FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Automated Creation of TLS Fingerprinting Database
Student:	Anton Aheyeu
Supervisor:	Ing. Karel Hynek
Study program:	Informatics
Branch / specialization:	Computer Security and Information technology
Department:	Department of Computer Systems
Validity:	until the end of summer semester 2021/2022

Instructions

Study the area of network monitoring based on deep packet inspection and (extended) IP Flows. Study the TLS protocol with focus on TLS fingerprinting techniques. Implement a plugin for the ipfixprobe flow exporter [1] to enrich IP flow data with information about application process identification (using, e.g., osquery [2]). By using the implemented plugin, create a TLS fingerprinting database. Develop a NEMEA [3] module capable of TLS fingerprinting of real TLS traffic using a created database. Evaluate the throughput of the implemented ipfixprobe plugin and the performance of the NEMEA module.

[1] https://github.com/CESNET/ipfixprobe
[2] https://osquery.io
[3] https://nemea.liberouter.org

Electronically approved by prof. Ing. Pavel Tvrdík, CSc. on 3 February 2021 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Automated Creation of TLS Fingerprinting Database

Anton Aheyeu

Department of Information Security
Supervisor: Karel Hynek

February 8, 2022

Acknowledgements

I would like to thank my supervisor Ing. Karel Hynek for valuable advice, patient guidance and willingness to help during my work on the thesis. I would also like to thank my family and friends for their love and support.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on February 8, 2022 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Aheyeu, Anton. *Automated Creation of TLS Fingerprinting Database.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Abstrakt

Tato práce se zabývá návrhem a implementací modulu pytrap pro systém NEMEA, který umožňuje identifikovat komunikující procesy pomocí databáze otisků TLS. Pro automatické vytváření databáze TLS otisků byl vyvinut zásuvný modul pro exportér síťových toků ipfixprobe, který využívá framework osquery. V teoretické části uvádíme základní pojmy a principy monitorování sítě a protokolu TLS, popisujeme, jak zásuvný modul získává informace o identifikaci procesů, a proces vytváření databáze otisků TLS. Na základě teoretické části jsme implementovali modul pytrap, zásuvný modul pro exportér toků a doplňkový program pro vytváření databáze otisků TLS, popsaný v praktické části práce. Výsledky testů potvrdily funkčnost a ukázaly úspěšnost vytvořených modulů.

Klíčová slova Monitorování sítě, TLS otisk, osquery, NEMEA, JA3.

Abstract

This thesis is about the design and implementation of the pytrap module for the NEMEA system which enables the acquisition of information about application process identification using the TLS fingerprinting database. The plugin for the ipfixprobe network flow exporter has been developed to automatically create a TLS fingerprint database, which uses the osquery framework. In the theoretical part, we introduce the basic terms and principles of network monitoring and the TLS protocol, describe how the plugin obtains information about process identification and the process of creating a TLS fingerprinting database. Based on the theoretical part, we have implemented the pytrap module, the plugin for the flow exporter and an additional program for creating a TLS fingerprinting database described in the practical part of the thesis. The test results confirm the functionality and show the success rate of the created module and plugin.

Keywords Network monitoring, TLS fingerprint, osquery, NEMEA, JA3.

Contents

List of Figures

List of Tables

Introduction

Network traffic monitoring is an essential prerequisite for proper computer network management. Captured traffic needs to be processed and analyzed for a variety of reasons. Such reasons can be, for example, providing basic network functionality or detecting potential security problems. Detecting malicious network traffic is equally important. Network monitoring is a prerequisite for maintaining network security and minimizing the overall security risks and threats.

TLS fingerprinting is a method that uses the metadata sent during a TLS negotiation to determine the communicating application. TLS fingerprint database is needed for the correct functioning of this method, where metadata are assigned to a specific application. A high-quality database contributes to more successful application identification and has great business potential.

In this thesis, we aim to develop tools that can create and fill such a fingerprint database. Moreover a module that can recognize not only the communicating application but also the system parameters of the device by TLS fingerprint using this database. This module will be implemented for an open-source monitoring system NEMEA and subsequently tested on a functional network.

Goals

In this work, we are implementing a software tool that allows users to get the system information of devices in computer networks knowing the TLS fingerprint. The tool will be used as a module in NEMEA, the system for network traffic analysis. This module will enrich the network traffic generated by observed devices using a database that we have purposely created.

Each row in the database contains a fingerprint and a list of system parameters. To obtain a fingerprint, we will use the existing TLS plugin provided by the ipfixprobe flow exporter. In order to get system parameters, it is necessary

to develop a new plugin, which will use the osquery framework to find system parameters belonging to a specific flow.

In addition, to combine the results into a common database, we need to create a program that can compare and merge the results of the existing plugin and with the one we created.

Structure of the Thesis

This thesis is divided into four chapters. Chapter 1 provides a theoretical background on the following topics: an overview of network traffic analysis, TLS protocol focusing on TLS fingerprinting techniques, and an introduction to osquery framework. The main part of our work is described in Chapters 2 and 3. Module integration methods and the process of selecting system parameters obtained using the osquery framework are described in Chapter 2. The details of implementation steps can be found in Chapter 3. Finally, Chapter 4 concludes the results of the work and evaluates the performance of the module.

Chapter 1

Background

This chapter provides an introduction to network traffic analysis, specifically network monitoring methods, and then illustrates how the TLS protocol works in detail and describes the basic concepts of the osquery framework.

1.1 Network monitoring

Network monitoring is essential for maintaining the security of the infrastructure. It usually consists of the use of monitoring systems (network probe), which is directly connected to the monitored infrastructure and analyzes its traffic. This allows to detect sources of attacks and protect network weaknesses, control whether users only use authorized services and do not exceed data limits.

Network monitoring can be divided into two categories depending on the approach to traffic analysis. The first type is deep packet inspection (DPI), which processes each packet separately [1]. This approach is described in the 1.1.1 section below. The main principle of the DPI analysis is that unwanted behaviour is mainly based on the content of the communication. There is another type of analysis, that examines data at a higher level of abstraction, called flow-based monitoring [2], which will be discussed in detail in the 1.1.2 section. This method analyzes the flow, which is a sequence of packets belonging to the communication and looks at it as a separate unit. In a network flow analysis, anomaly detection is performed using more general communication information, such as communication time duration and number of packets transmitted.

1.1.1 Deep packet inspection

The usual approach to monitoring network traffic of the DPI method is to record current traffic and then analyze individual packets and their contents [3]. Because deep packet inspection can inspect the contents of packets,

it can figure out where it came from, for example, the service or application that sent it. The detection algorithm looks for suspicious behavior or specific communication patterns and reports detected anomalies, attacks and other adverse events.

One of the main advantages of this method is evaluating the content of packets and creating rules according to which DPI handles these or those packets. In other words, the system can sort and filter packets. Intuitively, we notice the disadvantage of this method, which lies in the fact that careful processing of packages can slow down a network.

The task of this thesis implies the use of the NEMEA system [4], which uses flow monitoring and therefore this approach will be discussed in the next section.

1.1.2 Flow-based monitoring

Monitoring large networks using packet analysis is often impossible due to high demands on computing power. For this reason, the principle of network flow analysis, called traffic flow or just flow, is commonly used in such big networks.

Systems based on the principle of network flows analysis do not have detailed information about ongoing communication, as individual packet analysis can have, but they can detect other types of events due to their operation at a higher level of abstraction.

1.1.2.1 Traffic flow

In order to delve into the analysis of network traffic in the context of flow-based monitoring method, we must first define the basic terms. Traffic flow is defined by Aitken et al. [5] as "*a set of packets or frames passing an observation point in the network during a certain time interval. All packets belonging to a particular flow have a set of common properties*".

Each flow has its own properties, and in order to understand if a given packet belongs to the selected flow, it is needed to check whether the packet satisfies the flow's properties. Such properties can be, for example, the destination IP address or some characteristics of the packet. Flow keys are those properties by which packets will be related to a particular flow. For example, the traditional "5-tuple" flow key consists of source and destination IP address, source and destination transport port, and transport protocol as shown in the figure 1.1.

Also, in addition to flow keys, other flow parameters can be: the number of packets, the time of acceptance of the first and last packets, and so on.

Note that a flow can consist of several packets or be empty, because there may be no packets satisfying the properties of that flow.

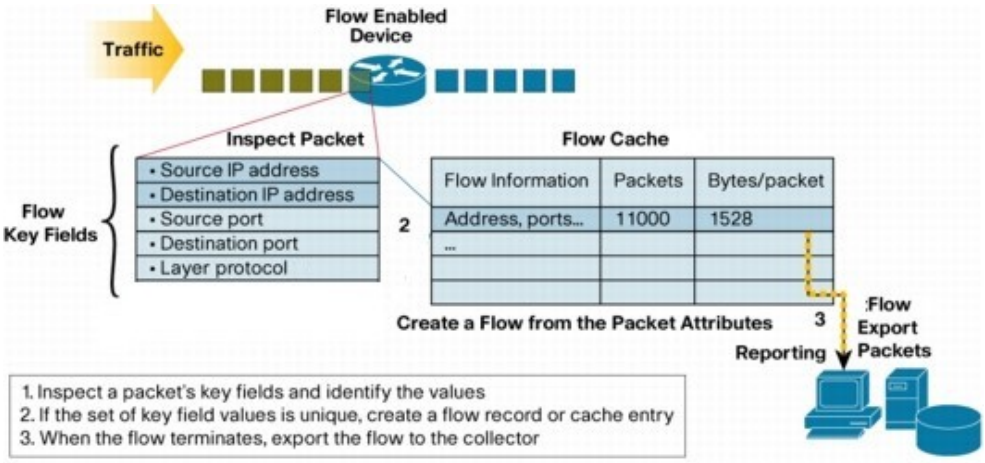

Figure 1.1: Traffic flow visualization

In addition to the usual unidirectional flow, there is also the bidirectional flow, called a biflow, which is packets flowing in both directions between two endpoints in the network [6]. This is the case because most of the application layer network protocols, such as POP3 or WebSocket, are inherently bidirectional. The bidirectional flow-based data model gives a better understanding of what's going on in the network and therefore provides opportunities for more effective security.

1.1.2.2 Flow enrichment

Flow enrichment is a process of adding or otherwise enhancing collected data with relevant context obtained from additional sources. For example, domains in DNS requests, HTTP headers, TLS extension values, the name of the process handling the connection or the host system parameters. An example of the extended flow fields are shown in figure 1.2. However, these information fields are not always formally defined.

1.1.2.3 Flow export formats

To ensure compatibility between different systems, there are several standardized protocols for sending network traffic records. Each protocol defines the format of the flow record and how data is exchanged between exporter and collector.

