VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2021 Samuel Kopecký

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

AUTOMATICKÝ NÁSTROJ PRO KONTROLU ÚROVNĚ ZABEZPEČENÍ WEBOVÉHO SERVERU
AUTOMATIC TOOL FOR CHECKING THE LEVEL SECURITY OF A WEB SERVER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Samuel Kopecký
AUTHOR

VEDOUCÍ PRÁCE — Ing. David Smékal
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Samuel Kopecký ***ID:*** 211799
Ročník: 3 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Automatický nástroj pro kontrolu úrovně zabezpečení webového serveru

POKYNY PRO VYPRACOVÁNÍ:

Téma je zaměřeno na implementaci automatických nástrojů pro testování zranitelnosti webových serverů. Automatické testy budou analyzovat využívající protokol SSL/TLS a použité certifikáty na straně webového serveru. Test zjistí použitou kryptografickou sadu a vyhodnotí její parametry. Dále budou provedeny testy na vhodně zvolené zranitelnosti z databáze CVE (Common Vulnerabilities and Exposures).
Testy budou naprogramovány pomocí programovacího jazyka Python. Každý test poběží v samostatném vláknu a nástroj bude využívat přepínačů pro zvolení jednotlivých testů. Nástroj bude ovládán jak pomocí grafického uživatelského rozhraní (GUI), tak i přes konzolové rozhraní.
Cílem práce je popsat možnosti analýzy a naprogramovat automatický nástroj pro analýzu několika typů bezpečnostních zranitelností webového serveru.

DOPORUČENÁ LITERATURA:

[1] PECINOVSKÝ, Rudolf. Python: kompletní příručka jazyka pro verzi 3.9. Praha: Grada Publishing, 2020. Knihovna programátora (Grada). ISBN 978-80-271-1269-2.

[2] RISTIC, Ivan. Bulletproof SSL and TLS: Understanding and Deploying SSL/TLS and PKI to Secure Servers and Web Applications. Feisty Duck, 2013, ISBN: 978-1907117046.

Termín zadání: 1.2.2021 ***Termín odevzdání:*** 31.5.2021

Vedoucí práce: Ing. David Smékal
Konzultant: Roman Kümmel (HACKER Consulting s.r.o.)

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

V dnešnej dobe je bežnou praxou mať bezpečne hostovaný webový server. Pre zabezpečenie webového servera je potrebné zabezpečiť, aby bola triáda CIA (Dôvernosť, Integrita, Dostupnosť) na mieste. Triáda CIA je splnená použitím rôznych mechanizmov, ako je symetrická kryptografia a infraštruktúra verejných kľúčov. Protokol TLS (Transport layer security) umožňuje zaistenie triády CIA a jeho funkcia je popísaná v prvej časti tejto práce spolu so základmi kryptografie. V druhej časti je predstavený nástroj napísaný v programovacom jazyku Python, ktorý umožňuje automatizáciu analýzy bezpečnosti algoritmov a parametrov, ktoré TLS používa. Tento nástroj je tiež schopný analyzovať podporované verzie protokolu TLS webovým serverom. Ďalšou funkciou nástroja je skenovanie špecifickej implementácie webového servera. Umožňuje tiež analýzu niekoľkých vybraných zraniteľností, ktoré sú tiež opísané v druhej časti. Na konci práce je popísané testovacie prostredie, ktoré bolo vytvorené na otestovanie nástroja.

KĽÚČOVÉ SLOVÁ

Analýza, Automatizácia, Bezpečnosť, SSL, TLS, Webový server

ABSTRACT

Nowadays it is common practice to have a securely hosted web server. To have a secure web server it is required to have the CIA (Confidentiality, Integrity, Availability) triad is in place. CIA triad is fulfilled by using mechanisms like symmetric cryptography and public key infrastructure. Protocol TLS (Transport layer security) enables the provision of the CIA triad and its function is described in the first part of this thesis along with the basics of cryptography. In the second part a tool written in the programming language Python is presented, which enables automation of security analysis of algorithms and parameters used by TLS. The tool is also capable of analyzing supported versions of the TLS protocol by a web server. Another feature of the tool is scanning a specific web server implementation. It can also analyze a few chosen vulnerabilities which are also described in the second part. At the end of the thesis the testing environment which was created to test the tool is described.

KEYWORDS

Analysis, Automation, Security, SSL, TLS, Web Server

KOPECKÝ, Samuel. *Automatický nástroj pro kontrolu úrovně zabezpečení webového serveru*. Brno, 2021, 73 s. Bakalárska práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedúci práce: Ing. David Smékal

VYHLÁSENIE

Vyhlasujem, že svoju bakalársku prácu na tému „Automatický nástroj pro kontrolu úrovně zabezpečení webového serveru“ som vypracoval samostatne pod vedením vedúceho bakalárskej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej bakalárskej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto bakalárskej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora

POĎAKOVANIE

Rád by som sa poďakoval vedúcemu bakalárskej práce pánu Ing. Davidovi Smékalovi za odborné vedenie, konzultácie, trpezlivosť a podnetné návrhy k práci. Rád by som sa poďakoval aj odbornému konzultantovi pánovi Romanovi Kümmelovi za podnetné návrhy a konzultácie k práci.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov

Úvod

V dnešnej dobe je webový prenos jedným z najčastejších typov komunikácie na internete, preto je dôležité, aby bol tento prenos chránený pred hrozbami. TLS protokol umožňuje túto ochranu a využíva sa ako prostriedok ochrany na všetkých zabezpečených webových serveroch. Od prvého vzniku protokolu vzniklo mnoho vylepšených verzií. Každá nová verzia protokolu zlepšovala bezpečnosť webových serverov a opravovala chyby, ktoré sa vyskytovali v predošlých verziách. Preto je dôležité, aby sa využíval protokol najnovšej verzie.

Nástroj, ktorý popisuje táto práca má za úlohu poskytnúť hodnotenie bezpečnosti TLS protokolu a jeho použitých parametrov (cipher suite parametre), ktoré sa používajú na danom webovom serveri. Ďalej je schopný aj analyzovať serverom podporované verzie TLS protokolu a konkrétne implementácie webového serveru. V práci je taktiež popísaných niekoľko zraniteľností z databáze CVE (Common Vulnerabilities and Exposures), na ktoré je nástroj schopný otestovať webový server.

Práca má taktiež v teoretickej časti poskytnúť základný prehľad o tom, ako protokol TLS a bezpečnosť webových serverov funguje. Práca taktiež popisuje, aké typy kryptografických základov protokol využíva k zaisteniu bezpečnosti prenosu cez otvorený internet.

Nástroj je vyvinutý v programovacom jazyku Python. Python je veľmi dobre čitateľný a prehľadný, takže umožňuje ľahkú orientáciu v zdrojovom kóde nástroja. Python je taktiež multiplatformový, čiže je ho možné spustiť na väčšine operačných systémoch. Nástroj ďalej využíva jazyk .json na ukladanie dát. Json je podobný jazyku XML, čiže má rovnakú funkcionalitu. Jediný rozdiel je, že používa syntax z Javascrip jazyka čo odpovedá aj jeho skratke *JavaScript Object Notation*.

1 Teoretické základy

Táto kapitola sa zaoberá teoretickými základmi informačnej bezpečnosti a mechanizmami, pomocou ktorých sa zaisťuje.

