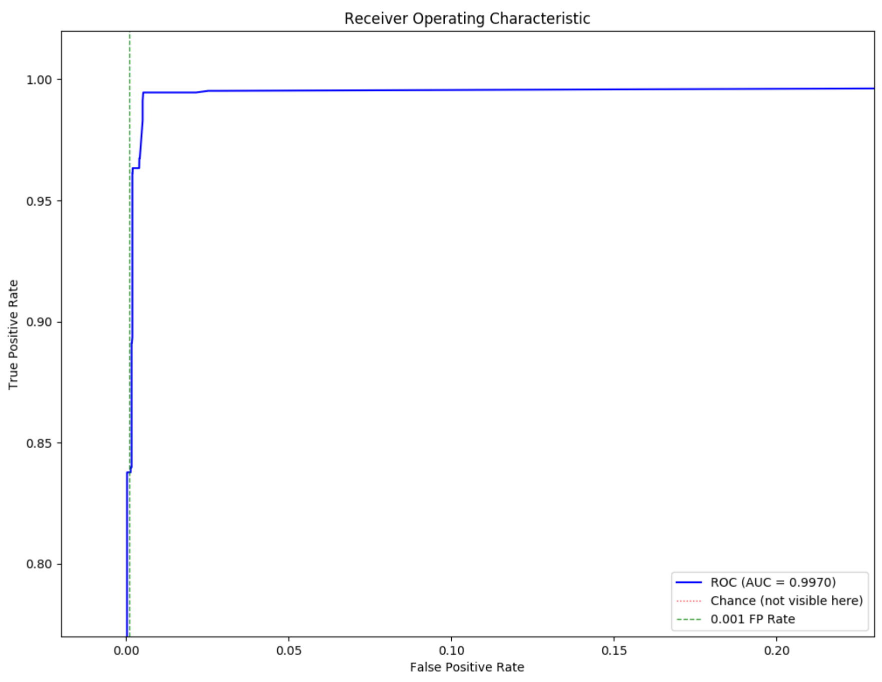


Figure 4.8: ROC Curve of fresh dataset

The ROC returns these values:

- For a threshold of **0.10**, the TPR is 0.9946 and the FPR is 0.0134. $P(I \mid A) = 0.0037$ (0.37%).
- For a threshold of **0.50**, the TPR is 0.9633 and the FPR is 0.0021. $P(I \mid A) = 0.0234$ (2.34%)
- For a threshold of **0.92**, the TPR is 0.8344 and the FPR is 0.0002. $P(I \mid A) = 0.1726$ (17.26%)

Keeping the default threshold of 0.5 would result in the low true positive rate of 2.34%. Even with a high threshold, the probability that there was an infection given an alert only reaches 17.3%. This figure, which may seem to contradict the good performances obtained previously, is due to the fact that the ability of the detector to find malicious flows is completely dominated by the number of benign flows. However this number must also be put in perspective: setting the threshold to 0.92 results in a false positive rate of 0.0002, which gives:

$$FP = FPR * N = 0.0002 \times 50000 \times (1 - \frac{5}{100000}) \approx 10$$

$$TP = TPR * P = 0.8344 \times 50000 \times \frac{5}{100000} \approx 2$$

There would be about 10 false alerts per day, which remains manageable and would be bearable if the detector is successful at detecting real threats often enough.

4.2.3 Influence of Features

Best Features

One of the advantages of the random forest classifier is that it is highly interpretable, meaning that we have access to the criteria on which the classifier bases its predictions. After training, each feature is given a coefficient indicative of its importance. This coefficient is computed from the mean decrease across all trees in the error rate (the *Gini index* for the random forest classifier) induced by the use of the feature [36].

Figure 4.9 presents the 50 most important features. The classifier was trained one hundred times and the coefficients shown in the figure are the average across all runs. The name associated to each code can be found appendix C.1 for ciphersuites, C.2 for extensions and C.3 for elliptic curve groups.

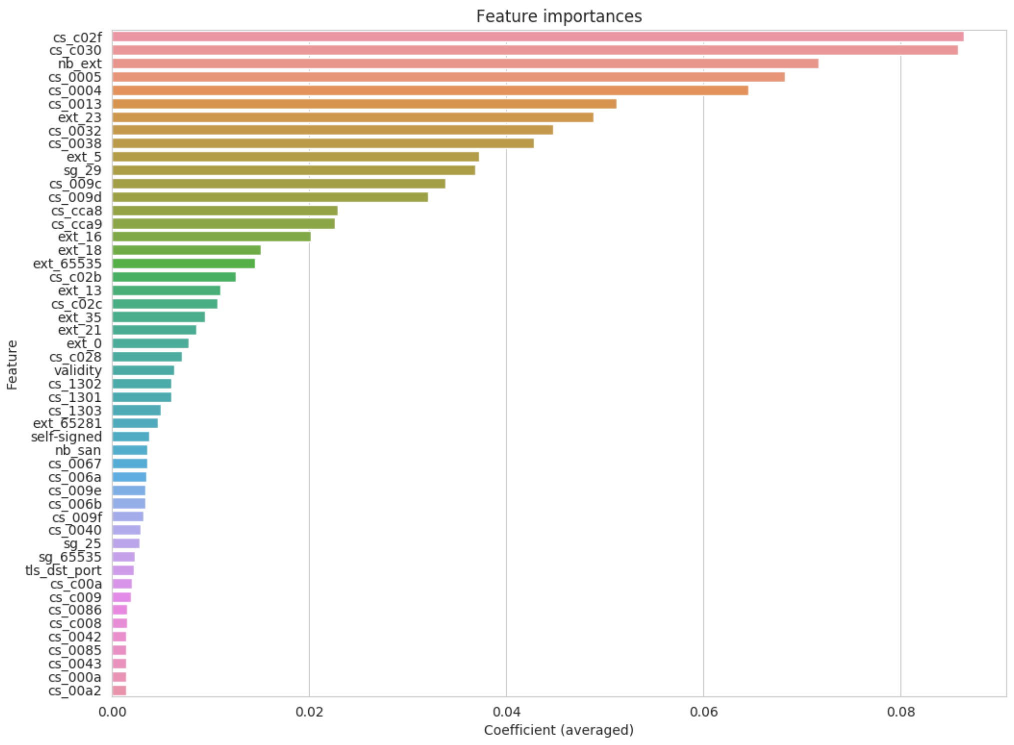


Figure 4.9: The top 50 features averaged across 100 runs

The most discriminant features are the ciphersuites and the extensions used by both classes, which account for 32 and 10 of the top 50 features respectively. The top 4 ciphersuites are:

1. `cs_c02f` (`TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`): a recommended ciphersuite [51] introduced in TLSv1.2, mostly used by normal TLS flows.
2. `cs_c030` (`TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`): another recommended ciphersuite used in benign traffic.
3. `cs_0004` (`TLS_RSA_WITH_RC4_128_MD5`): an insecure ciphersuites due to the use of the deprecated RC4 algorithm, mainly used by malware.
4. `cs_0005` (`TLS_RSA_WITH_RC4_128_SHA`): an insecure ciphersuites similar to the previous one.

The top 4 extensions are:

1. `ext_23` (`extended_master_secret`): mostly used by normal flows as seen figure 3.8b.
2. `ext_5` (`status_request`): mainly used by benign flows.
3. `ext_16` (`application_layer_protocol_negotiation`): mainly used by benign flows.
4. `ext_18` (`signed_certificate_timestamp`): mainly used by benign flows.
5. `ext_65535` (`unknown`): this extension is a placeholder for all extensions seen in the testing set but absent or ignored from the training set (see section 3.2.3 for more details). This feature is more representative of benign flows: this is due to Google Chrome's *GREASE* mechanism which inserts random extensions to make sure webservers ignore unknown values [1].

