Corso di Laurea in Informatica

Tesi di laurea

Malware Classification with Markov Chains

Relatore:
Luca Deri

Candidata:
Alessia Gagliano

Anno accademico 2023/2024

Contents

Abstract

The adoption of Transport Layer Security (TLS) has increased significantly in recent years, both in benign software and malware communications [1]. This trend has led to the development of new network analysis techniques, such as fingerprinting and certificate analysis, which can be employed without direct access to the encrypted segments of TLS communications.

This thesis aims to develop and evaluate a malware classification system capable of distinguishing among different malware families by analyzing their encrypted traffic.

The method employed is a fingerprinting system based on first-order Markov chains, as presented in Korczyński and Duda [2]. This system encompasses both TLS record type-based and packet length-based fingerprinting approaches, considering both bidirectional and unidirectional traffic to generate four distinct models. The findings presented in this thesis show that both TLS record types and packet lengths are valid classification parameters. Furthermore, our analysis suggests that methods using TLS record types perform better compared to those that use packet lengths.

Introduction

The use of the TLS has risen considerably in the last years [3], both in legitimate communication and in malware communication. Traffic encryption has been observed in many types of malware, particularly in the process of payload deposit, data exfiltration, and command and control communication. With TLS, malware traffic is indistinguishable from legitimate traffic, rendering traditional malware detecting techniques harder to use. Traditional packet inspection can still be used to analyze encrypted traffic, but it requires the ability to decrypt said traffic, which is not always feasible and defeats the purpose of TLS, rising important privacy and security issues. Given these constrains, new techniques have been developed, such as certificate analysis and fingerprinting, the process of selecting some traffic characteristics and creating a 'fingerprint' that can be used to identify future traffic with the same characteristics.

Malware detection techniques based on fingerprinting can use different parts of a network flow; many, for example JA4 [4] and Mercury [5], use the first part of a TLS communication, which is unencrypted (namely the ClientHello message or the ServerHello message). Several metadata can be extracted from these handshake messages and used to create a fingerprint, such as protocol version, Server Name Indication (SNI), and the lists of cipher suites and TLS extensions.
Other fingerprinting techniques use the entire network flow to create their fingerprints. Korczyński and Duda [2] in particular showed that a Markov chain fingerprint that considers TLS record type, and unidirectional traffic (only packets coming from a server) can reliably classify different applications. Ha and Roh [6] used a similar stochastic model, but considered packet lengths instead.

In this thesis we build a malware classifier based on first order Markov chains, as done in Korczyński and Duda [2], but we'll consider bidirectional and unidirectional traffic, as well as TLS record types and packet length, thus creating four models.

We used these models to create a fingerprint for four common malware that use TLS (Cobalt Strike, Dridex, Emotet, and Trickbot), and assess both their ability to classify malware, and to properly discriminate malware from legitimate communications, by validating the models against both types of traffic.

1. Background

This chapter describes some fundamental concepts underlying this thesis. Section 1.1 gives a description of how TLS works, section 1.2 talks about malware and its use of TLS, and section 1.3 goes over some fingerprinting techniques.

1.1 TLS

The main protocol used to encrypt network traffic is TLS, the successor of the now deprecated Secure Sockets Layer (SSL) protocol. TLS provides authentication, confidentiality, and data integrity, hence it secures the communication between two machines (usually a client and a server) communicating over the Internet. [7].

TLS sits between the transport layer and the application layer (see figure 1.1) ant secures the communication before any data is transmitted to or from the application layer. Its primary components are the TLS Handshake Protocol and, at a lower level, the TLS Record Protocol. The handshake protocol is responsible for negotiating encryption and security parameters, that will then be used by the record protocol to send the actual message.

During the handshake phase, a public-key cryptographic algorithm is used to exchange data (premaster secret) in order to construct a symmetric key (master secret) that will be used to secure the following communication.

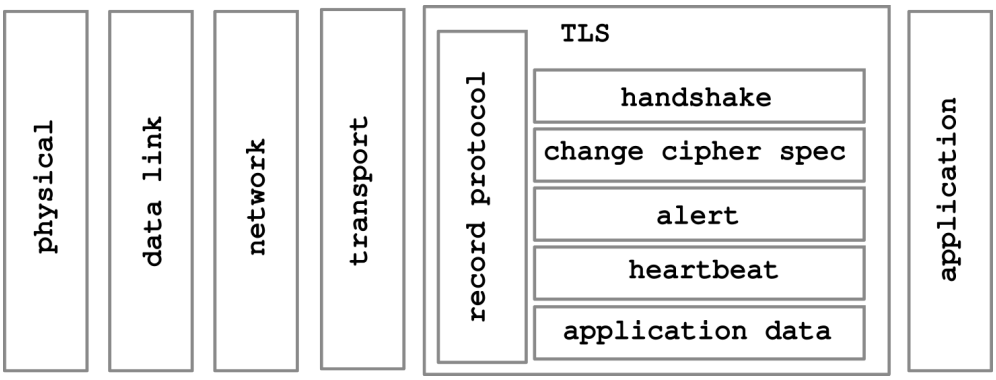

Figure 1.1: TLS in the TCP/IP stack

Different version of TLS have been written over the years. For the present work, the most important change comes with TLS version 1.3, which encrypts all handshake messages after ServerHello, while previous versions fully encrypt only the last handshake message. A summary of TLS versions can be found in table 1.1.

Version	Year	RFC
1.0	1999 - 2021	2246 [7]
1.1	2006 - 2021	4346 [8]
1.2	2008 - in use	5246 [9]
1.3	2018 - in use	8446 [10]

Table 1.1: TLS versions

The rest of this section will provide a more detailed description of how the different TLS components work.

- **TLS handshake protocol:** it is used by a client and a server to authenticate themselves, and to agree on the encryption and security parameters that will be used by the record layer.

  During the handshake, a connection is initiated by the client with a ClientHello message. This message includes the TLS version the client wants to use, a sequence generated by a random number generator, a session ID (empty if it's a new session, or a

specific identifier of a session the client wants to resume), a list of cipher suites supported by the client, in order of preference, and a list of compression methods supported by the client.

The server responds with a ServerHello message, which includes the chosen cipher suite, compression method, and TLS version, along with a random sequence, and the session ID corresponding to the connection to be created or resumed. The server sends a Certificate message (if the key exchange method is not anonymous) usually containing its certificate. If this last message does not contain enough information for the client to exchange a premaster secret, the server sends a ServerKeyExchange message containing cryptographic information. The server can optionally request the client's certificate with a CertificateRequest message.

This phase ends with a ServerHelloDone message: the server has finished its part of the key exchange communication. The client verifies the server certificate and the ServerHello security parameters, and, if requested, it sends its certificate in a ClientCertificate message. The client sends a ClientKeyExchange message, containing the premaster secret (PMS). With the premaster secret exchanged, both the client and the server can now independently compute the master secret (MS) using the premaster secret itself, and the client and server random sequences. Client and server notify each other they will next be using the negotiated security parameters with a ChangeCipherSpec message. Both the client and the server send a Finished message. This is the first message encrypted with the negotiated parameters. It is used to verify the correctness of the previous steps (key exchange and authentication). The full list of handshake content types can be found in table 1.3.

- **TLS record protocol:** the record layer transmits a message to its destination. The data is fragmented and (optionally) compressed using the compression algorithm negotiated by the handshake protocol. Before the transmission, the MAC is added and the encryption algorithm is applied to the message according to the handshake negotiation. The incoming data are processed in reverse order: they are decrypted, verified, and decompressed before being sent to a higher level.

Other TLS components that sit at the same level of the handshake protocol are:

- **Alert protocol:** it is used both to signal the end of a session (closure alerts: close_notify message), and to deliver alert messages (error alerts), along with their severity (warning, fatal).
- **Change cipher spec protocol:** it signals a change in the security parameters the record layer should use. It is not a handshake message, but its own content type.
- **Application data protocol:** this protocol handles application data messages. It delivers them to the TLS record layer, where they are fragmented, compressed, and encrypted.

