BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

EXTRACTION OF DECRYPTED DATA FROM SSL CONNECTION

EXTRAKCE DEŠIFROVANÉHO PROVOZU Z SSL SPOJENÍ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. JAKUB PASTUSZEK**
AUTOR PRÁCE

SUPERVISOR **Ing. MATĚJ GRÉGR, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2019

Brno University of Technology
Faculty of Information Technology

Department of Information Systems (DIFS) Academic year 2018/2019

Master's Thesis Specification

22185

Student: **Pastuszek Jakub, Bc.**
Programme: Information Technology Field of study: Computer Networks and Communication
Title: **Extraction of Decrypted Data from SSL Connection**
Category: Networking
Assignment:

1. Study tools used for SSL MiTM attacks - e.g. sslsplit, mitmproxy, sslsniff etc. - and select one of them based on features and performance.
2. Design an extension for the chosen tool able to mirror decrypted SSL traffic.
3. Implement the extension for the chosen tool and integrate the extended tool to NetX router platform.
4. Test the performance of your solution and evaluate results.

Recommended literature:

- Daniel Roethlisberger., *SSLsplit*. online, url: https://github.com/droe/sslsplit
- Ivan Ristic. *Bulletproof SSL and TLS*. London: Feisty Duck, 2014.
- Aldo Cortesi, Maximilian Hils, Thomas Kriechbaumer, and contributors. mitmproxy: A free and open source interactive HTTPS proxy, 2010-, https://mitmproxy.org/ [Version 4.0].

Requirements for the semestral defence:

- Items 1 and 2.

Detailed formal requirements can be found at http://www.fit.vutbr.cz/info/szz/

Supervisor: **Grégr Matěj, Ing., Ph.D.**
Head of Department: Kolář Dušan, doc. Dr. Ing.
Beginning of work: November 1, 2018
Submission deadline: May 22, 2019
Approval date: October 31, 2018

Abstract

The goal of the thesis is to develop an application able to decrypt a secure connection and mirror decrypted data to another node for analysis. The application encourages illegal purposes but the intended use of the resulting product is a legal interception. SSLsplit has been selected from the set of tools for this thesis because of its features and performance. This decision was based on tools' benchmarking and features comparison. SSLsplit signs the target server's certificates on the fly using a self-signed certificate. It runs as a transparent proxy directly on the central device in the network (router). SSLsplit performs a man-in-the-middle attack between a client and a server without any notice from either of them. The application sends the decrypted content of processed packets to a specific host in the network for further processing. Integration into the netc interface has been implemented for easier SSLsplit configuration. The application has been tested to determine its performance limits. Performance tests of the finished solution show a significant decline of transactions per second (TPS) when using SSLsplit in comparison to only forwarded traffic. The mirroring feature does not significantly affect the number of TPS or restrict SSLsplit itself. The results show that SSLsplit is capable of real operation with certain limitations.

Abstrakt

Cílem této práce je vyvinout aplikaci schopnou dešifrovat zabezpečená spojení a přeposlat dešifrovaná data na jinou stanici v síti pro další analýzu. Daná aplikace vybízí k nelegálním účelům, avšak zamýšleným použitím výsledného produktu jsou legální odposlechy. Pro tuto práci byla z množiny nástrojů vybrána aplikace SSLsplit díky jejím vlastnostem a výkonnosti. Toto rozhodnutí bylo na základě srovnávacích testů a porovnání vlastností. Pomocí vlastního certifikátu SSLsplit podepisuje certifikáty cílových serverů, které jsou vytvářené za běhu. Spuštěná aplikace běží v režimu transparentní proxy přímo na centrálním prvku dané sítě (routeru). SSLsplit provádí man-in-the-middle útok mezi klientem a serverem bez toho, aby to některá ze stran zaznamenala. Dále umožnuje dešifrovaný obsah odeslat na předem daný uzel v síti pro jeho další zpracování. Pro možnost snažší konfigurace SSLsplitu byla implementována integrace do netc rozhraní. Aplikace byla otestována za účelem zjištění jejich výkonnostních limitů. Výkonnostní testy výsledného řešení ukazují značný pokles počtu transakcí za sekundu (TPS) při použití SSLsplit v porovnání s pouhým přeposíláním provozu. Funkce zrcadlení významně neovlivňuje počet TPS ani neomezuje samotný SSLsplit. Výsledky ukazují, že SSLsplit je schopen reálného provozu s určitým omezením.

Keywords

mirror, SSL/TLS, SSL Proxy, HTTPS

Klíčová slova

zrcadlení, SSL/TLS, SSL Proxy, HTTPS

Reference

PASTUSZEK, Jakub. *Extraction of Decrypted Data from SSL Connection.* Brno, 2019. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Matěj Grégr, Ph.D.

Rozšířený abstrakt

Cílem této práce je vyvinout aplikaci schopnou dešifrovat SSL spojení a zrcadlit dešifrované data na jiný uzel sítě pro další analýzu. Práce je do značné míry spojena s protokolem HTTP sloužícím pro komunikaci s web serverem. Zabezpečení protokolu HTTP je prováděno pomocí TLS protokolu, který se v ISO/OSI modelu nachází v relační vrstvě. TLS používá "handshake" pro vyjednání šifrovacích algoritmů a dalších parametrů zabezpečeného spojení. Spojením protokolů HTTP a TLS vznikne protokol HTTPS. Digitální certifikáty jsou v HTTPS používané pro ověřování identity serveru.

Obsahem práce je, mimo jiné, správa certifikátů a to jak jejich vytvoření, tak i podepisování. Práce se zabývá také možnostmi revokace certifikátů a ověřováním jejich stavu a to hlavně z důvodu zabránění takovým akcím ze strany uživatele, aby nebylo prozrazeno použití výsledné aplikace. Vysvětlena je důležitost certifikační autority a kořenových (důvěryhodných) certifikátů při použití šifrovaného spojení a to jak z hlediska důvěryhodnosti, tak integrity vyměněných zpráv. Některým položkám v rámci certifikátů je v této práci věnovaná větší pozornost. Jednou z nich je "Common Name", který udává doménové jméno serveru, dále "Server Name Indication" umožňující více doménových jmen na jedné IP adrese a "Subject Alternative Name" umožňující uvést více různých doménových jmen do jednoho certifikátu. Dalším bodem jsou proxy servery, kterých je mnoho typů, ale v této práci jsou využity hlavně transparentní proxy servery.

Útoky Man-in-the-Middle (MitM) dovolují útočníkovi odposlouchávat komunikaci mezi dvěma zařízeními. Jistou ochranu proti tomuto typu útoku nabízejí politiky "HTTP Strict Transport Security" a "HTTP Public Key Pinning", které sdělují klientskému prohlížeči, že na stránku má přistupovat pouze pomocí protokolu HTTPS a že server může použít pouze určité certifikáty. Výsledná aplikace těmto ochranným mechanismům zabraňuje v působení tím, že žádosti serveru na jejich využití jsou odstraněny z hlaviček HTTP odpovědi. Nástroje umožňující provedení MitM útoku a při tom jsou schopny pracovat jako transparentní proxy servery jsou hlavní náplní této práce. Některé ze zkoumaných nástrojů jsou schopné generovat certifikáty serverů za běhu nebo blokovat žádosti o ověření platnosti daného certifikátu. Nástroj Mitmproxy je velice schopný nástroj umožňující hlavně živé zobrazení probíhající komunikace a případnou možnost změny její části. Nástroj SSLsplit je více optimalizovaný na výkon a umožňuje zaznamenávat více informací do souborů než ostatní nástroje. Velkou výhodou oproti ostatním má SSLsplit v tom, že již obsahuje implementované rozšíření pro zrcadlení provozu. Sslsniff je nástroj, který byl vytvořen zejména pro demonstraci některých ze známých útoků a jeho vývoj dále nepokračuje. Poskytované funkce nástrojů Mitmproxy a SSLsplit jsou skoro totožné, zatímco Sslsniff výrazně zaostává. Porovnání nástrojů pomocí výkonnostních testů ukázalo, že SSLsplit nejméně zatěžuje procesor a při tom dosahuje nejvyšších hodnot v propustnosti, proto byl vybrán pro integraci do netc rozhraní.

Implementace do NETX routeru, respektive do rozhraní příkazového řádku nazývaného netc je provedena pomocí několika souborů. Popisující soubor ve formátu YAML obsahuje různé kontexty nastavování nástroje SSLsplit a všechny možné volání funkcí pro zobrazení nápovědy, získání a nastavení hodnot a další. Všechny nakonfigurované hodnoty jsou ukládány do datové struktury DataTree a přepisovány dle potřeby. Servisní soubor obsahuje nastavení spuštění nástroje SSLsplit, které je, více méně, řízeno proměnnými prostředí nastavenými v konfiguračním souboru. Konfigurační skript je hlavním souborem a dalo by se říct, že i tím nejdůležitějším. Obsahuje výchozí hodnoty některých parametrů, všechny funkce pro nastavení a získání hodnoty z DataTree, funkce pro získání všech možných vstupních hodnot a tou nejdůležitější funkcí je vytvoření konfiguračního souboru obsahu-

jící proměnné prostředí. Skript pro statistiky využívá konfigurační soubor pro nalezení souboru obsahujícího řetězce připojení, ze kterého jsou získány veškeré statistiky. Certifikáty zmíněné dříve jsou téměř nepostradatelnou součástí zabezpečeného spojení a proto vytvoření podepsaného certifikátu pro nástroj SSLsplit je součástí této práce. Přesměrování provozu je v případě použití transparentní proxy velice důležité. Provoz který má být zpracován nástrojem SSLsplit musí být do něj určitým způsobem nasměrován.

Závěrečné výkonnostní testování prokázalo, že nástroj SSLsplit významně snižuje počet transakcí za sekundu (TPS) v porovnání s pouhým směrováním komunikace. V případě HTTPS komunikace je rozdíl v počtu TPS s použitím zrcadlení a bez něj téměř totožný. Z toho plyne, že funkce zrcadlení nijak výrazně nezpomaluje zpracování procházejících paketů. Při testování na fyzických strojích bylo zjištěno, že v případě HTTPS komunikace je SSLsplit schopen provést cirka 4 000 TPS. Vliv zapnutí funkcionality zrcadlení na zátěž procesoru v porovnání s vypnutou touto funkcionalitou byl změřen a stanoven na jednotky procent. Propustnost routeru za použití a zpracování komunikace nástrojem SSLsplit je snížena pouze v řádu procent. S ohledem na zjištěné skutečnosti je nástroj SSLsplit vhodný pro použití na jednoho klienta nebo malou síť.

Extraction of Decrypted Data from SSL Connection

Declaration

Hereby I declare that this master's thesis was prepared as an original author's work under the supervision of Ing. Matěj Grégr, Ph.D. All the relevant information sources, which were used during the preparation of the thesis, are properly cited and included in the list of references.

.....
Jakub Pastuszek
May 21, 2019

Acknowledgements

I would like to express my sincere gratitude to my superior Ing. Matěj Grégr, Ph.D. for his guidance, valuable advice, patience and willingness to help. Also, I would like to thank my family and friends for their support during the development of the thesis.

Contents

Chapter 1

Introduction

Nowadays, a secured connection is an important and inherent part of the Internet connection. Security is ensured by protocols such as SSL, its successor TLS or others. Security is important to retain everybody's privacy.

There are many tools that intercept secure connection and then reveal private information. These tools use different methods and approaches to gather information from secure connections, such as passive listening, extraction of interesting information like Common Name, form inputs and others. These tools are not always used only by attackers. Valid use cases exist, e.g., troubleshooting during application development with secured connections, testing or legal interception. The aim of the thesis is to develop a tool capable to perform man-in-the-middle (MitM) attack (2.4.3) and to decrypt intercepted data. The tool shall be installed as a transparent proxy on a router. The main goal in the thesis is to capture and mirror traffic for troubleshooting or legal compliance.

This thesis has the following structure. Chapter 2 describes important web applications' protocols with the main representative HTTP (2.1.1) and its secured extension HTTPS (2.1.2). There are also descriptions of TLS protocol, negotiation of security attributes (TLS handshake) and a digital certificate.

Applications which are capable of performing MitM attack (penetration tools) are discussed in Chapter 3. Each individual tool is tested for a performance and all its features (3.4) are considered in the selection of the best tool for the thesis. The evaluation of penetration tools take into account tools' features and the results from tools' benchmarking. SSLsplit is selected as the right tool for the thesis according to its features and for the most suitable performance in benchmarks.

Chapter 4 describes all files which were created as part of the solution of the thesis. In the chapter, the configuration file and certificate creation are described. A section about a proper forward and redirection setting of the clients' traffic is also present.

Results of the performance tests of SSLsplit are in Chapter 5. First tests were performed in the virtual environment where the limits of the virtual environment were hit but the essential overview was achieved. On physical machines was found a maximum of TPS with relation to a number of opened TCP connections and also was measured a drop in throughput when the tool was launched. Results from testing of the physical environment have greater value and therefore more emphasis is placed on the accuracy of results.

Chapter 6 summarises the thesis and contains an evaluation of the performance tests. The key part of the thesis was to implement the integration of the tool to NETX's netc command-line interface and the major part was to do a performance testing of the final solution.

Chapter 2

Secured Transport

This chapter is about how a secure connection could be established (TLS 2.1.3), what security mechanisms are nowadays widely used (2.2), how could anybody be at least slightly anonymous (proxies 2.3) and what preventing policies and protections are deployed to protect a computer on the network against attacks (2.4).

2.1 Protocols

A protocol is a convention or standard, according to which an electronic communication and data transfer between two endpoints take place. An accurate description of the protocol facilitates interoperability of different implementations of computer applications.

2.1.1 Hypertext Transfer Protocol

Hypertext Transfer Protocol [7] (HTTP) is an application layer protocol designated for communication with WWW servers. It serves for transfer of hypermedia documents in HTML, XML and other formats. The HTTP is a request-response protocol. The client (usually internet browser) submits an HTTP request to the server in a plain text format containing a type of document, HTTP version, domain name (*Host*), cookie, information about used internet browser (*User-Agent*), etc. The server afterwards returns a response to the client. The response contains completion status, information about the request (known as headers) and also may contain requested content (known as a body).

The HTTP is called a stateless protocol because each request is executed independently, without any knowledge of preceding requests. The standard port on the server side is 80 TCP. Status codes are an integral part of the HTTP protocol, they indicate whether a specific HTTP request has been completed successfully. The HTTP uses Uniform Resource Locator[1] (URL) which unambiguously specify a location of some source on the server.