The features related to TLS parameters dominate the top 50, though some of them are absent: elliptic curves point formats are far down the full ranking. TLS features are followed by features related to certificates which are all present in the top list. Regarding flow metadata, only the destination port is among the 50 bests and the source port is ranked 68. The bottom of the full list is filled with ciphersuites, extensions and elliptic curve groups that are equivalently or rarely used by the two classes.

Figure 4.10 shows how the model performs when considering only a subset of the features. A greedy approach was taken to built that subset, based on the ranking presented above: features were selected by group of five starting with the five first features then by going down the ranking, without considering potential interactions between features which may have resulted in better accuracy. The plot shows the average accuracy across 100 runs steadily increasing, reaching its maximum for the entire set of features. We have used all 208 features in this project, but it may be interesting to use only a subset of those, if storage is a scarce resource for instance or if the dataset becomes too large. According to the figure, the 80 top features would provide good accuracy (0.992) and after that number, accuracy does not increase as rapidly as before.

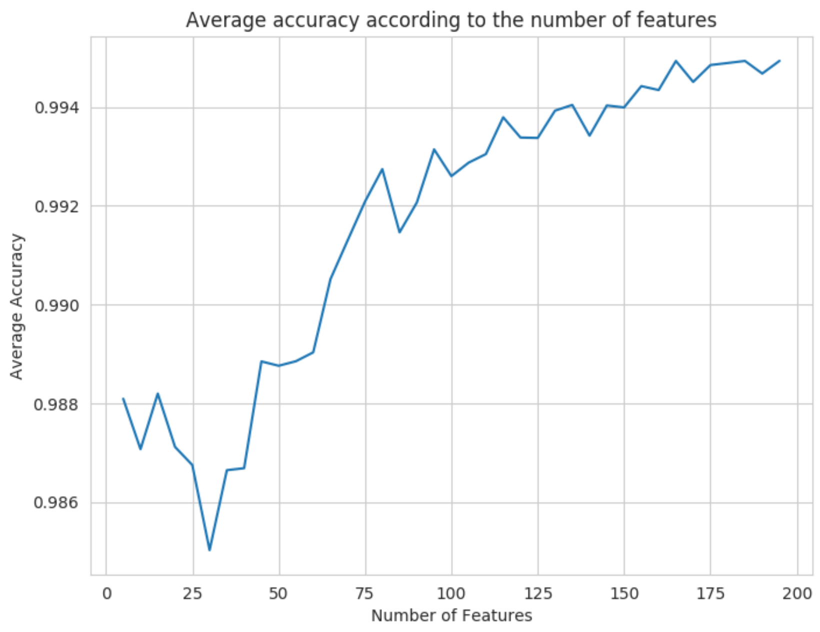


Figure 4.10: Accuracy across 100 runs according to the number of features

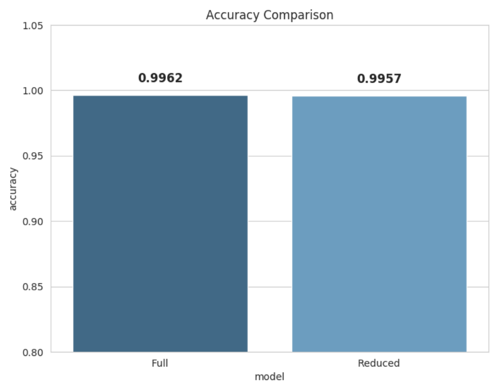
Reduced and Full Set of Features

All prior and later results were obtained with the reduced set of features. In this section, we consider the full set. Figure 4.11 shows the differences in accuracy, precision recall and TPR for $FPR = 0.02\%$ between the two.

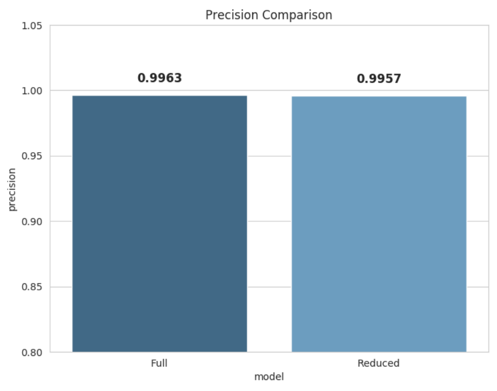
The results are quasi-identical when looking at accuracy, recall or precision. The main improvement is a sharp increase in the true positive rate when considering a very low FPR, which is not negligible as we have seen section 4.2.2.

The top 50 features for that set is presented 4.12. Having access to all $400+$ features for classification is not always possible which is the case for Lastline's serialized flows. However if they are available there is no reason they should not be used, at least for the benefit of an increase in the TPR.

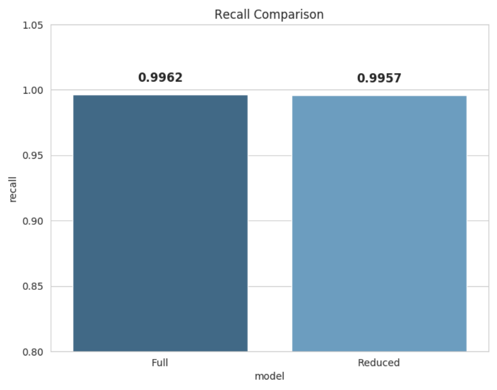
The first 15 best features basically do not change. Features related to the sequence of packet times and lengths do have an impact, as well as features related to byte distribution: number of inbound bytes is ranked 17th, entropy is 23rd and number of outbound bytes is 46th. This confirms that there are indeed differences in the frequencies of normal and infected clients' communications with a server. Other discriminant features are the client's public key length, which could already be seen figure 3.9d, and the flow duration (figure 3.6a). The bottom of the full ranking is filled with features related to the sequence of packet lengths and times. It is not surprising since these two categories alone account for 200 features, half of the full set.



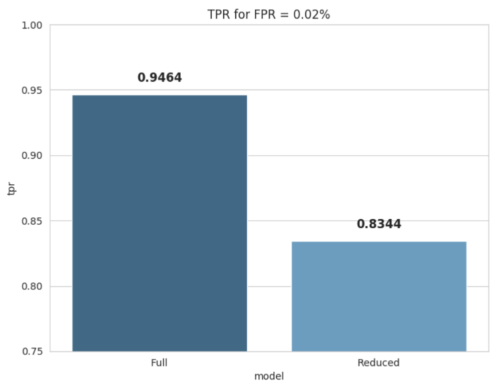
(a) Accuracy



(b) Weighted Precision



(c) Weighted Recall



(d) TPR for $FPR = 0.02\%$

Figure 4.11: Differences between the full and the reduced set of features

4.2.4 Comparison of Models

Five other classification models were tested, introduced section 3.3.2. The results are presented figure 4.13.

The results may seem very close to each other, but when considering a dataset of more than ten thousand flows, even a change in the third decimal covers more than ten flows. With that in mind, the best overall classifier is the random forest model which has the highest accuracy, recall and precision among all five models. The K-neighbors model (with $K = 5$ here) has a higher true positive rate when considering a FPR of 0.02% which makes it an interesting alternative to random forest and may be an evidence of a large spatial gap between the two datasets.