NetFlow v5 is a flow record format created by Cisco to capture network flows using Cisco routers and switches. It only supports a fixed set of exported fields in outgoing messages. As a result, extended network flows cannot be exported in this format. This is the most used version of NetFlow [7].

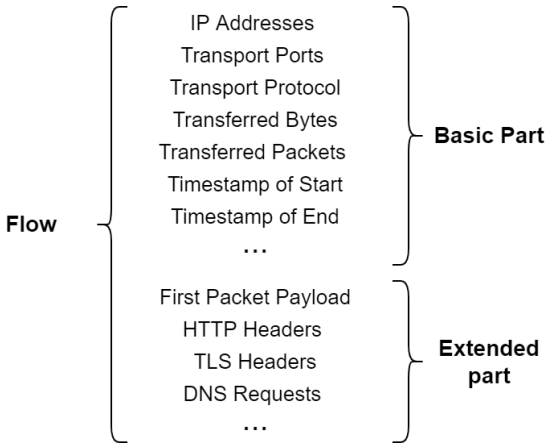

Figure 1.2: An example of an extended flow fields

NetFlow v7 is an enhancement to NetFlow v5 format that is not compatible with Cisco routers and only supports NetFlow with Cisco Catalyst 5000 series switches [8].

NetFlow v9 is an improved version of the NetFlow format that eliminates the disadvantages of previous versions by introducing templates. Due to the templates, the export is not limited by a fixed number of information fields in the message [9].

IPFIX is a standardized protocol for exporting network flows. It is also sometimes referred to as NetFlow v10. It brings the possibility of creating templates for flexible definition of fields for export. This protocol supports the export of extended network flows along with information from application layers. Moreover, it is possible to export multiple templates in one message [5].

UniRec is a format for efficient data transfer from the NEMEA system. Users can define any number of custom fields using a template. Only one template within one connection is allowed [4].

1.1.2.4 Flow monitoring infrastructure

A typical flow monitoring process consists of the following components:

Flow exporter — checks every packet of traffic passing through it, creates flow records by combining packet information and exports records to one or more collectors. Some of the existing flow exporters are:

Flowmon Probe is a powerful network exporter, developed by Flowmon Networks, that supports exporting information from the application layer in high-speed computer networks. It is a paid commer-

cial solution that can be delivered both as specialized stand-alone hardware and in the form of a virtualized tool [10].

ipfixprobe is an exporter written in the C++ programming language from the open-source project NEMEA, the extension of which is the subject of this work. It can be used to read packets stored in a pcap file, and also to obtain traffic from the network interface. Additionally, ipfixprobe has already a TLS plugin, with which it is possible to get a JA3 fingerprint [11].

yaf (Yet Another Flowmeter) is designed as a bidirectional network flow meter and used to capture flow information on a network. It processes packet data from packet captures (pcap) and exports obtained information in IPFIX format [12].

Cisco joy is software used for obtaining information about events that occur within a network flow, extracting data from packet capture for the protection of the networks being monitored with it. Moreover the obtained data is represented in JSON format since this format is supported and used by data analysis tools [13].

Flow Collector — is engaged in the collection, storage and primary processing of information received from a flow exporter. Below are examples of some flow collectors:

IPFIXcol2 is a flow data collector that supports such flow record formats as NetFlow v5, NetFlow v9 and IPFIX. The advantages of this collector include its open-source availability, an emphasis on parallel performance and no less important extensibility using plugins [14].

Flowmon Collector is an appliance (virtual and hardware) for collection, long-term storage and analysis of flow data. It can process data in NetFlow v5, NetFlow v9 and IPFIX formats too. One of the features is detection and diagnostics of operational and configuration issues [15].

Flow Analysis — identifies machines and devices that negatively affect network throughput, finds vulnerabilities in the system, and ultimately improves the overall efficiency of the network. Some of the existing solutions are:

NEMEA is an open-source modular system for flow detection and analysis of network traffic. The whole system consists of individual independent components, modules that can be interconnected via a communication interface. Each module has its specific task, for example: data acquisition and manipulation, detection of network

threats and malicious traffic, reporting incidents to system administrators. This solution allows scalability and very easy addition of new functionality through the modules to the system [4].

Flowmon ADS is a security solution that makes it possible to identify threats and infiltrate the network through various channels. It uses a large number of artificial intelligence algorithms such as dynamic standard behavior baseline, deviations, dynamic decision trees, machine learning and many others, to analyze multiple dimensions of network traffic flow [16].

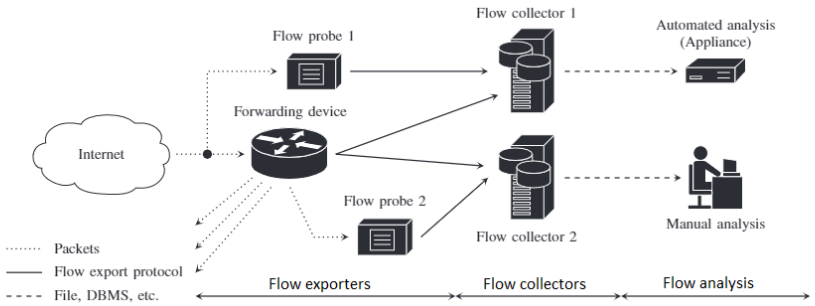

Figure 1.3: Architecture of a typical flow monitoring system [17]

1.2 Transport Layer Security

This section will describe the Transport Layer Security (TLS) protocol, the history of its development, as well as the basic concepts necessary to understand what TLS fingerprinting is.

TLS is a cryptographic protocol that is the successor to Secure Sockets Layer (SSL). TLS provides secure communication over a computer network using symmetric cryptography to encrypt the transmitted data. Keys are uniquely generated for each connection and are based on a shared secret negotiated at the start of a session, also known as a TLS handshake.
The protocol works at three levels of protection:

- **Confidentiality** using symmetric algorithms, TLS encrypts the data that is transmitted. If the data is intercepted, it will be impossible to read it;
- **Authentication** a guarantee that the exchange of data goes between those nodes for which the communication channel was originally created;
- **Integrity control** one-way hashing checks incoming information, excluding the possibility of substitution or distortion.

1.2.1 TLS history and versions

In the mid-90s, the Netscape Company released a protocol that improved the security of electronic payments. The protocol was named SSL and was the predecessor of the TLS protocol. Version 1.0 never went into production, being discarded during the testing phase. Version 2.0 [18] was released, but it had security holes. SSL 2.0 was deprecated in 2011.

In 1996, the shortcomings of version 2.0 were eliminated, and the world saw a completely working version — SSL 3.0 [19]. The protocol was implemented at the application level, over TCP. This allowed high-level protocols like HTTP to function correctly. SSL 3.0 was deprecated in 2015.

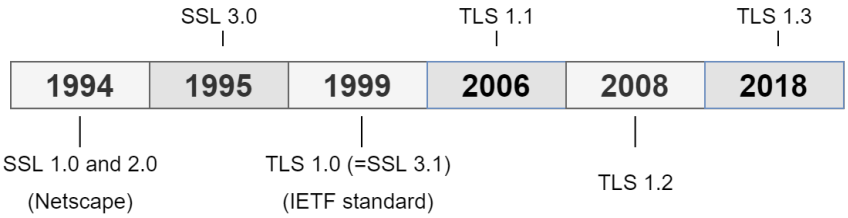

Figure 1.4: SSL/TLS protocol releases

In 1999, the next version was released, which is standardized by the Internet Engineering Task Force (IETF). The protocol is renamed TLS 1.0 [20].

7 years later, in the spring of 2006, the next version of the protocol was released — TLS 1.1 [21]. It has significantly expanded functions and eliminated current vulnerabilities. TLS 1.0 and TLS 1.1 have been deprecated in 2020 [22].

In 2008, TLS 1.2 [23] came out which changed the encryption methods. New block cipher modes have been introduced, and legacy cryptographic hashing methods have been prohibited.

The most recent version of the protocol to date is TLS 1.3 [24], released in 2018. It removed obsolete hashes, ciphers without authentication and open methods for obtaining keys to sessions. Deprecated options like helper messages and data compression have also been removed. A mandatory digital signature mode has been introduced, the approval and authentication processes have been separated. To improve the security of TLS, version 1.3 is not backward compatible with RC4 or SSL.

The main differences between versions 1.1 and 1.2 of TLS are security and performance improvements. To this end, in TLS 1.3, the key generation algorithm has been redesigned and known vulnerabilities have been fixed. The TLS 1.3 handshake also improves the message authentication and digital signature processes. Finally, in addition to phasing out old key generation or key exchange algorithms, TLS 1.3 removes old symmetric ciphers. TLS 1.3 has completely removed block ciphers.

1.2.2 TLS handshake

At the handshake stage, the connection parameters (protocol version, encryption method and others) are negotiated between the client and the server. The TLS 1.2 handshake process is shown in the figure 1.5.

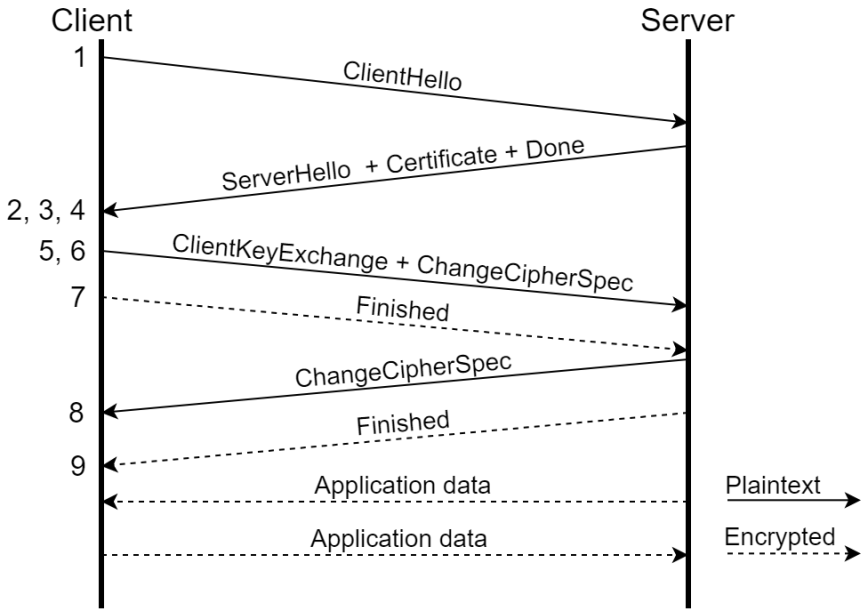

Figure 1.5: **TLS handshake** — the TLS handshake contains multiple messages sent in plain text, including ClientHello

According to the Husák et al. [25] and documentation [23] the handshake works in the following steps:

1. The client sends a **ClientHello** message indicating the latest version of the supported TLS protocol, a random number, and a list of supported cipher suites suitable for working with TLS;

2. The server replies with a **ServerHello** message containing: the protocol version selected by the server, a random number generated by the server, a selected cipher suite from the list provided by the client. If the client and server do not share cipher suites, then the connection fails;

3. The server sends a **Certificate** message that contains the server's digital certificate (depending on the encryption algorithm, this step can be skipped);

4. The **ServerHelloDone** message notifies the client that the server has finished transferring data;

5. The client then participates in generating the session key. The specifics of this step depend on the key exchange method that was selected in the original **ClientHello** message. The client sends a **ClientKeyExchange** message, which may contain a **PreMasterSecret**, a public key, or nothing (depends from the selected cipher). The client and server, using the **PreMasterSecret** key and randomly generated numbers, calculate the shared secret. All other information about the session key will be obtained from the shared secret;

6. The client sends a **ChangeCipherSpec** message, which indicates that all subsequent information will be encrypted with the algorithm established during the handshake using the shared secret;

7. The client sends the message **Finished**, which means that the handshake is complete on the client side. From this moment on, the connection is protected with a session key. The message contains data (MAC) that can be used to verify that the handshake has not been tampered with;

8. The server sends a **ChangeCipherSpec** message to notify that it is switching to an encrypted connection;

9. The server also sends a **Finished** message using the newly generated symmetric session key and checks the checksum to verify the integrity of the handshake.