2.1.2 Hypertext Transfer Protocol Secure

Hypertext Transfer Protocol Secure [16] (HTTPS) is a protocol allowing secure communication over a computer network. The HTTPS is widely used HTTP (2.1.1) extension where the HTTPS protocol is encrypted using TLS (2.1.3) or its predecessor SSL. The protocol is therefore also often called HTTP over TLS or HTTP over SSL. The HTTPS ensures authentication, confidentiality of transfer data and integrity. The standard port

[1] URL – https://tools.ietf.org/html/rfc1738

on the server side is 443 TCP. The degree of security depends on user behaviour (a user could bypass security alerts), protocol implementation in the client's internet browser (e.g. security holes, plugin vulnerabilities), the web server correctness of configuration and confidentiality of certification authorities (described later in this chapter). Table 2.1 summarises the advantages and disadvantages of using HTTPS over HTTP.

Pros	Cons
Identity verification	A need for certificates and their renewal
Confidentiality of transmitted data	Slightly complicated web server configuration
Content integrity	Greater overhead and latency

Table 2.1: HTTPS Pros and Cons (in comparison to HTTP)

Figure 2.1 shows a typical scenario when a web browser tries to connect to a web server by insecure protocol and the server responds with a request to redirect to secure one. While a web browser tries to connect to a secured site, firstly there has to be a TLS handshake (2.1.3) during which the web browser, among other things, verify the web server's certificate. After the handshake, the connection is secured by TLS and the communication could begin.

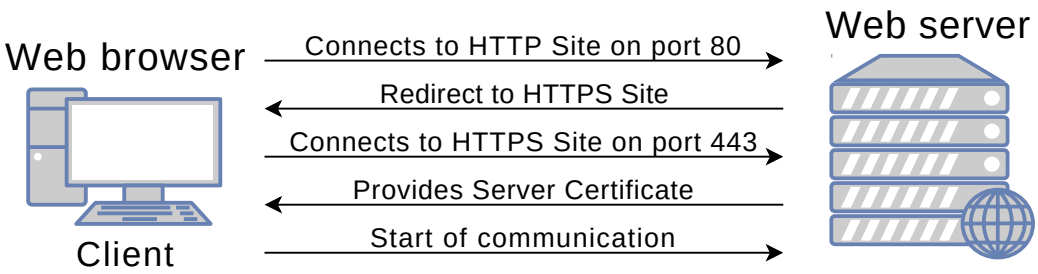

Figure 2.1: Typical web communication – connect to an insecure site and redirect to secure

2.1.3 TLS/SSL

Transport Layer Security [3] [17, p. 1–4] (TLS) and its predecessor Secure Sockets Layer (SSL), are security protocols designed to secure communication between a client and a server over a computer network. Despite the fact that SSL is now considered insecure, this term is still used for TLS[2].

TLS runs below the application layer and could add security to any protocol using a reliable transport protocol. TLS allows a computer to communicate over the computer network in a way which prevents eavesdropping and message forgery. There are multiple versions of TLS vary in security levels, amount of patched vulnerabilities and more. Authentication and privacy for endpoints are provided by cryptography. One of the protections against known attacks on TLS, such as an attempt to use less secured TLS protocol version or weaker ciphering algorithm, is a message ending initialisation (*Finished*) which contains a hash of all messages exchanged during the initialisation phase. To protect against Man-in-the-Middle attacks (2.4.3), the client compares the current DNS name of the server to the name of the certificate. Secure TLS connection is composed of three phases:

[2]The use of 'TLS' or 'SSL' refers to both SSL (outdated and insecure) and TLS (1.0 and up) in the generic sense, unless otherwise specified.

- endpoints' agreement on used algorithms
- keys exchange based on asymmetric cryptography using certificate(s)
- traffic encryption using symmetric cryptography

TLS Handshake

Handshake [17, p. 26–32] is automated negotiation which goal is to set up the parameters of a communication channel between two subjects before the start of sending relevant (useful) data. TLS handshake is a protocol to negotiate security attributes. Both endpoints could provide different ciphering and hashing algorithms or different data security. The goal of the handshake is to select the most secure option supported by both subjects.

Client		**Server**
Client Hello	$\longrightarrow$	
	$\longleftarrow$	Server Hello
	$\longleftarrow$	Server Certificate*
	$\longleftarrow$	Server Hello Done
Client Key Exchange	$\longrightarrow$	
Key Generation		Key Generation
Certificate Verify		
Change Cipher Spec	$\longrightarrow$	
Finished	$\longrightarrow$	
	$\longleftarrow$	ChangeCipherSpec
	$\longleftarrow$	Finished
Application Data	$\longleftrightarrow$	Application Data

Table 2.2: TLS handshake between the Client and the Server (*depends on cipher suite)

A typical initialisation of a TLS connection is shown in Table 2.2. The client sends a *ClientHello* message specifying the highest TLS protocol version which supports, random value and a list of suggested ciphering suites and compression methods. The server responds with a *ServerHello* message specifying selected TLS protocol version, its random value, a ciphering suite and a compression method selected from the client's list of suggestions. Then the server sends its *Certificate* (2.2). As the next step server sends a *ServerHelloDone* message to finish initialisation. The client responds a *ClientKeyExchange* message containing either a PreMasterSecret or nothing (regard to a selected ciphering suite). Both subjects could calculate a MasterSecret (described later in the section) from random values and a PreMasterSecret. The client verifies the server's certificate. If the certificate is not valid the client terminates the connection. The client sends a *ChangeCipherSpec* record, essentially telling the server 'everything I tell you will be encrypted'. Finally, the client sends an encrypted message *Finished* containing a hash and a Message Authentication Code (MAC) of previous initialisation messages. MAC is a short piece of information used to authenticate a message. The server attempt to decrypt the client's message and verify its hash and MAC. If the decryption or verification fails, the handshake is considered unsuccessful and the connection should be terminated. Finally, the server sends message *ChangeCipherSpec* and its encrypted *Finished* message and the client performs analogue decryption and verification.

At this point is TLS handshake complete and both endpoints have a shared key by which will be upcoming communication encrypted.

TLS Master Secret

A Master Secret [3, p. 64] is always 48 bytes. Four keys are derived from the value, two for a client and two for a server. One of them on each machine is a MAC key and the other is a write key. MAC keys are for the authentication and integrity within the MAC algorithm of a selected ciphering suite. Write keys are used for symmetrical encryption.

2.2 Digital Certificate

A digital certificate, also known as a public key certificate or an identity certificate (further referred as a certificate), is a digitally signed public cryptography key issued by Certification Authority (2.2.4). A certificate is saved in X.509 format [1] and contains information about the owner of the public key and an issuer of the certificate. Certificates are electronic documents used to identify a counterparty during the creation of the secured connection (proving the ownership of a public key). The owner of the certificate is called the subject. The subject could obtain a certificate by applying to a certificate authority with a Certificate signing request [12] (CSR).

2.2.1 Structure of a Certificate

Data of a certificate are expressed in Abstract Syntax Notation One[3] (ASN.1). The public key is linked to the certificate and the private key resides on a device where the certificate is installed. This ensures that data encrypted by the public key could be decrypted only by the paired private key and not by any other key. The main pieces of information stand in a certificate are the following:

- **Serial Number** – used to uniquely identify the certificate within CA's system
- **Subject** – the entity a certificate belongs to (a machine, an individual or an organisation), the field always has to contain a subfield named Common Name (CN) as a domain identifier, there are other subfields which are required for a higher certificate's validation level (2.2.6)
- **Issuer** – the entity that verified the information and signed the certificate
- **Not Before** – the earliest time and date on which the certificate is valid
- **Not After** – the time and date past which the certificate is no longer valid (the most common validity is a year)
- **Key Usage** – the purpose of the public key usage
- **Extended Key Usage** – the application in which the certificate may be used
- **Public Key Algorithm** – the algorithm used to create the public key
- **Public Key** – the public key belonging to the certificate subject

[3] ASN.1 – https://tools.ietf.org/html/rfc3641

- **Signature Algorithm** – the algorithm used to create the signature
- **Signature** – a digital signature of the public key created by CA (2.2.4)

2.2.2 Certificate Revocation List

Certificate revocation list (CRL) [1, p. 54–70] is a list of digital certificates that have been revoked by the issuing CA (2.2.4) prior to their claimed *'Not After'* date. A certificate on this list should no longer be trusted. The certificate is the most commonly revoked on the owner request. Revocation is done by adding the serial number of the certificate to the CRL list.

There are two states of revocation defined in RFC 3280[4]: *Revoked* – the certificate is irrevocably revoked and *Hold* – the certificate is temporarily revoked. A revocation request contains a reason for revocation according to RFC 5280[5].

A revocation restricts possible abuse of certificate and reason for this action are mainly: a change of information in the certificate, private key theft or password disclosure. Restricted certificate's validity protects against overloading CRL by many requests (a certificate past *'Not After'* is not valid).

2.2.3 Online Certificate Status Protocol

Online Certificate Status Protocol [19] (OCSP) is an internet protocol used for gathering a list of revoked X.509 digital certificates. This protocol was created as an alternative to CRL in order to eliminate some of its specific problems. OCSP is based on HTTP and it is encoded in ASN.1. The server providing OCSP service is called OCSP responder. The responder is commonly maintained by CA (2.2.4) which issued the certificate. The main advantage over CRL is smaller requirements on bandwidth because response usually contains less information.

OCSP Stapling

OCSP stapling [13] is known as the TLS Certificate Status Request extension. OCSP stapling is a method for quickly and safely determining whether the certificate is valid or not. It allows a web server to provide information on the validity of its own certificates by piggybacking validity status to the TLS handshake (2.1.3).

2.2.4 Certificate Authority

A certificate authority or a certification authority [17, p. 64] (CA) is an entity which issues digital certificates (electronically signed public cryptography keys) and ensures the truthfulness of the information provided in the certificate. A CA could issue multiple certificates. The list of certificates, from the root certificate to the end-user certificate, represents a certificate chain. There are three types of certificates within the certificate chain. The first one and the most important is the root certificate (described further) which is the top-most certificates of a tree. The next one is the intermediate certificate which inherits the trustworthiness of the root certificate. Intermediate certificates have the 'Certificate basic constraints' field set to 'CA'. And the last one is the end-user certificate which could

[4]Revocation states – https://tools.ietf.org/html/rfc3280
[5]Revocation reason – https://tools.ietf.org/html/rfc5280 (page 69)

be any other certificate. Figure 2.2 shows all three types of a certificate with appropriate connections between them. Every Root certificate, as well as every Intermediate certificate, could have more children certificates.

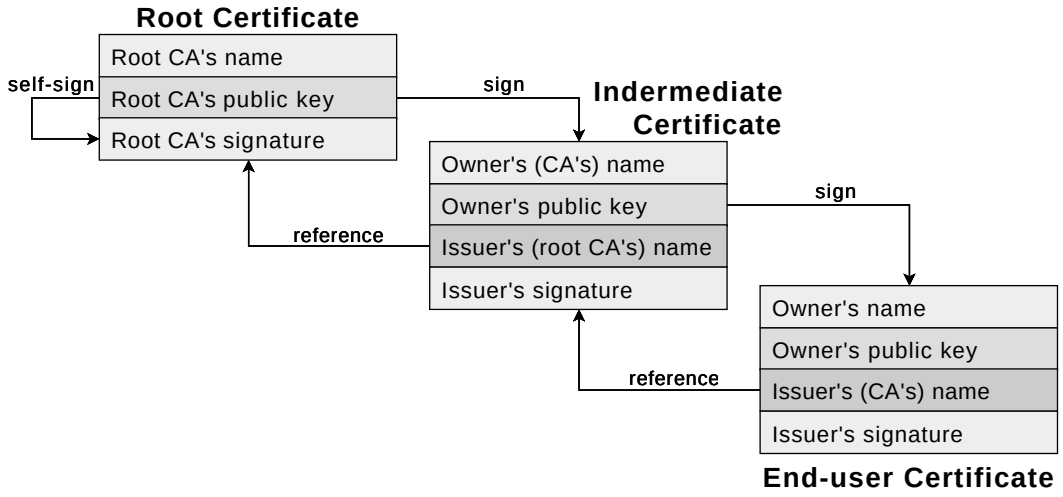

Figure 2.2: Example of the certificate chain

Based on the principle of transferred trust we could trust the information contained in digital certificate assuming we trust its CA. In other words, someone who trusts the root CA implicitly trusts all the intermediate CAs, and then by extension, all the client certificates issued by those intermediate CAs. One of the common use of CA is to sign certificates used in HTTPS (2.1.2), called end-user certificates. A CA is usually a company that charges customers to issue certificates for them although some free CA exists (e.g. Let's Encrypt). CAs are responsible for maintaining up-to-date revocation information (CRL 2.2.2/OCSP 2.2.3) about certificates they have issued. In the SSL system, anyone could generate a signing key and sign a new certificate with that signature.

Root Certificate

A root certificate is a public key certificate that has to be issued by a trusted CA. A trusted CA is an entity that has been entitled to verify that someone is effectively who claims to be. Root certificates are self-signed (2.2.5) and are on the top of the certificate chain. A root certificate is usually made trustworthy by some mechanism other than by signing. For example, operating systems and web browsers ship with a set of trusted root CAs.

2.2.5 Self-signed Certificate

A self-signed certificate [17, p. 255] is a certificate with a subject that matches an issuer and a signature that could be verified by its own public key. This type of certificate is used for testing (we trust the counterparty, we do not have to validate its identity) or for creating own CA (2.2.4). These certificates are not trusted by other applications and operating systems which could lead to authentications errors.

2.2.6 SSL Certificate Validation Level

A validation level determines the method adopted by the Certificate Authority (2.2.4) to confirm the identity of the certificate applicant. A certificate provider could issue three types of certificates as follows.

Domain Validated

A Domain Validated SSL Certificate (DV certificate) only validates the domain (specified in 'Common Name' described in section 2.2.1) and eventually its subdomains (specified in 'Subject Alternative Name' described in section 2.2.8) by proving some control over a DNS domain.

Organisation Validated

An Organisation Validated SSL Certificate (OV certificate) validates the domain ownership, plus the organisation information included in the certificate such a name, country, state and city.

Extended Validation

An Extended Validation SSL Certificate (EV certificate) validates the domain ownership, the organisation information, plus the legal existence of the organisation including manual verification by a human. This certificate is generally identified by a green address bar in the internet browser containing the company name.

2.2.7 Server Name Indication

A big limitation of TLS (2.1.3) is that each certificate requires its own IP address which means that is not possible for multiple domains with independent certificates to share the same IP address. The solution is Server Name Indication [4] [17, p. 57–58] (SNI) which lets the client specify the remote server name. SNI is an extension field of an SSL certificate by which the client indicates to the server which hostname is attempting to connect to. Using this field server could select the appropriate cipher key for indicated *VirtualHost*. The problem of SNI is that this field is not encrypted, so an eavesdropper could see which site is being requested.