The fact that the two best performing models are not linear, unlike the three others, supports the idea that a true label cannot be simply derived from a linear combination of the features. Non-linear approaches are therefore best suited for our goal of classifying flows defined by many parameters. Even so, all five models still yield satisfying results which at least go in favor of the hypothesis that the two classes are

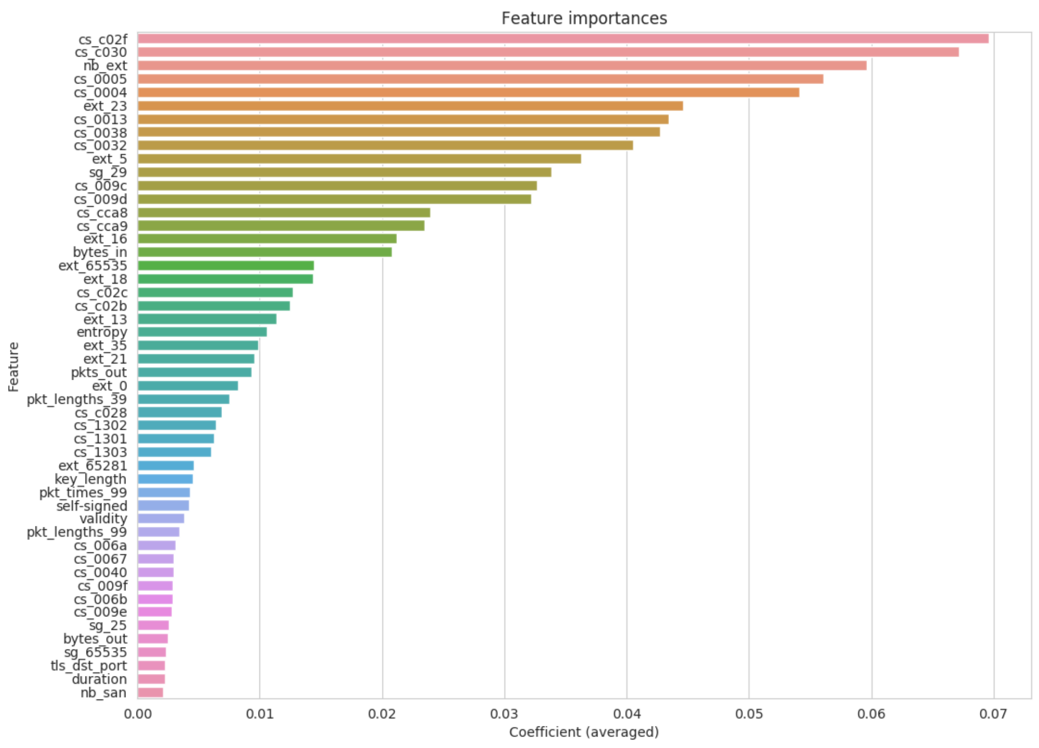


Figure 4.12: The top 50 features for the full set

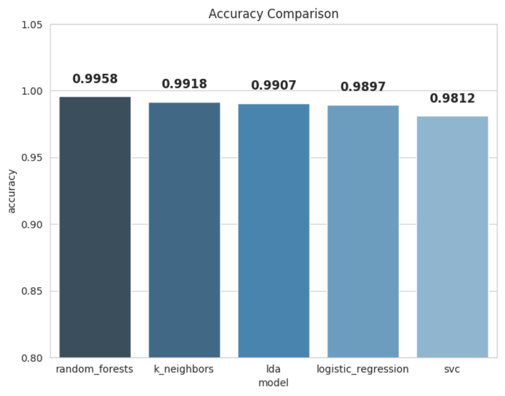
well-separated.

4.3 Lastline's Detector

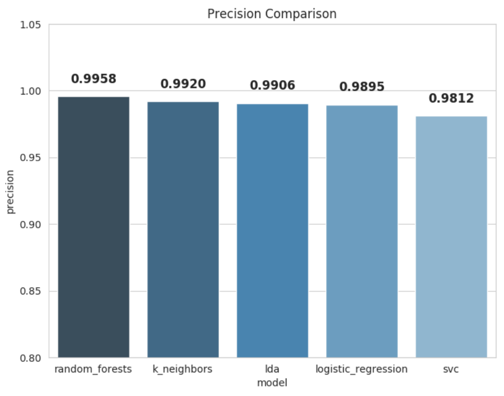
The classifier was finally integrated into Lastline's main product, *Lastline Defender*. Lastline Defender is a detection system that collects and analyses the traffic of customers looking for abnormal or malicious activities. Customers' traffic is sent to devices called *sensors* that pre-process and forward the network data to a service also hosted on sensors called *Llanta*, Lastline's Advanced Network Traffic Analysis. Llanta is composed of detectors that run periodically and raise alerts whenever an observation triggers their logic.

The integration took place in four steps:

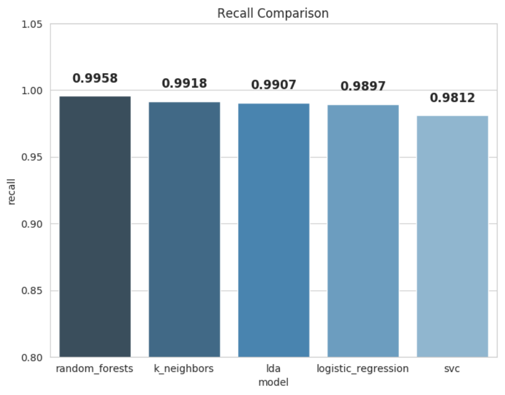
1. The trained classifier was converted into a Debian package that could be installed into Llanta and be used by any detector.
2. A Python module was created to convert a raw TLS flow object generated by a sensor into a TLS vector holding all features used by the classifier.
3. A detector was developed to make the connection between the data from the sensors and the classifier. It feeds on TLS flows only. How it operates is detailed



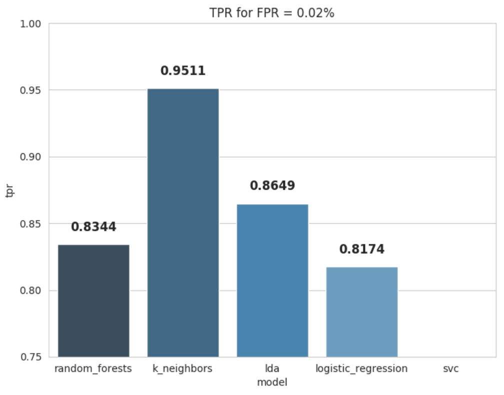
(a) Accuracy



(b) Weighted precision



(c) Weighted recall



(d) TPR for $FPR = 0.02\%$ (SVC does not provide FPR and TPR *w.r.t* threshold)

Figure 4.13: Performances of different models

below.

4. Finally, extensive documentation was written, both in the internal documentation platform (*Confluence*) and in the source code. All capture files and datasets used for testing and training were arranged and saved into Lastline's data repository. This step was to make improvements to the project easier to carry out and to facilitate subsequent trainings of the classifier.

The detector instantiates a TLS vector from each raw TLS flow it receives from the sensors. It then forwards the vector to the classifier which returns its prediction for that flow. When the detector has seen more than a certain number of flows labelled as malicious originating from a single host, an alert is raised. That *count threshold*, by default equal to five flows, is there to limit the number of false alerts and can be modified by the customer. A different threshold, the *classification threshold*, is used to classify a flow as malicious. It has been set by default to 0.92 for the reasons explained section 4.2.2. Unfortunately the detector was not yet fully deployed when the project was drawing to a close, so the results of the classifier in real conditions cannot be presented in this report.

4.4 Discussion

4.4.1 Related Work

The interest on the detection of TLS malware is relatively recent: some companies and individuals have just started to advertise their work on the subject [18, 54]. But as of August 2019, the most comprehensive (and public) study still comes from Cisco. They have published their results for several models [14, 12] including the random forest classifier with 125 trees. Table 4.2 summarizes the malware recall obtained by our classifier and Cisco's, based on the scores they have released in [12].

	Reduced set		Full set	
	0.5	**0.9**	**0.5**	**0.9**
Cisco	97.67%	80.76%	99.35%	85.80%
Lastline	96.13%	83.44%	97.11%	94.64%

Table 4.2: Malware recall of Lastline's and Cisco's classifier

The sets of features between our classifier and Cisco's, both reduced and full, differ. In particular, *reduced set* for Cisco refers to the full set they uses without SPL and SPT features. The features they have selected to train their models have been presented section 2.3.4.