Number	Description
20	change cipher spec
21	alert
22	handshake
23	application data
24	heartbeat

Table 1.2: TLS content types

Number	Description
0	hello request
1	client hello
2	server hello
4	new session ticket
8	encrypted extensions (1.3 only)
11	certificate
12	server key exchange
13	certificate request
14	server hello done
15	certificate verify
16	client key exchange
20	finished

Table 1.3: Handshake content types

1.2 Malware

The term ”malware” (malicious software) refers to any software intentionally designed to damage the functioning of a computer system or network, or to alter or steal information contained in the target machine.
Malware can be categorized based on what they do in the infected machine and how they gain access to the infected machine.

Some common types of malware are:

- *Virus:* a virus requires to be executed by the victim in order to begin its infection. When executed, it adds its own code to that of another, legitimate, program, through which it can replicate itself.
- *Worm:* a type of malware that actively spread itself over a network, without the victim's action.
- *Trojan horse:* a malware that disguise itself as a legitimate software, and hides its true intentions. It is commonly spread via social engineering, for example by email attachments, that once opened execute the malware. Once in the target system, it can download and install additional malware, in this case the trojan acts as a ”dropper”. A trojan subcategory is a Remote Access Trojan (RAT), a malware that has access to a machine through a network connection, and is installed without the user's knowledge.

In 2021, nearly 46 percent of malware detected by Sophos Labs used TLS [1]. Although malware can use TLS in the process of payload deposit, data exfiltration, and communication with a command and control server, the majority of TLS use is detected in the initial phase of an attack, when a main module of a malware is downloaded [1].
Malware generally uses encryption as a way to disguise itself, blending its traffic with that of legitimate programs, while benign traffic uses TLS for its security features. It is perhaps for this reason that malware TLS cryptographic parameters have in the past been found to be older and weaker than those of legitimate software [11].

What follows is a list of malware used in our fingerprinting system:

- **Cobalt Strike:** an offensive security tool used both by legitimate and malicious actors [12]. It is classified as a RAT. Its primary component are a ”team server”, which acts as a command and control (C2) server, and a client, used by the attacker (legitimate or not) to connect to the team server. [13] Cobalt Strike default payload is called ”beacon”, which connects to the team server, checking for additional commands.[14]

- **Dridex:** a banking trojan, that is a trojan that tries to steal banking information. Its primary method of infection is via attachments in phishing emails, namely MS Word documents containing macros. When executed, such macros contact a command-and-control server, from which they download additional malware modules or, in some instances, the main malware module itself [15]. Dridex uses TLS, usually over Hypertext Transfer Protocol (HTTP) on port default 443 to download additional modules, and exfiltrate the stolen data [1].

- **Emotet:** initially used as a banking trojan and information stealer, it has been repurposed as a dropper for other malware. The initial infection is through malicious attachments in phishing emails. Unlike other trojan, Emotet attempts to replicate itself within a network [16].

- **Trickbot:** a trojan and information stealer, it is also used as a dropper. It is spread mainly through spearfishing campaigns, that is phishing emails aimed at specific people. Trickbot exfiltrates stolen information to a command and control server, and is capable of downloading additional components and configuration files [17]. The communication with its command and control server is over Hypertext Transfer Protocol Secure (HTTPS).

Based on the analysis our malware samples 2.1, each of these malware use TLS, opening many TLS connection per attack event. All of them, with the exclusion of Cobalt Strike, connect to more than one server during an attack. Dridex and Cobalt Strike use mainly default TLS port 443, Trickbot uses non default ports 447 and 449, while Emotet uses mainly ports 8080, 80, and 7080.

1.3 Fingerprinting

Network fingerprinting aims to identify a certain number of network values and characteristics specific to a system. The ensemble of these values is called ”fingerprint”, which can be used for future identification of the same system.
Fingerprinting can be used to identify operating systems [18], clients [19], protocols [20], applications [21] [22], and even users [23]. Fingerprinting can be either passive or active. Active fingerprinting probes the target system by sending it packets and analyzing the corresponding replies. Passive fingerprinting analyses the traffic without interfering with it.

Fingerprinting techniques are particularly valuable in the classification of encrypted network traffic, as they use unencrypted metadata (mainly values from the handshake messages) to create their fingerprints, leaving all of TLS security properties unaltered.
The first example of fingerprinting applied to SSL/TLS is an Apache module developed in 2009 by Ivan Ristić [24] to extract client supported cipher suites from an SSL handshake [25] [26].
Other methods combine handshake values with packet lengths and packet interarrival times, or with related Domain Name System (DNS) or HTTP traffic to construct a unique fingerprint [27] [28].
Fingerprinting techniques can be also divided in two categories: those that use only the first packets to create a fingerprint, and those that consider the whole communication flow.

What follows are some common TLS fingerprinting tools belonging to the first category:

- **Mercury:** (successor of Joy [29]) a passive TLS fingerprinting tool, uses ClientHello messages to construct its fingerprints. The fingerprint is composed of: TLS version, list of cipher suites, list of extensions, as well as some ”contextual information”: destination IP address and port, and server name value [5] [30].

- **JA4:** (successor of JA3 [31]) a passive TLS client fingerprinting tool. The fingerprint elements are: TLS version, SNI, Application-Layer Protocol Negotiation (ALPN),

number of cipher suites and extensions, transport layer protocol, and cipher suites and extensions lists [32].

- **JARM:** an active TLS server fingerprinting tool [33]. It sends a server ten ClientHello messages and uses the responses to create a fingerprint string. The sent messages have different TLS versions, ciphers, and extensions in order to gather as many diverse responses as possible [34].

1.3.1 Markov chain fingerprinting

One fingerprinting system that uses the entire network flow is the one proposed in Korczyński and Duda [2]. They constructed a model consisting of a fist order, homogeneous Markov chain, where the states correspond to TLS record types, or TLS record type sequences observed in a TLS communication.

A Markov chain is a stochastic process that satisfies the Markov property:

$$P(X_{n+1} = i_{n+1} | X_n = i_n, X_{n-1} = i_{n-1}, ..., X_0 = i_0) = P(X_{n+1} = i_{n+1} | X_n = i_n)$$

that is, the probability distribution of X_{n+1} depends only on X_n, and not on past values of X. A Markov chain is homogeneous if:

$$P(X_{n+k} = j | X_k = i) = P(X_n = j | X_0 = i)$$

that is, if the transition probabilities don't depend on k.

In Korczyński and Duda [2], given a discrete-time random variable X_t, with $t = t_0, t_1, ..., t_k \in T$, that takes values $i_t \in \{1, ..., n\}$, that is the states set, the set consisting of the TLS record types or TLS record type sequence, the transition matrix is defined as: $[p_{i-j}]_{i,j=1,...,n}$ where $p_{i-j} = P(X_t = i_t | X_{t-1} = i_{t-1}) = P(X_t = j | X_{t-1} = i)$

To calculate the probabilities of entering and exiting the Markov chain, two vectors are used: an enter probability vector $Q = [q_1, q_2, ..., q_n]^T$, where $q_i = P(X_t = i)$ at time $t = t_0$ and an exit probability vector $W = [w_1, w_2, ..., w_n]^T$, where $w_i = P(X_t = i)$ at time $t = t_n$.

Thus, the probability of a states sequence $X_1, X_2, ..., X_T$ representing a TLS flow is: $P(X_1, ..., X_T) = q_{i_1} \cdot \prod_{t=2}^{T} p_{i_{t-1}-i_t} \cdot w_{i_T}$

2. Methodology and validation

This chapter describes how the fingerprints have been constructed using a Markov chain model. Section 2.1 describes how data is collected, section 2.2 describes how the Markov chain model is build and validated.

Four malware classifier based on Markov chains have been constructed, following the method described in section 1.3.1. Each model considers bidirectional traffic or unidirectional traffic, and its states consists of TLS record types or packet lengths.
Four types of malware have been chosen (Cobalt Strike, Dridex, Emotet, and Trickbot), for every one of them several pcap files [1] have been downloaded from Malware Traffic Analysis (MTA), filtered to contain only malicious traffic, and divided in two sets: a build set and a validation set. The build set contains elements used to create the models, and the validation set contains elements used to test the models. Figure 2.1 gives a summary of how data has been handled.