2.2.8 Subject Alternative Name

A Subject Alternative Name [17, p. 69] (SAN) is an extension field of an SSL certificate which allows securing more domain names by one SSL certificate. This field is practically a list of domain names beyond domain mentioned in a Common Name field (2.2.1). The certificate using this extension field is called a Multi-Domain Certificate or term coined by Microsoft the Unified Communication Certificate (UCC). If the certificate contains a SAN extension field many devices do not take into account a CN field. Skip a CN field and search only in SAN field – the result is that the domain mentioned in a CN field has to be also listed in the SAN's domain list. A certificate with a SAN field could secure more *VirtualHosts* sharing one IP address. Thanks to the SAN it is not necessary to assign the public IP address to each domain. A SAN is also very useful as a fix to www vs. no-www problem. DV (2.2.6) SAN certificates only support subdomains of the base domain

in Common Name whereas OV SAN certificates allow include any Fully Qualified Domain Name[6] (FQDN).

2.3 Proxy Server

In computer networks, a proxy server is a server that acts as an intermediary between the client and the target server. The proxy server translates client requests and forwards them to the target server while towards the server acts as the client. The server sends back a response and the proxy server forwards it to the client. The proxy server could be specialised hardware or software running on a common computer.

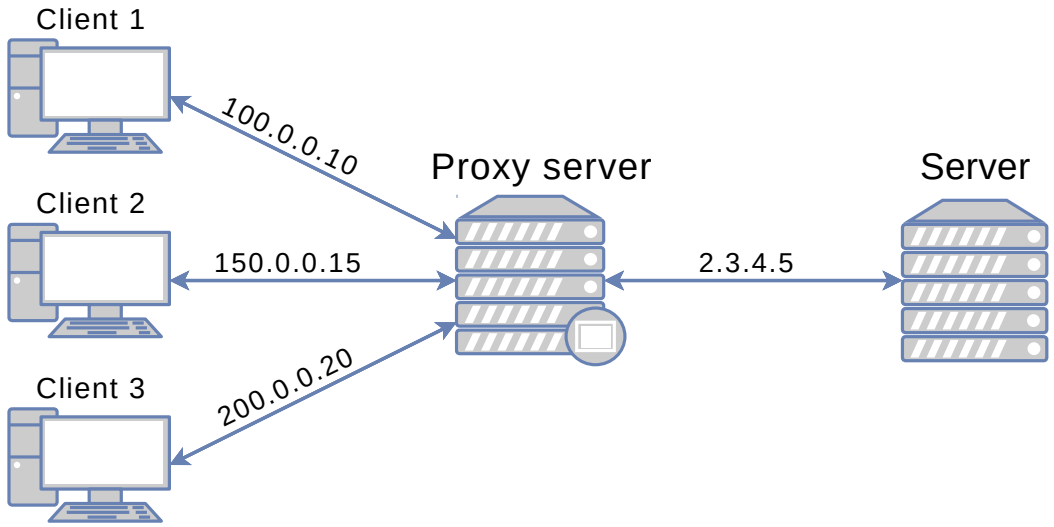

Figure 2.3: Schema of the network with the Proxy server between Clients and the Server

In this thesis are considered only proxies in a local computer network – further called local proxy. A local proxy separates a local computer network (a.k.a. intranet) from the internet. Local proxies could stand in the network as the security wall (similar to a firewall [20]) and restrict clients to connect to some servers or block using some protocols. Another benefit of using proxies in an intranet is hiding clients behind the proxy as well as NAT [8] does. Moreover, the proxy could hide the clients' geographical location, block cookies, etc.

There are three levels of proxies according to their anonymity:

- **Highly anonymous proxy** (level 1) – the server could not detect whether the client is using the proxy
- **Anonymous proxy** (level 2) – the server knows the client is using a proxy but cannot detect the client's real IP address
- **Transparent proxy** (level 3) – the server knows that the client is using the proxy and could also detect the client's real IP address. Transparent proxies are an unobtrusive way to add features and functionality to clients' browsing. Transparent proxies are considered transparent because the client is not aware of them. A transparent proxy allows implementing of a proxy without configuration of clients' devices.

[6]FQDN – https://tools.ietf.org/html/rfc1035

The proxy server could be located on the client computer or between the client and the target server. There are many different proxy types like:

- gateway or tunnelling proxy – does not modify the response body
- forward proxy – internet oriented, allows hide of the client's IP address
- reverse proxy – usually internet oriented, used to protect access to the server in private network, could do load balancing, authentication, ciphering, caching and more

Individual proxies are described further in this chapter.

2.3.1 HTTP Proxy

An HTTP proxy is specially made for HTTP (2.1.1) connections but could be used for other protocols as well. A client's browser sends an HTTP request to the proxy. The proxy forwards the actual request to the server. The server only sees the proxy as the submitter and answers the proxy as if responds to the client. The proxy receives the HTTP response and forwards it back to the client. This process is transparent and very close to a direct connection to the server. HTTP proxy could also forward secured communication but could neither view nor manipulate the content. The proxy then acts as a facilitator – blindly forwards data in both directions.

As the proxy could see the content of the message, it could change or add some data for various purposes. The proxy could add some headers identifying the client using a proxy like revealing the client's real IP address. As could be seen in Figure 2.3 there are three parties – there is not a direct connection between the client and the server. Both parties just communicate with the proxy between them. The client often trusts the proxy to be secure and not to disclose his identity.

2.3.2 SSL Proxy

It is possible to use SSL technology (2.1.3) between the client and the proxy server and then also use SSL between the proxy server and the server. In this case, the encryption and decryption occur twice, once on each 'hop'. It is possible to use the SSL proxy in the way the client connects unreliable to the proxy and then the proxy connects to the server using SSL encryption. The SSL proxy controls an HTTPS and other SSL traffic. The SSL proxy could represent the functionality of HSTS (2.4.1) so it rejects to connect by unreliable protocol (HTTP) to the server if it knows that the server is secured. The SSL proxy could have some list of secured domains which are already secured or at least the administrator wants clients to connect to them securely. In most applications, the proxy server acts as an intermediate CA (2.2.4). It is important to ensure that every device which connects to the proxy has imported proxy's certificate to the list of trusted CAs as the intermediate certificate may not be trusted.

2.3.3 Socks Proxy

A socks proxy is a versatile forward proxy but it is not as common as the HTTP proxy (2.3.1). That means it could work with various network protocol and could operate on any port. Most of socks proxies are highly anonymous due to the fact they are not as smart as HTTP proxy so they cannot interpret the information that is being sent and received.

There are two versions of socks proxies, version 4 (Socks4) which allows working with TCP [15] protocols only, whereas version 5 (Socks5) provides extra support for IPv6, UDP [14], DNS lookup[7] and more choices for authentication.

2.4 Attack and Protection

Pervasive monitoring [6] is according to the IETF community an attack on the privacy of Internet users and organisations. In other words, any unsolicited monitoring is an invasion of privacy. Exist many attacks of various types and capabilities that make monitoring feasible. There is an effort to prevent monitoring by introducing new protections and policies, some of them are described in this section.

2.4.1 HTTP Strict Transport Security

HTTP Strict Transport Security [10] [17, p. 295–303] (HSTS) is a web security policy mechanism that helps to protect communication between a web client and a web server against protocol downgrade attacks and simplifies protection against connection abduction (a.k.a. cookie hijacking). HSTS allows the web server to enforce using encrypted HTTPS (2.1.2) connection and exclude data transfer by the untrustworthy HTTP 3protocol. Web server conveys web client support of HSTS by adding HTTP header *'Strict-Transport-Security'* which defines time range during web client should approach to web server only via a secured connection. When a web application provides HSTS Policy to a User-agent, that agent behaves as follows. Automatically transfer all untrustworthy links which point to the web application to be trusted, before it connects to the web server. If there is a problem to create a trusted connection, an error message appears and the client will be restricted to enter the web application.

2.4.2 HTTP Public Key Pinning

HTTP Public Key Pinning [5] [17, p. 307–316] (HPKP) is protection against interchange certification authorities and server keys. In many ways, it is similar to HSTS (2.4.1) but HPKP is more specific. An HPKP enumerates a list of specific certificates (use their fingerprints) which could be used on the web server. A web client should not accept another certificate. A web server sends *'Public-Key-Pins'* header which by its value define maximum time from last connection during which should a web client check pinned keys against received and alternative fingerprints of keys which should be the only ones trusted for the future connection for the domain name (order of fingerprints does not matter). An HPKP is supported in Chrome, Firefox, Opera, but not in Internet Explorer. Chrome deprecated this protection with Chrome 67 release.

2.4.3 Man-in-the-Middle Attack

A Man-in-the-Middle attack [17, p. 18–22] (MitM) allows the attacker to eavesdrop on the communication between two targets, e.g., the client and the server. The basic idea is to pretend to be the server for the client and pretend to be the client for the server. The attacker tries to mask his presence and making it as if there is no other party involved in the communication. The goal of this attack is to read, insert or modify data within a

[7] DNS lookup – https://tools.ietf.org/html/rfc1034

communication. Typical MitM attack has to go through two phases, the first is to make a redirect of the communication and the second is to decrypt and re-encrypt the data. There are several types of redirecting methods like IP spoofing, DNS spoofing and others. In the case of decryption, there are also many possible techniques like SSL Hijacking, SSL Stripping (described further), various SSL vulnerabilities, etc.

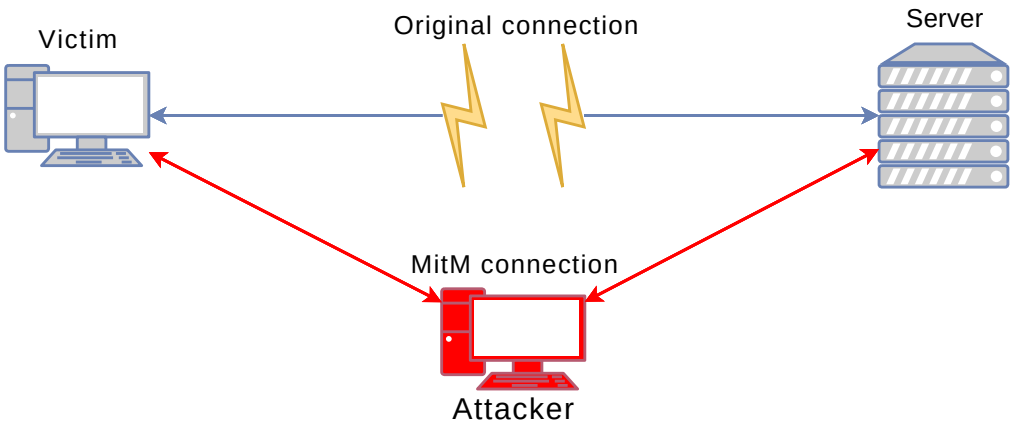

Figure 2.4: Schema of a Man-in-the-Middle attack

Critical for this scenario is that the client (the victim) is not aware of the attacker intercepting the communication. Schema of a MitM attack could be seen in Figure 2.4 where the victim wants to connect to the server (or was connected) and after a successful attack is connected to the server via the attacker's machine and the attacker could intercept a communication. The tricky part for a successful MitM attack on a secured connection is that a CA (2.2.4) system is designed to prevent exactly this type of attack, by allowing a trusted third party to cryptographically sign a server's certificate to verify that it is legit. In the case of bi-directional authentication, it is almost impossible to perform a MitM attack because you have to import generated CA certificate to both the client and the server.

There are several types of attacks. One of them is a method called ARP spoofing. The attacker is from the network point of view a client connected to this network (does not matter if wired or wireless). After connection, the attacker could use mentioned ARP spoofing attack, which in case of a successful attack deceive the victim that the attacker is a new gateway and every packet which goes outside of the network is sent to the attacker. In that state, the attacker could see all the victim's communication. Fortunately, these days we have a secure connection which is established directly between two communicating endpoints and thus the attacker could see only encrypted data.

SSL Stripping

SSL stripping [17, p. 125–126] (or HTTPS stripping) is a type of MitM attack. The attack is used to circumvent the security enforced by SSL certificates on HTTPS website. It is a technique that downgrades a connection from secure HTTPS to insecure HTTP. Against this circumvention exists already mentioned technique HSTS. HTTPS stripping attacks rely on the fact that most users begin their browsing on an insecure site (HTTP) and then are redirected to a secure one (HTTPS).

Chapter 3

Penetration Tools

This chapter contains a few tools which are capable of Man-in-the-Middle attack. The aim of the chapter is to select the tool which is suitable fro the thesis. It means that the tool has to be performant and has small system requirements. In the chapter, there are compared tools' features and benchmarking test were performed to be able to select the tool which will be subsequently integrated into netc command-line interface.

Using a tool as an HTTP proxy is the simplest and the most reliable way to intercept traffic. The client connects directly to the proxy and makes a standard HTTP request, then the proxy connects to the server and forwards the request on.

With respect to the current CA system, it is very hard to attempt a successful MitM attack these days. An application providing a successful MitM attack has to become a trusted CA. As the self-signed certificate is not in the client's list of trusted CAs. The application has to ensure proper manual registration of the certificate to the list. The best idea is a tool that generates interception certificates on the fly[1].

Some applications (including web browsers) employ HPKP to prevent MitM attacks. It means that a generated certificate will not be accepted by these applications. For an attacker, it is recommended to let pass through such traffic instead of interrupting the connection. There is an option for applications to log TLS Master Secrets so that external applications could decrypt TLS. One of them is, for example, Wireshark[2] in version 1.6 and higher.

3.1 Mitmproxy

Mitmproxy [2] is a free and open source HTTPS proxy. The development started at the beginning of 2010 and the last released version is 4.0 as of January 2019. There are over 250 contributors led by seven major developers. This tool is coded in Python and has very good documentation.

Mitmproxy has many features, therefore, it is complex. It could be used as a transparent proxy but there are options to use it as an HTTP or SOCKS proxy as well. Additionally, there is a simple Python scripting interface.

The tool is divided into three smaller tools. The main tool is named Mitmproxy and it is an interactive MitM proxy for HTTP and HTTPS with console interface which allows pausing, inspecting, editing, replaying or dropping flows. Mitmproxy keeps all flows in

[1] 'on the fly' – being changed while the process that the change affects is ongoing

[2] Wireshark – https://www.wireshark.org/

memory. The second tool is the command-line version of Mitmproxy called Mitmdump. Mitmdump provides tcpdump-like functionality to let user view, record and programmatically transform HTTP traffic. The last tool is a web-based user interface named Mitmweb which allows interactive examination and modification of HTTP traffic. Mitmweb, as well as Mitmproxy, keeps all flows in memory.

Mitmproxy has a feature to replay intercepted communication. Communication could be saved to a file for later use. There is a way to modify a communication by a script and eventually replay it. Mitmproxy also starts a web server with a special domain 'mitm.it' where a client could download a CA certificate (2.2.4) with setup instructions for a specific device which a client is using. If a client does not import the CA certificate he will see an SSL certificate warning every time he visits a new SSL domain. The following example shows mitmdump running in transparent mode:

```
mitmdump -v -p 10443 --mode transparent --showhost --set
    ↪ save_stream_file="connection.log"
```

Meaning of parameters used in the command is as follows:

-v	more verbose output
-p *number*	listen port
--mode transparent	mode of operation
--showhost *string*	shows host header instead of IP address
--set save_stream_file=*path*	connection log file

3.2 SSLsplit

SSLsplit [18] is an open source tool for MitM attacks against TLS encrypted network connections. The development started in 2009 and the actual version is 0.5.4 as of October 2018. There are over 20 contributors led by two major developers. SSLsplit depends on the OpenSSL[3] library. SSLsplit is written in ANSI C and works as a transparent proxy. It cannot act as an HTTP or SOCKS proxies. The tool terminates a TLS connection and initiates a new TLS connection to the original server while logging all data transmitted.