Overall the results are quite similar. Our classifier seems to perform better for the full set of features with a threshold of 0.9, but this kind of differences can be attributed to a lot of factors: the choice of features, the size of the training dataset, its contents, the number of trees used... There is no clearly better detector, however this comparison shows the performances a classifier targeting malicious TLS flows can expect to achieve.

4.4.2 Limitations

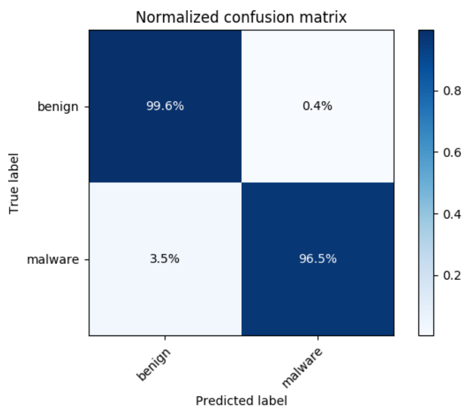
On Datasets

The performance of the classifier depends a lot on the quality of the training data. The malware dataset can be common to all usages since it has been built such that every malware using TLS is included in it indiscriminately. However it should be kept up to date with new malware samples while older ones should be removed regularly so that the classifier is able to detect recent threats. Finding malware samples is not an easy task since it is difficult to be sure that a flow is malicious in the first place and because manually collecting traffic from sandboxed malware is time-consuming and requires someone familiar with the procedure.

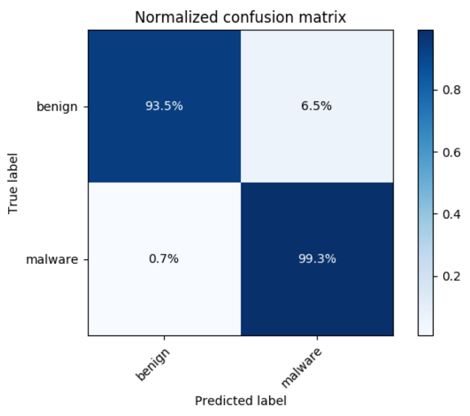
As a result of all this, the classifier is not suited to discover completely new threats since it has been trained on known malware only and because the usefulness of the malware dataset is limited in time. Malware come and go for periods spanning from weeks to years so the current dataset may only be valid for some months or a couple of years at best.

Benign flows should be easier to collect, however the dataset must be built from scratch for each different setting. Indeed, the nature of the traffic as well as operating systems clients are running impact a lot the performances of classifiers. To demonstrate this, figure 4.14 shows the normalized classification matrices of the random forest classifier in two scenarios:

1. In figure 4.14a, the training benign dataset was composed of flows from Lastline's London and Redwood offices and the model was tested on benign flows coming from Lastline's Redwood office only. London's office only has Linux and MacOS machines whereas in Redwood, the main OS is Windows with some machines running MacOS.
2. In figure 4.14b, the training benign dataset was from the London office only and the model was tested on the Redwood office.



(a) Training on London and Redwood datasets



(b) Training on London dataset only

Figure 4.14: Differences introduced by the change of the training dataset

We can see a sharp rise in the number of false positives in the second scenario. This is due to the fact that all machines in London's office are running Linux or MacOS, whereas the malware dataset contains a lot of traffic from Windows machines. It is very likely that in this scenario the classifier learns to distinguish between different operating systems rather than malware from normal traffic. Then, since the Redwood office is composed of a majority of Windows machines, the false positive rate increases.

On Classification

The classifier can be tricked. Section 3.2.4 details the robustness of each feature and from the table 3.3, we see that all features in the reduced set can be modified by a determined malware author so that communications would be almost indistinguishable from normal ones. One way to make this task harder for attackers would be to keep the list of features secret, or to use the full list of features (see table 3.2) which incorporates indicators harder to tamper with.

Another issue to mention is that it is quite difficult to validate results manually, without having prior access to the true labels. This is a direct consequence of the complexity of the original problem: isolating a TLS flow and classifying it as bad or not is a hard task for humans. This is also why the set of features is large (more than 200 at least), because there is no single parameter that is a sure sign of infection. This means that false alerts are particularly unwelcome since analysts responsible for checking on alerts would have to spend a lot of time on each of these false positives.

That is why the training phase and the choice of classification parameters should be given a lot of care. False positives cannot be avoided, however the classification threshold should be set so that there are as few as possible, as mentioned in section 4.2.2. The number of false positives is directly linked to the size of the network: more traffic generates more TLS flows, thus a bigger number of flows unintentionally labelled as malicious which can become unmanageable after a certain point. The best threshold is therefore unique to each setting. High thresholds might well result in more false negatives, but usability and trust in the results are also important factors to consider especially in a commercialized product. And there are still other detectors present to potentially intercept these false negatives using different methods.

4.4.3 Possible Improvements

On the Construction of Datasets

The capture files were collected manually which took a significant part of the time spent on the project. That process could be automated in the future. The benign dataset on which to train the classifier could be rebuilt periodically to ensure the freshness of the data. The difficulty lies in the fact that it is practically impossible to be certain that the whole capture is benign. One solution would be to collect traffic from a few trusted and well-protected hosts representative of the company's traffic that would be considered clean. Another idea would be to make use of other detectors to filter out as many bad flows as possible from capture files.

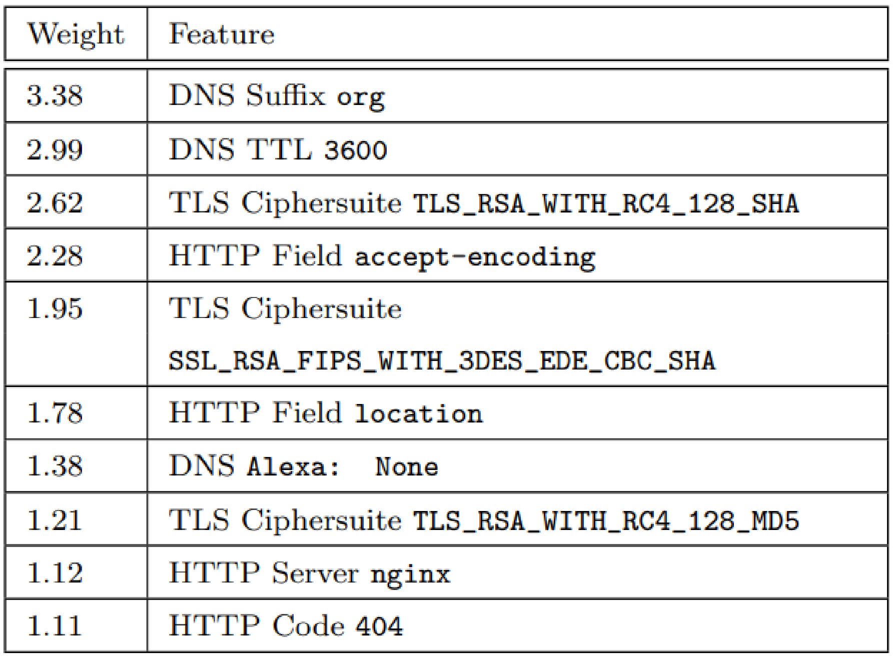
Collecting malware samples was a lengthy process, worsened by the lack of information from online providers on whether or not a malware actually uses TLS. A way to improve that phase would be to build and regularly update a collection of samples of malware that are known to abuse TLS. When a malicious flow would be detected, either by signature checking, by another detector or by the classifier itself (and after an analyst would have confirmed that the sample is indeed malicious), it would be saved and reused in future trainings. Such a database would also ease the search online for samples similar to malware families already present in the collection, which would in turn contribute to the growth of the database.