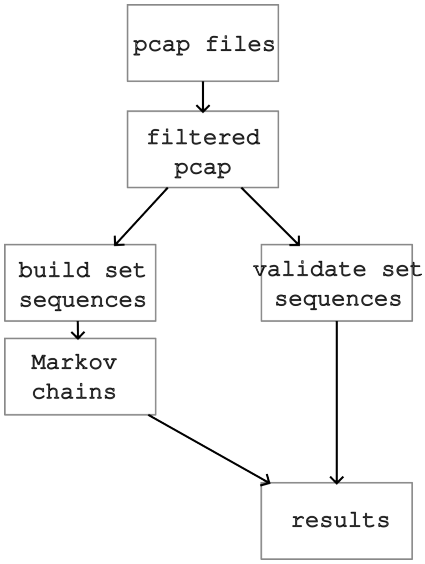

Figure 2.1: Data flow

[1] Files containing packet captures: every packet passing through a network interface during the capturing process. They can be read and generated with tools such as Wireshark [35] and Tshark [36].

2.1 Data collection

The malware chosen for this analysis are Cobalt Strike, Dridex, Emotet, and Trickbot. All packet captures have been downloaded from MTA [37]. The choice of these malware is based on the fact that they use TLS, and the sample availability on MTA. As shown in table 2.1, earlier captures are from 2019, and later capture are from 2023.

Every pcap file have been filtered in order to remove any non-malware related traffic. To this end, the indicators of compromise (IOC) relative to a pcap file is used to extract the IP addresses involved in the malicious traffic. This information is used to create a Tshark filter [36], and filter the original pcap. The result is a filtered pcap file containing only malicious traffic relative to a specific malware.

For every malware, its pcap files have been divided into two sets: a build set and a validate set.

Malware	Number of pcap files	Number of TLS flows	Time interval [years]
Cobalt Strike	63	21631	2020 - 2023
Dridex	10	121	2019 - 2021
Emotet	21	1397	2021 - 2023
Trickbot	17	425	2019 - 2021

Table 2.1: Collected data (total)

A small number of non-malware TLS traffic (labeled 'none') has been captured on a MS Windows machine, it consists of 394 flows. This traffic has been used in the validation phase, in order to test the models on legitimate traffic. The legitimate traffic is not used in the build phase, therefore the models can not explicitly classify a traffic flow as 'benign', but rather as 'belonging to none of the considered malware'. More specifically, the models consider the traffic associated with a classification probability lower than a given threshold as non-malware (see section 2.2.2 on the choice of this threshold). Therefore, we refer to this traffic as 'none', instead of 'benign', in order to avoid any misleading.

Table 2.2 details the data division between the build set and the validate set. The file division is approximately even, favoring the build set in the case of an uneven number of files. We chose to split the data at the file level rather than at the flow level, in order to preserve the atomicity of a malware infection event. This has led to an uneven distribution of the network flows between the build set and the validation set, but they are still mostly reasonably distributed, and when not (in the case of Cobalt Strike), the number of flows still favors the build set.

Malware	Build set [files]	Build set [flows]	Validate set [files]	Validate set [flows]
Cobalt Strike	32	15644	31	5987
Dridex	5	57	5	64
Emotet	11	608	10	789
Trickbot	9	238	8	187

Table 2.2: Data division

2.2 Model

Four models have been created, all based on first order homogeneous Markov chains. The difference between these models is the information used to construct the transition matrix states, and which packets have been considered.
For the packets we have two cases: bidirectional models, that consider the entire client-server communication (where 'client' refers to the malware infected machine, and 'server' refers to the remote machine contacted by the malware), and unidirectional models, that consider only packets coming from the server.

For the states we have the following two cases:

- *Type based models:* the Markov chain states coincide with the TLS record content types (table 1.2 and table 1.3). In the case of a packet containing multiple TLS records,

these have been concatenated together, resulting in a single state.

- *Length based models:* the Markov chain states are constructed based on the packets lengths. Following McGrew and Anderson [38] the lengths are divided into intervals of 150 bytes. Thus, state i, $(i > 0)$ represents packet lengths in $[(i-1) \cdot 150, i \cdot 150)$. In the bidirectional case, in order to account for the client component, the number of states are doubled: we consider positive states $(i > 0)$ for packets coming from the server, and negative states $(i < 0)$ for packets coming from the client [6].

By taking the combinations of flow directions and the types of states, we have four models to classify the chosen malware:

- **M1:** states are the TLS record content types, built considering client and server traffic (labeled: 'types + bidirectional').
- **M2:** states are the TLS record content types, built considering only traffic flows coming from the server (labeled: 'types + unidirectional').
- **M3:** states are the packet lengths, built considering client and server traffic (labeled: 'lengths + bidirectional').
- **M4:** states are the packet lengths, built considering only traffic flows coming from the server (labeled: 'lengths + unidirectional').

2.2.1 Sequence normalization

Several state sequences have been extracted from the pcap files described in section 2.1, the states being TLS record types or packet lengths, depending on the model used.
Given that the length of these sequences varies between 1 and 1363, a length analysis has been performed in order to restrict the interval of length sequences used to build and validate the models, ideally restricting it to only one sequence length. The reason being that longer sequences have an intrinsically lower probability of representing a TLS flow (see 1.3.1), thus confronting sequences of different lengths could be misleading. This length analysis process has been done separately for the unidirectional case and for the bidirectional case.

For each malware the four most common lengths have been retrieved, they are shown in table 2.3.

	Bidirectional				Unidirectional			
Malware	1st	2nd	3rd	4th	1st	2nd	3rd	4th
Cobalt Strike	**7**	6	9	8	**4**	3	6	2
Dridex	**7**	6	5	45	**3**	2	42	4
Emotet	**6**	8	7	9	**3**	4	2	5
Trickbot	**8**	9	7	6	**4**	3	5	6
none	**7**	6	5	8	**1**	3	2	4

Table 2.3: Four most common sequence lengths for each malware

From table 2.3, we can see that a good length for the bidirectional case is 7, and for the unidirectional is 3, given that they are common sequence lengths for every class.

Table 2.4 contains the number of flows used in model M1 and M3, and table 2.5 contains the number of flows used in model M2 and M4.

Malware	Build set [flows]	Validate set [flows]	Total [flows]
Cobalt Strike	11243	3708	14951
Dridex	30	44	74
Emotet	58	52	110
Trickbot	21	17	38
none	-	412	412

Table 2.4: Number of flows used in the bidirectional case

Malware	Build set [flows]	Validate set [flows]	Total [flows]
Cobalt Strike	1336	1531	2867
Dridex	40	49	89
Emotet	214	258	472
Trickbot	38	33	71
none	-	381	381

Table 2.5: Number of flows used in the unidirectional case

2.2.2 Validation

The models have been constructed using data from the build set, and validated using data from the validate set.

The validation phase consists on the application of the Maximum likelihood estimation: given a model and a flow in the validation set, its elements (packet length or TLS record type) are used to compute the transition probabilities relative to the four Markov chains, which represent a particular malware. The malware corresponding to the highest transition probability is used to classify the traffic flow.

If all the probabilities are 0 or lower than a specified threshold, the flow is classified as 'none', that is, not belonging to any of the malware considered.

3. Implementation: MalTLS

This chapter describes the code structure and implementation of MalTLS, the program written to build and validate the classification models. The last three sections correspond to the three main parts of the program, which creates the model 3.2, applies it to the validation data 3.3, and checks if the results are correct 3.4.

3.1 Code

The code for MalTLS can be found in [39]. The program takes two sets of pcap files, uses one set (build set) to build a Markov chain, representing the TLS content types sequence corresponding to a particular malware, and uses the second set (validation set) to validate the model. The program's file structure is described in table 3.1. Pcap files are parsed using Pyshark [40], a Tshark wrapper for python. The location of the pcap files and the output files is written in a .toml file.
Figure 3.1 contains an overview of the code structure.