1.1 OSI model

Základným modelom, podľa ktorého sa riadi celý internet a väčšina internetovej komunikácie, je model OSI (Open Systems Interconnection). Je to štandard, ktorý definuje protokoly a pravidlá, ktoré sa požívajú na sieťovú komunikáciu medzi sieťovými zariadeniami. Systém, ktorý implementuje túto protokolovú sadu sa nazýva protocol stack. Príklad protokol stacku je TCP\IP, ktorý je v dnešnej dobe implementovaný skoro na každom zariadení schopného sieťovej komunikácie.

OSI model vznikol v roku 1977 pod organizáciou ISO (International Organization for Standardization) [1]. Model sa skladá zo 7 vrstiev, ako je možné vidieť na obrázku 1.1. Prvá vrstva je Fyzická vrstva, ktorá je najbližšie k fyzickému médiu, cez ktoré sa prenášajú bity, naopak siedma vrstva nazývaná Aplikačná je najďalej od fyzického média a obsahuje najväčšie množstvo abstrakcie. Dáta, ktoré si vrstvy medzi sebou prenášajú, sa nazývajú PDU (Protocol Data Unit). Na každej vrstve sa PDU nazývajú inak, ako je zase možné vidieť na obrázku 1.1.

PDU	Názov vrstvy — číslo vrstvy	
Dáta	Aplikačná vrstva	7
Dáta	Prezentačná vrstva	6
Dáta	Relačná vrstva	5
Segmenty	Transportná vrstva	4
Pakety	Sieťová vrstva	3
Rámce	Spojová vrstva	2
Bity	Fyzická vrstva	1

Obr. 1.1: Vrstvy OSI modelu

Vrstvy medzi sebou komunikujú horizontálne, napríklad fyzická vrstva posiela prijaté bity spojovej vrstve ktorá, potom posiela rámce sieťovej vrstve a tak ďalej. Komunikácia taktiež prebieha vertikálne a to spôsobom takým, že transportná vrstva jednej stanice komunikuje s transportnou vrstvou druhej stanice.

1.2 Informačná bezpečnosť

Informačná bezpečnosť je pojem, ktorý je definovaný ako snaha chrániť informácie. Hlavným cieľom informačnej bezpečnosti je zabezpečiť takzvanú trojicu **CIA,** pričom každé písmeno znázorňuje jeden z hlavných účelov informačnej bezpečnosti [2]:

- **Confidentiality** (dôvernosť) – chránenie informácii pred neautorizovaným vstupom.
- **Integrity** (integrita) – chránenie informácií pred zmenou.
- **Availabillity** (dostupnosť) – dostupnosť informácií vždy, keď je to potrebné.

1.2.1 Ochrana sieťovej komunikácie

Pri vzniku Internetu sa nedávalo veľa dôrazu na integrovanie bezpečnosti do komunikačného modelu. Protokoly boli vytvorené s predpokladom dôvernosti a poctivosti. Tento predpoklad bol nesprávny, a preto musela vzniknúť určitá ochrana sieťovej komunikácie.

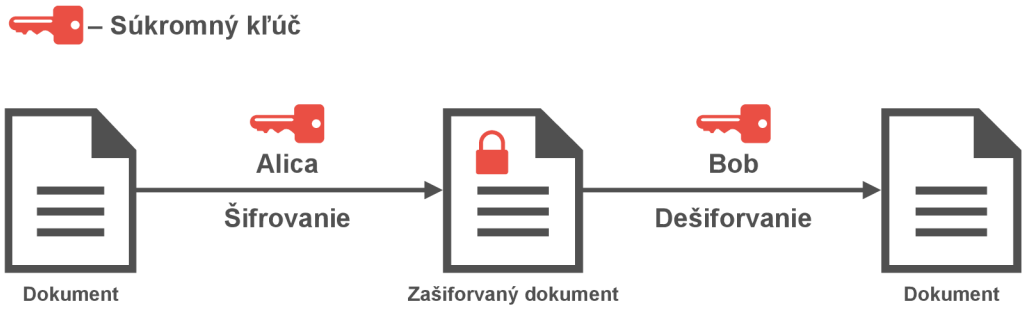
Preto vznikol protokol SSL (Secure Socket Layer) a neskoršie protokol TLS (Transport Layer Security). Tieto protokoly majú za úlohu zaistiť informačnú bezpečnosť. V praxi to znamená, že ak sa užívateľ chce spojiť so serverom, ktorý hostuje webovú stránku tak, si môže byť istý, že jeho prenos nebude odpočúvaný, menený a že dorazí tam, kde mal. Bezpečnosť ale nie je jediným cieľom týchto protokolov [3]:

- **Interoperability** (Interoperabilita) – možnosť kompatibility nových programov a knižníc.
- **Extensibility** (Rozšíriteľnosť) – nezávislosť na kryptografických základoch, možnosť výmeny týchto základov bez potreby rozsiahlej zmeny funkčnosti protokolov.
- **Efficiency** (Efektívnosť) – snaha dosiahnuť všetky ostatné ciele a požiadavky efektívne bez zbytočne veľkých nákladov.

1.3 Symetrická kryptografia

Symetrická kryptografia je metóda, ktorá umožňuje prenos zašifrovaných informácií medzi dvoma stranami s podmienkou že nikto iný bez znalosti súkromného kľúča nemôže zistiť obsah poslanej informácie. Obidve strany musia vlastniť pred začatím komunikácie spoločný privátny kľúč, ktorý sa potom používa na šifrovanie a dešifrovanie prenosu.

Keďže privátny kľúč nemôže byť posielaný v otvorenej forme, existujú algoritmy, ktoré umožňujú navzájom nazdielať tento privátny kľúč pred začatím šifrovanej komunikácie (sekcia 1.5.3).



Obr. 1.2: Princíp symetrickej kryptografie

Na obrázku 1.2 je možné vidieť proces symetrického šifrovania. Majme 2 užívateľov Alicu a Boba, ktorí si pred začatím komunikácie vygenerovali spoločný súkromný kľúč. Na sieti si chcú posielať navzájom tajné informácie. Tento prenos odpočúva zlá Eva. Alica zašifruje svoju správu súkromným kľúčom a pošle ju Bobovi. Eva odchytí túto správu ale, nemôže ju prečítať pretože bola zašifrovaná súkromným kľúčom. Keď Bob dostane zašifrovanú správu, je schopný ju dešifrovať súkromným kľúčom, ktorý si vopred dohodol s Alicou.

1.3.1 Kryptografické štandardy

Aby boli kryptografické algoritmy použiteľné a účinné, je potrebné aby všetci účastníci mali znalosť o tom, ako funguje daný algoritmus, na ktorom sa dohodli. Preto vznikli kryptografické štandardy. To ale znamená, že útočník tiež bude poznať, ako fungujú tieto algoritmy, na druhej strane, čím viac ľudí bude vedieť o danom algoritme, tým viac bude bezpečnejší, keďže bude viac testovaný a prepracovaný.

Znakom dobrého kryptografického štandardu je situácia, kedy útočník pozná všetko ohľadom použitého algoritmu okrem súkromného kľúča a aj tak nie je možné zistiť zatajenú informáciu [4].