After these steps, the TLS handshake is complete. Both parties now have a session key and can communicate over an encrypted and authenticated connection.

1.2.3 ClientHello message

The first message in the TLS handshake is the ClientHello message. This message is sent unencrypted and allows the client to specify a list of parameters and features that it supports [26]. This includes, the versions of TLS that the client supports, a list of supported compression methods and cipher suites that the client is willing to use, a random number used to protect against replay attacks, and a list of extensions.

Extensions allow the client and server to negotiate additional features and parameters. Although there are more than 20 extensions listed in different versions of TLS, we will focus on some of them that can be used for creating TLS fingerprint.

Server Name Indication (SNI) allows the client to specify the requested domain in the ClientHello message, allowing the server to send the appropriate certificate if the server supports multiple hosts. Since this is sent before the TLS handshake, SNIs will be sent unencrypted.

Application Layer Protocol Negotiation (ALPN) enables clients to negotiate the application protocols they support over TLS. For example, a web browser can specify HTTP/1.1 and HTTP/2.

Elliptic Curve Point Format defines the encoding formats supported by the client for sending or receiving elliptic curve points.

Supported Groups this extension defines a list of supported math groups that a client can use for authentication and key exchange.

Signature Algorithms clients can choose the combinations of hashing and signature algorithms they can support to authenticate other participants. They are usually in the form of a pair of signature and hashing algorithms, such as rsa_sha256 or ecdsa_sha512 [24].

Capturing ClientHello packets is a good way of TLS fingerprinting for many reasons:

- Since ClientHello is the first packet in a TLS connection, by analyzing it, we can make a security decision before starting the data exchange process;
- ClientHello packets are rare, so all ClientHello packets on the network can be captured for analysis using a relatively small storage size as opposed to capturing all packets [27].

1.2.4 TLS fingerprinting

The TLS fingerprint was developed to increase the visibility of encrypted TLS traffic by being able to identify the process that communicates using a given TLS connection. Based on the fact that ClientHello messages remain static from session to session for each client, capturing packets containing ClientHello makes it possible to create a fingerprint to recognize a specific client in further sessions.
There are two main approaches to creating a TLS fingerprint:

- **Cisco Mercury fingerprint** — captured fields are used to create the fingerprint: TLS version, TLS record version, cipher suites, compression options and a list of extensions [28];
- **JA3 fingerprint** — uses a similar approach as Cisco Mercury, but additionally collects data from three specific extensions: signature algorithms, elliptic curves and elliptic curve format [29]. This approach will be described in more detail below.

1.2.4.1 JA3 fingerprint

JA3 fingerprints [30] is a hash (SSL/TLS client application fingerprint). This approach makes it easy and efficient to classify client applications.

To initiate a session, the client sends a ClientHello packet [29]. If the server accepts connection, it will respond with a ServerHello packet, thereby continuing the encryption negotiation. Because TLS negotiations are sent unencrypted, client applications can be traced and identified using data from the ClientHello packet.

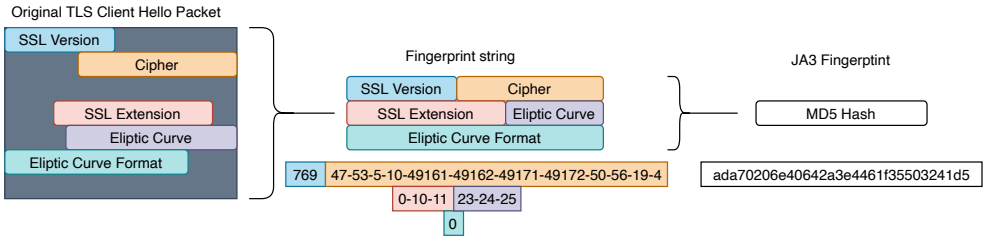

Figure 1.6: JA3 fingerprint creation process [31]

JA3 collects decimal byte values for the following fields: TLS version, accepted ciphers, extension list, elliptic curves and elliptic curve formats. The use of this combined data is not only reliable in terms of being static for a particular client, but also provides a greater granularity than evaluating cipher suites alone, which has significantly more fingerprint conflicts. It then concatenates these values to one string in the following order: SSLVersion, Ciphers, Extensions, EllipticCurves, EllipticCurvePointFormats as it is shown on the figure 1.6. If there are no TLS extensions in the ClientHello message, the corresponding field is left blank.

This string is MD5 hashed then to create a 32-character fingerprint that can be easily consumed and shared.

1.3 Osquery framework

Osquery is an open-source security tool that looks at the operating system as a single database with tables that can be queried using SQL statements [32]. Using these SQL queries, user can check the integrity of files, check the status and configuration of the firewall, perform a security check on the target server, and much more. It also provides an opportunity to obtain information about running processes, loaded kernel modules and opened network connections.

Osquery is cross-platform, and since osquery has the edge on low-level operating systems and includes many SQL tables for them, users can install and use it on Ubuntu, CentOS, MacOS and also on Windows systems [33].

1.3.1 Osquery features

One of the features of the osquery framework is `osqueryd` [34], which is a daemon for scheduling queries and recording the changes in the state of OS. This daemon accumulates and logs query data that can be viewed as a snapshot of the system configuration and status. This data can be further used for security analysis and for the system health check.

Another feature of the framework is `osqueryi` [35], which is the osquery interactive query console for working with SQL queries. This is a good tool for system diagnostics as it provides many tables that can be accessed using SQL syntax, which is intuitive to use.

Another equally important feature is the provision of public API, called the `SDK osquery` [36]. This is a set includes only the bulk of the source code called the core.

1.3.1.1 Osqueryd

The main task of the `osqueryd` daemon is to execute scheduled requests. Scheduled query is a list of SQL queries and their intervals (the approximate frequency of query execution, given in seconds).

The principal feature of the `osqueryd` is to treat the request as parameters of the operating system at a certain period of time. This means that the response to the first query, while the database has not been initialized, will contain every row from the resulting table. After a certain interval, a query will be executed again and log the monitored events that caused the changes. But if an event occurs and ends faster than the scheduled interval, the event will not be logged when the query is executed. In other words, the daemon is not designed for queries that track frequently occurring events.

1.3.1.2 Osqueryi

Interactive query console `osqueryi` works in completely standalone mode. Since osqueryi is an interface for working with SQL queries, it makes it easy to get system information in real time. These queries can be executed without administrator privileges, however, some tables may return fewer results.

The console is stateless and is not connected to the osqueryd daemon. Therefore, to systematically execute queries and record changes in the state of the operating system, it is advisable to consider creating a query schedule using osqueryd.

1.3.1.3 SDK osquery

This functionality is designed to create osquery extensions. To register tables, the extension must provide to the osquery core information about the structure of the new table and how to obtain the data. The whole communication works

in a way that there is an osquery core and extensions communicated with it within different processes. Extensions can be written in any language that support Thrift, such as C++, Python or Go.

Chapter 2

Analysis and Design

This chapter contains all the possible design decisions that needed to be made during the creation of this bachelor thesis.

2.1 Overall architecture

Figure 2.1 provides an abstract overview of the workflow of modules we need to implement. The blocks we need to create will be grayed out in the figure for better visibility.

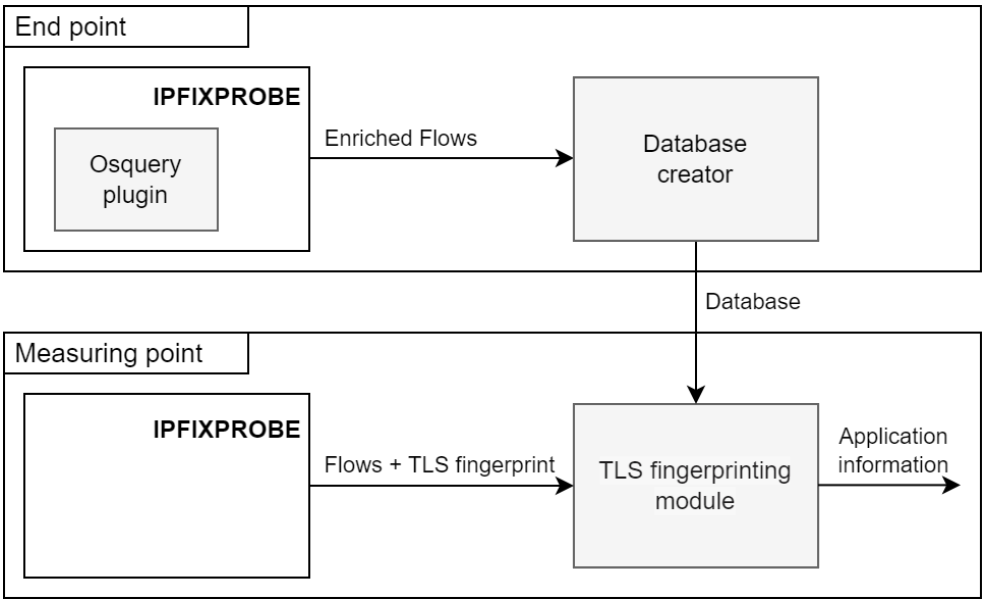

Figure 2.1: Overall architecture

To create a TLS fingerprint database, we need to implement a database creator. Such a database creator will take enriched flows that ipfixprobe will

provide us with. Ipfixprobe will enrich the flows using the already existing TLS plugin and the osquery plugin we created. The TLS fingerprinting module will use the TLS database we created to determine information about application process identification based on the fingerprint provided by the TLS plugin built into ipfixprobe.