SSLsplit supports plain TCP, TLS, HTTP and HTTPS connections over both IPv4 and IPv6 addresses. SSLsplit also supports SNI and dynamic upgrade from plain TCP to TLS, called AutoSSL. Depending on the version of OpenSSL, SSLsplit supports SSL 3.0, TLS 1.0, TLS 1.1 and TLS 1.2. TLS 1.3 is not yet supported as developers of SSLsplit extended the tool by an extension capable of mirroring traffic in October 2019.

SSLsplit generates and signs forged certificates on the fly, based on the original server certificate. It is also possible to use existing certificates of which the private key is available. SSLsplit implements several features which help to stay undetected: deny OCSP requests and modify headers to prevent switching to HTTP/2 or WebSockets. For HTTP and HTTPS connections, SSLsplit mangles headers to avoid server-instructed HSTS and prevent HPKP response headers. Intercepted connections could be saved into log files which include 'connection log' and 'content log' files as well as Pcap files[4] and mirroring traffic to a network

[3] OpenSSL – https://www.openssl.org/
[4] Pcap file – https://en.wikipedia.org/wiki/Pcap

interface. Additionally, certificates, Master Secrets and local process information could be logged as well. The following example shows SSLsplit running with all possible log files and HTTPS proxy:

```
sslsplit -Z -P -d -s NULL:RC4:AES128:-DHE -l connection.log -L
    ↪ content.log -M master-key.log -c ca.pem https 0.0.0.0 10443
```

Used parameters in the command:

-Z	disable TLS compression on all connections
-P	let pass-through TLS connection if it cannot be split
-d	run in daemon mode (in the background)
-s *string*	use given cipher suite
-l *path*	log one-line summary per connection to a file
-L *path*	log full data to a file
-M *path*	log Master Secrets to a file
-c *path*	CA certificate file

Master Secrets (2.1.3) log file is in SSLKEYLOGFILE format[5]. CA certificate signs forged certificates. Proxy defines which type of connection will be intercepted: `"https"`, on which address SSLsplit will listen: `"0.0.0.0"` (all possible local addresses) and what port will have assigned: `"10443"`.

3.3 Sslsniff

Sslsniff [11] is a tool to perform a MitM attack which could only intercept HTTP and HTTPS connections. The development was done from 2008 to 2011 and since that time there were only a few forks but no further extension has been done. There was only one major developer. Sslsniff could run in two distinguished modes. In authority mode, Sslsniff acts as if it is a CA and dynamically generates certificates on the fly for whatever server. In targeted mode, Sslsniff is given a directory full of certificates and is used for targeted MitM attack against domains those certificates are in the directory.

The main goal of this tool was to demonstrate and exploit the vulnerability to a *'Basic-Constraints'* certificate's extension and later for demonstration *Null-prefix* attack[6]. Sslsniff could be configured to deny OCSP requests from clients in order to stay undetected. Moreover, Sslsniff implements several attacks on some older versions of Firefox auto-updates. Parameters used in the following example of use are described further. The following example shows Sslsniff running in authority mode:

During the configuration and compilation, there were problems which had to be fixed by changing linked libraries and rewriting part of the source code. From a perspective, this tool is not appropriate for further development as it has no community created and its development stagnates.

```
sslsniff -a -s 10443 -w content.log -c ca.pem
```

[5]SSL key log – https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/Key_Log_Format
[6]Null-prefix attack – https://moxie.org/papers/null-prefix-attacks.pdf

The parameters used in the command are as follows:

-a	authority mode and generation certificates on the fly
-s *number*	listen port
-w *path*	content log file
-c *path*	CA certificate file

3.4 Comparison of Tools' Features

The main advantage of SSLsplit is the capability of mirroring intercepted traffic and creating Pcap file output. Mitmproxy is great for live analyses and optional modification or replaying a communication and inserting new messages into traffic. While all tools support IPv4, Sslsniff lags behind the others in support of IPv6. All relevant features are depicted in Table 3.1. Logging Master Secrets is a way how to decrypt saved traffic in a Pcap file. Whereas SSLsplit has a built-in feature to log Pcap files, Mitmproxy does not have the feature and another application has to be used to save traffic to a Pcap file. HPKP preventing, HSTS overcoming and OCSP blocking enhance the tool's ability to successful traffic interception. Mitmproxy's feature to ignore explicit hosts is great when a connection to a certain host should not be split nor intercepted. All compared tools have a feature to log the content of the traffic going through. The table does not contain a complete list of tools' features but only features relevant to this thesis.

	Mitmproxy	**SSLsplit**	**Sslsniff**
IPv4	✓	✓	✓
IPv6	✓	✓	
CA certificate	✓	✓	✓
SSL stripping	✓	✓	✓
HSTS overcoming	✓	✓	
HPKP preventing	✓	✓	
OCSP blocking	✓	✓	✓
Ignore explicit hosts	✓		
Mirror traffic		✓	
Pcap output		✓	
Master Secrets logging	✓	✓	
Content logging	✓	✓	✓

Table 3.1: Comparison of provided features of individual tools

Sslsniff neither supports Master Secrets logging nor Pcap logging and regard the fact that the development stagnates the tool is not the right candidate to be selected for further development in this thesis. Both Mitmproxy and SSLsplit perform the same kind of MitM (2.4.3) attack on TLS (2.1.3). Mitmproxy is more powerful and has more features than SSLsplit, whereas SSLsplit has a more dedicated purpose, it is simpler and faster. SSLsplit has being developed in ANSI C which is a structured programming language mainly used for hardware related application development like Network Drivers or Operating Systems, whereas Mitmproxy has been written and developed in Python which is a general-purpose

programming language mainly used for Machine Learning, Natural language processing, etc. Python is robust and slower than ANSI C – in the target application is essential to provide fast operations to handle traffic on high-speed links. Regard to tools' features SSLsplit seems to be the best candidate for the development of the extension in this thesis.

3.5 Tools' Benchmarking

Benchmarking is used to measure the performance of a tool and then compare to others. Various benchmarking metrics exist but for network applications the most interesting are, e.g., a number of transactions per second, memory usage and CPU load during a test.

3.5.1 Nginx

Nginx[7] is an HTTP and reverse proxy server. It is free and open-source software. In this testing, Nginx is used solely as an HTTP server providing files of various sizes. Nginx is focused on high performance and small memory requirements. The main goal is fast delivery of static pages and to distributing the load to other servers according to a priority. Nginx has configurable cache so that recurrent requests could be served very fast. There is an option to set up a limit of connections per IP address to mitigate Denial of Service [9] (DoS) attacks.

3.5.2 Wrk

Wrk[8] is an HTTP benchmarking tool. A capability of Wrk is to generate significant load while running on a single multi-core CPU. Multiple connections are handled by a single thread. Wrk distributes the total number of connections evenly among the threads. It is useful for testing a number of connections per time period – requests per second (RPS). The tool could be also used for testing a secured connections' RPS, a number of transactions per second TPS or the server throughput and latency. Wrk prints statistics about the measured minimum, maximum, average and median values at the end of its run. Wrk could be extended by LuaJIT script, using the extension there is an option to script benchmark tests. Wrk has the following parameters which are used in tests:

-c *number*	number of connections to create
-d *number*	duration of the traffic generation
-t *number*	number of threads used

3.5.3 Sar

Sar[9] (stands for System Activity Report) is a command-line utility to write selected cumulative activity counters in the operating system. The utility could report data on the fly every second. In the following tests sar is used with only one parameter, namely `"1"` stands for grabbing machines statistics every second. Sar is used in these tests to capture CPU load on the machine where one of the tools is running.

[7] Nginx – https://www.nginx.com/
[8] Wrk – https://github.com/wg/wrk
[9] sar – https://linux.die.net/man/1/sar

3.5.4 Testing

Tests of tools described in this chapter are necessary to be able to select the right tool for this thesis. The right tool has to be performant and its requirement on CPU has to be small. To achieve relevant results it is necessary to disable Nginx's caching procedure and Nginx's DoS attacks mitigation. During the tests were not used all provided features of Wrk (3.5.2) because tested was a performance of individual tools, not server's. Testing was inspired by 'NGINX Plus Sizing Guide'[10] for performance tests of Nginx Reverse Proxy server.

In order to properly test tools described in this chapter, both ends have to be configured well. One end machine has to be configured as a web server which response on HTTP requests. The other end machine has to be configured as a client where the benchmarking tool is present.

Topology

All tests were done using three separate machines connected together with 10 GE links. As could be seen in Figure 3.1, there is a machine with a transparent proxy (2.3) in the middle which is operated by one of the tools described in sections 3.1, 3.2 and 3.3. To generate traffic from the client machine, Wrk (3.5.2) was used. All traffic was directed to the transparent proxy. A tool operating the transparent proxy intercepted and forwarded the traffic to the server.

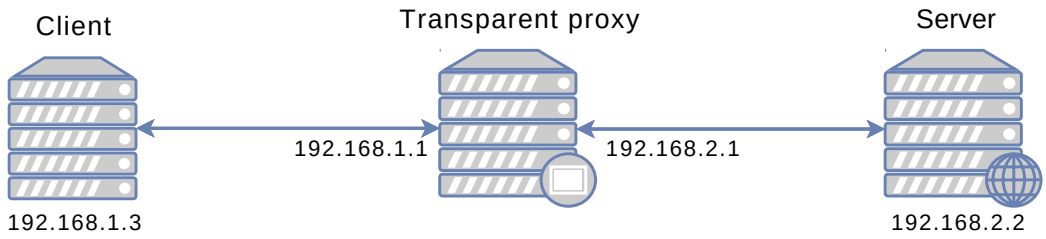

Figure 3.1: Topology used for tools' benchmark testing

Environment

Testing was performed on VMware virtual machines. All machines had the same hardware parameters, namely *CPU Intel Xeon E5-2660 @ 2.20 GHz*, *4 GB RAM* and a standard HDD. Installed OS was *NetXOS 7* with kernel *Linux 3.10.0* and architecture *x86-64*. Connections between machines were 10 GE.

Nginx – Server Configuration

Properly configured web server is needed in order to have meaningful tests results. Nginx 1.15.8 was installed on the server. The server is set to listen on two TCP ports. The first port is 80 (HTTP) and the second is 443 (HTTPS). Next, it is necessary to set up paths to the private key and certificate. There are two more necessary pieces of information which have to be set in order to HTTPS work properly. The first one is a list of available

[10] https://www.nginx.com/blog/nginx-plus-sizing-guide-how-we-tested/

SSL protocols and the second one is a set of SSL ciphers. Every mentioned setting has been configured in file '/etc/nginx/conf.d/default.conf'. Compression is also disabled on the server.

To assure the intended traffic will pass through, the following command has been executed two times, each time with a different value of a variable `<port>`. The variable values are the same as TCP ports which were set in the Nginx configuration file, namely 80 and 443.

```
iptables -A INPUT -p tcp --dport <port> -m conntrack --ctstate
    ↪ NEW,ESTABLISHED -j ACCEPT
```

Performance Metrics

The following metrics were measured:

- **TLS transactions per second (TPS)** – measures the tool's ability to process new TLS connections (file sent as a response has zero size)
- **Speed of re-encryption** – measures the tool's ability to fast re-encrypt traffic, evaluated was one large file of size 1 GB sent as a response

Running Tests

There had to be some files to test on the server. To create a file of one-kilobyte size filled with all zeros the following command was used.

```
dd if=/dev/zero of=1kb.bin bs=1KB count=1
```

A private key has been created on the server together with a public key which becomes a part of the server certificate. Generating these keys and certificate was done by following commands. The first command generates a private 2048 bit RSA key. The second command ensures the creation of a public key and a certificate with serial number 666 and expiration one year.

```
openssl genrsa -out localhost-nginx.key 2048
openssl req -new -x509 -days 365 -key localhost-nginx.key -out
    ↪ localhost-nginx.crt -set_serial 666
```

Wrk was used to generate all traffic from the client's machine. To measure requests per second (RPS) the following script was run:

```
./wrk -t 1 -c 50 -d 30s http://192.168.2.2/1kb.bin
```

To measure TLS transactions per second (TPS) the following script was run:

```
./wrk -t 1 -c 50 -d 30s -H 'Connection: close'
    ↪ https://192.168.2.2/0kb.bin
```

To measure CPU load and how fast a tool could re-encrypt traffic (measured time specifies how fast was the file downloaded) the following script was run:

```
curl -k http://192.168.2.2/1gb.bin > /dev/nul
```

Results

Figure 3.2 depicts the relationship between CPU load and time during the sixty seconds long TPS test. From the figure could be read that SSLsplit has slightly lesser CPU requirements than Mitmproxy, whereas the third tool Sslsniff crashed every time during this test with Segmentation fault error. This test runs three times and results were averaged. Results in Table 3.2 are from the same TPS test case as is Figure 3.2.

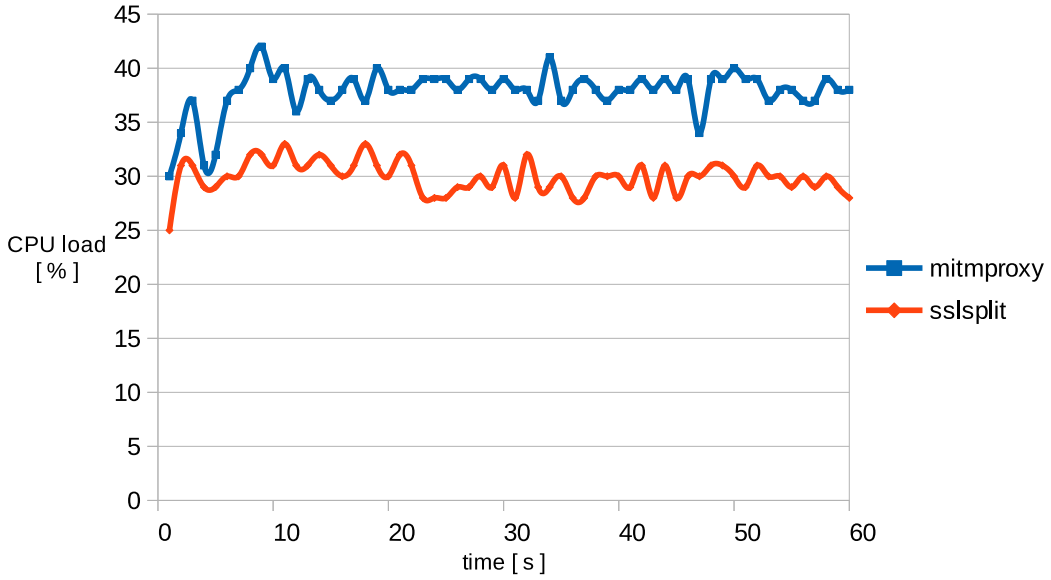

Figure 3.2: TPS graph of relation between CPU load and time

		Mitmproxy	**SSLsplit**
Latency	[ms]	960	45.6
Requests/s	[-]	31.6	373
Transfer/s	[kB]	7.5	88.7

Table 3.2: Average latency, RPS and transfer per second while measuring TPS

Figure 3.3 depicts the relationship between a CPU load and time during re-encryption test. A scale of time is in percent in order to be able to directly compare the progress of the download in different tools. On the lines, there could be seen points. Each point shows a sample captured one second after a preceding sample and thus in the figure could be seen only ten points for SSLsplit and eighteen points for Mitmproxy (corresponding to Table 3.3). The figure shows that the progress of the CPU load during downloading in SSLsplit is relatively even for the whole time, whereas in the case of sslsniff it is straight in the first half and it has a small peak in the second half. Sslsniff has for the whole time a bit higher CPU load but in comparison to Mitmproxy in download time (according to Table 3.3) sslsniff is better. Table 3.3 shows the times required to download the whole 1 GB file using different tools. The best results have SSLsplit using which the download was almost two times quicker than by using Mitmproxy.