On the Classifier’s Performances

Alone, the detector may raise a lot of false positives, especially if the decision threshold and the classifier’s parameters are not carefully chosen (sections 4.2.2,4.4.2). A way to limit that phenomenon would be to combine the TLS detector with other detectors which also look into TLS flows. For instance, Lastline has developed a *Domain Generation Algorithm* detector which picks up random-looking domain names hinting at the presence of malware. This detector could be used on the SNI (*Server Name Indication*) TLS extension which is not currently used by the classifier except to filter out flows (section 3.2.1).

Another detector developed within the scope of this project, not mentioned in this report to avoid going off-topic, is a JA3 profiling detector. The detector extracts JA3 (section 2.3.3) from TLS flows and builds a profile of typical JA3 hashes for each host in the network. Then after that training period, new JA3 hashes absent from a host’s profile and its neighbors’ are flagged as suspicious. Combining other detectors with the classifier would serve to improve precision and lower the false positive rate.

Robustness against determined attackers, discussed section 3.2.4, also contributes to the usefulness of an intrusion detection system. To improve robustness, the classifier could take into account features from other types of flows rather than limit itself to TLS. This has been done in another paper from Cisco [13]. In this paper, DNS queries made prior to the TLS handshake as well as headers from "*HTTP flows originating from the same source IP address within a 5 minute window*" were used as features. The paper includes a top 10 features indicative of malware which is reported figure 4.15.



Weight	Feature
3.38	DNS Suffix `org`
2.99	DNS TTL `3600`
2.62	TLS Ciphersuite `TLS_RSA_WITH_RC4_128_SHA`
2.28	HTTP Field `accept-encoding`
1.95	TLS Ciphersuite `SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA`
1.78	HTTP Field `location`
1.38	DNS `Alexa: None`
1.21	TLS Ciphersuite `TLS_RSA_WITH_RC4_128_MD5`
1.12	HTTP Server `nginx`
1.11	HTTP Code `404`

Figure 4.15: Top 10 features from different protocols (taken from [13])

These contextual features improve robustness by including hard-to-spoof features, such as whether or not the domain queried is present in Alexa top 1 million list. The accuracy does not improve significantly compared to results of section 4.2.2, but the TPR when considering a very low FPR increases greatly: from 77.88% (83.44% for our classifier as a reference) to 99.98%. Therefore expanding the set of features given as inputs to the classifier would prove very beneficial in all aspects.

Finally, the classifier currently only knows to distinguish between "good" and "bad" flows, ignoring the malware family a flow would belong to. Modifying the classifier so that it is able to assign a malware family to a malicious flow would help a lot the response phase. Indeed, knowing what type of malware has infected a machine is very valuable to contain it and to understand what may have been compromised. However this refinement in classification is only possible if the training data is correctly labelled in the first place, which requires a lot more care and verification when collecting malware samples.

Chapter 5

Conclusion

In the past decade our online presence has increased drastically. From this evolution has risen the need of a fast and resilient protocol to protect users' privacy. TLS aims to fulfill that role and TLS version 1.2 has quickly become the standard of web encryption. But these last years we have seen the emergence of new kinds of malware leveraging TLS to hide their malicious communications among normal traffic. Such behavior in malicious applications is on the rise and we can expect it to grow even more in the years to come.

However we have seen that malware authors often overlook the configuration of their TLS servers and malware, such that they present several characteristics that make them stand out from normal traffic. While these dissimilarities are not easy to detect for humans, they make possible the creation of a classifier which, once trained, can reliably detect malicious flows.

This project's contributions to the current landscape of intrusion detection systems can be summed in three points:

1. The construction of a curated dataset of capture files from malware known to use TLS to hide their communications (section 3.1). This dataset has furthermore been arranged by dates, malware families and sources to facilitate any use that might be made of it. It can be found on Google Drive[1].
2. The development of a pipeline that can process and filter capture files in order to extract relevant classification features. This pipeline takes the form of several scripts packaged together, each with its own purpose detailed section 3.1.4.
3. The creation of a classifier capable of detecting, with high accuracy, malicious TLS flows in a company's network traffic. The classifier was later combined with a detector integrated into Lastline's intrusion detection platform.

The random forest classifier presented in this project takes advantage of the net separation between bad and good datasets. It achieves 99.5% accuracy on never-seen before flows. When limiting the number of false positives is the main concern, which is often the case in large networks, the model reaches a true positive rate of 83.44% when allowing one flow in 5000 benign ones to potentially raise a false alert.

[1]https://tinyurl.com/tlsmalware

The classifier does have limitations, and could be reworked to improve robustness and to reduce even more the false positive rate. However it still has the significant advantage over traditional detection techniques that it does not require the decryption of each packet to perform analysis. This results in a valuable speed gain while respecting the privacy of users. Security companies such as Lastline have become aware of the adoption of TLS by malware and are turning to machine-learning based detection systems for these kind of benefits.

However the fight against malware is not going to end anytime soon. While detection capabilities improve and awareness of the threats posed by malware spreads, hostile agents adapt and responds by advancing their tools and techniques. Some malware abusing TLS have already been spotted tampering TLS parameters to evade a basic detection technique based on JA3 hashes [8] and we can expect future TLS malware to become even more stealthy.

The rise of the new version of TLS, TLSv1.3, is also to take into account. At the moment, no malware samples using TLSv1.3 were found and TLSv1.3 adoption still remains low. The classifier developed in this project was trained exclusively on TLSv1.2 samples and will probably stay relevant for the months to come. But abuse of this new version will without a doubt happen anytime soon and, along other intrusion detection systems, the classifier will have to be adapted and retrained to stay useful.

Appendix A

Malware using TLS

Table A.1 presents the list that was used to collect malware samples online and from Lastline's repositories. It was built using Cisco's papers, various reports found online about TLS malware detection and MITRE ATT&CK database[1]. Type might not be accurate.

[1]https://attack.mitre.org/software/

Name	Type	Name	Type
adwind	RAT	naid	Trojan
ammyy	RAT	necurs	Trojan
andromeda	Trojan	nidiran	RAT
auditcred	Malicious DDL	njrat	RAT
badcall	RAT	parite	Worm
bbsrat	RAT	pasam	RAT
bebloh	Trojan	petya	Ransomware
bergat	Trojan	powerduke	RAT
bestafera	Trojan	powerton	RAT
bisonal	Trojan	proxysvc	Trojan
briba	Trojan	qadars	Trojan
bunitu	Trojan	ratankba	RAT
carbanak	Trojan	razy	Trojan
comnie	RAT	redyms	Trojan
cryptowall	Ransomware	rerdom	Trojan
deshacop	Trojan	retefe	Trojan
dragonfly	RAT	sality	Virus
dridex	Trojan	shylock	Virus
dynamer	Trojan	skeeyah	Trojan
dyre	Trojan	spambot	Trojan
dyreza	Trojan	stype	RAT
emotet	Trojan	symmi	Trojan
felixroot	RAT	tescrypt	Ransomware
gamarue	Trojan	teslacrypt	Ransomware
geodo	Trojan	tinba	Trojan
gh0st	RAT	tofsee	RAT
gootkit	Trojan	toga	Trojan
gozi	Trojan	torrentlocker	Ransomware
hardrain	Trojan	trickbot	Trojan
hizor	RAT	troldesh	Ransomware
jbifrost	RAT	typeframe	Trojan
jigsaw	Ransomware	uboat	Trojan
kazy	Trojan	upatre	Trojan
kelihos	Trojan	urlzone	Trojan
keymarble	Trojan	virlock	Ransomware
kins	Trojan	virtob	Ransomware
locky	Ransomware	yakes	Trojan
lowball	Trojan	zbot	Trojan
misdat	Trojan	zusy	Trojan
miuref	Trojan		

Table A.1: Malware using TLS

Appendix B

Common TLS Destination Ports

Table B.1 was built using the Wikipedia list of usual ports [40]. The list is used to generate the *destination port* feature. If the destination port of a flow is present in the table, the feature is set to 1, otherwise 0.