By end point, we mean the device on which ipfixprobe with the osquery plugin is installed. The database collected at the end point makes it possible to annotate other devices on the network. Databases can be generated at different end points and subsequently merged into one. In turn, the measuring point is a device with a TLS fingerprinting module installed, for example a router, capable of recognizing information about applications using a database collected at end points.

2.2 Selection of TLS fingerpint algorithm

Since there are several approaches to creating a TLS fingerprint, in this thesis we need to choose one of them. In the Background chapter, we described two main approaches: Cisco Mercury and JA3 fingerprinting. Our choice fell on JA3 fingerprint, which is more prevalent and appeared earlier than Cisco Mercury. Also its code is publicly available and supported by Flowmon exporter, ipfixprobe or Suricata [37]. Another advantage is the existence of software solutions using JA3 fingerprints, for example, a service that provides JA3 fingerprint blacklist [38], which is often updated and can be used to identify already known malware.

2.3 Osquery extension

This section analyzes the creation of an extension for ipfixprobe and the integration of the osquery framework.

2.3.1 Osquery extension architecture

According to the assignment of this bachelor's thesis we needed to implement the osquery plugin for ipfixprobe. This plugin will enrich the flow with system data using osquery framework.

Ipfixprobe is a flow exporter whose task is to create flows from network packets. The osquery plugin will be built inside ipfixprobe and will process the already created flows. To enrich the flow, the plugin will use the osquery framework to get system information. Figure 2.2 schematically shows a high-level view of the plugin design.

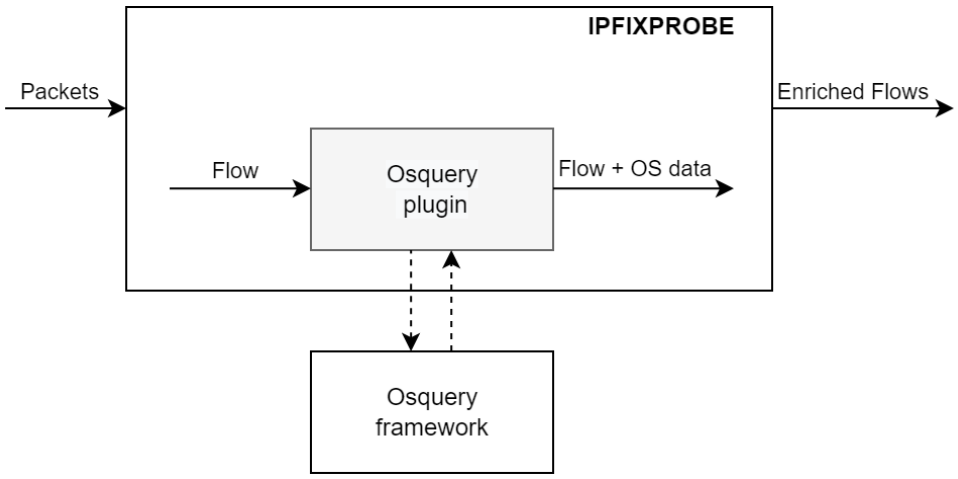

Figure 2.2: Osquery plugin

2.3.2 Tables in osquery and their portability

Osquery currently has 277 tables [39] for various operating systems, most of them are available for macOS and Linux, but there are also tables available for Windows. Since this work is being developed on Linux, we select six tables that are available for the mentioned operating system. They are: os_version, system_info, kernel_info, users, processes and process_open_sockets. A detailed description of the tables is written in appendix D.

2.3.3 Selection of relevant information for flow extension by osquery framework

In this work, we use the flow exporter ipfixprobe, which can create network flows. Moreover, it can enrich the flow by adding JA3 fingerprint information with the already existing TLS plugin.

We also use osquery framework to enrich the flow with information about application process identification. The goal is to select rarely changed system parameters to reduce the chance of collision. First of all, we decided to make a list of constant parameters for an individual device regardless of the flow parameters. The next step is to define parameters that depend on the flow information, for example, source and destination ports. And finally, we collect constant and flow-dependent parameters in one table, which will be added to the standard flow information with the new osquery plugin.

2.3.3.1 Constant parameters

To select these parameters, we have chosen osquery tables, which contain information about the system and do not require administrator privileges.

The `kernel_info` table D.1 provides basic information about the active kernel. We will take only the *kernel version* parameter, since the rest of the parameters may change depending on the system settings.

A large set of information about device identification and hardware specifications are contained in the `system_info` table D.2. We decided not to use hardware parameters, because these parameters are easily changeable (e.g. updating a driver) and, in practice, they are often optional. From this table we will take the *hostname* parameter.

The rest of the parameters will be taken from the `os_version` table D.3. This table contains a single row with operating system parameters. We have selected only those parameters that do not depend on the settings of a specific device, except the *build-specific* parameter, which can be set by the user. As a result, the following parameters will be obtained from the `os_version` table: *distribution or product name*, *major release version*, *minor release version*, *optional build-specific*, *OS platform*, *closely related platforms*, *OS architecture*.

2.3.3.2 Flow-dependent parameters

To obtain information about the identification of the process that handles the connection, we will use the data carried by the flow, such as the source and destination IP addresses and ports.

The `process_open_sockets` table D.4 contains a list of all processes with open network sockets. One entry from this table contains: network and transport protocol versions, socket file descriptor and inode numbers, pid (process id), the source and destination IP addresses and ports. We are only interested in one parameter — pid, which will be used to obtain further information.

The list of all processes currently running on the system is contained in the `processes` table D.5. Since we know the process id, we can get the the *program name* that is serving the connection. We will also take uid (user id) from this table for further user identification.

The `users` table D.6 contains all the local user accounts on the system. Using the previously obtained uid, we can get various information about the user such as username, home directory, group id and others. To enrich the flow, we will take the *username* parameter from this table.

2.3.3.3 Summary

At this step, the system parameters have already been selected. For clarity, the following table 2.1 illustrates the list of parameters, which were chosen for the flow enrichment.

Table 2.1: List of parameters exported by the osquery plugin

Column	Type	Description
PROGRAM_NAME	string	Program name
USERNAME	string	The user name who starts the process
OS_NAME	string	Distribution or product name
OS_MAJOR	integer	Major release version
OS_MINOR	integer	Minor release version
OS_BUILD	string	Optional build-specific
OS_PLATFORM	string	OS Platform or ID
OS_PLATFORM_LIKE	string	Closely related platforms
OS_ARCH	string	OS Architecture
KERNEL_VERSION	string	Kernel version
SYSTEM_HOSTNAME	string	Network hostname including domain

2.3.4 Osquery limitations

There are several limitations with flow recognition using the osquery framework. The first one is the inability to recognize short flows because they live less than the average framework request polling interval so the framework does not have time to recognize them.

The next limitation is the inability to obtain information about application process identification if the program does not open a network socket directly. For example, if the program needs to request a DNS server, it will initiate communication with the system resolver via localhost, so such flows cannot be recognized correctly. It is possible to implement a tool to solve this problem, but in agreement with the supervisor of this work, it was decided to leave this problem, since communication with the localhost still does not use the TLS protocol.

2.3.5 Osquery integration

Osquery provides several ways to get information about the operating system. The first is an interactive console, where the result of running an SQL query is displayed in different formats, for example, JSON or CSV. The second is to schedule queries with a specific interval using the osquery daemon. And the third is the osquery SDK, which allows users to create an extension — a table with additional specific information.

Using the osquery SDK seems like a convenient solution, because it is possible to use the full capabilities of osquery directly from the extension and create custom tables. There is a fairly detailed instruction for developing extensions. But there are also disadvantages: a complicated configuration process, the need to download the all source codes of the osquery core and compile them, which makes it impossible to use it as a dynamically linked

library. Because of the desire to create a lightweight plugin, we abandoned this solution. Also as discussed earlier in section 1.3.1.1, osqueryd was designed to run queries at a given interval, log the results and is not suitable for frequently executing queries with variable values.

The decision was made to use osqueryi. The interactive console is executed in the terminal and accepts an SQL request on standard input, and sends a response to standard output. The idea is to run osqueryi as a child process and be able to use the file descriptors, that osqueryi understands as standard input and output. The big advantage of osqueryi is that user just needs to start it once and all subsequent communication will be performed within one session without the need to restart the application to process each request.

2.3.6 Implementation of extension into ipfixprobe

The flow exporter ipfixprobe implies the ability to enrich the flow with additional parameters, for example, the TLS plugin adds JA3 fingerprint information to the flow.

For the convenience of creating plugins, the ipfixprobe developers have automated this process by creating an interactive script `create_plugin.sh`, that performs the routine work to simplify the creation of new plugins. This script will generate template source and header files, and show a quick guide on how to create a new plugin.

The header file declares a new class inherited from the `FlowCachePlugin` class, witch is responsible for packet processing and flow creation; and a structure inherited from the `RecordExt`, witch handles data buffering. These class and structure will be described in the following subsection.

2.3.6.1 Class FlowCachePlugin

This is the base class for all extensions that provides a list of standard methods for working with the flow. The following are the main ones:

init initializes data processing.

pre_create provides the ability to analyze the first packet in network communication. This method is called before creating a new flow record.

post_create provides information about communication in both directions using the created flow.

pre_export is called before exporting the flow record from the cache. Can be used for final processing or statistics collection.

finish used by the final data processing before the end of the job.

We have divided the data obtained by osquery plugin into two categories: constant and flow-dependent parameters.

We will use the `init` method to read the constant parameters from osquery. Since these parameters remain constant regardless of the flow data, it is sufficient to obtain it once before starting processing.

The osquery plugin uses communication information such as source and destination IP addresses and ports to obtain flow-dependent parameters. This information is available both from the flow and from the first packet in communication. We will not use the `pre_create` method, as we need to add data to a flow that has not yet been created. Instead, we will use the `post_create` method, because the flow already exists and we can immediately add the data obtained from the osqeury. This will reduce the work.

At the end of the plugin's work, we want to show the user the number of successful requests, that is, the number of flow records for which additional information has been obtained using the plugin. We can use the `pre_export` method, which makes it possible to analyze the flow before exporting it, but it is much easier to increment the number of successful requests directly in the `post_create` method. We will use the `finish` method to display this number to the user.

2.3.6.2 Structure RecordExt

Structure `RecordExt` is a base structure of one extension record which is a linked list. It contains information about the type of extension and a pointer to the next extension. This structure has a `addExtension` method, that adds a extension to the end of the list, the virtual methods `fillUnirec`, which fills the unirec record with the stored extension data, and `fillIPFIX`, which is responsible for the correct writing of the extension data to the ipfixprobe buffer.

2.4 Database creator

This section analyzes the method of creating the database of TLS fingerprints and the automated tool for its creation.