3.2 Build

build.py uses parse_pcap.py to extract a list of TLS content types from the pcap files in the build set. It then calls build_markov.py to create the transition matrix, the enter probability vector, and the exit probability vector (see section 1.3.1), which are saved to a file

- *parse_pcap.py* takes a pcap file and searches for all TCP streams containing a TLS ServerHello message. For all of these TCP streams, it constructs a TLS content type sequence, or a packet length sequence, depending on the mode specified by the caller.
- *build_markov.py* takes as input a list of TLS content type sequences or a list pf packet lengths, as returned by parse_pcap.py. It considers the first and last element of every

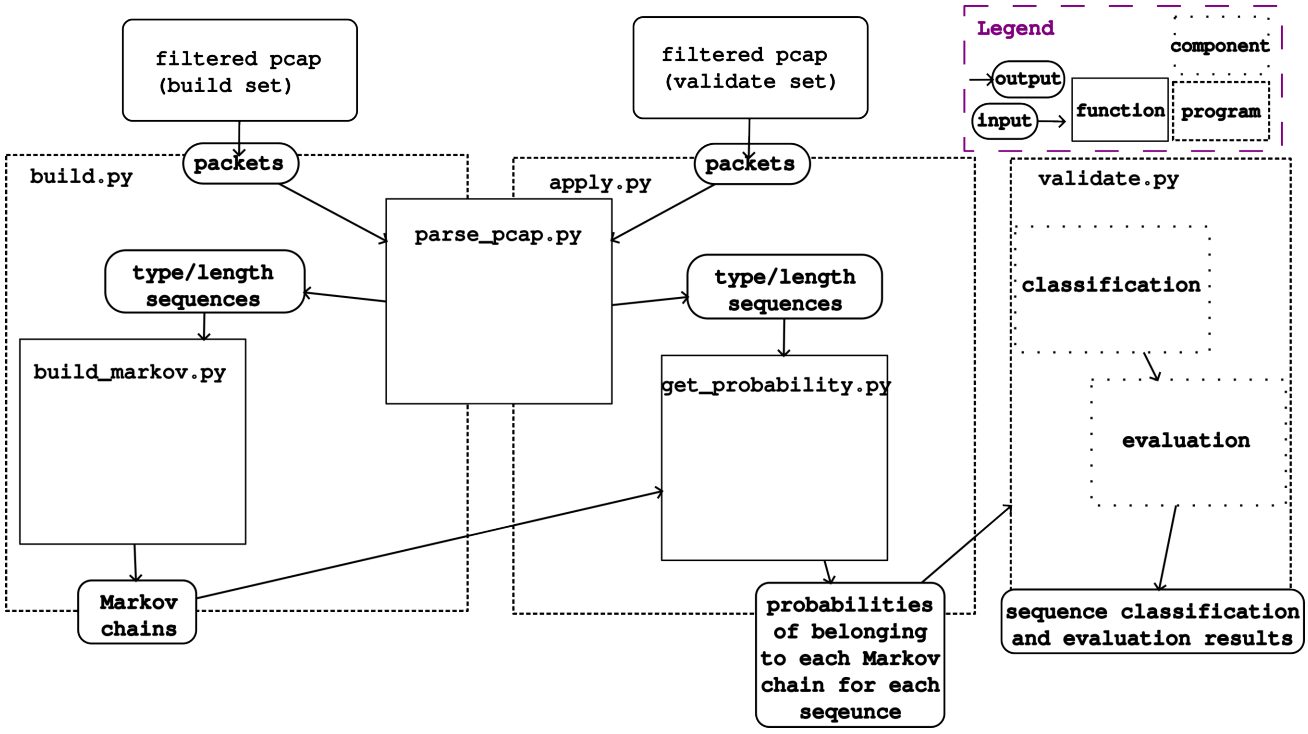

Figure 3.1: Code structure

sequence to compute the enter and exit probability vectors, respectively. The exit and enter vectors are represented by a dictionary: its keys are the Markov chain's states, and its values are the probability of entering or exiting the Markov chain with that specific state. The rest of the sequence elements is used to compute the transition matrix, which is represented by a nested dictionary. The first keys represent a state in the Markov chain; their values are themselves a dictionary, containing all the states reachable from the fist key (state), and their respective probability.

3.3 Apply

apply.py uses parse_pcap.py to extract a list of TLS content types or packet lengths from the pcap files in the validation set. For each malware, it retrieves the Markov chain computed by build.py and then calls get_probability.py to compute the probability that the sequence belongs to a particular malware.

- *parse_pcap.py* is described in the previous section.

Name	Description	Input	Output
apply.py	apply model	pcap files in validate set	classified sequences
build_markov.py	compute Markov chain	type or length sequences	Markov chain
build.py	build model	pcap files in build set	Markov chain model
get_probability.py	calculate probability of sequence belonging to a malware	Markov chain	a probability
parse_pcap.py	pcap file parsing	pcap file	type or length sequences
validate.py	validate apply results	apply results	model evaluation
maltls.toml	configuration file	-	-

Table 3.1: Files structure

- *get_probability.py* takes an entry probability vector, a transition matrix, an exit probability vector, and a sequence of TLS content types, or packet lengths. It uses its first three input elements to calculate the probability that the sequence belongs to the malware represented by this particular Markov chain.

3.4 Validate

validate.py takes the results of apply.py, and checks if the pcap have been correctly categorized. The result is a dictionary containing the number of true positives, false positives, and total number of sequences of each malware.
It additionally outputs the actual and predicted classification of each sequence, among with its classification probability for each class.

4. Evaluation

This chapter describes the results obtained from the four models. Section 4.1 contains the Markov chains corresponding to each malware. Sections 4.2 and 4.3 show the models' results when applied to the validation set. Section 4.4 contains a discussion of the aforementioned results.

In this chapter we show the output models of the Markov chain algorithm described in chapter 2. The models are presented by diagrams that graphically display the entry and exit probability vectors, as well as the transition matrix of the Markov chain. Each diagram shows a series of transition states between an entry state vector and exit state vector. The entry vector goes to the transition states according to the entry probability, and the transition states are linked to each other by transition probabilities. The exit vector is reached from the transition states according to the exit probability. These diagrams are shown in section 4.1.

To determine the performance of the results, we compute the confusion matrices corresponding to each model. Rows represent the actual classes, columns represent the predicted classes. The confusion matrices are used to calculate a series of parameters that allow to quantitatively assess the robustness of the results; such parameters are:

- **Recall:** it measures the number of samples correctly classified as one class out of all the samples actually belonging to that class, that is: $recall = TP \div (TP + FN)$, where TP is the number of true positives, and FN is the number of false negatives.
- **Precision:** it measures the number of samples actually belonging to a specific class out of all the samples classified as that specific class, that is: $precision = TP \div (TP + FP)$, where FP is the number of false positives.
- **F1-score:** the harmonic mean of precision and recall. $F1\ score = (2 \cdot precision \cdot recall) \div (precison + recall)$.

- **Accuracy:** the number of samples correctly classified among all the samples in the validate set, that is: $accuracy = (TP + TN) \div (TP + FP + TN + FN)$.
- **Micro F1-score:** the harmonic mean of the global precision and the global recall (that is, precision and recall calculated considering the sum of the class-wise TP, FP, and FN).
- **Recall mean:** we calculate the recall measure for our five categories (Cobalt Strike, Dridex, Emotet, Trickbot, and 'none'), and then compute its arithmetic mean: $recall\ mean = (\sum_{t=1}^{5} Recall_i) \div 5$.

Table 4.1 summarizes the performance for each model.

Model	Recall mean	Micro F1-score
M1 (types+bidirectional)	**0.9435**	0.9577
M2 (types+unidirectional)	0.8461	**0.9600**
M3 (lengths+bidirectional)	0.6937	0.9428
M4 (lengths+unidirectional)	0.7366	0.7877

Table 4.1: Recall mean and micro F1-score of every model

4.1 Malware's Markov chains

What follows are the Markov chains build for the four models, along with their enter and exit probability vectors.

Nodes represent the transition matrices states, edges represent the transition probabilities: if lower than 1, their values are written on the edges. Enter and exit probabilities are represented with dotted edges.

Cobalt Strike M3 and M4 Markov chains, and Emotet M3 Markov chain are not shown for the sake of clarity, as they contain too many transitions.