Najznámejšie kryptografické štandardy a algoritmy:

1. **DES** (Data Encryption Standard) – štandardizovaný v roku 1977, využíva model Feistelovej siete, veľkosť kľúču 56 bitov, pracuje s blokmi veľkými 64 bitov. Tento protokol sa už v súčasnosti nevyužíva kvôli jeho nepostačujúcej bezpečnosti [5].
2. **TDES** (Tripple Data Encryption Standard)– veľmi podobný štandardu DES. Jediným rozdielom je, že algoritmus štandardu DES sa opakuje 3-krát pre každý blok správy pre zlepšenie bezpečnosti štandardu. Kľúč môže mať veľkosť 168, 112 alebo 56 bitov, čo vyplýva z opakovania algoritmu. Algoritmus sa taktiež už nevyužíva.[5]
3. **AES** (Advanced Encryption Standard) – v dnešnej dobe najpoužívanejší štandard symetrickej kryptografie, založený na šifre Rijndael[1], možnosť použitia kľúča v 3 módoch 128 bit, 192 bit a 256 bit, v každom móde pracuje s blokmi veľkosti 128 bitov, dodnes je šifra neprelomená a považuje sa za bezpečnú. Šifra sa v dnešnej dobe využíva skoro v každom systéme, ktorý zabezpečuje dôveryhodnosť [6].
4. **ChaCha20** – prúdová šifra ktorá je podla dokumentu RFC 7539 [7] priemerne 3-krát rýchlejšia ako Rijndael algoritmus. Využíva 256 bitový kľúč a 96 bitový inicializačný vektor. Šifra sa taktiež považuje za neprelomenú a využíva sa spolu s algoritmom Poly1305 v niektorých kryptografických sadách protokolu TLS 1.3. Viacej o protokole TLS bude nasledovať v sekcii 2.

1.3.2 Typy operácie symetrických šifier

Existujú 2 typy operácií symetrických šifier. Prúdová šifra (Stream Cipher) a bloková šifra (Block Cipher).

Prúdová šifra funguje na princípe logickej operácie XOR. Jednotlivé bity správy a bity keystreamu sa skombinujú pomocou operácie XOR. Keystream je reťazec bitov, ktorý je derivovaný zo zdieľaného privátneho kľúča a má rovnakú bitovú dĺžku ako plaintext[2], ktorý sa šifruje. Pri používaní prúdovej šifry je potrebné generovať unikátny keystream pre každú novú správu, pretože v správe sa môžu opakovať sekcie ako napríklad hlavičky HTTP požiadaviek, z ktorých je potom útočník schopný zistiť časť keystreamu.

Bloková šifra rozdelí jednu správu na niekoľko blokov, ktoré potom samostatne šifruje pomocou súkromného kľúča. Spôsob, akým sa bloky šifrujú, závisí na zvolenej šifre. Keďže je veľmi malá pravdepodobnosť, že plaintext bude mať dĺžku, ktorá je deliteľná veľkosti blokov bez zvyšku, tak je nutné použiť operáciu nazývanú

[1] kombinácia mien vynálezcov *Joan Daemen* a *Vincet Rijmen*

[2] reťazec bitov z ktorých sa skladá prenášaná správa

padding. Padding je operácia, ktorá doplní chýbajúce bity do posledných blokov správy, ktoré sú menšie ako zadefinovaná veľkosť bloku pri šifrovaní. Aké bity sa doplnia, zase záleží na zvolenej metóde paddingu. Jednou nevýhodou blokových šifier je predvídateľnosť, pretože výstup rovnakých blokov bude vždy rovnaký pri použití rovnakého kľúča. Túto nevýhodu riešia módy blokových šifier:

- **Counter Mode** – využíva počítadlo, ktoré sa inkrementuje po každom bloku, ktorý slúži ako vstupný blok do algortimu. Plaintext sa ďalej XORuje s výstupom tohto algoritmu.
- **CBC** (Cipher Block Chaining) – každý blok sa XORuje s predošlým ciphertextom[3]. Pri prvom bloku ako vstup sa používa IV (Inicialization Vector). IV je náhodný reťazec bitov a má rovnakú dĺžku ako veľkosť bloku algoritmu.
- **CFB** (Cipher FeedBack) – podobný princíp ako CBC. Využíva ciphertext minulého bloku ako vstup do ďalšieho bloku. Plaintext sa XORuje s výstupom algoritmu.
- **GCM** (Galois/Counter Mode) – modifikácia módu Counter mode ktorá, zaisťuje autentifikáciu (overenie identity).

1.4 Kontrola integrity informácií

Aj keď je možné zabezpečiť symetrickou kryptografiou, kedy útočník nebude čítať obsah správy, útočník stále vie zmeniť zašifrovaný obsah komunikácie. Preto je nutné pre zaistenie integrity použiť hašovacie funkcie. Hašovacia funkcia je algoritmus, ktorý vytvorí z ľubovoľne dlhej správy odtlačok (message digest alebo hash) s fixnou veľkosťou. Tento odtlačok môže mať rôzne velikosti. Ako algoritmus vytvorí tento odtlačok, už záleží na konkrétnom algoritme viď sekcia 1.4.2.

V kryptografii sa na overovanie integrity používajú Kryptografické hašovacie funckie, ktoré musia mať nasledovné vlastnosti:

- **Jednocestnosť** (one-way) – odtlačok zo správy sa dá jednoducho vypočítať, z hashu sa však nedá naspäť vypočítať originálnu zprávu.
- **Bezkolíznosť** (collision-free) – nie je možné nájsť 2 rôzne odtlačky, ktoré patria dvom rôznym správam.

Hašovacie funkcie sú tiež užitočné na porovnávanie veľkých súborov. Keby vznikla potreba porovnávať 2 veľké súbory, ktoré majú 2 GB, tak by to trvalo oveľa dlhšie ako sa vytvorili odtlačky z týchto súborov a porovnali len vytvorené odtlačky.

[3] zašifrovaný plaintext

1.4.1 MAC

Problém so zaisťovaním integrity počas komunikácie je ten, že ak by zlá Eva chcela zmeniť obsah zašifrovanej správy, tak stále vie vypočítať nový odtlačok z upravenej správy a poslať spolu so správou upravený odtlačok. Nezistí síce obsah správy, ale obsah správy vie stále upraviť.

Tento problém vyrieši mechanizmus MAC (Messege Authentication Code). MAC je kryptografický algoritmus, ktorý zaisťuje integritu aj autentifikáciu správy. Funguje podobne ako hašovacia funkcia. Keď sa vytvára odtlačok zo zašifrovanej správy, pridá k nej aj súkromný kľúč, ktorý je zdieľaný medzi komunikujúcimi stranami. Týmto je docielené to, že keď zlá Eva bude chcieť upraviť správu, tak nedokáže z upravenej správy vytvoriť odtlačok ktorý by patril k upravenej správe bez znalosti súkromného kľúča.

MAC algoritmy, ktoré využívajú kryptografickéý odtlačok funkcie, sa nazývajú HMAC (Hashed-based Message Authentication Code) [8]. Tieto algoritmy sa potom označujú podľa toho, aký odtlačok algoritmus MAC algoritmus používa, napríklad:

- **HMAC-MD5**
- **HMAC-SHA**
- **HMAC-SHA256**
- **HMAC-SHA512**

Algoritmy MAC môžu taktiež využívať aj blokové šifry ako napríklad CMAC-AES.

1.4.2 Kryptografické hash funkcie

Jednou z prvých kryptografických hash funkcií ktorá, vznikla, bola MD5, navrhol ju Ron Rivest v roku 1991, výstupný odtlačok má veľkosť 128 bitov. V dobe návrhu bol tento algoritmus bezpečný, ale v dnešnej dobe sa už považuje za prelomený a ďalej už nespĺňa vlasnosť bezkolíznosti [9][10].