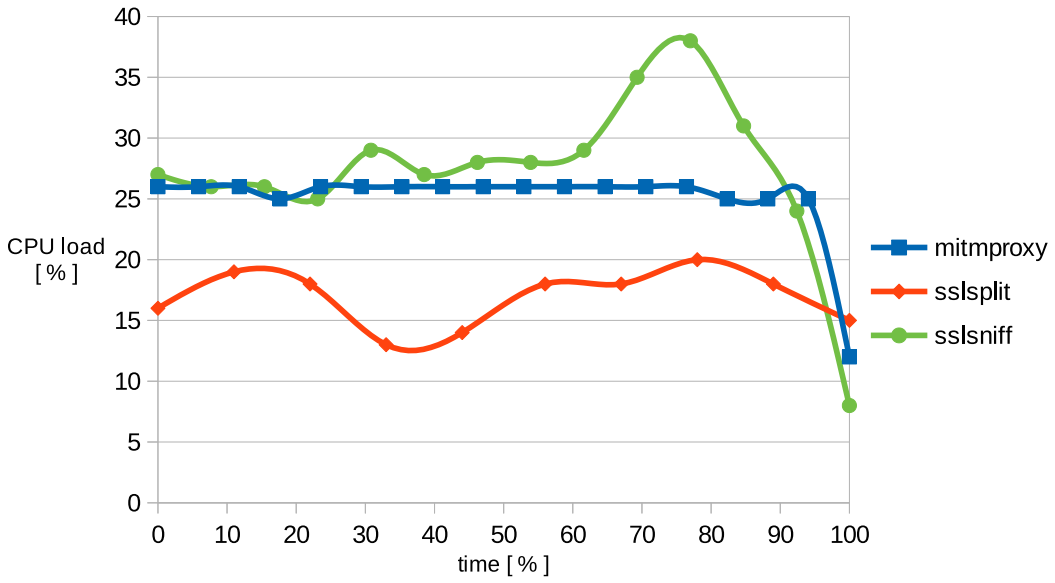

Figure 3.3: Re-encryption graph of relation between CPU load and time (rescaled)

		Mitmproxy	**SSLsplit**	**Sslsniff**
Download time	[s]	18	10	14

Table 3.3: Time of downloading a 1 GB file

3.6 Tools' Evaluation

All relevant features and tests were taken into account to evaluate the most suitable candidate to further implementation of integration (described in Chapter 4) into netc. Regard to available features mentioned in section 3.4 Mitmproxy and SSLsplit are advisable tools. Sslsniff due to its restricted amount of features, no support for IPv6 and stagnant development is not suitable at all. The thesis is aimed to HTTPS traffic so the feature SSL stripping is essential to be part of the selected tool. In this case, all tools have the feature to split SSL traffic so according to this criterion, no decision could be made.

SSLsplit has the feature which mirror traffic to another machine in the network. Instead of mirroring, or simultaneously with it, SSLsplit could log traffic to a Pcap file and log each connection per one line into a file. This feature is great for automatically analysing the file for further utilisation of the data.

Mitmproxy has the ability to modify, remove and create traffic and is more user-friendly thanks to its web-based interface. The fact that Python is interpreted language is great for some applications but not for applications which require high performance and high throughput like network applications usually do.

From the performed tools' benchmarking could be deduced that SSLsplit is a great candidate to be selected. From Table 3.2 and Figure 3.2, there is clearly visible dominance of SSLsplit in both lower CPU load, number of TPS and latency. Table 3.3 shows a huge difference in the speed of re-encryption on behalf of SSLsplit.

The results of the section show that the selected tool is SSLsplit which has all needed and advisable features. SSLsplit is written in fast, compiled ANSI C and its development is still in progress.

Chapter 4

Implementation

This chapter is about the implementation of SSLsplit extension and its integration to netc. Several files had to be created for the integration. The integration was done in order to SSLsplit could be configurable via united netc interface which is almost exclusively used on the NETX platform instead of the standard command-line interface. Firstly the NETX platform and its netc command line interface are described. Then follows a description of a describing YAML file and description of data structure DataTree. A service file with its all settings followed by configuration and statistics scripts written in Perl[1] programming language with a short description of a configuration file are all described in this chapter. In the next section of this chapter is described the creation of the CA certificate. The chapter is finished by necessary Linux forwarding and redirecting.

4.1 NETX

NETX Smart Router[2] series is an open-source routing platform. NETX was developed in cooperation with Brno University of Technology to provide high-performance routing with the capability of forwarding tens of gigabits per second. A machine running NETX could be managed via RESTful API or through command-line interface named netc. The operating system is based on GNU/Linux.

4.1.1 Netc

Netc[3] is a command line interface for NETX (4.1) products. Netc runs as a configuration shell, similarly to Cisco, Juniper or Mikrotik. Netc allows configuring a Linux-based system in the same way as network devices. Netc is written in script language Perl, also is modular and thus easily extensible and uses two types of configurations. One of them is running configuration which is actually loaded and used by netc and the other is startup configuration which is saved in a file and used upon a boot or a restart.

Various configuration contexts are used for some commands within netc. The current running or startup configuration could be displayed using the *'show'* command. The command *'show this'* could be used within any configuration context to show only the relevant part of the running configuration. To display a list of available commands or to provide

[1] Perl – https://www.perl.org/
[2] NETX – https://netx.as/
[3] Netc documentation – http://docs.netx.as/

help to partially entered command a question mark '?' could be used. To complete itemised command 'TAB' key could be used. Examples are present later in this chapter.

4.2 Configuration of Sslapp

Sslapp is a netc context which contains all possible configurations of SSLsplit. This section contains several possible configurations of sslapp. In the following code is shown an example configuration of sslapp in netc:

```
netx# sslapp
netx(sslapp)# ca-cert /home/user/sslapp/ca.pem
netx(sslapp)# cipher ECDH-ECDSA-RC4-SHA
netx(sslapp)# log connection /var/log/sslapp/connection.log
netx(sslapp)# mirror eth0 192.168.100.10
netx(sslapp)# proxy
netx(sslapp-proxy)# https ipv4 0.0.0.0 10443
netx(sslapp)# sslapp enable
```

SSLsplit's mirroring feature has to know on which interface will be the traffic mirrored. SSLsplit sends the mirrored packets with their original IP addresses. In the mirror configuration, there has to be also an IP address in order to SSLsplit could find destinations' link-local address where the mirrored packets will be sent.

For example, the second row of the code above cause calling 'xcfg-netc-sslapp set-cacert /home/user/sslapp/ca.pem'. The first part of the command defines a script file which will process the rest of the command (parameters). The rest of the command contains action and its parameter – in this case, a path to a CA certificate file.

A request to an HTTPS server usually targets port 443, but the code above causes SSLsplit to listen on port 10443. Traffic redirection (described later in this chapter) is required to redirect such request to the port where SSLsplit is listening.

4.3 Forward and Redirect

In order to be ensured a proper function of the transparent proxy (2.3) on the machine where SSLsplit (3.2) runs, there have to be correctly set forwarding and redirection. Linux offers many tools for taking care of redirection. In this case, iptables[4] was chosen. Iptables is a utility that allows a root user to configure a table for IPv4 provided by the Linux kernel firewall and the chains and rules it stores.

The first thing that has to be done is to enable both IPv4 and IPv6 forwarding. The first line of the following commands executed under root user cause enabling of IPv4 packets forwarding. The second one is for IPv6. If the `"sysctl -w"` part is removed and the remainders of the commands are added to the end of the file "/etc/sysctl.conf", then that new configuration will survive a reboot.

```
sysctl -w net.ipv4.ip_forward=1
sysctl -w net.ipv6.conf.all.forwarding=1
```

[4]iptables – https://www.netfilter.org/projects/iptables/index.html

Setting up the routing directly via iptables is in case of using netc (4.1.1) not necessary. By using netc commands which are very similar to iptables commands is routing setup even easier with context switching and autocomplete netc options. In the netc is sufficient to write the following command and all traffic from `<addr>/<mask>` network targeting to port number `443` and using TCP protocol will be forwarded to port `10443`. In this case, there is only listed a command for HTTPS (2.1.2) traffic, command for HTTP (2.1.1) would be analogical with port number `80` and would be forwarded, for example, to port `1008`. In the command are used variables closed in 'angle brackets' and in Table 4.1, there are listed all possible values of the variables which could be used in the command.

```
<IP-proto> firewall table nat chain PREROUTING action REDIRECT src
    ↪ <addr>/<mask> proto tcp dport 443 in <interface> to-port 10443
```

IP-proto	addr	mask	interface
'ipv4'	IPv4 address	IPv4 mask	machines' interface
'ipv6'	IPv6 address	IPv6 prefix-length	

Table 4.1: Possible values of variables in netc firewall command

4.4 YAML – Descriptive File

YAML[5] is a human-readable data serialisation language. Created file within this thesis is named according to netc guidance as 'sslapp.yml' because extends netc with commands for configuration context *'sslapp'*. This file is within 'netc.d' directory containing all YAML descriptive files.

The YAML structure is simple but powerful. On the beginning of the file is a short part of use cases. The breakdown is mostly self-explanatory and the whole file is divided into individual sections. Each section contains element NODE which describes the actual item in the configuration tree. Within NODE string there could be a parameterized part which is defined in netc as interface, IPv4 or IPv6 address, string and other. Another very useful element is DESCR serving as a describing text within the file to know what the item is about and also is listed beside item name while a user let netc display a list of available commands (a.k.a. autocomplete). The following example shows context switching and printed help within netc:

```
netx# sslapp
netx(sslapp)# proxy
netx(sslapp-proxy)# https ?
    ipv4 - set ipv4
    ipv6 - set ipv6
```

Netc is able to switch the configuration context as described in section 4.1.1. For that option, there are defined two elements. The first one is named SUBCONTEXT containing only a boolean value which specifies whether the item has subcontext or not. The second is PROMPT which defines the exact form of how the CLI prompt will look like. Not so

[5] YAML – https://yaml.org/

much important element is ORDER which indicates in what order will be the item listed within *'show config'* extract. The following example shows a part of the descriptive file, more accurately the proxy item containing subcontext:

```
NODE: 'sslapp/proxy'
DESCR: 'configure Proxy specification'
PROMPT: '%h(sslapp-proxy)'
SUBCONTEXT: '1'
```

Get and set elements are more interesting in terms of functionality. All the following elements are optional. The SET contains a command or a script with optional parameters which sets a certain item. The opposite of a previous element is an element named UNSET which contains a command or a script with optional parameters but as the name suggests it unsets the item. Other elements are used to get some information from the item. An element GET gets the configured value of the item or in case of subcontext gets whether the subcontext is already set or not. Very similar to GET is an element GETALL which returns all possible values which could be set. Result of GETALL is listed while is inserted question mark '?' after the actual command. The element COND contains command or script returning a boolean value which determines whether the item will be present in the printed help. The possibility of using the script within elements is shown in the following example where SET use two out of three variables as parameters and the leftover parameter is a literal as well as all parameters in the GET element:

```
NODE: 'sslapp/proxy/https/ipv4/%IP4/%NUM'
SET: 'xcfg-netc-sslapp set-proxy-https %1 %2'
GET: 'xcfg-netc-sslapp get-proxy-https port'
```

An element SHOW is the last very common element within *'sslapp'*. SHOW is used for showing actual configuration. The main difference between GET and SHOW is that the first mentioned gets information from a configuration file and primary target of the extract is running configuration, the later gets information from the DataTree.

```
NODE: 'sslapp/proxy/https/'
SHOW: 'xcfg-netc-sslapp show-proxy-https'
```

4.4.1 DataTree

DataTree is netc package which let the developer create tree hierarchies. The building blocks of these structures are Node objects. The package allows logically divide the whole application data into separated structures. For saving sslapp data, DataTree structure called *'sslapp'* is created. An example of using DataTree is shown below. A new node named *'sslapp'* is created. A string containing path is stored into the *'cacert'* subnode, then the existence of *'cacert'* subnode is tested and if it exists a message containing a path to CA certificate is printed.

```
my $DT = NetC::DataTree->new('sslapp');
$DT->Set('cacert', $path);
if ( $DT->Exists('cacert') ) {
```

```
        printf "%s: %s\n", "cacert path", $DT->GetLast('cacert');
}
```

DataTree is also used in the statistics module which is described further in this chapter. The package also provides methods for creating new nodes, getting a subtree, setting and getting list values and other. Every node has no restriction on a number of children. Each leaf node contains only two values, one of them is lastly set value and the other is previously set value. Both values contain timestamp when were set.

4.5 Service File

The service file named 'sslsplit.service' is used for initiating SSLsplit process and is placed in 'systemd' directory. Most Linux distributions use systemd[6] as a system and service manager and NETX is not an exception.