- **M1:** types + bidirectional

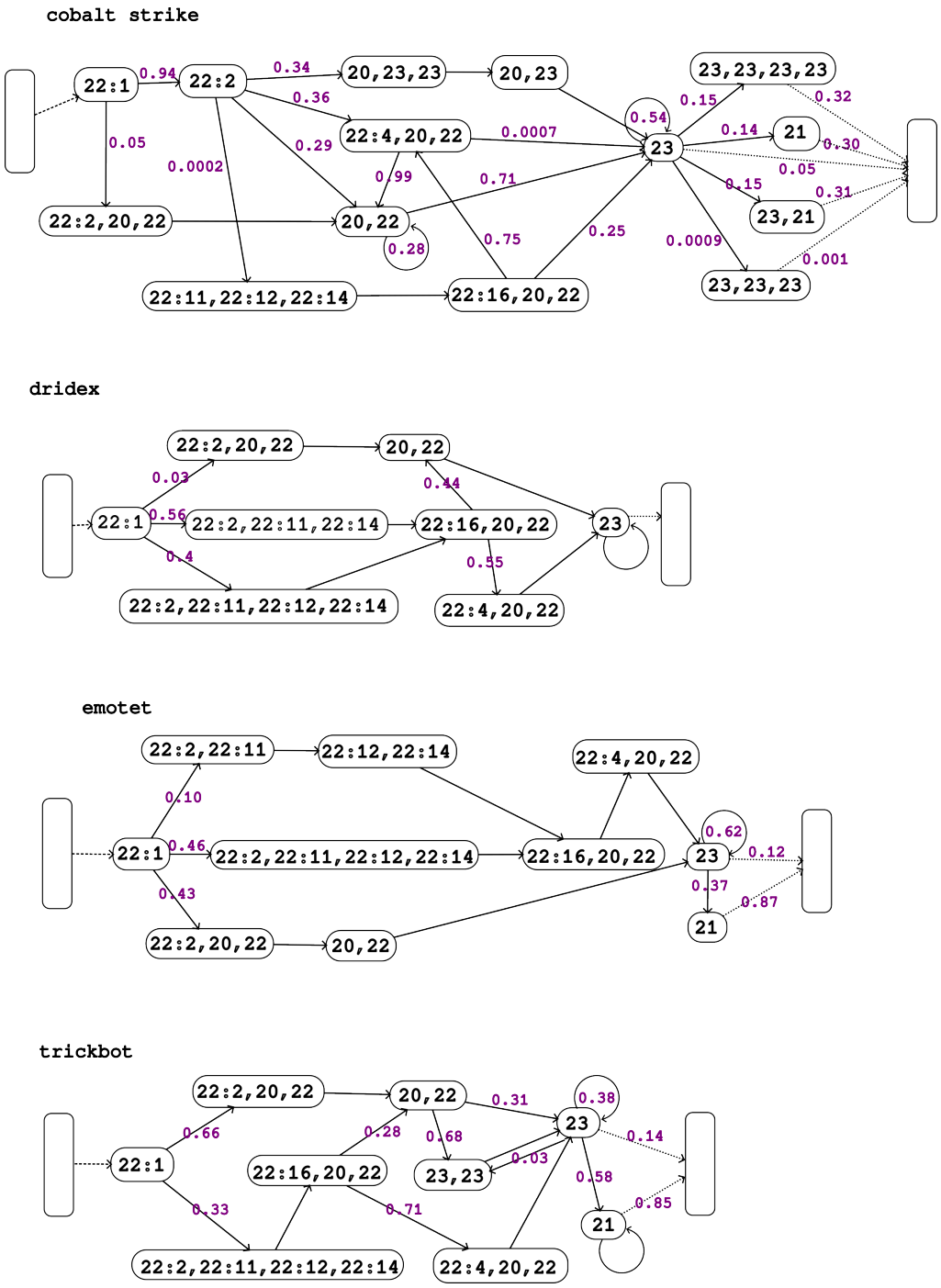

Figure 4.1: M1 model - Markov chains

- **M2:** types + unidirectional

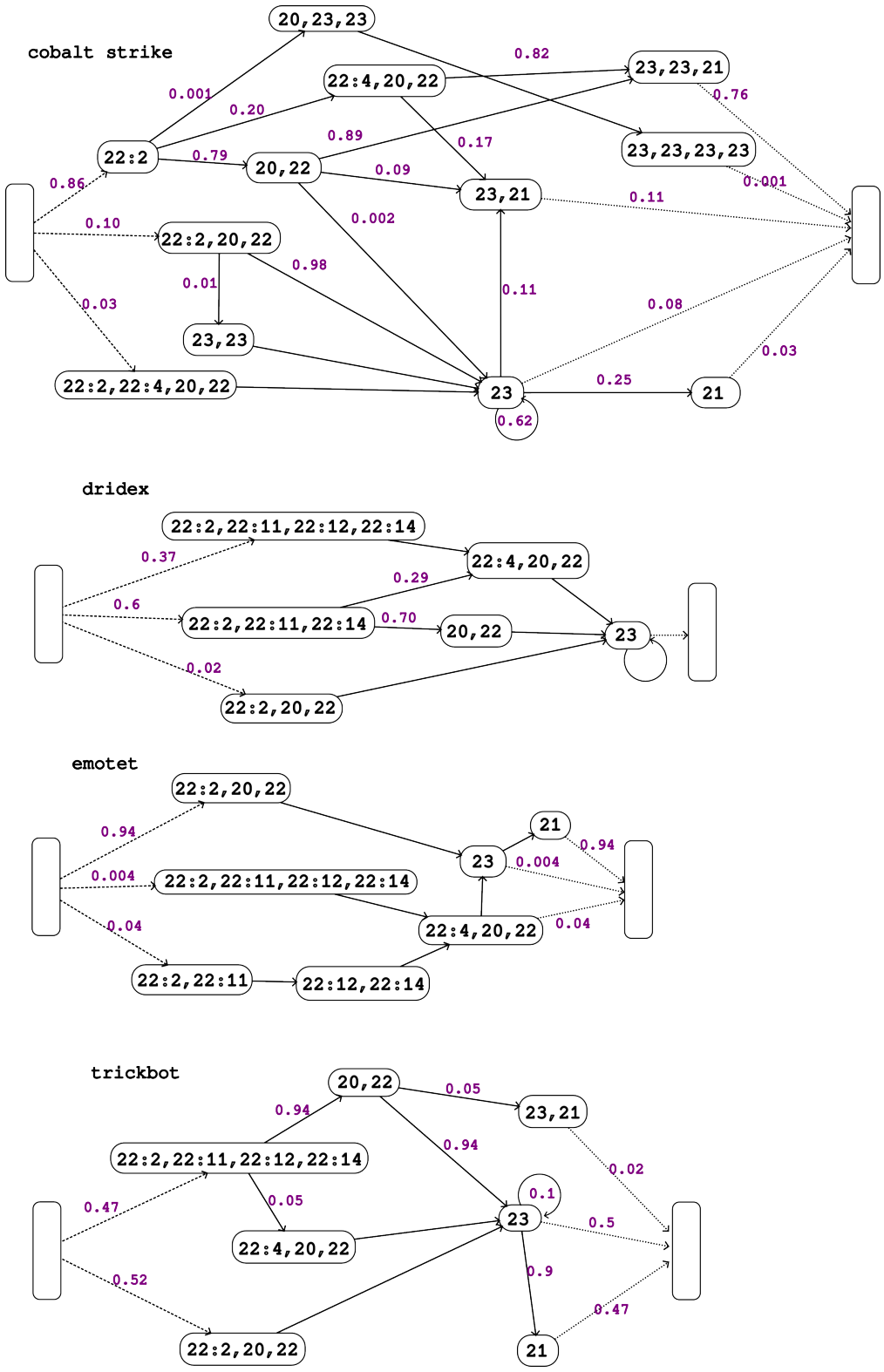

Figure 4.2: M2 model - Markov chains

- **M3:** lengths + bidirectional

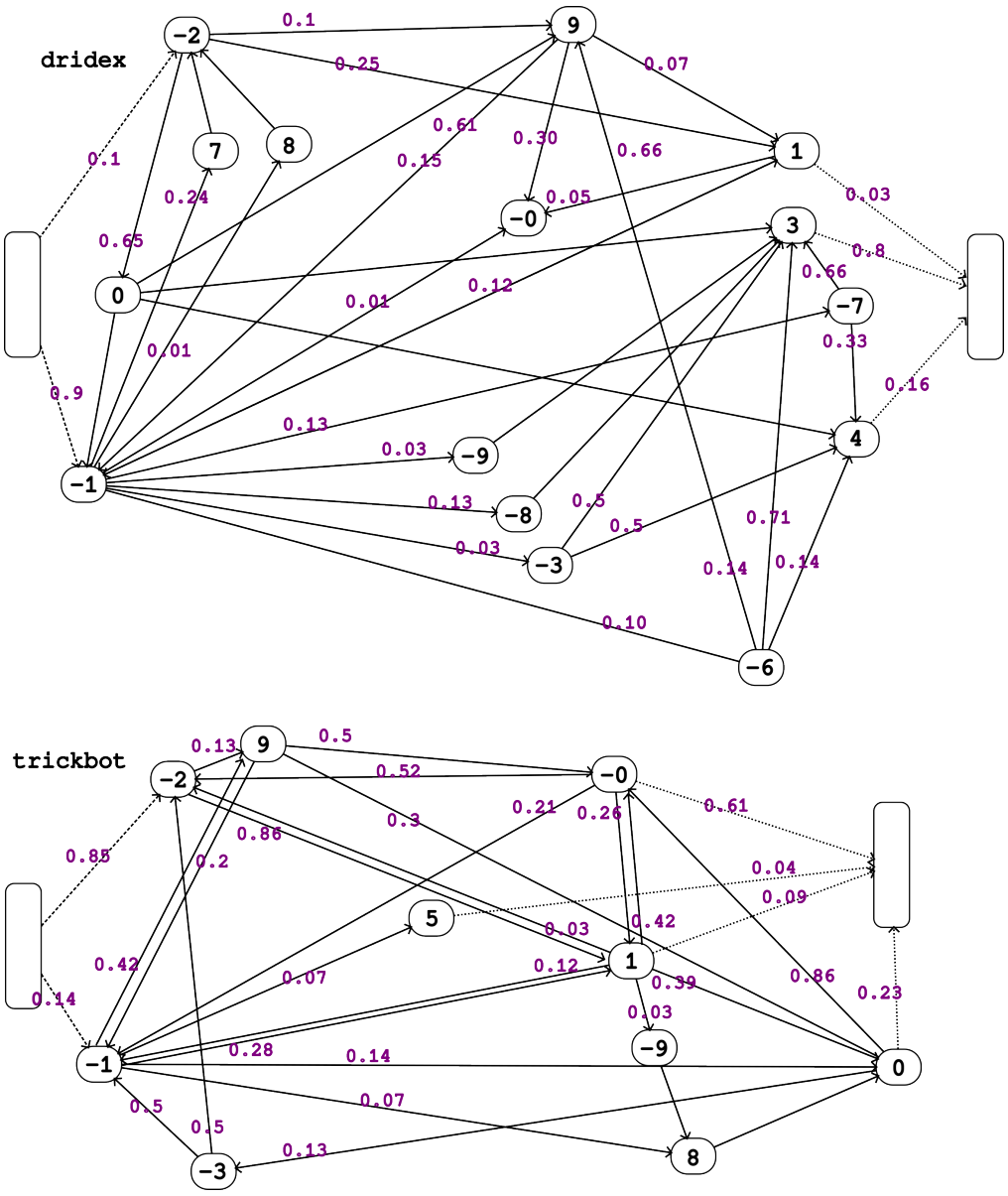

Figure 4.3: M3 model - Dridex and Trickbot Markov chains

- **M4:** lengths + unidirectional