Nasledovníkom MD5 je algoritmus SHA-1 (Secure Hashing Algorithm), ktorý vylepšuje na bezpečnosti MD5. Napriek tomu existujú metódy, ktoré umožňujú znížiť počet možností útokom hrubou silou na 2^{69} operácií, čo je stále veľa ale tento algoritmus sa už neodporúča používať [11].

Preto bola navrhnutá rodina algoritmov SHA-2 ktoré existujú v niekoľkých módoch, ktoré určujú veľkosť odtlačku. Tieto módy je možné vidieť v tabuľke 1.1. Skupina algoritmov SHA-2 sa v dnešnej dobe považuje za vhodnú náhradu algoritmu SHA-1.

V roku 2015 vznikla nová rodina kryptografických hash algoritme SHA-3 [13]. Algoritmy v rodine SHA-3 sú založené na algoritme Keccak. SHA-3 bola navrhnutá s úmyslom ľahkej výmeny s algoritmom SHA-2, keďže SHA-3 obsahuje rovnaké módy veľkosti odtlačku ako SHA-2 (tabuľka 1.1).

Tab. 1.1: Rodina SHA-2 algoritmov [12]

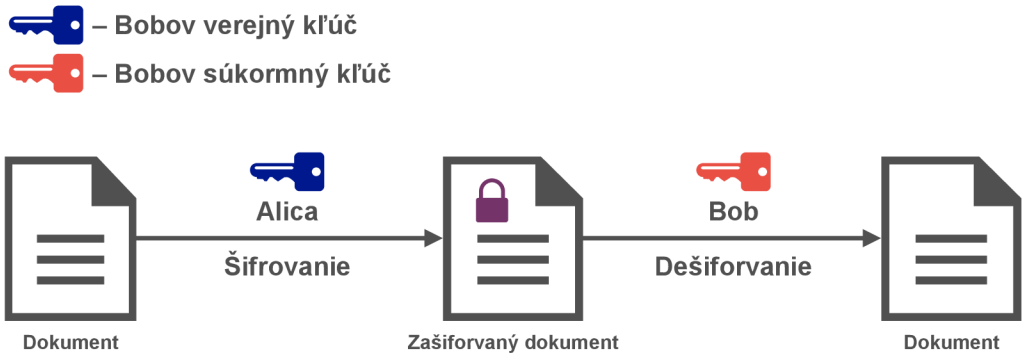
Algoritmus	maximálna veľkosť správy (bity)	veľkosť odtlačku (bity)
SHA-224	2^{64}	224
SHA-256	2^{64}	256
SHA-384	2^{128}	384
SHA-512	2^{128}	512
SHA-512/224	2^{128}	224
SHA-512/256	2^{128}	256

1.5 Asymetrická kryptografia

Asymetrická kryptografia taktiež ako symetrická kryptografia sa môže použiť na šifrovanie správ. Naviac pri niektorých algoritmoch sa ešte môže použiť aj na zdieľanie privátneho kľúča pre symetrickú kryptografiu alebo na autentifikáciu užívateľov pomocou digitálnych podpisov.

Na rozdiel od symetrickej kryptografie ktorá používa 1 kľúč na šifrovanie, asymetrická kryptografia používa pár kľúčov. Jeden z nich sa označuje ako súkromný kľúč (SK) a druhý ako verejný kľúč (VK). Užívateľ vygeneruje obidve kľúče, súkromný kľúč udržuje v tajnosti a verejný kľúč zdieľa druhej strane.

Pár kľúčov je matematicky navrhnutý tak, aby správa bola zašifrovaná verejným kľúčom. Potom iba ten, kto vlastní privátny kľúč, ju môže dešifrovať. Obrázok 1.3 vysvetľuje tento princíp. Ilustrovaný mechanizmus umožňuje šifrovať správy a posielať ich bez znalosti kľúča ktorým sa správa dešifruje, čo zmenšuje riziko oproti symetrickej kryptografii kde je nutná znalosť privátneho kľúča obidvoch strán.



Obr. 1.3: Princíp šifrovania pomocou asymetrickej kryptografie

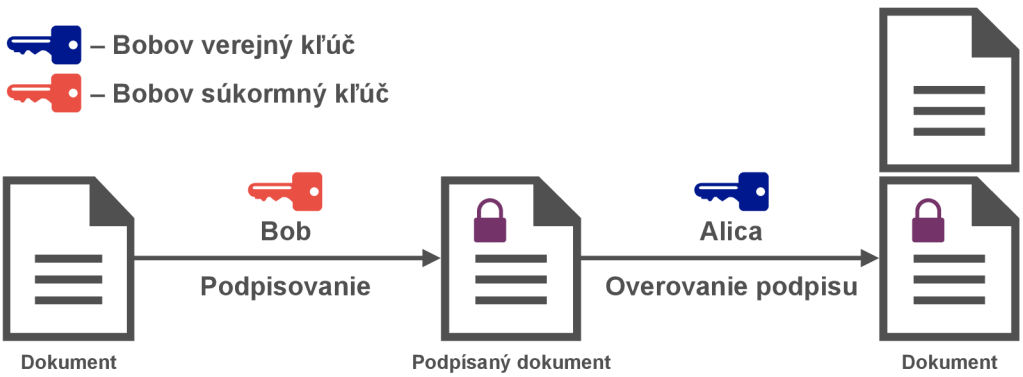
Môže potom vzniknúť otázka prečo, vôbec používať symetrickú kryptografiu? Asymetrická kryptografia má jednu nevýhodu, a to je veľkosť kľúčov, ktoré musia byť väčšie, aby šifra nebola náchylná na útok hrubou silou. Z tohto dôvodu trvá

šifrovanie väčších súborov alebo správ oveľa dlhšie oproti symetrickej kryptografii. Preto sa asymetrická kryptografia často nepožíva na šifrovanie.

1.5.1 Digitálne podpisy

Ako bolo spomínané v sekcii 1.5, asymetrická kryptografia sa dá použiť aj na autentifikovanie užívateľov pomocou digitálnych podpisov. Princíp digitálnych podpisov je podobný princípu šifrovania. V systéme stále existuje súkromný a verejný kľúč, pričom zdieľaný je iba verejný. Jeho úlohou je dokázať autentickosť poslanej správy ostatným užívateľom. Digitálny podpis by sa dal prirovnať k podpisu normálnemu, kde osoba, ktorá podpisuje dokument, zaisťuje svojím podpisom dôveryhodnosť dokumentu. Ak si ostatní účastníci chcú overiť, že dokument práve podpísala daná osoba, je to možné.

Na obrázku 1.4 je možné vidieť, ako presne funguje digitálny podpis. Užívateľ, ktorý podpisuje správu, ju podpíše svojím súkromným kľúčom. Môže ale nastať situácia, pri ktorej overovaná správa bude veľký dokument a podpisovanie veľkých súborov asymetrickou kryptografiou je pomalé a neefektívne. Preto sa musí najprv vytvoriť odtlačok z tejto správy, ktorý sa bude podpisovať oveľa rýchlejšie. Ak si prijímateľ správy chce overiť pôvod správy, vypočíta odtlačok rovnakým algoritmom ako odosielateľ z doručenej správy a pomocou verejného kľúča overí podpísaný odtlačok. Ak sa tieto dva odtlačky rovnajú, autentickosť správy bola overená.



Obr. 1.4: Princíp digitálneho podpisu

1.5.2 Algoritmy asymetrickej kryptografie

Najpoužívanejšie asymetrické algoritmy sú algoritmy RSA (Rivest Shamir Adleman) a DSA (Digital Signature Algorithm). Tieto dva štandardy sa popisujú detailnejšie v norme FIPS PUB 186-4 [14].