Port	**Description**
443	HTTPS (HTTP over TLS)
465	SMTPS (SMTP over TLS)
563	NNTPS (NNTP over TLS)
636	LDAPS (LDAP over TLS)
853	DNS over TLS
989	FTPS (data) (FTP over TLS)
990	FTPS (control) (FTP over TLS)
992	Telnet over TLS
993	IMAPS (IMAP over TLS)
995	POP3S (POP version 3 over TLS)

Table B.1: TLS destination ports

Appendix C

TLS Parameters Codes

The tables list TLS parameters extracted from the training datasets. It does not encompass all available options: for instance only 145 ciphersuites out of 344 were present in the training dataset.

C.1 Ciphersuites

Table C.1 lists all ciphersuites extracted from the training dataset, which represents 145 distinct ciphersuites. Their robustness can be evaluated on ciphersuite.info [51].

Table C.1: TLS ciphersuites

TLS Ciphersuites		
Hex.	**Dec.**	**Description**
3	3	TLS_RSA_EXPORT_WITH_RC4_40_MD5
4	4	TLS_RSA_WITH_RC4_128_MD5
5	5	TLS_RSA_WITH_RC4_128_SHA
6	6	TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5
7	7	TLS_RSA_WITH_IDEA_CBC_SHA
8	8	TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
9	9	TLS_RSA_WITH_DES_CBC_SHA
a	10	TLS_RSA_WITH_3DES_EDE_CBC_SHA
d	13	TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA
10	16	TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA
11	17	TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
12	18	TLS_DHE_DSS_WITH_DES_CBC_SHA
13	19	TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
14	20	TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
15	21	TLS_DHE_RSA_WITH_DES_CBC_SHA
16	22	TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
2f	47	TLS_RSA_WITH_AES_128_CBC_SHA
30	48	TLS_DH_DSS_WITH_AES_128_CBC_SHA

TLS Ciphersuites (continued)

Hex.	**Dec.**	**Description**
31	49	TLS_DH_RSA_WITH_AES_128_CBC_SHA
32	50	TLS_DHE_DSS_WITH_AES_128_CBC_SHA
33	51	TLS_DHE_RSA_WITH_AES_128_CBC_SHA
35	53	TLS_RSA_WITH_AES_256_CBC_SHA
36	54	TLS_DH_DSS_WITH_AES_256_CBC_SHA
37	55	TLS_DH_RSA_WITH_AES_256_CBC_SHA
38	56	TLS_DHE_DSS_WITH_AES_256_CBC_SHA
39	57	TLS_DHE_RSA_WITH_AES_256_CBC_SHA
3c	60	TLS_RSA_WITH_AES_128_CBC_SHA256
3d	61	TLS_RSA_WITH_AES_256_CBC_SHA256
3e	62	TLS_DH_DSS_WITH_AES_128_CBC_SHA256
3f	63	TLS_DH_RSA_WITH_AES_128_CBC_SHA256
40	64	TLS_DHE_DSS_WITH_AES_128_CBC_SHA256
41	65	TLS_RSA_WITH_CAMELLIA_128_CBC_SHA
42	66	TLS_DH_DSS_WITH_CAMELLIA_128_CBC_SHA
43	67	TLS_DH_RSA_WITH_CAMELLIA_128_CBC_SHA
44	68	TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA
45	69	TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA
67	103	TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
68	104	TLS_DH_DSS_WITH_AES_256_CBC_SHA256
69	105	TLS_DH_RSA_WITH_AES_256_CBC_SHA256
6a	106	TLS_DHE_DSS_WITH_AES_256_CBC_SHA256
6b	107	TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
84	132	TLS_RSA_WITH_CAMELLIA_256_CBC_SHA
85	133	TLS_DH_DSS_WITH_CAMELLIA_256_CBC_SHA
86	134	TLS_DH_RSA_WITH_CAMELLIA_256_CBC_SHA
87	135	TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA
88	136	TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA
96	150	TLS_RSA_WITH_SEED_CBC_SHA
97	151	TLS_DH_DSS_WITH_SEED_CBC_SHA
98	152	TLS_DH_RSA_WITH_SEED_CBC_SHA
99	153	TLS_DHE_DSS_WITH_SEED_CBC_SHA
9a	154	TLS_DHE_RSA_WITH_SEED_CBC_SHA
9c	156	TLS_RSA_WITH_AES_128_GCM_SHA256
9d	157	TLS_RSA_WITH_AES_256_GCM_SHA384
9e	158	TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
9f	159	TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
a0	160	TLS_DH_RSA_WITH_AES_128_GCM_SHA256
a1	161	TLS_DH_RSA_WITH_AES_256_GCM_SHA384
a2	162	TLS_DHE_DSS_WITH_AES_128_GCM_SHA256
a3	163	TLS_DHE_DSS_WITH_AES_256_GCM_SHA384
a4	164	TLS_DH_DSS_WITH_AES_128_GCM_SHA256
a5	165	TLS_DH_DSS_WITH_AES_256_GCM_SHA384

TLS Ciphersuites (continued)

Hex.	**Dec.**	**Description**
ba	186	TLS_RSA_WITH_CAMELLIA_128_CBC_SHA256
bd	189	TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA256
be	190	TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA256
c0	192	TLS_RSA_WITH_CAMELLIA_256_CBC_SHA256
c3	195	TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA256
c4	196	TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA256
ff	255	TLS_EMPTY_RENEGOTIATION_INFO_SCSV
1301	4865	TLS_AES_128_GCM_SHA256
1302	4866	TLS_AES_256_GCM_SHA384
1303	4867	TLS_CHACHA20_POLY1305_SHA256
1304	4868	TLS_AES_128_CCM_SHA256
c002	49154	TLS_ECDH_ECDSA_WITH_RC4_128_SHA
c003	49155	TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA
c004	49156	TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
c005	49157	TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA
c007	49159	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
c008	49160	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
c009	49161	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
c00a	49162	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
c00c	49164	TLS_ECDH_RSA_WITH_RC4_128_SHA
c00d	49165	TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA
c00e	49166	TLS_ECDH_RSA_WITH_AES_128_CBC_SHA
c00f	49167	TLS_ECDH_RSA_WITH_AES_256_CBC_SHA
c011	49169	TLS_ECDHE_RSA_WITH_RC4_128_SHA
c012	49170	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
c013	49171	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
c014	49172	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
c016	49174	TLS_ECDH_anon_WITH_RC4_128_SHA
c017	49175	TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA
c018	49176	TLS_ECDH_anon_WITH_AES_128_CBC_SHA
c019	49177	TLS_ECDH_anon_WITH_AES_256_CBC_SHA
c023	49187	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
c024	49188	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
c025	49189	TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256
c026	49190	TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384
c027	49191	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
c028	49192	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
c029	49193	TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256
c02a	49194	TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384
c02b	49195	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
c02c	49196	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
c02d	49197	TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256
c02e	49198	TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384