Network service is the service's only dependency. SSLsplit service has to start after the network is set (there is no suitable use of the tool without a network). A path to an environmental file is defined in the service file. The file contains all parameters configured by a user and necessary to start the tool. There is also defined a path to a PID file which ensures only one instance of the application is running. If another SSLsplit application would start via the service than existing PID file containing process ID would stop initiating a new process.

```
PIDFile=/var/run/sslapp.pid
```

While SSLsplit application starts, the process forks and the parent exits. For that reason, the service is a 'forking' type. Another property provides information to the service manager about all successful exit codes of the application. Without that information, every termination of SSLsplit via signal would look like the application failure. In the snippet below, there is shown part of the service file containing successful exit codes.

```
SuccessExitStatus=1 SIGTERM SIGQUIT SIGHUP SIGINT
```

The command to run the SSLsplit application contains environmental variables defined in the environmental file and some important parameters. Two necessary parameters to run SSLsplit as a service are `"-d"` which runs the application as a daemon (in the background) and a parameter `"-p"` with a path to the PID file. Another useful parameter is `"-P"` that let pass through traffic which cannot be split and the last preset parameter is `"-Z"` that disables SSL compression and therefore saves CPU time. The following command is a part of the service file and is executed when the service starts:

```
/usr/bin/sslsplit -Z -P -d -p /var/run/sslapp.pid $CIPHER $CA_CERT
    ↪ $OTHER $LOG $MIRROR $PROXY
```

[6] systemd – https://www.freedesktop.org/wiki/Software/systemd/

4.6 Configuration Script

Perl script file named 'xcfg-netc-sslapp' is placed in 'bin' directory and contains configuration functions. Default values used during sslapp configuration in netc are saved in the file. A substantial part of the script is formed by action commands. An action is sent to the script as the first parameter and indicates which action will be performed. Actions are divided into logically separated sections which are based on elements in the descriptive file. There are sections starting with the string 'set', 'unset', 'get', 'getall' and 'show' which are used for the action as the name suggests for each individual element. These sections are described further in this section. There are also two special sections named 'config' and 'start', former one builds the configuration and exits, the other builds the configuration and starts SSLsplit service. Default values saved in the configuration script are the following.

```
my $DEFAULT_CIPHER = 'NULL:RC4:AES128:-DHE';
my $DEFAULT_PROXY = 'https 0.0.0.0 10443';
my $DEFAULT_LOG_DIR = '/var/log/sslapp';
my $DEFAULT_LOG_CONNECTION = '/var/log/sslapp/connection.log';
my $DEFAULT_CACERT = '/etc/netc/sslapp/ca.pem';
```

The most of 'set' actions only save parameters to the DataTree. The action *'set-cacert'* firstly verify if the parameter contains an absolute or relative path and save the absolute path to the DataTree. The absolute path is created by concatenating default absolute path (according to context) with a given relative path. The same principle is used while setting log files. Setting a proxy type is a bit different. Within a proxy setting, there are two parameters, IPv4 or IPv6 address and port number. Due to those specific parameters, a validation has to be done. For check of IP address is used regular expressions for both IPv4 and IPv6 addresses. The port number is checked for positive numbers ending at 65535 because the value is stored in a 16-bit TCP field. All 'unset' actions only remove the value from the DataTree. In the following snippet, there is a part of a function to set a log type with a path. The snippet ensures enabling logging within sslapp if any other log was not set.

```
# if logging was never used - enable it
if ( !$DT->Exists('log') ) {
    $DT->Set('log', 1);
}
```

All 'get' actions using the same function which reads data from the configuration file and returns the right value. The function reads the configuration file and looks for a specified environment variable.

Actions starting on *'getall'* using system commands to get all possible values which could be set within the element. The cipher element prints all possible Cipher suites[7]. Element mirror prints all possible interfaces which could be chosen as an outgoing interface. Further, there are commands to print all possible IPv4 addresses including address 0.0.0.0 which stands for all possible addresses (SSLsplit receive traffic on the specified port regardless

[7] OpenSSL cipher suites – https://www.openssl.org/docs/man1.0.2/apps/ciphers.html

incoming interface) and all possible IPv6 addresses. The following system call returns all used IPv4 addresses which are used as 'getall' result.

```
system("ip -o address show | awk '{if (\$3 == \"inet\") print \$4}' |
    ↪ sed -e 's;/[[:digit:]]\\{1,3\\}\$;;'");
```

Actions 'show' are very similar to actions 'unset' in their simplicity. In the case of a CA, certificate or cipher element is first checked if the value is in the DataTree and if not, the default value is printed. The setting of a log is shown gradually, firstly is printed if logging is enabled or not and then individual types one by one. The only log type which has a default value is a connection type. The proxy setting is printed similar to log with only one exception, it is necessary to determine if any proxy type is set. If no proxy type is set and even if the whole proxy is unset, the default proxy setting is printed. Proxy type is printed as a list with each pair of IP address and the port number on the separated line. The action *'show-all'* calls all possible 'show' commands and thus print all 'show' sections. In the snippet below there is a part of the *'show-proxy-type'* function which contains get subnode 'type' of the element 'proxy' and then if there are any values print first a name of the type and in a foreach cycle, each iteration gets port number and prints both IP address and port number values. With regard to the type of IP address, a different output is displayed.

```
my @values = $DT->Keys('proxy', $type);

if ( @values ) {
    printf "%s:\n", $type;

    foreach my $address (@values) {
        my $port = $DT->GetLast('proxy', $type, $address);

        if (index($address, ':') != -1) {
            printf "%4s[%s]:%s\n", "", $address, $port;
        } else {
            printf "%4s%s:%s\n", "", $address, $port;
        }
    }
}
```

When the action 'start' is performed the first step is to build the configuration file. At first, the proxy variable is set. If the proxy element is unset the default value is used. The same situation occurs when the proxy element is set but none proxy type is set. If any proxy type is within the DataTree, all values are concatenated into the proxy variable. Proxy types *'tcp'* and *'ssl'* are similar to *'http'* and *'https'* except they do not modify HTTP (2.1.1) headers. At second, the log variable is set. The situation is very similar to the proxy element. All log types with their parameters are concatenated into the log variable. If the connection log is not set and even if the log is disabled the default value for connection log is used. Connection logging is always active due to statistics (more description in section 4.7). Cipher and CA certificate elements are put in their variables.

For both elements exist default values which are used when none value is set in the DataTree. Other variable contains all other SSLsplit (3.2) parameters which are set through netc. For example, there could be a URL to the CRL (2.2.2) server. The last variable mirror is filled with a preconfigured outgoing interface and target IPv4 address.

After the configuration file is successfully build, check whether the application is running is performed. If the SSLsplit is not running, the system starts the service. Then the script waits for one second and checks whether the application successfully started and run. In case of any failure, an error message is printed. Check whether the service is running is performed by executing the following system call.

```
system("systemctl | grep sslsplit.service | grep running &>/dev/null")
```

4.6.1 Configuration File

The configuration file 'sslapp.conf' has a predetermined location in '/etc/netc/sslapp/' directory. This file consists of six environmental variables. Variable PROXY contains all settings regarding proxy. For a successful start of SSLsplit process have to be set at least one proxy type with IP address and port number. There could be more proxies separated by space. Each proxy has to have a type, IPv4 or IPv6 address and port number. Another variable LOG contains all types of log options. There have to be a parameter `"-l"` with a path to a file, which is source file for statistics, within the LOG variable. Mandatory parameters `"-c"` with a path to the certificate and `"-s"` with ciphering suite are included in variables CA_CERT and CIPHER, respectively. Variables OTHER and MIRROR are not necessary. These variables only add extra features to SSLsplit process. The parameter `"-O"` denies all OCSP requests (2.2.3) within OTHER variable whereas variable MIRROR contains parameters `"-I"` and `"-T"` to determine where traffic is mirrored. Example of a configuration file is depicted below.

```
# config file created by netc
PROXY = "https 0.0.0.0 10443"
LOG = "-l /var/www/sslapp/connection.log"
CA_CERT = "-c /etc/netc/sslapp/ca.pem"
CIPHER = "-s NULL:RC4:AES128:-DHE"
OTHER = "-O "
MIRROR = "-I ve3 -T 192.168.2.2"
```

4.7 Statistics Script

Perl script file 'statsd-sslapp' containing functions to get statistics from SSLsplit. The file is placed in 'stats.d' directory. The script execution interval is set to sixty seconds in a statistics configuration file within netc. As a result of the setting could be shown all SSLsplit statistics per minute.

Statistics script works as well as the configuration script with the DataTree. The script opens configuration file for reading and finds within LOG environmental variable a parameter `"-l"` which is followed by a path to the log file. The file for logging information about connections is always available even when a user did not configure it – it is set by default in

that case. Statistics are saved into *'sslapp'* node which is contained within the pre-created node *'stats'*.

```
my $DT = NetC::DataTree->new('stats', 'sslapp');
```

When the name of the log file is known, it is split into two parts. The first part contains the name of the file without extension and the second contains only the file extension. The next step is to get PID of SSLsplit process. When the script got all information, the log file could be renamed to a file with the name composed of a timestamp in between two previously mentioned parts (further called stats file). This act ensures that any other log could not be added to the log file. Immediately after renaming a signal (SIGUSR1) is sent to SSLsplit process which causes reopening log file and the process continues on logging. When the signal is sent, the script continues to parse the stats file. In the snippet below there could be seen renaming of connection log file followed by sending the signal to SSLsplit process. Before those actions, a date is obtained in a specific format which is used as a name for a new statistics file.

```
my $datestring = strftime "%Y-%m-%d-%H-%M-%S", localtime;

rename $LOG_file, $LOG_file_name.'-'.$datestring.$LOG_file_ext;
kill 'SIGUSR1', $PID_SSLsplit;
```

Statistics are gathered for all types of connection within SSLsplit, namely HTTP, HTTPS, SSL, TCP, AutoSSL and all other communications as 'pass-through'. Statistics for these types count the number of connections carried through SSLsplit process. Other statistics are gathered only for HTTP (2.1.1) and HTTPS (2.1.2). This other type of statistics includes counting volumes of communication passing through SSLsplit process.

When all lines of stats file are parsed the next step is to save these data to the DataTree. The first action is to get the last value by type of communication, then add value from current stats file and save back to the DataTree. If the value is not present in the DataTree the initial value is zero and value from current stats file is added and saved.

4.8 Certificate

In order to successfully use SSLsplit (3.2) as an SSL (2.1.3) connection terminating and recreating tool, there is a need to create a private key. Next is needed to create a SS certificate (2.2.5) from the private key which is essential for smooth running SSLsplit application. SSLsplit creates any server's certificate which the client wants to connect to. Then the created certificate is signed by own CA certificate (2.2.4). By the command on the first line of the following snippet is generated 2048 bit RSA key. The other line cause creating the SS certificate from the key with attributes validity and serial code. Validity is restricted to one year from now and serial code of the certificate is set to `666`. By the command on the last line is created PEM[8] format file which contains the concatenated certificate and the key files.

[8] PEM file format – https://tools.ietf.org/html/rfc1421

```
openssl genrsa -out ca_cert.key 2048
openssl req -new -x509 -days 365 -key ca_cert.key -out ca_cert.crt
    ↪ -set_serial 666
cat ca_cert.crt ca_cert.key > ca_cert.pem
```

For comfortable creation of the certificate is created a bash script which could be executed as is or could accept a parameter which determines the key, certificate and PEM file names. The script facilitates the creation of the certificate by its preset values submitted as parameters to `openssl`. Only the name of files is configurable from outside as a parameter. The script uses similar commands as are shown above. Name of the script is 'create-cert.sh'. It is recommended to change the certificate information (described in section 2.2) in the script as needed. In the script could be changed also the size of RSA private key, number of days of validity and general information about certificate subject and issuer. The path to the target destination of the files is created and the non-existence of the key file is verified.

Executing the script or the commands generate files of three types. The first generated file is 'ca_cert.key' which contains private key in Base64[9] encoding. Another file encoded in Base64 is 'ca_cert.crt' which contains a single certificate. The last generated file is 'ca_cert.pem' containing the key concatenated with the certificate. The file is encoded in improved Base64 encoding called PEM format.

[9] Base64 – https://cs.wikipedia.org/wiki/Base64

Chapter 5

Performance Testing

Performance test is such a type of tests where the server is loaded with plenty of requests for a period of time and try to achieve maximum utilisation of resources on the server side. It is not about finding software bugs or defects. Many types of performance tests exist but only Load testing is used in the thesis. Load testing measures system performance as the workload increases. There are several tools and scripts focused on performing performance tests from single-use scripts to very complex applications.

This chapter describes how SSLsplit was tested, which additional tools and applications were used and the results of measurements. Tests were performed firstly in the virtual environment (5.1) to basically test SSLsplit and its capabilities and subsequently on physical machines (5.2) to utilise all available resources.

5.1 Virtual Environment

Virtualization is useful, for example, for testing some new features or settings before buying new technologies, modelling new planned topology and much more. Virtualization has many advantages like the easier implementation of new servers, effortless backups and migrations and so on. Like all roads, the virtualization has its pitfalls. The disadvantage, paradoxically, results from advantages. In a situation where are more servers and applications consolidated on one physical hardware and this hardware fails, successive breakdown means immediate failure the entire infrastructure.

A virtual environment is ideal for developing. It is able to remotely shut down, reboot and add network interfaces. Virtualization in this thesis is used mainly for easier development and testing without the need for time-consuming cabling and customising new servers (which would be unnecessary reserved for one use only).

5.1.1 Testing in a Virtual Environment

Testing was done using three separate machines connected together with 10 Gb links. The topology and whole testing process are almost the same as in the tools' performance testing section (3.5.4). The only difference is that there have to be one more computer acts as a receiver of mirrored packets. As could be seen in Figure 5.1, there is a machine with the transparent proxy in the middle which is operated by SSLsplit application. Wrk (described in section 3.5.2) was used to generate traffic from the client machine. All traffic was directed to the server IP address more accurately to Nginx server running on the server. The traffic went through the transparent proxy where SSLsplit processes it and sent to the server. In

case of testing with the mirroring feature, the target of the traffic was the server. There was also tested plain HTTP and HTTPS transfers without using SSLsplit application. The test was performed in a way that SSLsplit application was terminated and the machine in the centre acts as a normal router. VMware was used as a virtualization technology with its server virtualization platform named vSphere.