RSA algoritmus sa hlavne využíva pre digitálny podpis. Organizácia NIST odporúča dĺžku kľúčov na digitálny podpis veľkosti 2048 alebo 3072 bitov. Taktiež je možné použiť RSA ako šifrovací algoritmus ale ako bolo spomínané v sekcii 1.5, je to veľmi neefektívne. Algoritmus RSA sa využíva aj ako algoritmus na výmenu súkromného kľúča pri symetrickej kryptografii. Viac pozornosti k algoritmom výmene kľúča bude venované v nasledujúcej sekcii 1.5.3.

DSA algoritmus je použiteľný iba ako algoritmus digitálneho podpisu. Bol navrhnutý ako odpoveď na niektoré nevýhody algoritmu RSA. Vytvorenie digitálneho podpisu pomocou RSA je náročnejšie na výpočetný výkon zariadenia, DSA toto rieši možnosťou výpočtom nezávislej hodnoty z určených parametrov komunikácie pred samotným začatím komunikácie. Vypočítanú hodnotu je potom možné použiť na zrýchlený výpočet podpisu. Vďaka tejto vlastnosti je možné použiť algoritmus na zariadenia s malým výpočetným výkonom ako napríklad čipové karty [15].

DSA sa taktiež dá použiť z použitím eliptických kriviek, takýto algoritmus sa potom nazýva **ECDSA**. Hlavnou výhodou eliptických kriviek je možnosť používať menšie kľúče bez straty bezpečnosti. Ďalšou výhodou je rýchlosť výpočtu podpisu keďže, kľúče majú menšiu veľkosť. Jednou nevýhodou je, že takýto algoritmus môže byť zložitejší na implementáciu.

1.5.3 Algoritmy pre výmenu kľúčov

Aby dvaja užívatelia mohli komunikovať šifrovanou komunikáciou je najprv potrebné, aby mali zdieľaný súkromný kľúč. Samozrejme je nemožné, aby bol posielaný v otvorenej forme, ak by útočník zachytil túto komunikáciu, je schopný dešifrovať celý prenos.

Jedným z prvých protokolov, ktorý umožnil bezpečnú výmenu kľúčov, bol protokol **Diffie-Hellman**. Algoritmus funguje na princípe asymetrickej kryptografie. Algoritmom RSA je taktiež možné zdieľať privátny kľúč medzi 2 užívateľmi. Princíp je rovnaký ako šifrovanie pomocou **RSA**.

Majme dvoch užívateľov Alicu a Boba. Alica vygeneruje symetrický súkromný kľúč, ktorý potom zašifruje verejným kľúčom Boba. Keď Bob dostane zašifrovaný súkromný kľúč, rozšifruje ho svojím asymetrickým súkromným kľúčom. Tieto algoritmy stále ale nezaručujú autentifikáciu užívateľov. Autentifikácia sa dá zaistiť pomocou certifikátov a PKI (Public Key Infrastructure) a digitálnych podpisov. Viacej informácií o PKI bude nasledovať v kapitole 3.

2 Protokol SSL/TLS

Protokol SSL/TLS je kryptografický protokol, ktorý bol navrhnutý s cieľom vytvoriť zabezpečené sieťové spojenie medzi dvoma užívateľmi na úrovni aplikačnej vrstvy. SSL 1.0 (Secure Socket Layer) protokol bol vyvinutý ako prvá iterácia v roku 1990 spoločnosťou Netscape Communications [10]. V nasledujúcich rokoch vznikali ďalšie verzie protokolu SSL, menovite SSL 2.0 a SSL 3.0, ktoré sa považujú za kryptograficky nebezpečné.

V roku 1999 vznikol protokol TLS (Transport Layer Security), ktorý bol nástupcom protokolu SSL. Protokol bol síce premenovaný na TLS 1.0, ale v skutočnosti to bol stále taký istý protokol, ktorý sa líšil len minimálne od verzie SSL 3.0. Vďaka tomuto sa niekedy protokol TLS 1.0 nazýva aj ako SSL 3.1. Protokol sa ďalej vyvíjal a v roku 2006 vznikla verzia TLS 1.1 a o 2 roky neskôr aj verzia 1.2 [10].

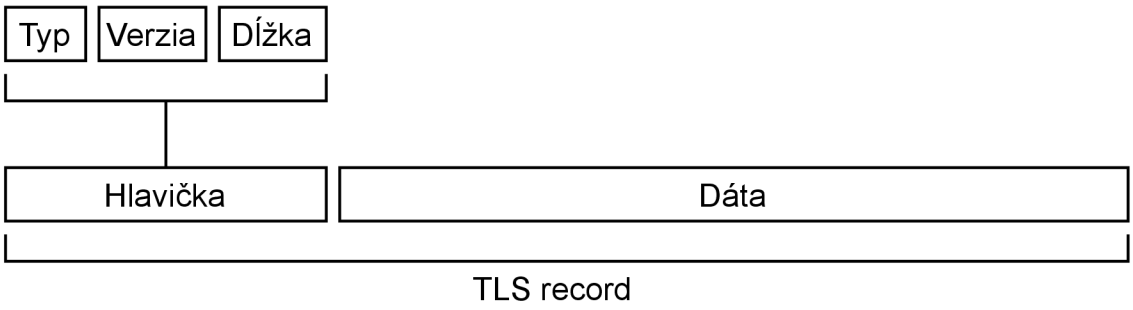
Najnovšia verzia protokolu TLS 1.3 vznikla nedávno v roku 2018, ktorá má za účel predefinovať a radikálne zmeniť starú špecifikáciu protokolu TLS 1.2 [16]. Jednou z mnohých zmien je menší počet hanshake správ. Čo sú to handshake správy a ako funguje protokol TLS, bude vysvetlené v nasledujúcich sekciách.

2.1 Record protokol

Record protokol má za úlohu enkapsulovať dáta z vyšších vrstiev sieťového modelu (aplikačná vrstva) do fragmentov a ďalej šifrovať tieto fragmenty podľa dohodnutých kryptografických algoritmov. K týmto fragmentom sa pridá hlavička, ktorá obsahuje:

- **typ správy** (Type),
- **verziu protokolu** (Version),
- **dĺžku správy** (Length).