TLS Ciphersuites (continued)		
Hex.	**Dec.**	**Description**
c02f	49199	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
c030	49200	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
c031	49201	TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256
c032	49202	TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384
c050	49232	TLS_RSA_WITH_ARIA_128_GCM_SHA256
c051	49233	TLS_RSA_WITH_ARIA_256_GCM_SHA384
c052	49234	TLS_DHE_RSA_WITH_ARIA_128_GCM_SHA256
c053	49235	TLS_DHE_RSA_WITH_ARIA_256_GCM_SHA384
c056	49238	TLS_DHE_DSS_WITH_ARIA_128_GCM_SHA256
c057	49239	TLS_DHE_DSS_WITH_ARIA_256_GCM_SHA384
c05c	49244	TLS_ECDHE_ECDSA_WITH_ARIA_128_GCM_SHA256
c05d	49245	TLS_ECDHE_ECDSA_WITH_ARIA_256_GCM_SHA384
c060	49248	TLS_ECDHE_RSA_WITH_ARIA_128_GCM_SHA256
c061	49249	TLS_ECDHE_RSA_WITH_ARIA_256_GCM_SHA384
c072	49266	TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_CBC_SHA256
c073	49267	TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_CBC_SHA384
c076	49270	TLS_ECDHE_RSA_WITH_CAMELLIA_128_CBC_SHA256
c077	49271	TLS_ECDHE_RSA_WITH_CAMELLIA_256_CBC_SHA384
c07a	49274	TLS_RSA_WITH_CAMELLIA_128_GCM_SHA256
c07b	49275	TLS_RSA_WITH_CAMELLIA_256_GCM_SHA384
c07c	49276	TLS_DHE_RSA_WITH_CAMELLIA_128_GCM_SHA256
c07d	49277	TLS_DHE_RSA_WITH_CAMELLIA_256_GCM_SHA384
c086	49286	TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_GCM_SHA256
c087	49287	TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_GCM_SHA384
c08a	49290	TLS_ECDHE_RSA_WITH_CAMELLIA_128_GCM_SHA256
c08b	49291	TLS_ECDHE_RSA_WITH_CAMELLIA_256_GCM_SHA384
c09c	49308	TLS_RSA_WITH_AES_128_CCM
c09d	49309	TLS_RSA_WITH_AES_256_CCM
c09e	49310	TLS_DHE_RSA_WITH_AES_128_CCM
c09f	49311	TLS_DHE_RSA_WITH_AES_256_CCM
c0a0	49312	TLS_RSA_WITH_AES_128_CCM_8
c0a1	49313	TLS_RSA_WITH_AES_256_CCM_8
c0a2	49314	TLS_DHE_RSA_WITH_AES_128_CCM_8
c0a3	49315	TLS_DHE_RSA_WITH_AES_256_CCM_8
c0ac	49324	TLS_ECDHE_ECDSA_WITH_AES_128_CCM
c0ad	49325	TLS_ECDHE_ECDSA_WITH_AES_256_CCM
c0ae	49326	TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8
c0af	49327	TLS_ECDHE_ECDSA_WITH_AES_256_CCM_8
cca8	52392	TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
cca9	52393	TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
ccaa	52394	TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256
End of Table		

C.2 Extensions

Table C.2 lists all extensions seen in the training dataset. It represents 15 out of 44 extensions.

Hex.	Dec.	Description
0	0	server_name
5	5	status_request
a	10	supported_groups
b	11	ec_point_formats
d	13	signature_algorithms
f	15	heartbeat
10	16	application_layer_protocol_negotiation
11	17	status_request_v2
12	18	signed_certificate_timestamp
15	21	padding
16	22	encrypt_then_mac
17	23	extended_master_secret
18	24	token_binding
23	35	session_ticket
ff01	65281	renegotiation_info

Table C.2: TLS extensions

C.3 Elliptic Curve Groups

Table C.3 lists all elliptic curve groups seen in the training dataset. It represents 35 out of 47 groups.

Hex.	Decimal	Description	Hex.	Dec.	Description
1	1	sect163k1	13	19	secp192r1
2	2	sect163r1	14	20	secp224k1
3	3	sect163r2	15	21	secp224r1
4	4	sect193r1	16	22	secp256k1
5	5	sect193r2	17	23	secp256r1
6	6	sect233k1	18	24	secp384r1
7	7	sect233r1	19	25	secp521r1
8	8	sect239k1	1a	26	brainpoolP256r1
9	9	sect283k1	1b	27	brainpoolP384r1
a	10	sect283r1	1c	28	brainpoolP512r1
b	11	sect409k1	1d	29	x25519
c	12	sect409r1	1e	30	x448
d	13	sect571k1	100	256	ffdhe2048
e	14	sect571r1	101	257	ffdhe3072
f	15	secp160k1	102	258	ffdhe4096
10	16	secp160r1	103	259	ffdhe6144
11	17	secp160r2	104	260	ffdhe8192
12	18	secp192k1			

Table C.3: TLS elliptic curve groups

Bibliography

[1] Internet Engineering Task Force (IETF), ed. *Applying GREASE to TLS Extensibility*. Jan. 2019. URL: https://tools.ietf.org/html/draft-ietf-tls-grease-02 (visited on 08/12/2019).

[2] Internet Engineering Task Force (IETF), ed. *The DTLS Protocol – Version 1.2*. Jan. 2012. URL: https://tools.ietf.org/html/rfc6347 (visited on 05/02/2019).

[3] Internet Engineering Task Force (IETF), ed. *The TLS Protocol – Version 1.0*. Jan. 1999. URL: https://tools.ietf.org/html/rfc2246 (visited on 05/02/2019).

[4] Internet Engineering Task Force (IETF), ed. *The TLS Protocol – Version 1.2*. Aug. 2009. URL: https://tools.ietf.org/html/rfc5246 (visited on 04/29/2019).

[5] Internet Engineering Task Force (IETF), ed. *The TLS Protocol – Version 1.3*. Aug. 2018. URL: https://tools.ietf.org/html/rfc8446 (visited on 04/29/2019).

[6] International Telecommunications Union (ITU), ed. *Internet X.509 Public Key Infrastructure Certificate*. May 2008. URL: https://tools.ietf.org/html/rfc5280 (visited on 05/03/2019).

[7] abuse.ch. *SSL Blacklist*. 2019. URL: https://sslbl.abuse.ch/ (visited on 05/12/2019).

[8] Akamai. *Bots Tampering With TLS to Avoid Detection*. May 2019. URL: https://blogs.akamai.com/sitr/2019/05/bots-tampering-with-tls-to-avoid-detection.html (visited on 05/16/2019).

[9] Selim Aksoy and Robert M. Haralick. "Feature Normalization and Likelihood-based Similarity Measures for Image Retrieval". In: *Pattern Recogn. Lett.* (2001), pp. 563–582. DOI: 10.1016/S0167-8655(00)00112-4.

[10] Kevin Allix et al. "Are Your Training Datasets Yet Relevant?" In: *Engineering Secure Software and Systems*. 2015, pp. 51–67. ISBN: 978-3-319-15618-7.

[11] John Althouse. *TLS Fingerprinting with JA3 and JA3S*. Jan. 2019. URL: https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967 (visited on 04/29/2019).

[12] Blake Anderson. *Detecting Encrypted Malware Traffic (Without Decryption)*. June 2017. URL: https://blogs.cisco.com/security/detecting-encrypted-malware-traffic-without-decryption (visited on 04/29/2019).

[13] Blake Anderson and David McGrew. "Identifying Encrypted Malware Traffic with Contextual Flow Data". In: *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*. 2016, pp. 35–46. DOI: 10.1145/2996758.2996768.

[14] Blake Anderson and David McGrew. "Machine Learning for Encrypted Malware Traffic Classification: Accounting for Noisy Labels and Non-Stationarity". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2017, pp. 1723–1732. DOI: 10.1145/3097983.3098163.

[15] Blake Anderson, Subharthi Paul, and David McGrew. "Deciphering Malware's use of TLS (without Decryption)". In: (2016). arXiv: 1607.01639. URL: http://arxiv.org/abs/1607.01639.

[16] *Area under the curve*. July 2019. URL: https://en.wikipedia.org/wiki/Receiver_operating_characteristic#Area_under_the_curve (visited on 08/01/2019).

[17] Stefan Axelsson. "The Base-rate Fallacy and the Difficulty of Intrusion Detection". In: *ACM Trans. Inf. Syst. Secur.* (2000), pp. 186–205. DOI: 10.1145/357830.357849.

[18] Barac. *Encrypted Traffic Visibility*. 2017. URL: https://barac.io (visited on 08/10/2019).

[19] David Bianco. *Pyramid of Pain*. Mar. 2013. URL: http://detect-respond.blogspot.com/2013/03/the-pyramid-of-pain.html (visited on 07/03/2019).