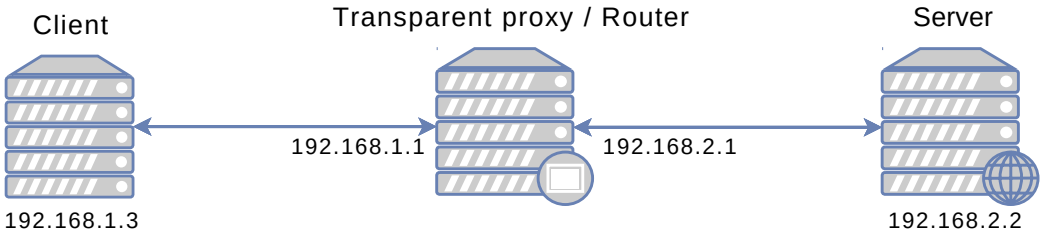

Figure 5.1: The topology used for performance testing in a virtual environment

Results

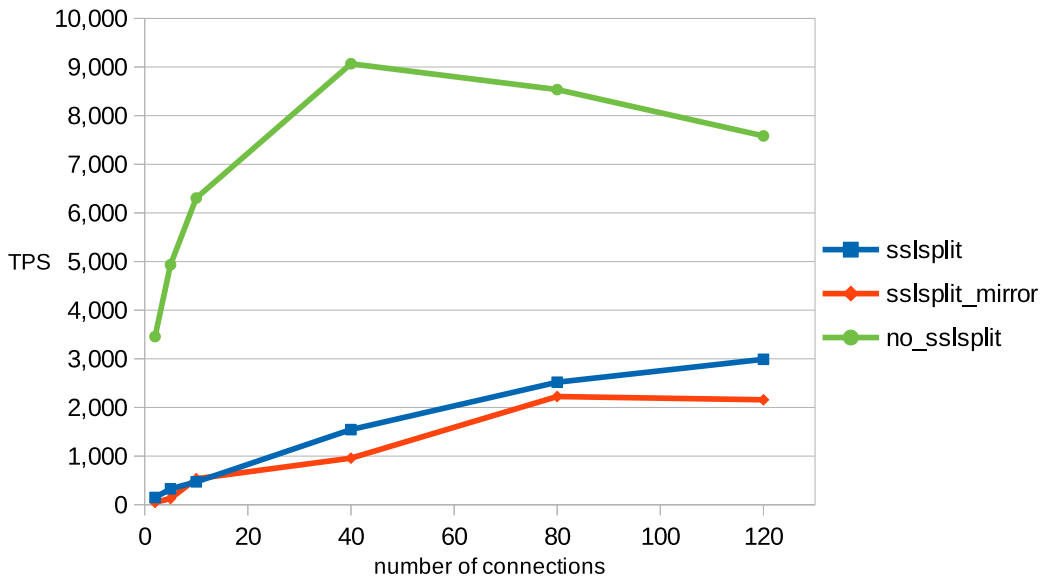

Figure 5.2: HTTP – Transactions per second

Figure 5.2 shows a number of transactions per second (TPS) while using HTTP queries in relation to a number of TCP connections. TPS of HTTP queries without using SSLsplit is much higher than with using SSLsplit regardless of using mirroring. SSLsplit process each HTTP query and parse it to get individual information to log into a file. The number of TPS while using only forwarded HTTP queries in comparison to HTTP queries passing through SSLsplit is four times larger. There is a trend depicting the number of queries which is spoiled by testing in a virtual environment as a decrease of TPS while increasing the connection number. The same decrease would occur with using SSLsplit while increasing the number of connections for the reason was selected an appropriate number of connections.

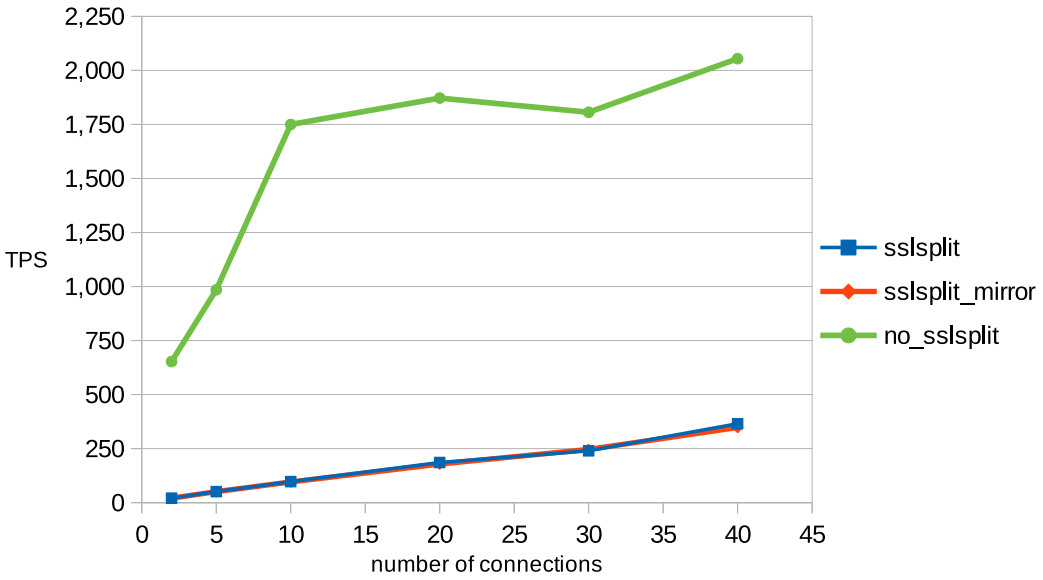

Figure 5.3: HTTPS – Transactions per second

Figure 5.3 shows a similar comparison as in Figure 5.2 with a difference in the used protocol as HTTPS was used in this case. Curves which depict SSLsplit with and without mirror are almost equal – they overlap each other in the figure. In the figure, there could be seen the same trend as is in Figure 5.2, however, no rapid decrease has appeared but the line showing measured HTTPS without using SSLsplit is mostly straight starting at ten TCP connections. There is a significant difference between sending HTTPS requests using SSLsplit and without it. The difference is quantified as multiple of five and is caused presumably by terminating the connection and creating the new connection to the originally intended server.

Figure 5.4 depicts CPU load while HTTP protocol used eighty opened TCP connections. Measurement was performed in a thirty-second long interval and due to measurement inaccuracy at the beginning and end, the result was shortened by five seconds on each side. In the figure, there could be seen a high CPU load while using SSLsplit in comparison to communication without the SSLsplit. There is nearly five times higher CPU load which is caused by SSLsplit parsing mechanisms mentioned in this chapter.

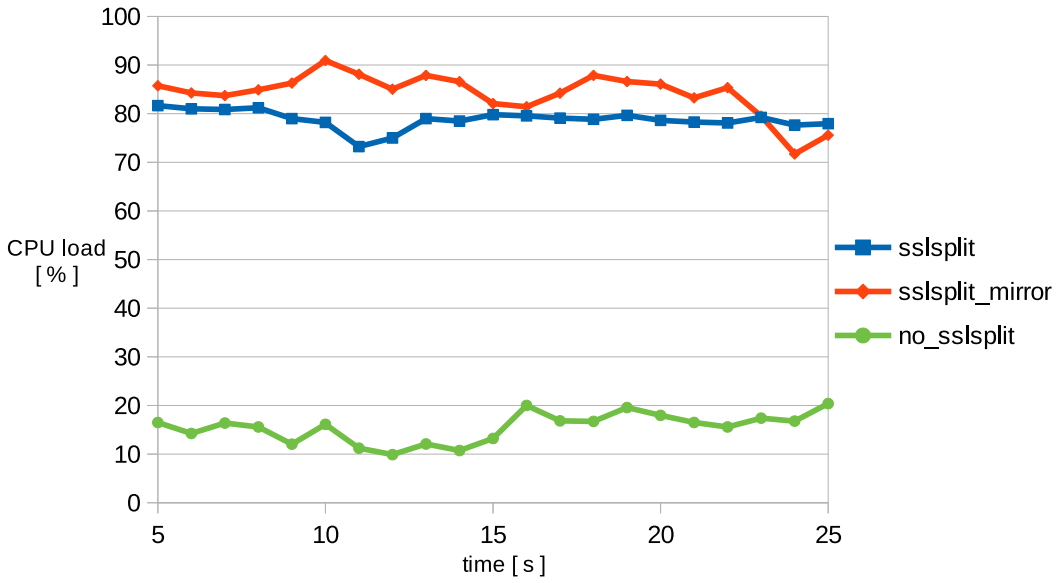

Figure 5.4: HTTP – 80 TCP connections

A similar comparison as in Figure 5.4 is shown in Figure 5.5 with a difference in the protocol used. HTTPS was used in this case. In the figure, there could be observed almost the same CPU load in the case of test with SSLsplit and without it. There is almost none difference in comparison of SSLsplit with active traffic mirroring and with disabled mirroring feature. The main areas of the thesis are an evaluation of results of an HTTPS traffic passing through SSLsplit and measurement of CPU load difference during traffic being mirrored and without that feature.

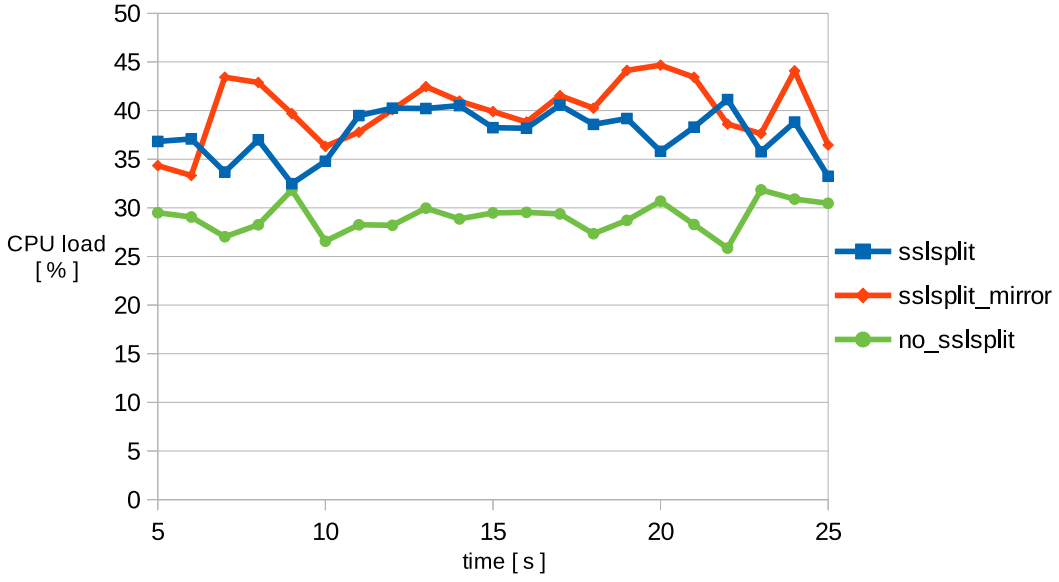

Figure 5.5: HTTPS – 30 TCP connections

Evaluation

The main goal of testing in a virtual environment was a comparison of SSLsplit (3.2) with and without enabled traffic mirroring feature depending on the number of connections. Looking at the figures, it could be said that the effect of turning on SSLsplit's traffic mirroring feature is not significantly reducing the number of TPS nor increasing CPU load. SSLsplit is quite restrictive compared to traffic that is not processed by SSLsplit due to the processing of each connection.

5.2 Physical Environment

A physical environment has some advantages over a virtual. Physical server has its dedicated hardware and only it uses all resources. Testing on physical machines has to be performed in order to have applicable and relevant results. Performance testing is necessary to understand the limitations of the tool being tested and to find out the tool's maximum possible utilisation. As was written in the introduction of the chapter, Load testing was performed.

5.2.1 Testing in a Physical Environment

The physical topology is almost identical to virtual topology (5.1). There could be seen three machines with the only difference in the addressing scheme in Figure 5.6. Different machines with different hardware components are used compared to the virtual environment. Figure 5.6 shows the router acting as the transparent proxy which is operated by SSLsplit application, client machine which runs benchmark tests and server machine where runs Nginx.

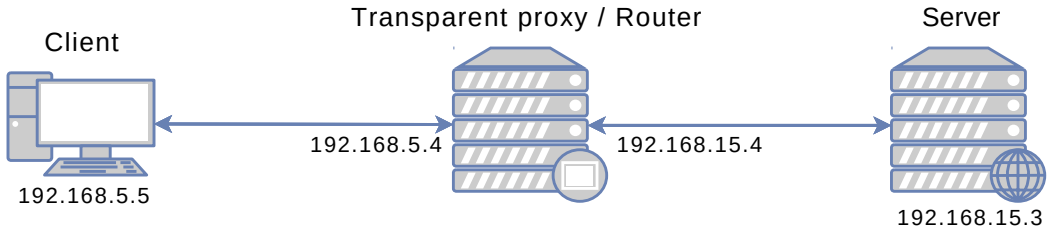

Figure 5.6: The topology used for performance testing in a physical environment

All machines had the same *x86-64* architecture and other configuration listed in Table 5.1. Connections between machines were 10 GE. Benchmark tool Wrk (described in section 3.5.2) predominantly run on the client machine and *taskset* command and *wget* tool were also used for testing.

	OS	Kernel	RAM	CPU	Disc
Client	CentOS 7	Linux 3.10.0	8 GB	i3-4160 @ 3.80GHz	SSD
Router	NetXOS 7.6.1810	Linux 3.10.0	16 GB	D-1537 @ 4.00GHz	SSD
Server	NetXOS 7.5.1804	Linux 4.20.10	16 GB	D-1587 @ 4.00GHz	SSD

Table 5.1: Software and hardware configuration of the machines

Linux Tuning

To be able to achieve higher TPS with using SSLsplit the *'File Descriptor Limit'* on the router have to be set to value at least 4,096. Very important is to enlarge a range of ports used for outgoing TCP connections because the tool needs open many new outgoing TCP connections (in tests the range was set from 10,000 to 64,999). Another advice is to lower time before TCP/IP can release a closed connection and reuse its resources (during tests the value was set to 5). All mentioned values could be set via 'sysctl' command.

Running Tests

Description of parameters of Wrk tool is in section 3.5.2. The parameter `"-t"` set to value `"4"` starts the Wrk on four threads. The following command was used to test transactions per second (TPS):

```
./wrk -t 4 -c 250 -d 30s -H 'Connection: close'
    ↪ https://192.168.15.3/0kb.bin
```

The following code was used to generate a high load on the link to measure throughput. A `for`-loop is there in order to set a correct CPU affinity in `taskset` command. A loop is executed four times – for each CPU core once. Within each iteration is executed a command `"taskset"` with a parameter `"-c"` set to loop variable value. The command `taskset` assigns the selected CPU core to the process. `Wget` tool is for retrieving files using HTTP, HTTPS, FTP and FTPS. The tool does not support parallelization. `Taskset` assigns one process to CPU core so each core runs one `wget` process for maximum utilisation. `Wget` has many parameters but in this case, only two was used. The first one is the URL of the target file and the second is a parameter `"-O"` with 'path' which sets the path where the downloaded file will be saved. At the very end, there have to be a character `"&"` which causes the start of the command in the background.

```
for t in `seq 0 3`; do
    taskset -c $t wget https://192.168.15.3/10gb.bin -O /dev/null &
done
```

A number of received bytes saved in the file are a great source for verifying throughput. A bash script was used to get a number of received bytes for a ten-second interval. The script on the beginning read the value from the following file, then waits for ten seconds and read the value again. Difference between the values is ten times higher than needed in order to eliminate deviations and measurement errors. The resulting number has to be divided by ten to get throughput per second. The script run on the client's machine and received bytes was evaluated. The file resides on the following path containing an actual number of received bytes by the interface named 'enp1s0f0' on which the measurement was performed.

```
/sys/class/net/enp1s0f0/statistics/rx_bytes
```

On the server, there have to be created a ten-gigabyte file in order to have large enough file to test throughput on a 10 GE link. To create a file of ten-gigabyte size filled with all zeroes was used the following command.

```
dd if=/dev/zero of=10gb.bin bs=1GB count=10
```

Results

Figure 5.7 shows a number of transactions per second (TPS) during HTTPS communication between the client and the server in relation to a number of connections. Achieved TPS is slightly above 16,000 according to this measurement. Wrk was used to generate requests and subsequent evaluation of TPS. According to preliminary tests results, a maximum of 350 open TCP connections was selected. The difference between communication processed by SSLsplit and only forwarded communication is quite striking. In the case of 40 TCP connections, the number of TPS while using SSLsplit is sixteen times lower. The difference between SSLsplit with enabled mirroring feature and SSLsplit without that feature is very small. The first measured number of TCP connections was ten and the second one was forty. The curve showing a number of TPS in case of using only forwarded traffic (without SSLsplit) shows a big step between these two measurements. Continued growth would occur if there were no system limitations which hold the maximum of TPS on value about 16,000.