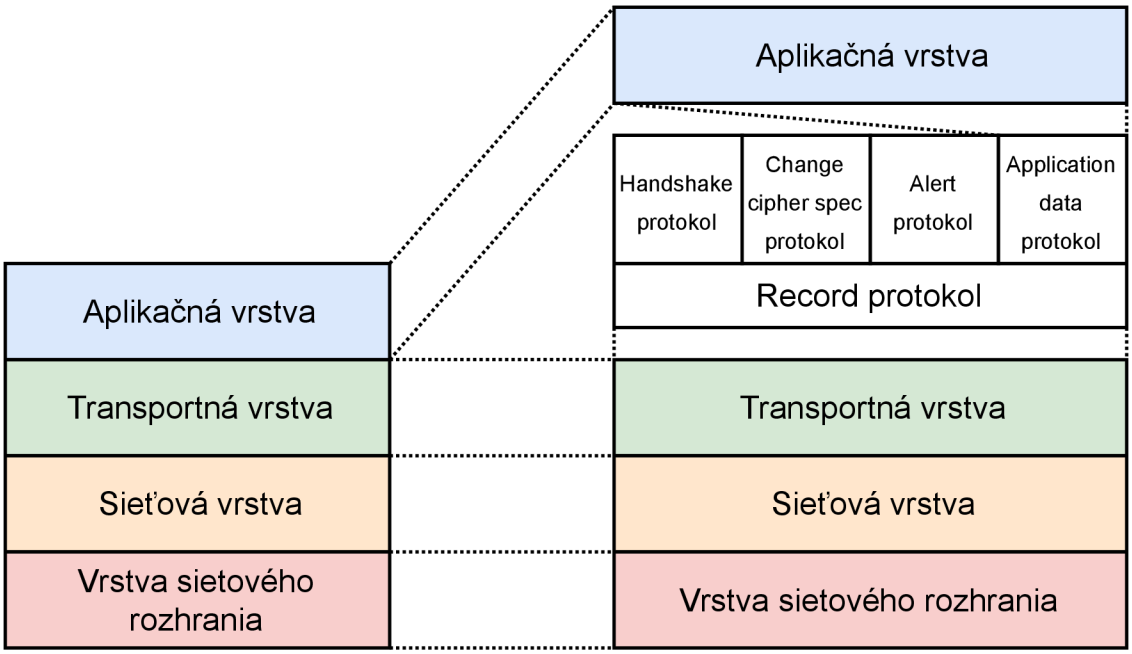
Celá táto štruktúra sa potom nazýva *TLS record* a možno ju vidieť na obrázku 2.1. Každý TLS record má priradené sekvenčné číslo, ktoré sa ale neprenáša po sieti. Každá strana si ho samostatne udržuje a sleduje aj sekvenčne číslo druhej strany.



Obr. 2.1: TLS record

TLS protokol bol navrhnutý tak, aby mohol zabezpečovať bezpečný prenos pre všetky protokoly aplikačnej vrstvy. Z tohto dôvodu nemôže protokol pracovať na vrstve aplikačnej, ale pod aplikačnou vrstvou. Umiestnenie TLS protokolu v modeli TCP/IP je možné vidieť na obrázku 2.2. Z obrázku je tiež možné vidieť že nad protokolom TLS sa nachádzajú ďalšie 4 sub-protokoly:

- **Handshake protocol** – princíp Handshake protokolu bude vysvetlený v nasledujúcej sekcii 2.2.
- **Change cipher spec protocol** – protokol umožňuje zmenu ustanovenej kryptografickej sady. Informácie o kryptografických sadách budú nasledovať v sekcii 2.3
- **Application data protocol** – tento protokol má na starosti prenos dát z aplikačnej vrstvy do Record protokolu ktorý už ďalej spracováva tieto dáta.
- **Alert protocol** – dovoľuje posielanie signalizačných správ.



Obr. 2.2: TLS protokol v modelu TCP/IP

2.2 Hanshake protokol

Každé TLS spojenie začína výmenou správ pomocou protokolu handshake. Keď chce klient naviazať zabezpečenú komunikáciu so serverom, tak pošle ako prvý správu o tom, že by sa chcel spojiť so serverom. Server potom odpovedá na túto požiadavku.

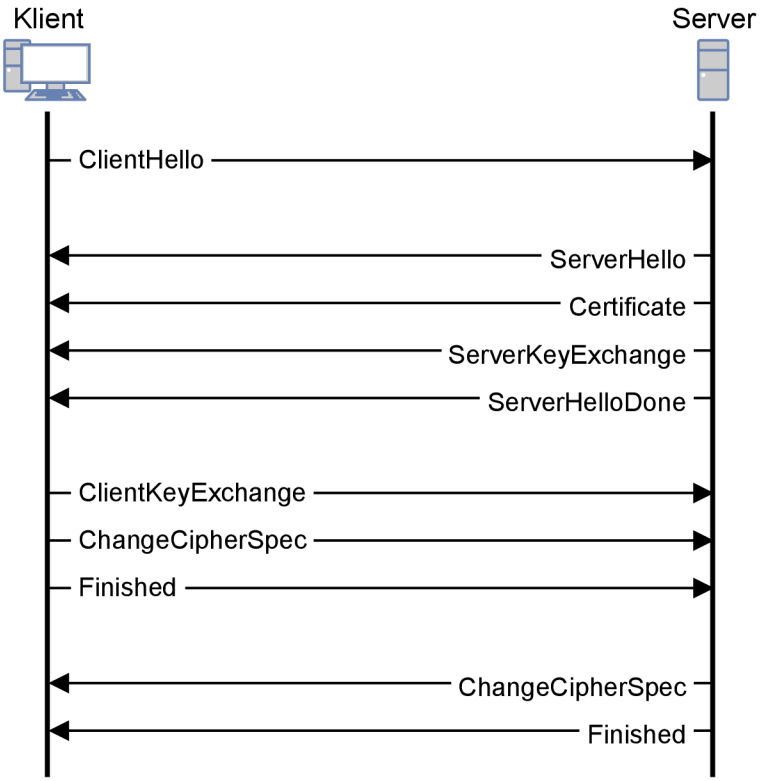
Protokol handshake má za úlohu splniť 4 základné funkcie:

- vzájomná výmena možností použitia kryptografických prostriedkov a voľba týchto prostriedkov ktoré budú použité počas celého spojenia,
- validácia certifikátov servera prípadne klienta,
- ustanovenie takzvaného *master key*, ktorý sa bude nasledovne používať počas celého spojenia,
- zaistenie integrity doteraz posielaných správ.

Princíp prvotného ustanovenia TLS spojenia so serverom je možné vidieť na obrázku 2.3. V tomto prípade sa autentifikuje len server klientovi, ale existuje druhý variant, kde sa musí autentifikovať aj klient serveru. Detailnejší popis správ posielaných pri TLS handshake:

- **ClientHello** – klient oznamuje serveru, že chce naviazať spojenie a pošle serveru list kryptografických sád, ktoré podporuje a môže použiť. Spolu s týmto pošle verziu SSL/TLS protokolu, ktorú chce použiť, čas odoslania správy a náhodný reťazec bitov.
- **ServerHello** – server odpovie na požiadavku klienta s kryptografiou sadou, ktorá sa bude naďalej používať, ustanovenú verziu SSL/TLS protokolu a podobne ako ClientHello pošle čas a jeho náhodný reťazec bitov.
- **Certificate** – server posiela klientovi svoju reťaz certifikátov [1] ktoré si klient overí. Táto správa nieje povinná, keďže niektoré kryptografické sady nepoužívajú infraštruktúru PKA na autentifikáciu servera.
- **Client/ServerKeyExchange** – obsahom týchto správ sú potrebné parametre na ustanovenia hlavného kľúča (master key). Keď si obidve strany vymenia tieto parameter, vypočítajú si hlavný kľúč samostatne už na ich fyzických strojoch a ďalej ho neposielajú druhej strane. Parametre ktoré si vymieňajú, záležia na dohodnutej kryptografickej sade.
- **ServerHelloDone** – server oznamuje klientovi že poslal všetky potrebné správy a čaká na odpoveď klienta.
- **ChangeCipherSpec** – správa znázorňuje, že strana ktorá, posiela túto správu, má dostatok informácií a že začne šifrovať všetku komunikáciu od tohto bodu ďalej. Táto správa ale nie je súčasťou handshake protokolu, ale je súčasťou Change cipher spec protokolu.
- **Finished** – obidvaja účastníci posielajú túto správu keď signalizujú druhej strane, že handshake komunikácia sa skončila. Správa taktiež slúži ako kontrola integrity doteraz prenášaných správ.