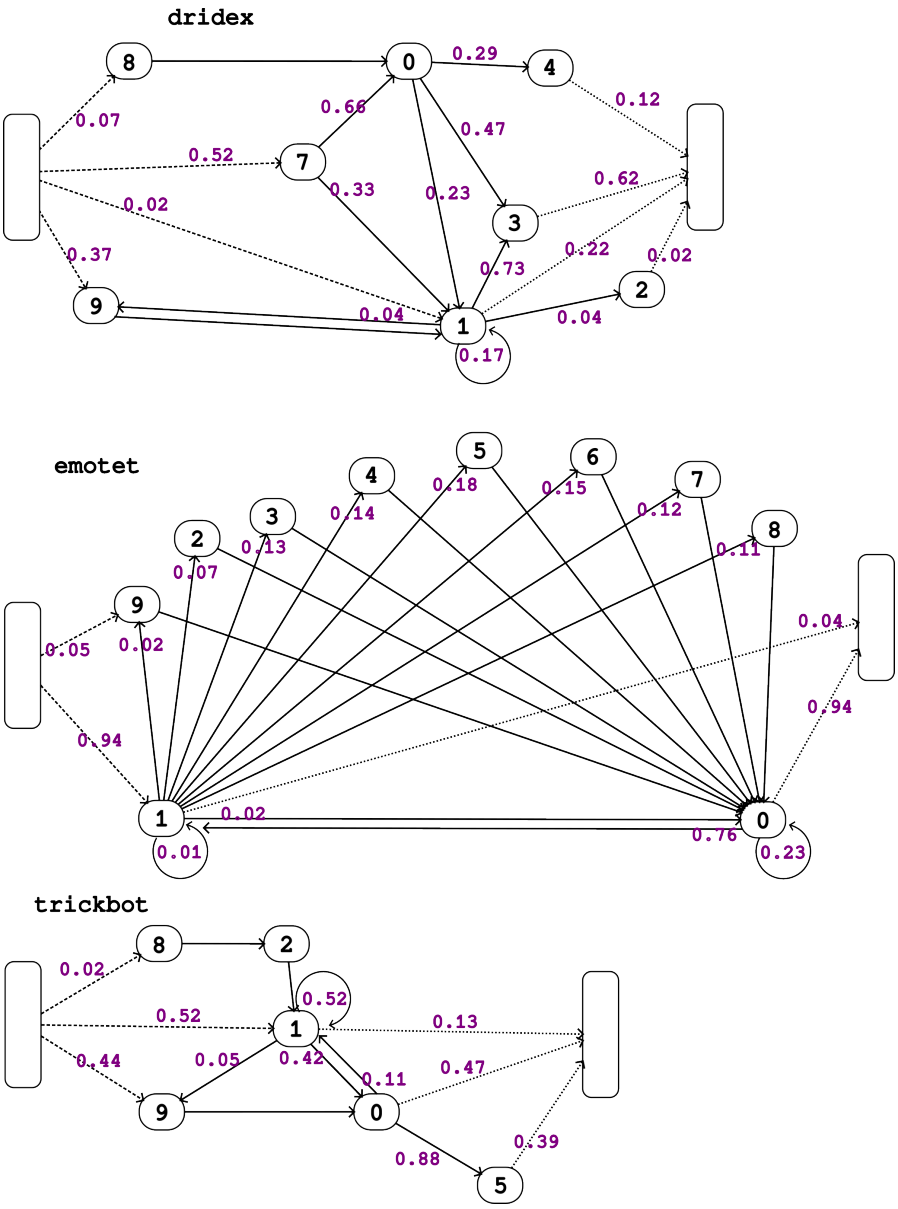

Figure 4.4: M4 model - Dridex, Emotet, and Trickbot Markov chains

4.2 Type based models results

This section contains the results for the two type based models: M1 and M2. Table 4.2 contains the results of the evaluation measures applied to each class. Figure 4.5 and figure 4.6 show the confusion matrices for model M1 and model M2, respectively.