2.4.1 Database creator architecture

In this subsection, we provide an overview of the database creator architecture. The database creator takes enriched flows from the ipfixprobe, using the already existing TLS plugin and our previously created osquery plugin. A simplified schematic of the database creator workflow is shown in figure 2.3.

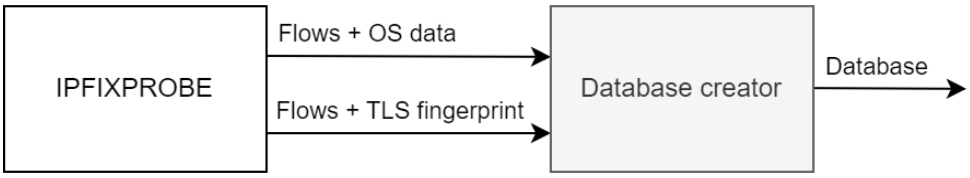

Figure 2.3: Database creator

2.4.2 Database creator design

One of the goals of this work is to create the TLS fingerprint database. We need to decide which programming language will be used for development, in what format the database will be created and define its structure.

To develop the program, we can use any programming language, for example C++, which is also used to develop the ipfixprobe extension. Since the input files, that are used to create the database, are in CSV format, the final solution is to use Python, which can handle CSV files in a more easy way.

The database can be created using an existing SQL database engines such as SQLite [40] or MySQL [41]. We can also use file formats such as JSON, CSV or XML. The simplest solution would be to use the same format as the input files have, but we will use the JSON format since this format is often supported and used by data analysis tools. In addition, the Python language has functionality for working with the JSON.

One record in the database will have a JA3 fingerprint obtained from the TLS plugin as a string and all data received by the osquery plugin, these data is corresponding to table 2.1. Also, to find more relevant records with the same JA3 fingerprint, an additional parameter has been added that counts and holds the number of identical records in the database. The structure of a record of the database is shown in the listing 2.1.

```
{
  "FINGERPRINT":"JA3_fingerprint",
  "PROGRAM_NAME":"program_name",
  "USERNAME":"username",
  "OS_NAME":"os_name",
  "OS_MAJOR":os_major,
  "OS_MINOR":os_minor,
  "OS_BUILD":"os_build",
  "OS_PLATFORM":"os_platform",
  "OS_PLATFORM_LIKE":"os_platform_like",
  "OS_ARCH":"os_arch",
  "KERNEL_VERSION":"kernel_version",
  "SYSTEM_HOSTNAME":"system_hostname"
  "COUNT":number_of_identical_records
}
```

Listing 2.1: Structure of a database record

2.5 TLS fingerprinting module

This section analyzes the creation of a NEMEA module capable of recognizing information about application process identification using a created database of TLS fingerprints.

2.5.1 TLS fingerprinting module architecture

According to the assignment of this bachelor's thesis we need to create an fingerprinting module capable of enriching the flow with system parameters.

The TLS fingerprinting module will use flows enriched with the TLS plugin for the ipfixprobe flow exporter. Such flows will already carry TLS fingerprints and based on which the module will obtain system data from a previously created database. The workflow of the fingerprinting module is shown schematically in figure 2.4.

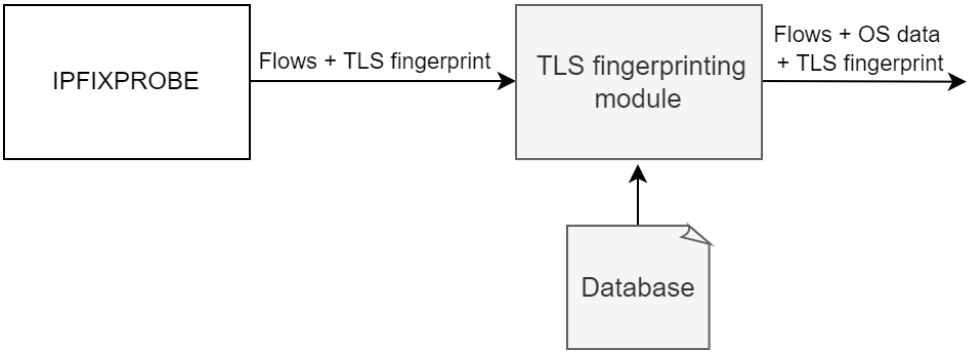

Figure 2.4: TLS fingerprinting module

2.5.2 Design of TLS fingerprinting module

The main purpose of the new module for the NEMEA system is to enrich the flow with system information obtained from the previously created TLS fingerprint database. The module will succeed if the flow has fingerprint information, otherwise, the module should return a format error.

For each flow, the module must find the relevant record based on the JA3 fingerprint and add system information to the flow. If the record does not exist, empty text will be added to the flow to match the output format.

The osquery plugin obtains system parameters, that can be found in the table 2.1. We can add all the parameters to the flow, but for further analysis it is advisable to exclude those that are too detailed. For example, username can be easily changed and be irrelevant for a given flow. It seems that this parameter does not carry important information, but it is necessary at the stage of creating or updating the database in order to respond in time to changes in the system parameters of devices.

Finally, it was decided to add the following parameters to the flow: OS architecture and platform, build-specific parameters and the program name that handles the connection.

Chapter 3

Realisation

This chapter describes in detail the technologies used to develop the ipfixprobe flow exporter plugin, the method of creating the TLS fingerprint database and the process of implementing a new module for the NEMEA system.

3.1 Implementation into ipfixprobe

The process of creating a plugin for the flow exporter can be roughly divided into two stages. The first is creating a new plugin and integrating it correctly. The second is the integration of osquery into a previously created plugin.

3.1.1 Plugin design

To create the plugin, we used an interactive script that will require user to enter the name and developer information. We named the plugin `osquery`. After the script finishes, template files `osqueryplugin.h` and `osqueryplugin.cpp`, and a short guide are generated. Let's take a closer look at all the points that are indicated in the guide and also explain them:

1. Add plugin header and source files to Makefile. The new plugin can work only if the osquery framework is installed on the device. The Makefile has been improved to include a new argument `WITH_OSQUERY`. Thus, when using this argument, a check will be made to see if the osquery framework is installed, otherwise the installation process will be aborted;

2. Add an osquery entry to the list of available extension types. This step is required to inform ipfixprobe about creating a new extension;

3. Include `osqueryplugin.h` in `main.cpp`. Also add osquery to the list of supported plugins for -p parameter and add plugin support to the function responsible for parsing parameters;

4. Add unirec fields to the `UR_FIELDS` macro in `osqueryplugin.cpp` to define the data type for each field. This is necessary for the correct calculation of the expected amount of data;

5. Define the new `IPFIX` fields. After this step, for each parameter obtained using osquery, a field will be created with a unique id, a specific length and, optionally, a pointer to copy the data;

6. Implement the `fillIPFIX` function in `osqueryplugin.h` to fill fields to IPFIX message. This function strictly defines the way how each field should be written to the buffer;

7. Implement the necessary methods provides by `FlowCachePlugin` class. Based on the analysis from the previous chapter 2.3.6.1, we used the following methods: `init`, `post_create` and `finish`, unused ones can be removed.

After completing these steps, we can assume that the plugin works correctly. But it still does not know how to get information about the operating system.

3.1.2 Integration of osquery into the plugin

An additional structure `OsqueryRequestManager` has been developed to interact with osquery framework. This structure is responsible for running osquery, reading and writing data to it, handling errors, and creating a record (the structure inherited from `RecordExt` and responsible for storing and recording data) for this plugin.

The `OsqueryRequestManager` has a private pointer to a record that will be added to the flow. This solution makes it possible to read constant parameters from osquery and immediately write them to this record without the need to create additional variables. Further flow-dependent parameters will be added to record. Additionally, there are three public methods in this structure: `readInfoAboutOS` gets constant information using osquery; `readInfoAboutProgram` gets flow-dependent parameters; `getRecord` returns the record that will be added to the flow.

3.1.2.1 Running osquery

As decided earlier in section 2.3.5, we will be using the interactive console mode — osqueryi, to get system information.

We will run osquery as a child process and replace standard input and output with our file descriptors for future reference. We also want to communicate with osquery in JSON format without seeing errors, so we redirect standard error to null device. The final command to run osquery is as follows: `osqueryi --json 2>/dev/null`.

We wrote an `popen2` function to run osqueryi and to create duplicate file descriptors. After starting this function, an attempt will be made to create unidirectional data channels for inter-process communication using the `pipe` function [42]. After successfully creating data channels, a new child process will be created using the `fork` function [43]. The child process will close unused file descriptors and create copies of the remaining one. These copies will be returned for future use. Listing 3.1 shows the startup process in detail.

```
#define READ_FD     0
#define WRITE_FD    1

pid_t popen2(const char *command, int *inFD, int *outFD)
{
    int p_stdin[2], p_stdout[2];
    pid_t pid;

    if (pipe(p_stdin) != 0 || pipe(p_stdout) != 0)
        return -1;

    pid = fork();

    if (pid < 0) return pid;

    if (pid == 0) {
        close(p_stdin[WRITE_FD]);
        dup2(p_stdin[READ_FD], READ_FD);
        close(p_stdout[READ_FD]);
        dup2(p_stdout[WRITE_FD], WRITE_FD);
        execl("/bin/sh", "sh", "-c", command, NULL);
        perror("execl");
        exit(1);
    }

    inFD == NULL ? close(p_stdin[WRITE_FD]) :
                   *inFD = p_stdin[WRITE_FD];
    outFD == NULL ? close(p_stdout[READ_FD]) :
                    *outFD = p_stdout[READ_FD];

    return pid;
}
```

Listing 3.1: Running osquery as a child process and creating duplicates of file descriptors

3.1.2.2 Reading from osquery

To get data from osquery, we created the `readFromOsquery` function that uses the output file descriptor copied by `popen2` function. A simplified version of the code for this function can be found in listing 3.2.

To detect events occurring on a specific file descriptor, we use the `poll` function [44]. Before using this function, we need to initialize the `pollfd` structure that means setting the file descriptor and tracked events. Before each use, we will reset the list of events that have happened.

We can now call the `poll` function to detect events. The function will return a value used as a flag for errors, such as a timeout or poll error. If there were no errors and the `POLLIN` event occurred (there is data to read), we will try to read data from the file descriptor using the `read` function [45]. The function will return the number of bytes of information read. Since osqueryi is running in json mode, we will always get a response, at least an empty JSON string. Therefore, we will check if the length of the read string is more than four bytes (the size of the minimum osqueryi response in json mode), then we will write it to the buffer, otherwise we will set the read error flag and clear the buffer.