[1] viacej o certifikátoch nasleduje v sekcii 3.1



Obr. 2.3: Princíp TLS handshake protokolu

2.3 Kryptografické sady

Kryptografická sada (Cipher suite) je výber kryptografických algoritmov a metód, ktoré sa budú implementovať v danom systéme komunikácie. Najčastejšie kryptografická sada definuje tieto typy základných kryptografických metód a algoritmov:

- algoritmus autentifikácie,
- algoritmus výmena kľúča,
- algoritmus šifrovania, veľkosť použitého kľúča,
- blokový mód šifrovacieho algoritmu,
- MAC algoritmus,
- pseudonáhodná metóda generácia čísel,
- hash funkcia.

Samozrejme nie vždy bude sada definovať všetky z týchto metód a algoritmov napríklad blokový mód šifry sa uplatní, iba ak sa používa bloková šifra pre algoritmus šifrovania. Príklad kryptografickej sady je možné vidieť tu[2]:

`TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`

[2]príklad používa formát od organizácie IANA.

Metódy a algoritmu ktoré definuje sada, sú oddelené znakom "_" a je možné ich zo sady vyčítať:

- **ECDHE** – algoritmus výmena kľúča.
- **RSA** – algoritmus autentifikácie.
- **AES_128** – algoritmus šifrovania, veľkosť použitého kľúča.
- **GCM** – blokový mód šifrovacieho algoritmu.
- **SHA256** – hash funkcia.

Doplňujúce parametre ohľadom konkrétnych implementácií sa potom dajú nájsť už v súvisiacich RFC (Request For Comment) dokumentoch. Zoznam všetkých kryptografických sád je možné nájsť na stránke organizácie IANA [17].

Ako sa definujú a zapisujú kryptografické sady, taktiež záleží aj od konkrétnej implementácie protokolu TLS. Napríklad implementácia OpenSSL používa iný formát zapisovania kryptografický sád ako IANA.

2.4 TLS rozšírenia

TLS rozšírenia (TLS extensions) umožňujú možnosť pridať dodatočnú funkcionalitu do protokolu SSL/TLS. Táto funkcionalita sa pridáva na koniec ClientHello a ServerHello správ v podobe štruktúry, ktorá obsahuje typ rozšírenia a dáta k rozšíreniu. List všetkých rozšírení udržiava organizácia IANA [18]. Príklad TLS rozšírení:

- **Server Name Indication** – umožňuje špecifikovať meno serveru s ktorým sa chce užívateľ spojiť, ak je na jednej IP adrese viacero virtuálnych serverov.
- **Signature Algorithms** – umožňuje užívateľovi zvoliť iný typ algoritmu pre digitálny podpis, ako je definovaný v kryptografickej sade.

2.5 Vlastnosti verzií protokolov SSL/TLS

Ako bolo spomínané v úvode kapitoly 2, protokol SSL/TLS existuje v niekoľkých verziách. Potreba stále vyvíjať nové verzie vznikla z neustáleho vývoja výpočetnej sily počítačov a vývoja poznatkov z oblasti kryptografie a bezpečnosti. Práve preto stále vznikali nové verzie SSL/TLS protokolu, ktoré vylepšovali vlastnosti predošlých protokolov, ako je možné vidieť v nasledujúcom zozname verzií protokolov.

SSL 1.0 [10]

- Používal slabú prúdovú šifru RC4 na šifrovanie dát.
- Na overenie integrity dát používal jednoduché mechanizmy ako CRC Cyclic Redundancy Check.
- Protokol obsahoval mnoho ďalších nedostatkov, kvôli čomu nebol ani dostupný pre verejné použitie.

SSL 2.0 [10][20]

- Slabá kontrola integrity bola nahradená hašovacou funkciou MD5, čo je ale stále ľahko preľomiteľný hašovací algoritmus.
- Dovoľuje výmenu iba jedného certifikátu v reťazi certifikátov, znamená to, že certifikát musí byť podpísaný koreňovým certifikátom.
- Rovnaký kľuč je použitý pre šifrovací algoritmus a HMAC algoritmus.
- Handshake správy nie sú chránené žiadnym mechanizmom, čo umožňuje uskutočnenie MitM (Man in the Middle) útoku pomocou, ktorého je útočník schopný vynútiť zvolenie slabej kryptografickej sady.

SSL 3.0 [21][22]

- Zraniteľnosť v blokovom móde CBC čo umožňuje útočníkovi dešifrovať časť správy. Táto zraniteľnosť sa využíva v útoku POODLE pomocou ktorého sa degraduje verzia SSL protokolu na 3.0.
- Protokol stále používa slabé hašovacie algoritmy ako MD5 a SHA-1.
- Stále umožňuje použitie RC4 prúdovej šifry, ktorá je ľahko preľomiteľná.
- Nepodporuje využitie DTLS protokolu, ktorý pracuje na UDP spojení namiesto TCP.
- Umožnuje útok MitM pomocou mechanizmu session resumption.

TLS 1.0 [21][10]

- Upravuje konštrukciu blokového módu CBC, ktorá odstraňuje zraniteľnosť POODLE.
- Definuje rozšírenie (`TLS_FALLBACK_SCSV`[3]), ktorým je možné predísť tkz. downgrade útok.
- Umožňuje použitie algoritmov, ktoré využívajú eliptické krivky ako napríklad ECDH (Eliptic Curve Diffie-Hellman).
- Táto verzia protokolu neprináša žiadne radikálne zmeny, keďže je veľmi podobná verzii SSL 3.0.

TLS 1.1 [23]

- Generácia Inicializačného vektora sa zmenila z implicitnej metódy ne explicitnú pri blokovom móde CBC. Explicitná generácia znamená, že po určitom počte bytov sa bude generovať nový inicializačný vektor. Táto zmena zabraňuje útoku BEAST [24].
- Ak sa spojenie nečakane skončí, tak je zase možné obnoviť spojenie.
- Do RFC dokumentu boli pridané informácie o útokoch na TLS protokol.

[3]Rozšírenie bolo definované až v roku 2015 a nie pri vzniku protokolu

TLS 1.2 [25]

- Algoritmy MD5 a SHA-1, ktoré sa vždy používali v pseudonáhodných funkciách, sú vymenené algoritmami, ktoré sú špecifikované v kryptografickej sade.
- Podpora blokových módov, ktoré dovoľujú autentifikáciu ako napríklad GCM.
- Pridané HMAC-SHA256 kryptografické sady.
- Odstránenie symetrických šifier DES a IDEA.
- Definuje užšiu kontrolu na rôzne požiadavky.

TLS 1.3 [16]

- Algoritmy výmen kľúča RSA a statický Diffie-Hellman boli odstránené.
- Všetky handshake správy sa šifrujú okrem `ServerHello`.
- Hašovacie funkcie v kryptografických sadách sa teraz používajú v MAC algoritmoch aj v Key-derivation algoritmoch.
- Kryptografické sady určujú len algoritmus, blokový mód, veľkosť kľúča symetrickej šifry a hašovaciu funkciu.
- Zmenšený počet handshake správ, čo umožňuje rýchlejšie ustanovenie spojenia.