	Bidirectional (M1)				Unidirectional (M2)			
Malware	Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score	Accuracy
Cobalt Strike	1.0000	0.9539	0.9770	0.8537	1.0000	0.9621	0.9807	0.7380
Dridex	0.2756	0.9772	0.4300	0.5026	0.8596	1.0000	0.9245	0.5055
Emotet	0.9792	0.9038	0.9400	0.5025	0.9257	0.9651	0.9450	0.5285
Trickbot	0.8824	0.8824	0.8824	0.5008	0.1493	0.3030	0.19999	0.4985
none	0.8677	1.0000	0.9290	0.5260	0.9870	1.0000	0.9935	0.5463

Table 4.2: Per-class evaluation measures in type based models

- **M1:** types + bidirectional

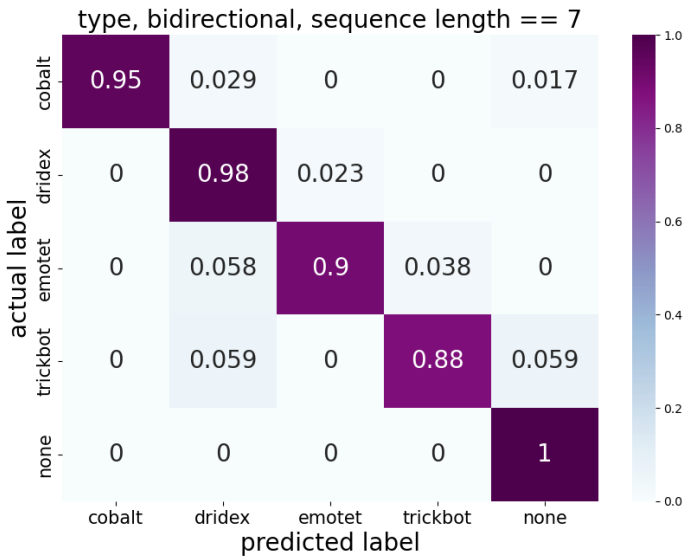

Figure 4.5: M1 model - confusion matrix

- **M2:** types + unidirectional

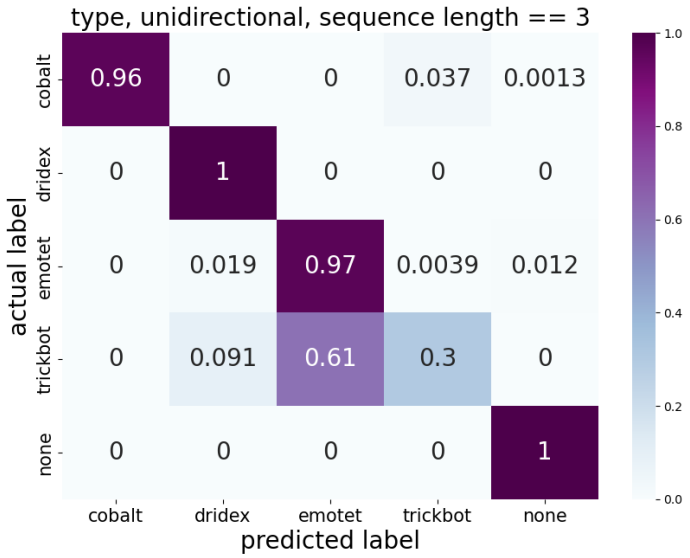

Figure 4.6: M2 model - confusion matrix

4.3 Length based models results

This section contains the results for the two length based models: M3 and M4. Table 4.3 contains the results of the evaluation measures applied to each class. Figure 4.7 and figure 4.8 show the confusion matrices for models M3 and M4, respectively.

	Bidirectional (M3)				Unidirectional (M4)			
Malware	Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score	Accuracy
Cobalt Strike	0.9977	0.9520	0.9743	0.8534	0.8020	0.9288	0.8608	0.7891
Dridex	1.0000	0.3864	0.5574	0.4994	0.8654	0.9184	0.8911	0.5046
Emotet	1.0000	0.3654	0.5352	0.4992	0.9725	0.9612	0.9669	0.5281
Trickbot	1.0000	0.7647	0.8667	0.5005	0.8387	0.7879	0.8125	0.5021
none	0.6378	1.0000	0.7788	0.5272	0.2340	0.0866	0.1264	0.4597

Table 4.3: Per-class evaluation measures in length based models

- **M3:** lengths + bidirectional

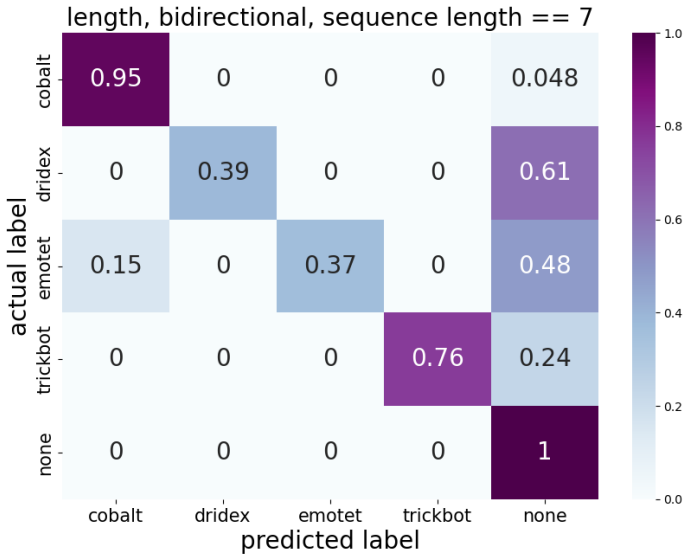

Figure 4.7: M3 model - confusion matrix

- **M4:** lengths + unidirectional

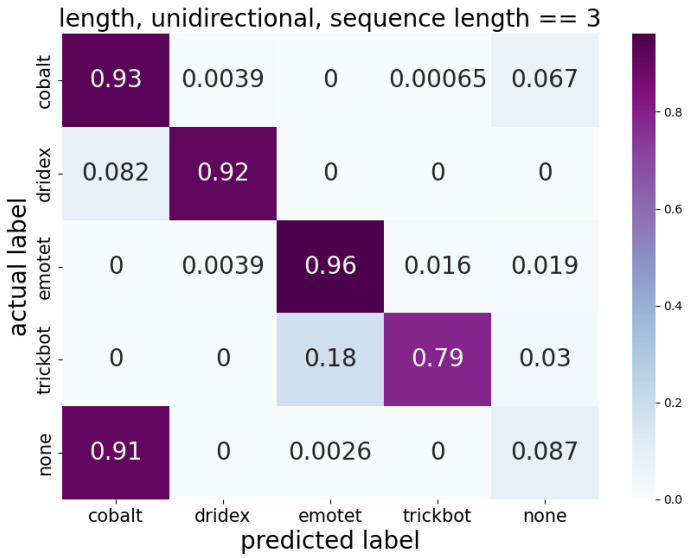

Figure 4.8: M4 model - confusion matrix

4.4 Discussion

- **M1 (types + bidirectional)):** this model has the highest recall mean (0.9435), and the second highest micro F1-score (0.9577). It behaves well even with Trickbot, which has the least training data among all the classes in the bidirectional case.
- **M2 (types + unidirectional):** it has the highest micro F1-score (0.9600), but can not accurately classify Trickbot, which is most of the time classified as Emotet.
- **M3 (length + bidirectional):** it has the lowest recall mean (0.6937), and can not reliably classify Dridex and Emotet. It has a particularly high number of not recognized malware samples, therefore classified as 'none'.
- **M4 (lengths + unidirectional):** it has the lowest micro F1-score among all the models (0.7877). It is the only one with a low recall for the class 'none'.

All type based models have no false negatives for the 'none' class, that is no benign traffic flow was classified as belonging to a malware. There are however many false positives for the 'none' class, especially in M3.
Trickbot accuracy is relatively low in all the models, which is likely caused by its lower number of flows in our build set.

The type based methods confirm that TLS record type sequences are valid fingerprint parameters when considering an entire TLS traffic flow.

The length based methods behave overall poorly, even though Ha and Roh [6] reported packet length sequence as more representative that TLS record type. Model M3 in particular, the one that can be directly compared with one of the models studied in Ha and Roh [6] [41], has a lower recall mean than expected, when considering only the malware classes (excluding 'none'). This could be perhaps be explained by the higher number of samples in their training set.

5. Future works

Future works could include:

- Testing the models on more diverse benign traffic, given that this traffic was captured from a single machine and in a limited time frame.
- Building and testing the bidirectional models on traffic not generated in a sandbox, in order to study the classification model behavior on traffic generated by diverse clients.
- Considering different versions of the same malware. Given the continuous development of the types of malware considered here, it would be useful to extend the classification models to ”sub-categories” of a specific malware.
- Adding the computation of an external fingerprint, such as JA4 [4], to provide an additional check, and perhaps decrease the number of false positive cases for the ’none’ class.