After successfully reading the data, additional functions that can parse information from the buffer will set the corresponding values received from osquery.

```
#define READ_SIZE         1024
#define POLL_TIMEOUT      200

pollfd *pfd = new pollfd;
pfd->fd     = outFD;
pfd->events = POLLIN;

size_t readFromOsquery()
{
    ssize_t bytesRead = 0;
    pfd->revents = 0;

    int ret = poll(pfd, 1, POLL_TIMEOUT);
    if (ret == -1 || ret == 0) return 0;

    if (pfd->revents & POLLIN)
        bytesRead = read(outFD, buffer, READ_SIZE);

    return bytesRead < 5 ? 0 : bytesRead;
}
```

Listing 3.2: Reading from osquery

3.1.2.3 Writing to osquery

Writing to osquery is much easier than reading from it. We use the `write` function [46] to write data to the file descriptor copied by the `popen2` function. Listing 3.3 shows an example of the code for the function responsible for writing.

The function first of all determines the length of data to send and sends them using the `write` function. It checks the number of sent bytes and, based on this value, decide whether the data has been sent successfully.

```
bool writeToOsquery(const char *query)
{
    ssize_t length = strlen(query);
    ssize_t n      = write(inFD, query, length);

    return (n != -1 && n == length);
}
```

Listing 3.3: Writing to osquery

3.1.2.4 Error handling

Errors may occur while the plugin is running, for example osquery has stopped responding or cannot be started (because it is already running).

Errors must be monitored and handled correctly. For these purposes, an additional structure `OsqueryStateHandler` has been implemented, which is responsible for storing information about existing errors.

3.2 Database creation

To create the database, a program called `TLSDatabaseCreator` has been developed. The main functionality of this program is to merge two CSV files into one file in JSON format.

The program expects two files in CSV format at the input, which are the results of the TLS and osquery plugins for the ipfixprobe flow exporter. The format of the input files will be checked, and if any of these files does not match the format, the program will be interrupted. Format checking is necessary to ensure that the input data has not been corrupted and has common parameters for comparison with each other.

For correct JA3 fingerprint and system information association, the program will generate a list of common parameters. This list will be used to compare records from the both input files. If two lists of parameters are equal, then the pair of JA3 fingerprint and system information from these records will be written to the database. The fingerprint will be taken from the results of the TLS plugin, and the system data, in turn, obtained using the

osquery plugin, will be taken from the second file. The program also counts how many times the same data has occurred and writes the number to the corresponding record in the database. The structure of its one record is shown in the listing 2.1

The program can work in several modes, in addition to merging two files, the program can also merge two databases or merge files and a database at the same time. The help page on how to use this program can be found in appendix E.1. The choice of the program operation mode is determined by the control parameters. The following is a detailed description of them:

--merge FILE Specifies the path to the database file that will be used for merging with CSV files. If no output file is specified, FILE will be used.

--output OUTPUT Specifies the path to the output file, the file will be overwritten. If the output file does not exist, a new file will be created.

--csv FILE1 FILE2 CSV file merge mode. The files should be the output of the osquery and TLS plugins. At least one option `--merge` or `--output` must be defined.

--database FILE1 FILE2 Database file merge mode. The files must be valid JSON database files. If no output file is specified, FILE1 will be used.

--help Show help page.

3.3 Implementation of pytrap module

The TRAP extension [47] for python3, called pytrap, can be used to implement a new NEMEA module. This module consists of two main classes: `TrapCtx` and `UnirecTemplate` [48]. `TrapCtx` provides an interface for communication, `UnirecTemplate` can be used to access and manipulate data. An example of using pytrap module is shown in the listing 3.4.

The first step of implementation is to initialize `TrapCtx` class. We use `init` method to define the input and output interfaces [49] and the amount of them. Next, we define a list of parameters that record must contain, otherwise an exception will be thrown. After initializing all the necessary parameters, it is possible to read from the input interface, process and send data to the output interface.

```
import pytrap

c = pytrap.TrapCtx()
c.init(["-i", "u:socket1,u:socket2"], 1, 1)

c.setRequiredFmt(0, pytrap.FMT_UNIREC, "ipaddr SRC_IP")

rec = pytrap.UnirecTemplate("ipaddr SRC_IP")

try:
    data = c.recv()
if len(data) <= 1:
    pass
else:
    rec.setData(data)

c.send(data)
c.finalize()
```

Listing 3.4: A simple example of using the pytrap module

3.3.1 TLS fingerprinting module

One of the main goals of this work is to create a new pytrap module that will use the previously created database. The database contains a JA3 fingerprint, system information obtained from osquery and a frequency of occurrence of this pair.

The new module will be getting the network flow enriched with JA3 fingerprint on the input interface. A flow will be sent to the output interface extended with system parameters from osquery.

Since the new module needs to know the JA3 fingerprint to work correctly, we must add the `bytes TLS_JA3` parameter to the list of required ones. Thus, an exception will be thrown when trying to process data that does not contain a fingerprint.

As mentioned earlier in section 2.5, the fingerprinting module will only take some of the data that the osquery plugin is capable of providing. We enrich the flow with four new parameters: OS architecture, build-specific parameters, OS platform and program name.

One of the ways to generate incoming data for the new module can be the use of the TLS plugin for ipfixprobe. The results of the TLS plugin will be stored into a pcap file, which will be processed by the module and enriched with system data.

To use the module, it is necessary to specify the format of the input and output interfaces (if no output interface is specified, a blackhole interface will be used) and the path to the file with the database. The help page on how to use TLS fingerprinting module can be found in appendix E.2.

Chapter 4

Testing

This chapter describes the results of testing the osquery plugin and module for the NEMEA system. The program has also been tested to create a database of TLS fingerprints. All tests in this chapter were performed on a Linux Mint distribution using a computer with a 2.2GHz Intel Core i7-8750H processor and 16GB of RAM.

4.1 Testing the osquery plugin

The main purpose of the osquery plugin is to enrich the flow with system data. A flow is considered successfully processed if flow-dependent parameters were found for it using osquery.

Determining the performance of a plugin is not a trivial task. Since the plugin works on the fly, namely, it checks the list of open network sockets at a specific time, it is impossible to create a test file, because it may contain outdated communication information that is no longer available for retrieval.

We ran a series of tests, for each of which we considered: the number of processed packets, flows accepted for processing, the number of successfully enriched flows and elapsed time since the beginning of the test in minutes. A standard set of user programs that use the network, such as Chrome, Spotify, or Slack, was executed on the testing virtual machine. The first test works independently and without user interaction with the running programs, the second on the contrary tests the work of the plugin when actively using the same programs. By active use we mean, for example, sending messages in messengers, surfing the Internet and so on.

The first test showed an average value of successfully enriched flows of 52.1 %, the test results are shown in table 4.1. The results of the second test were better, the average success rate of flow enrichment is about 75.4 %, the test results are shown in table 4.2. Although the results are not perfectly accurate, we consider them satisfactory. The explanation for this is the osquery limitations described in 2.3.4 section.

Table 4.1: Osquery plugin test results without active network use

Test №	Packets	Flows	Enriched	Time	Success rate
1	10 749	2467	1381	60	56.0 %
2	13 795	2650	1401	60	52.9 %
3	19 684	3434	1714	120	49.9 %
4	21 842	3156	1569	120	49.7 %
5	27 561	4207	2180	180	51.8 %

Table 4.2: Osquery plugin test results with active network use

Test №	Packets	Flows	Enriched	Time	Success rate
1	24 185	373	292	5	78.3 %
2	54 128	962	644	10	66.9 %
3	56 968	1087	734	10	67.5 %
4	430 956	2834	2495	20	88.0 %
5	480 004	1217	926	20	76.1 %

4.2 Database creation

To create a database, we use the results of osquery and TLS plugins work previously converted to CSV format. The database creation program merges input files into a common database, in addition the program can merge several databases into one. The format of a single record in the database can be seen in listing 2.1. The created database can be found on the attached DVD, file path `data/dataset.json`.

Data for further creation of the database was obtained from the six devices. Based on the test results in the previous section, it can be seen that the osquery plugin produces a fairly large amount of enriched flows to create an informative database. But in practice it turned out that most of them have the same pair of JA3 fingerprint and system data, therefore they are not duplicated in the database. During testing, it was possible to obtain about 30 unique records in the database from one device, the number of records is directly related to the number of running programs using the network.

This database creation program has been fully tested with all of its supported options. See appendix E.1 for a list of options.

4.3 Testing the TLS fingerprinting module

The main purpose of the fingerprinting module is to enrich the flow with system information based on the JA3 fingerprint using a previously created database. To obtain test data, we used the TLS plugin for the ipfixprobe flow exporter.

We performed a series of tests to check the efficiency of the new module. During testing, we paid attention to certain parameters: the number of flows enriched with JA3 fingerprint, the number of unprocessed flows, the number of successfully processed flows and elapsed time since the beginning of the test in minutes. By unprocessed flows, we mean those flows for which no corresponding record was found in the database. In turn, by the concept of successfully processed flow, we mean a flow for which there is a record in the database and the set of all parameters exported by the fingerprinting module (OS architecture and platform, build-specific parameters and program name) are true for this device. The tests were run on the same device that was previously analyzed using the ipfixprobe flow exporter.

The first tests were run using a database containing records only for the mentioned device. The average module success rate is 99.6 %. The test results are shown in table 4.3.

The next set of tests was run using the database containing records for the six devices. The average success rate of recognized fingerprints is about 87 %, the test results are shown in table 4.4. We consider these results to be quite high but remember that the tests were performed in an ideal environment and when tested in real networks the results could potentially be lower.

During testing, we found that the parameters exported by the fingerprinting module have been recognized with different accuracy. For example, program name was recognized for all flows except the unprocessed ones, the final accuracy is 99.5 %. There is also a problem of correct identification of the OS platform for the same program running on different operating platforms. For example, in our experiments, the Yandex browser was launched on the Linux Mint and Ubuntu. For each platform, the JA3 fingerprint was the same, but the OS platform parameter was different, so there are two records in the database. Ultimately, the module will select the record that was most often encountered when filling the database. In other words, if the flow related to the Yandex browser has been processed more times on Ubuntu than on Linux Mint, the module will consider the record from the database with the Ubuntu operating platform parameter to be correct.

The module has been checked for compliance with the declared capabilities, which can be viewed in appendix E.2 or in the `README.md` file in the `src/TLSFingerprintingModule` directory on the attached DVD.