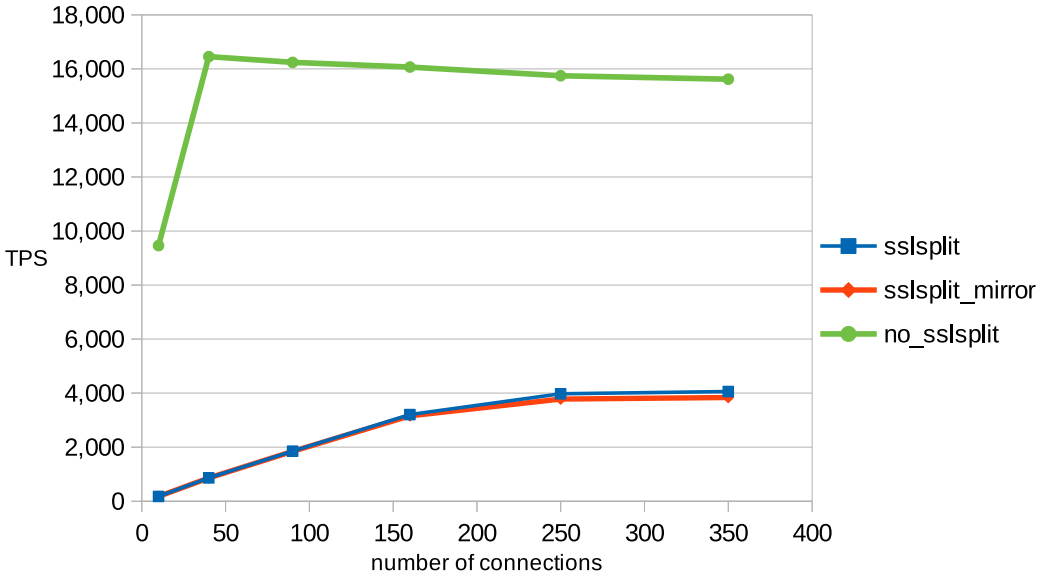

Figure 5.7: HTTPS – Transactions per second

Figure 5.8 shows the CPU load over time in HTTPS communication. Measurement was performed in a thirty-second long interval and due to measurement inaccuracy at the beginning and end, the result was shortened by five seconds on each side. This shortening does not have any effect on results. Wrk generated traffic uses 160 TCP connections and CPU load was almost the same all the time. The difference in CPU load between SSLsplit with and without enabled mirroring feature is about seven percent. In the case of forwarded only traffic, CPU load is so small because the router does not have to 'send' traffic to a user-space.

In Table 5.2, there are only two values which depict throughput in the network. The higher value is for communication without any processing on the router – traffic is only forwarded to the server. And the other one is for a state when communication is on

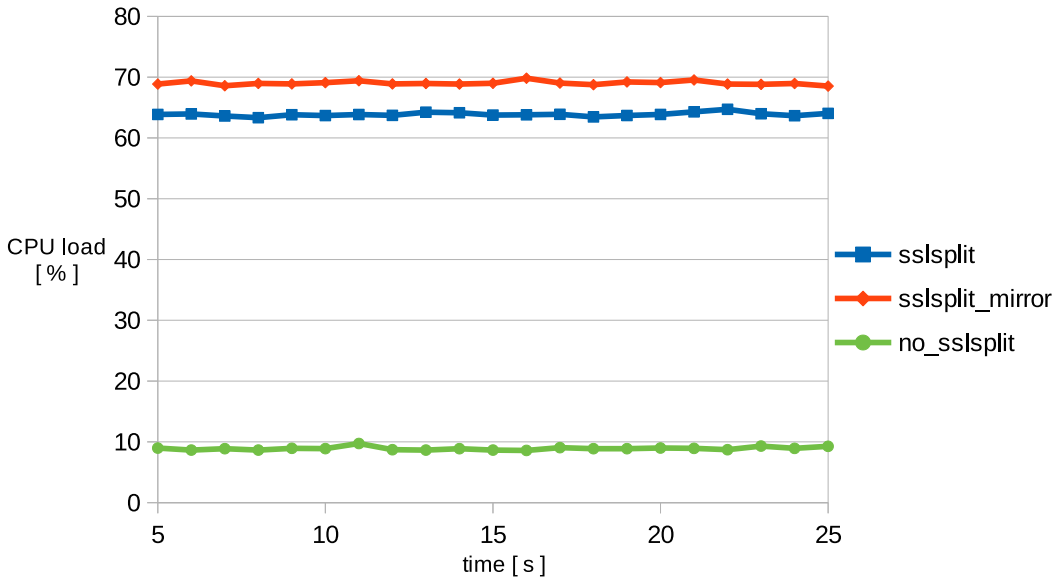

Figure 5.8: HTTPS – 160 TCP connections

		'forwarded'	**SSLsplit**
Throughput	[Mbps]	9,934.91	9,795.84

Table 5.2: Comparison of throughput only forwarded and processed by SSLsplit

the router processed by SSLsplit. Those two values could be compared and the drop in throughput could be evaluated. In this case, the decrease is 1.4 %.

Evaluation

Tests of HTTP communication were not successful on physical machines. SSLsplit from time to time crashed during high load and the reason was not found nor fixed.

The main goal of the Load testing on physical machines was to find limitations and a maximum number of TPS of SSLsplit (3.2) with and without enabled traffic mirroring feature. From the results depicted by figures could be said that the effect of turning on SSLsplit's traffic mirroring feature is not significantly reducing the number of TPS nor increasing CPU load. Getting higher numbers of TPS was restricted by CPU. On the other hand, SSLsplit performed very well in the throughput test. When SSLsplit began processing traffic, the drop in throughput was only less than two percent.

Chapter 6

Conclusion

The main goal of the thesis is to implement an application, which is capable to mirror decrypted traffic to another node within a network. The node has to be known prior to launch the tool in order to ensure proper traffic mirroring.

Chapter 2 contains a description of some protocols, digital certificates, proxies and some network attacks and protection against them. Firstly there is described an insecure protocol HTTP (2.1.1) and then a protocol HTTPS (2.1.2) which is secured by the transport layer protocol TLS (2.1.3). Next in the chapter is described a digital certificate (further only certificate) with its structure (2.2.1). There are also sections about certificate revocations, certificate validation level (2.2.6), self-signed certificate (2.2.5), a certification authority (2.2.4) with an explanation of root certificate, certificate chain and extension fields of a certificate. Another big section is proxy (2.3) which contains a description of the individual proxy types with a focus on HTTP proxy (2.3.1), SSL proxy (2.3.2) and Socks proxy (2.3.3). The last section of the chapter covers attacks and protections against them (2.4). As a representative of the attacks, there is described Man-in-the-Middle (MitM) attack (2.4.3) and one of its type named SSL stripping. In the chapter, there are also described two representatives of protection against MitM attack.

In Chapter 3 are compared and described tests of penetration tools which are capable of MitM attack. The chapter also contains examples of use and a basic description of each individual tool. One of the main parts of the chapter is a comparison of tools' features (3.4) which describes the advantages and disadvantages of tools and provides a summary Table 3.1. The second main part is the tools' benchmarking (3.5) in a virtual environment. As a benchmarking tool on the client's machine was used Wrk (3.5.2), on the server's machine run Nginx (3.5.1) web server and for measuring CPU load was used sar (3.5.3). In the transaction per second test was found that tool Sslsniff (3.3) crashes during the test and from Figure 3.2 was read that SSLsplit (3.2) has lesser CPU requirements than Mitmproxy (3.1). In the speed of re-encryption test, the best results were achieved by SSLsplit using which was the download fastest (Table 3.3). According to the features' comparison, the language used to develop and performed benchmark was selected SSLsplit. SSLsplit has all needed and advisable features. It is written in fast, compiled ANSI C and its development still continues.

Description of all created files and procedures is in Chapter 4. In the chapter, there is described integration of the tool to the NETX (4.1) router to be fully operable within NETX's netc (4.1.1) interface. Firstly, there is presented NETX platform with its command line interface called netc followed by a description of YAML – descriptive file (4.4) defining a new configuration context within netc. In the descriptive file section, there are shown

fragments of the file with explanations. There are also described all important parts of the file including the DataTree (4.4.1) structure. Created service file (4.5) with explained certain parts and commands together with the configuration script (4.6) are present in the chapter. In the service file section is described the need for PID file and explicit specification of exit statuses. The configuration script section contains all essential default values which are part of the script. There are also described some extraordinary fragments of code and all functions listed in the descriptive file. The configuration file (4.6.1) generated by a configuration script with an example content of the file as well as the statistics script (4.7) taking care of generating statistics from connection log file are included in the chapter. The statistics script section contains the need for renaming and sending a signal to SSLsplit process in order to properly handle the connection log file. The chapter also includes script and commands for creation of a certificate (4.8) using OpenSSL and a description of created files including their encoding. The last section of the chapter is devoted to Linux forwarding a redirecting (4.3). There is written the need for enabling a forwarding and a proper redirection by using iptables utility.

All necessary performance testing is summarised in Chapter 5. Tests in a virtual environment showed a decrease in transactions per second (TPS) while using SSLsplit which is caused by the fact that SSLsplit processes each packet and thus it is much slower. Results from testing on physical machines showed a similar decrease in TPS but in higher values. Highly appreciated is a fact that the mirroring feature does not significantly slow down or restrict SSLsplit itself. The whole testing proved SSLsplit to be useful and usable for one client or a small network.

Bibliography

[1] Cooper, D.; Santesson, S.; Farrell, S.; et al.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280. RFC Editor. May 2008. [Accessed 2019-05-01].
Retrieved from: http://www.rfc-editor.org/rfc/rfc5280.txt

[2] Cortesi, A.; Hils, M.; Kriechbaumer, T.; et al.: mitmproxy: A free and open source interactive HTTPS proxy. 2010–. [Accessed 2019-05-08].
Retrieved from: https://mitmproxy.org/

[3] Dierks, T.; Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246. RFC Editor. August 2008. [Accessed 2019-05-01].
Retrieved from: http://www.rfc-editor.org/rfc/rfc5246.txt

[4] Eastlake, D.: Transport Layer Security (TLS) Extensions: Extension Definitions. RFC 6066. RFC Editor. January 2011. [Accessed 2019-05-01].
Retrieved from: http://www.rfc-editor.org/rfc/rfc6066.txt

[5] Evans, C.; Palmer, C.; Sleevi, R.: Public Key Pinning Extension for HTTP. RFC 7469. RFC Editor. April 2015. [Accessed 2019-05-01].
Retrieved from: http://www.rfc-editor.org/rfc/rfc7469.txt

[6] Farrell, S.; Tschofenig, H.: Pervasive Monitoring Is an Attack. BCP 188. RFC Editor. May 2014. [Accessed 2019-05-01].
Retrieved from: http://www.rfc-editor.org/rfc/rfc7258.txt

[7] Fielding, R. T.; Gettys, J.; Mogul, J. C.; et al.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616. RFC Editor. June 1999. [Accessed 2019-05-01].
Retrieved from: http://www.rfc-editor.org/rfc/rfc2616.txt

[8] Francis, P.: Network Address Translation (NAT). *ACM SIGCOMM Computer Communication Review*. vol. 45, no. 2. 2015: pp. 50–50. ISSN 0146-4833.

[9] Handley, M.; and, E. R.: Internet Denial-of-Service Considerations. RFC 4732. RFC Editor. December 2006. [Accessed 2019-05-01].
Retrieved from: http://www.rfc-editor.org/rfc/rfc4732.txt

[10] Hodges, J.; Jackson, C.; Barth, A.: HTTP Strict Transport Security (HSTS). RFC 6797. RFC Editor. November 2012. [Accessed 2019-05-01].
Retrieved from: http://www.rfc-editor.org/rfc/rfc6797.txt

[11] Marlinspike, M.: SSLsniff. 2008–2011. [Accessed 2019-05-08].
Retrieved from: https://moxie.org/software/sslsniff/

[12] Nystrom, M.; Kaliski, B.: PKCS #10: Certification Request Syntax Specification Version 1.7. RFC 2986. RFC Editor. November 2000. [Accessed 2019-05-01].
Retrieved from: https://www.rfc-editor.org/rfc/rfc2986.txt

[13] Pettersen, Y.: The Transport Layer Security (TLS) Multiple Certificate Status Request Extension. RFC 6961. RFC Editor. June 2013. [Accessed 2019-05-01].
Retrieved from: http://www.rfc-editor.org/rfc/rfc6961.txt

[14] Postel, J.: User Datagram Protocol. STD 6. RFC Editor. August 1980. [Accessed 2019-05-01].
Retrieved from: http://www.rfc-editor.org/rfc/rfc768.txt

[15] Postel, J.: Transmission Control Protocol. STD 7. RFC Editor. September 1981. [Accessed 2019-05-01].
Retrieved from: http://www.rfc-editor.org/rfc/rfc793.txt

[16] Rescorla, E.: HTTP Over TLS. RFC 2818. RFC Editor. May 2000. [Accessed 2019-05-01].
Retrieved from: http://www.rfc-editor.org/rfc/rfc2818.txt

[17] Ristic, I.: *Bulletproof SSL and TLS*. London: Feisty Duck. 2014. ISBN 978–1907117046.

[18] Roethlisberger, D.: SSLsplit. 2009–. [Accessed 2019-05-08].
Retrieved from: https://www.roe.ch/SSLsplit

[19] Santesson, S.; Myers, M.; Ankney, R.; et al.: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 6960. RFC Editor. June 2013. [Accessed 2019-05-01].
Retrieved from: http://www.rfc-editor.org/rfc/rfc6960.txt

[20] Weber, W.: Firewall basics. In *4th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services. TELSIKS'99 (Cat. No.99EX365)*, vol. 1. IEEE Publishing. 1999. ISBN 0-7803-5768-X. pp. 300–305.

Appendix A

CD Content

The thesis contains a CD with the following content:

- src/ – source codes
  - create-cert.sh – script to create self-signed certificate
  - sslapp.yml – describing file to add new entry to netc
  - sslsplit.service – file for initiating SSLsplit application
  - statsd-sslapp – script to get statistics from SSLsplit
  - xcfg-netc-sslapp – script file with configuration functions
- data/ – data from tests
- latex-src/ – source files of the thesis in latex
- thesis.pdf – the thesis in pdf format