[20] Álvaro Castro-Castilla. *Traffic Analysis of an SSL/TLS Session*. Dec. 2014. URL: http://blog.fourthbit.com/2014/12/23/traffic-analysis-of-an-ssl-slash-tls-session (visited on 04/29/2019).

[21] Canadian Institute for Cybersecurity. *Datasets*. URL: https://www.unb.ca/cic/datasets/index.html (visited on 06/25/2019).

[22] Darktrace. *Beyond the hash: How unsupervised machine learning unlocks the true power of JA3*. June 2018. URL: https://www.darktrace.com/en/blog/beyond-the-hash-how-unsupervised-machine-learning-unlocks-the-true-power-of-ja-3/ (visited on 04/29/2019).

[23] Brendan Dolan-Gavitt. *Reproducible Malware Analyses for All*. Dec. 2014. URL: http://moyix.blogspot.com/2014/12/reproducible-malware-analyses-for-all.html (visited on 06/25/2019).

[24] Michael Driscoll. *The New Illustrated TLS Connection*. Mar. 2019. URL: https://tls13.ulfheim.net/ (visited on 05/02/2019).

[25] Brad Duncan. *malware-traffic-analysis*. URL: https://www.malware-traffic-analysis.net/ (visited on 06/25/2019).

[26] Let’s Encrypt. *Why ninety-day lifetimes for certificates?* Nov. 2015. URL: https://letsencrypt.org/2015/11/09/why-90-days.html (visited on 07/18/2019).

[27] *Encrypted Traffic Analytics*. White paper. Cisco, Jan. 2019. URL: https://www.cisco.com/c/en/us/solutions/enterprise-networks/enterprise-network-security/eta.html (visited on 04/29/2019).

[28] *Ephemeral Port*. Jan. 2018. URL: https://en.wikipedia.org/wiki/Ephemeral_port (visited on 07/09/2019).

[29] Mozilla Foundation. *Security/Server Side TLS*. July 2019. URL: https://wiki.mozilla.org/Security/Server_Side_TLS (visited on 07/03/2019).

[30] Open Information Security Foundation. *Suricata*. July 2010. URL: https://suricata-ids.org/ (visited on 05/13/2019).

[31] Alessandro Ghedini. *Encrypt it or lose it: how encrypted SNI works*. Sept. 2018. URL: https://blog.cloudflare.com/encrypted-sni/ (visited on 04/29/2019).

[32] Google. *HTTPS encryption on the web*. Jan. 2019. URL: https://transparencyreport.google.com/https/overview (visited on 04/29/2019).

[33] *Hidden Threats in Encrypted Traffic*. Research report. Ponemon Institute, May 2016. URL: https://www.ponemon.org/library/hidden-threats-in-encrypted-traffic-a-study-of-north-america-emea (visited on 04/28/2019).

[34] IANA. *TLS Cipher Suites*. 2005. URL: https://www.iana.org/assignments/tls-parameters/tls-parameters.xml#tls-parameters-4 (visited on 06/27/2019).

[35] Stratosphere IPS. *Malware Capture Facility Project*. URL: https://www.stratosphereips.org/datasets-malware (visited on 06/25/2019).

[36] Gareth James et al. *An Introduction to Statistical Learning: With Applications in R*. Springer, 2014. ISBN: 1461471370.

[37] Brian Krebs. *Half of all Phishing Sites Now Have the Padlock*. Nov. 2018. URL: https://krebsonsecurity.com/2018/11/half-of-all-phishing-sites-now-have-the-padlock/ (visited on 05/08/2019).

[38] Christopher Kruegel and Giovanni Vigna. “Anomaly Detection of Web-based Attacks”. In: *Proceedings of the 10th ACM Conference on Computer and Communications Security*. 2003, pp. 251–261. DOI: 10.1145/948109.948144.

[39] SSL Labs. *SSL Pulse*. Apr. 2019. URL: https://www.ssllabs.com/ssl-pulse/ (visited on 05/02/2019).

[40] *List of TCP and UDP port numbers*. Aug. 2019. URL: https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers (visited on 08/23/2019).

[41] Marc-Etienne M.Léveillé. *TorrentLocker*. White paper. ESET, Dec. 2014. URL: https://www.welivesecurity.com/wp-content/uploads/2014/12/torrent_locker.pdf (visited on 05/09/2019).

[42] Gabriel Maciá-Fernández et al. "UGR'16: A new dataset for the evaluation of cyclostationarity-based network IDSs". In: (2018). DOI: `https://doi.org/10.1016/j.cose.2017.11.004`. URL: `http://www.sciencedirect.com/science/article/pii/S0167404817302353`.

[43] D. McGrew and B. Anderson. *Joy*. 2016. URL: `https://github.com/cisco/joy` (visited on 06/26/2019).

[44] David McGrew and Blake Anderson. "Enhanced telemetry for encrypted threat analytics". In: *Proceedings of the 2016 IEEE 24th International Conference on Network Protocols (ICNP)*. 2016, pp. 1–6. DOI: `10.1109/ICNP.2016.7785325`.

[45] Neel Mehta. *The Heartbleed Bug*. Apr. 2014. URL: `http://heartbleed.com/` (visited on 08/20/2019).

[46] T. T. T. Nguyen and G. Armitage. "A survey of techniques for internet traffic classification using machine learning". In: *IEEE Communications Surveys Tutorials* (2008), pp. 56–76. DOI: `10.1109/SURV.2008.080406`.

[47] Dick O'Brien. *Dridex*. White paper. Symantec, Feb. 2016. URL: `https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/dridex-financial-trojan.pdf` (visited on 05/09/2019).

[48] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* (2011).

[49] Feargus Pendlebury et al. "TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time". In: *28th USENIX Security Symposium*. 2019.

[50] Sam Richman. *This New Covert Channel Uses SSL/TLS Handshakes*. Feb. 2018. URL: `https://www.extrahop.com/company/blog/2018/stop-ssl-tls-exfil/` (visited on 04/29/2019).

[51] Hans Christian Rudolph and Nils Grundmann. *Security of Cipher Suites*. 2019. URL: `https://ciphersuite.info/cs/` (visited on 07/09/2019).

[52] Salesforce. *JA3 – A method for profiling SSL/TLS Clients*. June 2017. URL: `https://github.com/salesforce/ja3` (visited on 04/29/2019).

[53] Robin Sommer and Vern Paxson. "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection". In: *Proceedings of the 2010 IEEE Symposium on Security and Privacy*. 2010, pp. 305–316. DOI: `10.1109/SP.2010.25`.

[54] Aragorn Tseng. *Malicious Encrypted Traffic Detection*. Jan. 2019. URL: `https://data.hackinn.com/ppt/HITCON2018/day1/Malicious-Encrypted-Traffic-Detection.pdf` (visited on 08/10/2019).

[55] Can I Use. *TLSv1.3 Browser Support*. May 2019. URL: `https://caniuse.com/#feat=tls1-3` (visited on 05/02/2019).

[56] Nigel Williams, Sebastian Zander, and Grenville Armitage. "A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification". In: *SIGCOMM Comput. Commun. Rev.* (Oct. 2006), pp. 5–16. DOI: `10.1145/1163593.1163596`.

[57] Sebastian Zander, Thuy Nguyen, and Grenville Armitage. "Automated Traffic Classification and Application Identification Using Machine Learning". In: *Proceedings of the The IEEE Conference on Local Computer Networks 30th Anniversary*. 2005, pp. 250–257. DOI: `10.1109/LCN.2005.35`.

[58] *ZeuS trojan FAQ*. 2011. URL: `https://github.com/Visgean/Zeus/blob/c55a9fa8c8564ec196604a59111708fa8415f020/manual_en.html#L778` (visited on 07/09/2019).