Table 4.3: TLS fingerprinting module test results for one device

Test №	**JA3 flows**	**Unprocessed**	**Succeed**	**Time**	**Accuracy**
1	76	0	76	5	100 %
2	96	0	96	5	100 %
3	127	1	126	5	99.2 %
4	161	0	161	10	100 %
5	387	5	382	20	98.7 %

Table 4.4: TLS fingerprinting module test results for the six devices

Test №	**JA3 flows**	**Unprocessed**	**Succeed**	**Time**	**Accuracy**
1	49	0	44	5	89.8 %
2	62	0	57	5	91.9 %
3	87	0	73	5	83.9 %
4	101	0	93	5	92.1 %
5	229	0	192	10	83.7 %
6	240	1	225	10	93.6 %
7	243	0	190	10	78.2 %
8	344	1	301	20	87.5 %
9	423	8	339	20	80.1 %
10	716	17	641	30	89.5 %

Conclusion

One of the techniques that can increase visibility into encrypted traffic is TLS fingerprinting. To identify applications, TLS fingerprinting tools use a database, where TLS metadata is assigned to a specific application. The motivation for our work is the lack of high-quality fingerprint databases, the use of which does not enable the identification of the application with sufficient accuracy.

To solve this problem, we have developed a tool for enriching flows with information about application process identification. The enriched flow is then used to fill our fingerprint database. In fact, this work can be divided into three stages: data obtaining, data collection and flow enrichment with that data in real time.

At the first stage, a new plugin for the flow exporter ipfixprobe was designed and implemented. This plugin is used to enrich the flow with system data obtained using the osquery framework.

As a part of the second stage, we have implemented a program for creating a TLS fingerprint database. This database contains system data and TLS fingerprints associated with them. We get system data using our previously created osquery plugin and fingerprints using the existing TLS plugin for ipfixprobe.

Finally, we created a module that takes a TLS fingerprint enriched flow and extends it with system data. This module uses a ready-to-use fingerprint database and enriches the flow in real time.

It was verified that the module can correctly enrich the flow with the full set of system parameters we have chosen with an average success rate of 87 %. If we take into account the correctness of enrichment only by the application name parameter, the accuracy is 99.5 %. These results are favourable and prove that the module works as expected, but on real, larger networks this accuracy will decrease.

This bachelor thesis presents a functional module, which is going to be a valuable part of the NEMEA framework ecosystem, and also a tool for

creating a TLS fingerprint database, which can be used as a reference solution for future research.

Future work

As part of future work, the module could be tested on larger and more congested networks, which would contribute to obtaining more accurate results of success. To improve the work of the plugin, there is also a space for investigation of the problem of communication between applications and system resolvers on localhost, since osquery framework cannot determine which program opened a network socket to communicate. It is also possible to experiment with the structure of the database to find relevant records more efficiently.

Bibliography

[1] Geere, D. How deep packet inspection works. Accessed: 2021-11-18, [online]. Available from: https://www.wired.co.uk/article/how-deep-packet-inspection-works

[2] Umer, M.; Ramzan, M. S.; et al. Flow-based intrusion detection: Techniques and challenges. *Computers & Security*, volume 70, 06 2017, doi: 10.1016/j.cose.2017.05.009.

[3] Alexander, L. Deep Packet Inspection: is this the Future of Analysers? 2021, accessed: 2021-11-18, [online]. Available from: https://pandorafms.com/blog/deep-packet-inspection/

[4] Cejka, T.; Bartos, V.; et al. NEMEA: A framework for network traffic analysis. In *2016 12th International Conference on Network and Service Management (CNSM)*, 2016, pp. 195–201, doi:10.1109/CNSM.2016.7818417.

[5] Aitken, P.; Claise, B.; et al. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC 7011, Sept. 2013, doi:10.17487/RFC7011. Available from: https://rfc-editor.org/rfc/rfc7011.txt

[6] Trammell, B.; Boschi, E. Bidirectional Flow Export Using IP Flow Information Export (IPFIX). RFC 5103, Jan. 2008, doi:10.17487/RFC5103. Available from: https://rfc-editor.org/rfc/rfc5103.txt

[7] Cisco Systems, Inc. NetFlow v5. Accessed: 2021-11-18, [online]. Available from: https://www.cisco.com/c/en/us/td/docs/net_mgmt/netflow_collection_engine/3-6/user/guide/format.html

[8] Caligare s.r.o. Netflow — Version 7. 2006, accessed: 2021-11-18, [online]. Available from: https://netflow.caligare.com/netflow_v7.htm

[9] Claise, B. Cisco Systems NetFlow Services Export Version 9. RFC 3954, Oct. 2004, doi:10.17487/RFC3954. Available from: https://rfc-editor.org/rfc/rfc3954.txt

[10] Progress Software Corporation. Kemp Flowmon Probe. Accessed: 2021-11-18, [online]. Available from: https://www.flowmon.com/en/products/appliances/probe

[11] CESNET, a.l.e. ipfixprobe - IPFIX flow exporter. Accessed: 2021-11-18, [online]. Available from: https://github.com/CESNET/ipfixprobe

[12] Computer emergency response team. YAF Documentation. Accessed: 2021-11-18, [online]. Available from: https://tools.netsa.cert.org/yaf/docs.html

[13] Cisco Systems, Inc. Cisco joy. Accessed: 2021-11-18, [online]. Available from: https://github.com/cisco/joy

[14] CESNET, a.l.e. IPFIXcol2: High-performance NetFlow v5/v9 and IPFIX collector. Accessed: 2021-11-18, [online]. Available from: https://github.com/CESNET/ipfixcol2

[15] Progress Software Corporation. Kemp Flowmon Netflow Collector. Accessed: 2021-11-18, [online]. Available from: https://www.flowmon.com/en/products/appliances/netflow-collector

[16] Progress Software Corporation. Kemp Flowmon ADS. Accessed: 2021-11-18, [online]. Available from: https://www.flowmon.com/en/products/software-modules/anomaly-detection-system

[17] Hofstede, R.; Čeleda, P.; et al. Flow Monitoring Explained: From Packet Capture to Data Analysis 7 With NetFlow and IPFIX. *IEEE Communications Surveys Tutorials*, volume 16, no. 4, 2014: pp. 2037–2064, doi:10.1109/COMST.2014.2321898.

[18] Polk, T.; Turner, S. Prohibiting Secure Sockets Layer (SSL) Version 2.0. RFC 6176, Mar. 2011, doi:10.17487/RFC6176. Available from: https://rfc-editor.org/rfc/rfc6176.txt

[19] Barnes, R.; Thomson, M.; et al. Deprecating Secure Sockets Layer Version 3.0. RFC 7568, June 2015, doi:10.17487/RFC7568. Available from: https://rfc-editor.org/rfc/rfc7568.txt

[20] Allen, C.; Dierks, T. The TLS Protocol Version 1.0. RFC 2246, Jan. 1999, doi:10.17487/RFC2246. Available from: https://rfc-editor.org/rfc/rfc2246.txt

[21] Dierks, T.; Rescorla, E. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346, Apr. 2006, doi:10.17487/RFC4346. Available from: https://rfc-editor.org/rfc/rfc4346.txt

[22] Moriarty, K.; Farrell, S. Deprecating TLS 1.0 and TLS 1.1. RFC 8996, Mar. 2021, doi:10.17487/RFC8996. Available from: https://rfc-editor.org/rfc/rfc8996.txt

[23] Rescorla, E.; Dierks, T. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, Aug. 2008, doi:10.17487/RFC5246. Available from: https://rfc-editor.org/rfc/rfc5246.txt

[24] Rescorla, E. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, Aug. 2018, doi:10.17487/RFC8446. Available from: https://rfc-editor.org/rfc/rfc8446.txt

[25] Husák, M.; Cermák, M.; et al. Network-Based HTTPS Client Identification Using SSL/TLS Fingerprinting. In *2015 10th International Conference on Availability, Reliability and Security*, 2015, pp. 389–396, doi:10.1109/ARES.2015.35.

[26] T, M. Network Forensic Investigation of HTTPS Protocol. *International Journal of Engineering Research*, 08 2013.

[27] Brotherston, L. TLS fingerprinting. 2015, accessed: 2021-11-18, [online]. Available from: https://blog.squarelemon.com/tls-fingerprinting/

[28] Cisco Systems, Inc. Mercury: network metadata capture and analysis. Accessed: 2021-11-18, [online]. Available from: https://githubplus.com/cisco/mercury

[29] Althouse, J. Open Sourcing JA3 — SSL/TLS Client Fingerprinting for Malware Detection. [online] https://engineering.salesforce.com/open-sourcing-ja3-92c9e53c3c41, 2017.

[30] Althouse, J.; Atkinson, J.; et al. JA3 is a standard for creating SSL client fingerprints in an easy to produce and shareable way. Accessed: 2021-11-18, [online]. Available from: https://github.com/salesforce/ja3

[31] Hynek, K. IP Flow extension for increased visibility in encrypted network traffic. 2021, Doctoral Study Report, Faculty of Information Technology, Czech Technical University in Prague.

[32] Meta Platforms Inc. osquery — Easily ask questions about your Linux, Windows, and macOS infrastructure. Accessed: 2021-11-18, [online]. Available from: https://osquery.io/

[33] Meta Platforms Inc. Introducing osquery - Engineering at Meta. 2014, accessed: 2021-11-18, [online]. Available from: https://engineering.fb.com/2014/10/29/security/introducing-osquery/

[34] Meta Platforms Inc. osqueryd (daemon) — osquery. Accessed: 2021-11-18, [online]. Available from: https://osquery.readthedocs.io/en/latest/introduction/using-osqueryd/

[35] Meta Platforms Inc. osqueryi (shell) — osquery. Accessed: 2021-11-18, [online]. Available from: https://osquery.readthedocs.io/en/latest/introduction/using-osqueryi/

[36] Meta Platforms Inc. SDK and Extensions — osquery. Accessed: 2021-11-18, [online]. Available from: https://osquery.readthedocs.io/en/latest/development/osquery-sdk/

[37] Open Information Security Foundation. Suricata Home Page. Accessed: 2021-11-18, [online]. Available from: https://suricata.io/

[38] Abuse project. SSLBL — Blacklist. Accessed: 2021-11-18, [online]. Available from: https://sslbl.abuse.ch/blacklist/#ja3-fingerprints-csv

[39] Meta Platforms Inc. osquery — Schema. Accessed: 2021-11-18, [online]. Available from: https://osquery.io/schema/5.0.1/

[40] SQLite team. About SQLite. Accessed: 2021-11-18, [online]. Available from: https://sqlite.org/about.html

[41] Oracle Corporation. About MySQL. Accessed: 2021-11-18, [online]. Available from: https://www.mysql.com/about/

[42] Linux commands. Pipe linux command man page. 2014, accessed: 2021-11-18, [online]. Available from: https://www.commandlinux.com/man-page/man7/pipe.7.html

[43] Linux commands. Fork linux command man page. 2014, accessed: 2021-11-18, [online]. Available from: https://www.commandlinux.com/man-page/man2/fork.2.html

[44] Linux commands. Poll linux command man page. 2014, accessed: 2021-11-18, [online]. Available from: https://www.commandlinux.com/man-page/man2/poll.2.html