Conclusions

In this thesis we have implemented and evaluated four Markov chain based malware classification methods, based on TLS record type sequences or packet length sequences, and considering both bidirectional and unidirectional traffic.
We used confusion matrices to estimate the performance of each model, by computing the micro F1-score and recall mean. This analysis show that all models have a micro F1-score of at least 0.7877, with the type based models performing particularly well with most classes.

Cobalt Strike precision and recall are among the highest across all models, suggesting that the number of training data was enough to create a reliable model for this malware.
In the type based models case, it is interesting to note that Dridex recall is high even if the number of build sequences for this class is relatively low.
The bidirectional length based model (M3) has the lower recall mean, and fails to reliably classify two out of four malware. Furthermore in this model the number of sequences in the build set does not always correlate to a class performance.

Future developments of the present work will likely require a more in-depth analysis of the build dataset, in order to properly determine the best set of parameters to build the fingerprints.

List of Acronyms

List of Figures

List of Tables

Bibliography

[1] S. Gallagher. Nearly half of malware now use TLS to conceal communications. [Online]. Available: https://news.sophos.com/en-us/2021/04/21/nearly-half-of-malware-now-use-tls-to-conceal-communications/

[2] M. Korczynski and A. Duda, "Markov chain fingerprinting to classify encrypted traffic," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. IEEE, pp. 781–789. [Online]. Available: http://ieeexplore.ieee.org/document/6848005/

[3] 11+ latest SSL certificates statistics december 2024: TLS stats. [Online]. Available: https://sslinsights.com/ssl-certificates-statistics/

[4] J. Althouse. JA4+ network fingerprinting. [Online]. Available: http://blog.foxio.io/ja4+-network-fingerprinting

[5] mercury/doc/npf.md at main · cisco/mercury. [Online]. Available: https://github.com/cisco/mercury/blob/main/doc/npf.md

[6] J. Ha and H. Roh, "Experimental evaluation of malware family classification methods from sequential information of TLS-encrypted traffic," vol. 10, no. 24, p. 3180, number: 24 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/2079-9292/10/24/3180

[7] C. Allen and T. Dierks, "The TLS protocol version 1.0," num Pages: 80. [Online]. Available: https://datatracker.ietf.org/doc/rfc2246

[8] T. Dierks and E. Rescorla, "The transport layer security (TLS) protocol version 1.1," num Pages: 87. [Online]. Available: https://datatracker.ietf.org/doc/rfc4346

[9] E. Rescorla and T. Dierks, "The transport layer security (TLS) protocol version 1.2," num Pages: 104. [Online]. Available: https://datatracker.ietf.org/doc/rfc5246

[10] E. Rescorla, "The transport layer security (TLS) protocol version 1.3," num Pages: 160. [Online]. Available: https://datatracker.ietf.org/doc/rfc8446

[11] B. Anderson, S. Paul, and D. McGrew, "Deciphering malware's use of TLS (without decryption)." [Online]. Available: http://arxiv.org/abs/1607.01639

[12] Cobalt strike | adversary simulation and red team operations. [Online]. Available: https://www.cobaltstrike.com

[13] Cobalt strike | defining cobalt strike components & BEACON. [Online]. Available: https://cloud.google.com/blog/topics/threat-intelligence/defining-cobalt-strike-components

[14] MAR 10339794-1.v1 – cobalt strike beacon | CISA. [Online]. Available: https://www.cisa.gov/news-events/analysis-reports/ar21-148a

[15] Dridex malware | CISA. [Online]. Available: https://www.cisa.gov/news-events/cybersecurity-advisories/aa19-339a

[16] Emotet malware | CISA. [Online]. Available: https://www.cisa.gov/news-events/cybersecurity-advisories/aa20-280a

[17] TrickBot malware | CISA. [Online]. Available: https://www.cisa.gov/news-events/cybersecurity-advisories/aa21-076a

[18] B. Anderson and D. McGrew, "OS fingerprinting: New techniques and a study of information gain and obfuscation," in *2017 IEEE Conference on Communications and Network Security (CNS)*, pp. 1–9. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8228647/keywords#keywords

[19] M. Husák, M. Čermák, T. Jirsík, and P. Čeleda, "HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting," vol. 2016, no. 1, p. 6. [Online]. Available: https://doi.org/10.1186/s13635-016-0030-7

[20] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli, "Traffic classification through simple statistical fingerprinting," vol. 37, no. 1, pp. 5–16. [Online]. Available: https://dl.acm.org/doi/10.1145/1198255.1198257

[21] X. Yang, J. Xu, and G. Li, "Efficient fingerprinting attack on web applications: An adaptive symbolization approach," vol. 12, no. 13, p. 2948, number: 13 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/2079-9292/12/13/2948

[22] J. Zirngibl, F. Gebauer, P. Sattler, M. Sosnowski, and G. Carle, *QUIC Library Hunter: Identifying Server Libraries Across the Internet*.

[23] T. Bujlow, V. Carela-Español, J. Solé-Pareta, and P. Barlet-Ros, "Web tracking: Mechanisms, implications, and defenses." [Online]. Available: http://arxiv.org/abs/1507.07872

[24] "ssllabs/sslhaf," original-date: 2012-07-19T20:37:44Z. [Online]. Available: https://github.com/ssllabs/sslhaf

[25] Ivan ristić: HTTP client fingerprinting using SSL handshake analysis. [Online]. Available: https://blog.ivanristic.com/2009/06/http-client-fingerprinting-using-ssl-handshake-analysis.html

[26] Qualys SSL labs - SSL pulse. [Online]. Available: https://www.ssllabs.com/ssl-pulse/

[27] B. Anderson and D. McGrew, "Identifying encrypted malware traffic with contextual flow data," in *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*. ACM, pp. 35–46. [Online]. Available: https://dl.acm.org/doi/10.1145/2996758.2996768

[28] Z. Gancheva, "TLS fingerprinting techniques," medium: PDF Publisher: Chair of Network Architectures and Services, Department of Computer Science, Technical University of Munich. [Online]. Available: https://www.net.in.tum.de/fileadmin/TUM/NET/NET-2020-04-1/NET-2020-04-1_04.pdf

[29] "cisco/joy," original-date: 2016-01-08T20:30:43Z. [Online]. Available: https://github.com/cisco/joy

[30] B. Anderson and D. McGrew, "Accurate TLS fingerprinting using destination context and knowledge bases." [Online]. Available: http://arxiv.org/abs/2009.01939

[31] "salesforce/ja3," original-date: 2017-06-13T22:54:10Z. [Online]. Available: https://github.com/salesforce/ja3

[32] Advancing threat intelligence: JA4 fingerprints and inter-request signals. [Online]. Available: https://blog.cloudflare.com/ja4-signals

[33] "salesforce/jarm," original-date: 2020-07-09T22:19:04Z. [Online]. Available: https://github.com/salesforce/jarm

[34] J. A. Lindeman, Laura. Easily identify malicious servers on the internet with JARM. [Online]. Available: https://engineering.salesforce.com/easily-identify-malicious-servers-on-the-internet-with-jarm-e095edac525a/

[35] Wireshark · go deep. [Online]. Available: https://www.wireshark.org/

[36] Tshark | tshark.dev. [Online]. Available: https://tshark.dev/

[37] malware-traffic-analysis.net. [Online]. Available: https://www.malware-traffic-analysis.net/index.html

[38] D. McGrew and B. Anderson, "Enhanced telemetry for encrypted threat analytics," in *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/7785325

[39] blasevich/maltls. [Online]. Available: https://github.com/blasevich/maltls

[40] D. Green, "KimiNewt/pyshark," original-date: 2013-12-28T14:38:22Z. [Online]. Available: https://github.com/KimiNewt/pyshark

[41] H. Kim, M. Kim, J. Ha, and H. Roh, "Revisiting TLS-encrypted traffic fingerprinting methods for malware family classification," in *2022 13th International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 1273–1278, ISSN: 2162-1241. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9952872