[45] Linux commands. Read linux command man page. 2014, accessed: 2021-11-18, [online]. Available from: https://www.commandlinux.com/man-page/man2/read.2.html

[46] Linux commands. Write linux command man page. 2014, accessed: 2021-11-18, [online]. Available from: https://www.commandlinux.com/man-page/man2/write.2.html

[47] Cejka, T.; CESNET, a.l.e. Welcome to pytrap's documentation. Accessed: 2021-11-18, [online]. Available from: https://nemea.liberouter.org/doc/pytrap/index.html

[48] CESNET, a.l.e. Nemea-Framework - pytrap. Accessed: 2021-11-18, [online]. Available from: https://github.com/CESNET/Nemea-Framework/tree/master/pytrap

[49] CESNET, a.l.e. TRAP Interface Specifier. Accessed: 2021-11-18, [online]. Available from: https://nemea.liberouter.org/trap-ifcspec/

Appendix A

Acronyms

ALPN Application Layer Protocol Negotiation

API Application Programming Interface

CSV Comma-separated Values

DNS Domain Name System

DPI Deep Packet Inspection

HTTP Hypertext Transfer Protocol

IETF Internet Engineering Task Force

IPFIX Internet Protocol Flow Information Export

JSON JavaScript Object Notation

OS Operating System

POP Post Office Protocol

SDK Software Development Kit

SNI Server Name Indication

SQL Structured Query Language

SSL Secure Sockets Layer

TLS Transport Layer Security

XML Extensible Markup Language

Appendix B

Contents of enclosed DVD

- `readme.txt` ..... the file with DVD contents description
- `src` ..... the directory of source codes
  - `ipfixprobe` ..... ipfixprobe source codes
  - `nemea` ..... NEMEA system source codes
  - `TLSDatabaseCreator` ..... database creator source codes
  - `TLSFingerprintingModule` ..... fingerprinting module source codes
  - `thesis` ..... the directory of LaTeX source codes of the thesis
  - `osquery_5.0.1-1.linux_amd64.deb` ..... osquery framework installer
- `doc` ..... the directory with documentation
  - `ipfixprobe` ..... ipfixprobe documentation
  - `TLSDatabaseCreator` ..... database creator documentation
  - `TLSFingerprintingModule` ..... fingerprinting module documentation
- `examples` ..... the directory with examples of input and output files
  - `TLSDatabaseCreator` ..... database creator file examples
  - `TLSFingerprintingModule` ..... fingerprinting module file examples
- `data` ..... the directory with TLS database
  - `dataset.json` ..... testing TLS database
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format

Appendix C

Installation manual

The recommended system to install the ipfixprobe flow exporter and the NEMEA system — Debian/Ubuntu.

C.1 Dependencies

The following libraries are required to install NEMEA system:

- autoconf
- automake
- gcc
- gcc-c++
- libtool
- libxml2-devel
- make
- pkg-config
- libpcap-dev
- libidn11-dev
- bison
- flex

This command can be used to install the required libraries:

```
sudo apt-get install -y gawk bc autoconf automake gcc \
g++ libtool libxml2-dev make pkg-config libpcap-dev \
libidn11-dev libssl-dev bison flex
```

Also, additional installation of python3, pip and pandas is required, for the TLS fingerprint database creator to work correctly. This can be done with the following command:

```
sudo apt-get install python3 python3-pip
pip install pandas
```

C.2 Installation

All source codes can be found on the attached DVD. Before starting the installation, copy the `src/` directory from the DVD to your home directory.

1. **Install the osquery framework**
   There are several ways to install the osquery framework. The easiest is to visit the official website [32] and download the installer for your system. Also for installation, you can use the installer located in the attached DVD. Be careful, this installer is compatible with `Debian x86_64`, if you are using another system, please download the appropriate installer from the official website. To install, you can use the following command (change the filename if you are using a different installer):

   ```
   cd ~/src/
   sudo dpkg -i osquery_5.0.1-1.linux_amd64.deb
   ```

2. **Install the NEMEA system**
   Use the following instructions to install the NEMEA system:

   ```
   cd ~/src/nemea/
   ./bootstrap.sh
   ./configure --enable-repobuild --prefix=/usr \
   --bindir=/usr/bin/nemea --sysconfdir=/etc/nemea \
   --libdir=/usr/lib/x86_64-linux-gnu
   make
   sudo make install
   ```

   Please note that the `--libdir` parameter value may be different, replace it with the appropriate library path for your system.

3. **Install the ipfixprobe**
   After successfully installing the NEMEA system, install the ipfixprobe flow exporter. At this step, it is very important to have the osquery framework installed. Otherwise, it is not possible to install ipfixprobe with osquery plugin support. Use the following instructions to install the ipfixprobe:

```
cd ~/src/ipfixprobe/
autoreconf -i
./configure --with-nemea --with-osquery \
--libdir=/usr/lib/x86_64-linux-gnu
make
sudo make install
```

Please note that the `--libdir` parameter value may be different, replace it with the appropriate library path for your system.

After successful installation of ipfixprobe, the path to the executable file is `/src/ipfixprobe/ipfixprobe`. For more convenient use, you can create a link to the executable file, for example, with the following command:

```
ln -s ~/src/ipfixprobe/ipfixprobe ~/ipfixprobe
```

4. **Install the pytrap module**
   Since the module interacts with the NEMEA system, it must already be installed on your device. You can install the pytrap module engine with the following instructions:

```
cd ~/src/nemea/nemea-framework/pytrap/
sudo python3 setup.py install
```

C.3 Usages

To obtain data for the database, we use the ipfixprobe flow exporter, namely the existing TLS and the new osquery plugins. The following is an example of getting data using ipfixprobe:

- The first step is to run ipfixprobe. There are different ways of configuring, for example, the `-p` parameter allows you to select which plugin will be used; the `-i` parameter defines the format of the input and output interfaces [49]; the `-I` parameter says that the data for analysis will be taken from the network interface. More information about the parameters can be obtained using the `ipfixprobe -h` command. The following is an example of using ipfixprobe:

```
sudo ~/ipfixprobe -p osquery,tls \
-i f:osquery_out,f:tls_out -I wlo1
```

- To export the processed data to CSV format, a `logger` is used, which is part of the NEMEA system. The following is an example of its use:

```
/usr/bin/nemea/loger -t -i f:osquery_out > osquery.csv
/usr/bin/nemea/loger -t -i f:tls_out > tls.csv
```

For a detailed acquaintance with the capabilities of the pytrap module and database creator read the `README.md` in the `src/TLSFingerprintingModule` and `src/TLSDatabaseCreator` directories, respectively.

Appendix D

Osquery tables

Table D.1: Table kernel_info

Column	Type	Description
version	text	Kernel version
arguments	text	Kernel arguments
path	text	Kernel path
device	text	Kernel device identifier

Table D.2: Table system_info

Column	Type	Description
hostname	text	Network hostname including domain
uuid	text	Unique ID provided by the system
cpu_type	text	CPU type
cpu_subtype	text	CPU subtype
cpu_brand	text	CPU brand string
cpu_physical_cores	integer	Number of physical CPU cores
cpu_logical_cores	integer	Number of logical CPU cores
cpu_microcode	text	Microcode version
physical_memory	bigint	Total physical memory in bytes
hardware_vendor	text	Hardware vendor
hardware_model	text	Hardware model
hardware_version	text	Hardware version
hardware_serial	text	Device serial number
board_vendor	text	Board vendor
board_model	text	Board model
board_version	text	Board version
board_serial	text	Board serial number
computer_name	text	Friendly computer name (optional)
local_hostname	text	Local hostname (optional)

Table D.3: Table os_version

Column	Type	Description
name	text	Distribution or product name
version	text	OS version
major	integer	Major release version
minor	integer	Minor release version
patch	integer	Optional patch release
build	text	Optional build-specific
platform	text	OS Platform or ID
platform_like	text	Closely related platforms
codename	text	OS version codename
arch	text	OS Architecture
install_date	bigint	The install date of the OS.
pid_with_namespace	integer	Pids that contain a namespace
mount_namespace_id	text	Mount namespace id

Table D.4: Selection of relevant columns from a process_open_sockets table

Column	Type	Description
pid	integer	Process (or thread) ID
fd	bigint	Socket file descriptor number
socket	bigint	Socket handle or inode number
family	integer	Network protocol (IPv4, IPv6)
protocol	integer	Transport protocol (TCP/UDP)
local_address	text	Socket local address
remote_address	text	Socket remote address
local_port	integer	Socket local port
remote_port	integer	Socket remote port

Table D.5: Selection of relevant columns from a processes table

Column	Type	Description
pid	bigint	Process (or thread) ID
name	text	The process path or shorthand argv[0]
path	text	Path to executed binary
cmdline	text	Complete argv
state	text	Process state
cwd	text	Process current working directory
root	text	Process virtual root directory
uid	bigint	Unsigned user ID
gid	bigint	Unsigned group ID

Table D.6: Selection of relevant columns from a users table

Column	Type	Description
uid	bigint	User ID
gid	bigint	Group ID (unsigned)
username	text	Username
description	text	Optional user description
directory	text	User's home directory
shell	text	User's configured default shell

Appendix E

Listings

```
Description:
  Merge two CSV files, merge CSV files with an existing database
  file, or merge two database files and save them in JSON format.

Usage:  TLSDatabaseCreator.py -c FILE1 FILE2 [-m FILE] [-o OUTPUT]
  or:   TLSDatabaseCreator.py -d FILE1 FILE2 [-o OUTPUT]

Options:
  -c  --csv FILE1 FILE2       CSV file merge mode.
                              The files should be the output of the
                              osquery and TLS plugins. At least one
                              option(-m or -o) must be defined.

  -d  --database FILE1 FILE2  Database file merge mode.
                              The files must be valid JSON database
                              files. If no output file is specified,
                              FILE1 will be used.

  -m  --merge FILE            Specifies the path to the database file
                              that will be used for merging with CSV
                              files. If no output file is specified,
                              FILE will be used.

  -o  --output OUTPUT         Specifies the path to the output file,
                              the file will be overwritten. If the
                              output file does not exist, a new file
                              will be created.

  -h  --help                  Show this help.
```

Listing E.1: TLSDatabaseCreator help page

```
Description:
  Osquery pytrap module. Enriches the flow with information about
  the system based on the JA3 fingerprint contained in this flow.

Usage:  TLSFingerprintingModule.py -i IFC -o IFC -d FILE
  or:   TLSFingerprintingModule.py -i IFC -d FILE

Options:
  -i  --input IFC             TRAP input interface format.

  -o  --output IFC            TRAP output interface format.
                              If no output IFC is specified,
                              blackhole interface ("b:")
                              will be used.

  -d  --database FILE         Database file name.

  -h  --help                  Show this help.
```

Listing E.2: TLSFingerprintingModule help page