2.6 Implementácie protokolu SSL/TLS

SSL/TLS je sám o sebe nepoužiteľný v praxi, pretože RFC dokumenty, ktoré popisujú a vysvetľujú princíp týchto protokolov, neponúkajú konkrétne implementácie. Preto vzniklo niekoľko knižníc, ktoré implementujú protokol SSL/TLS. Medzi tieto knižnice patria:

- **OpenSSL** (`www.openssl.org`)
- **wolfSSL** (`www.wolfssl.com`)
- **LibreSSL** (`www.libressl.org`)
- **GnuTLS** (`www.gnutls.org`).

3 Infraštruktúra verejných kľúčov

PKI alebo *Public Key Infrastructure* je model, ktorý umožňuje distribuovať certifikáty medzinárodne a umožňuje užívateľom a zariadeniam vytvoriť bezpečnú komunikáciu na internete. PKI model funguje na princípe asymetrickej kryptografie a digitálnych podpisov (sekcia 1.5). Presnejšie, využíva princíp pár kľúčov a to súkromného a verejného. Pričom verejné kľúče sa ukladajú do certifikátov, ktoré sú ďalej sprístupnené verejnosti. Tieto certifikáty sa ďalej podpisujú súkromnými kľúčmi a užívatelia sú schopní si overiť identitu pomocou verejných kľúčov. Práve tento princíp umožňuje vytvorenie bezpečného spojenia so stranami.

Zodpovednosť za vystavovanie týchto certifikátov nesú takzvané Certifikačné autority. Viacej informácií o certifikačných autoritách bude nasledovať v sekcii 3.3.

3.1 Certifikáty

Certifikát je digitálny dokument, ktorý umožňuje účastníkom sieťovej komunikácie vymieňať potrebné informácie na overenie identity strán. Štandard, ktorý definuje, čo bude obsahovať certifikát, je x509 (RFC 5280 [19]). Certifikát obsahuje informácie o dvoch subjektoch, ktoré sa označujú pomocou DN (Distinguished name) parametru [10]. V certifikáte sa nachádzajú 2 typy týchto subjektov:

- **Vydavateľ** (Issuer) – označuje subjekt, ktorý vytvoril certifikát.
- **Predmet** (Subject) – označuje subjekt, ktorému bol certifikát vystavený.

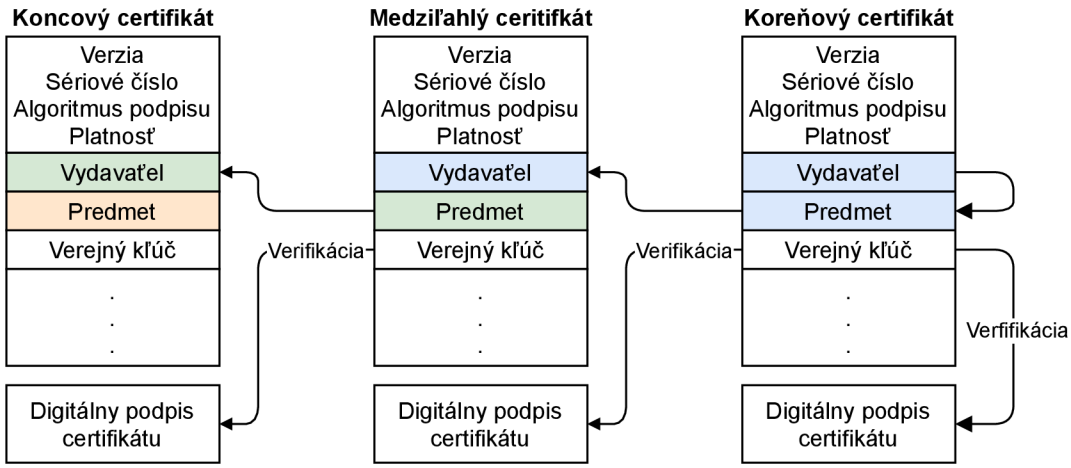
Viacej informácií ohľadne týchto subjektov bude nasledovať v sekcii 3.2. Niektoré ďalšie parametre ktoré certifikát obsahuje:

- **Verzia** – verzia certifikátu. Momentálne existujú len verzie $1, 2$ a 3.
- **Sériové číslo** – číslo, ktoré unikátne identifikuje certifikát.
- **Algoritmus použitý pre podpis** – špecifikuje algoritmy použité na vytvorenie podpisu. Napríklad `shaWithRSAEncryption`.
- **Digitálny podpis predošlého certifikátu** – digitálny podpis, ktorý sa používa na overenie integrity certifikátu.
- **Verejný kľúč** – verejný kľúč, ktorý sa používa na overovanie digitálnych podpisov počas samotnej komunikácie alebo na overenie predošlého certifikátu v reťazi certifikátov.
- **Platnosť** – platnosť certifikátu, ktorá sa udáva v dvoch parametroch. Tieto parametre určujú začiatok a koniec platnosti certifikátu.

Existuje niekoľko formátov a štruktúr, ako sa môžu certifikáty kódovať, ukladať a prenášať. Jedným z najviac používanejších formátov je formát `PEM` (Privacy-Enchanced Mail). Ďalšie používané formáty sú `ASN.1` (Abstract Syntax Notation One) a `DER` (Distinguished Encoding Rules) [3].

3.2 Zreťazenie certifikátov

Zreťazenie certifikátov umožňuje vytvoriť reťaz certifikátov (certificate chain). Tento mechanizmus slúži na overenie dôveryhodnosti verejného kľúča posledného certifikátu v reťazi. Princíp fungovania zreťazenia certifikátov je ilustrovaný na obrázku 3.1.



Obr. 3.1: Princíp zreťazenia certifikátov

Overenie dôveryhodnosti je možné, pretože ako prvý certifikát v reťazi je vždy koreňový certifikát (root certificate). Tento koreňový certifikát je vydávaný certifikačnými autoritami (sekcia 3.3). Tomuto certifikátu implicitne dôveruje každý užívateľ alebo server, ktorý ho majú vstavaný do operačného systému. Ďalej v reťazi sa nachádzajú ďalšie medziľahlé certifikáty (intermediate certificate). Medziľahlé certifikáty sú digitálne podpísané súkromným kľúčom, ktorého verejný kľúč sa nachádza v koreňovom certifikáte. Z toho vyplýva, že dôveryhodnosť koreňového certifikátu sa prenáša na medziľahlé certifikáty. Ako posledné sa v reťazi nachádzajú koncové certifikáty (end-entity certificate), ktoré sú opäť podpísané súkromným kľúčom, ktorý patrí verejnému kľúču medziľahlým certifikátom. Celá táto reťaz certifikátov a ich digitálne podpisy sa potom dajú spätne overiť verejnými kľúčmi certifikátov [3].

V reťazi certifikátov sa taktiež prenášajú parametre vydavateľ a predmet. V koreňovom certifikáte sa tieto parametre rovnajú, keďže vydavateľ certifikátu je aj predmet certifikátu. Koreňový certifikát sa taktiež nazýva aj *self-singed certificate* keďže digitálny podpis je vytvorený súkromný kľúčom patriacemu verejnému kľúču samotného certifikátu. Ďalšie certifikáty v reťazi majú v parametri vydavateľ predmet minulého certifikátu ako je možné vidieť na obrázku 3.1.

3.3 Certifikačné autority

Certifikačná autorita je organizácia, ktorá slúži na vystavovanie koreňových certifikátov. Ako bolo spomínané v minulej sekcii, tieto certifikáty sú uložené na operačných systémoch zariadení. Toto znamená, že v rámci TLS handshake správy `Certificate` sa neposiela koreňový certifikát v reťazi. Príklad niekoľkých certifikačných autorít:

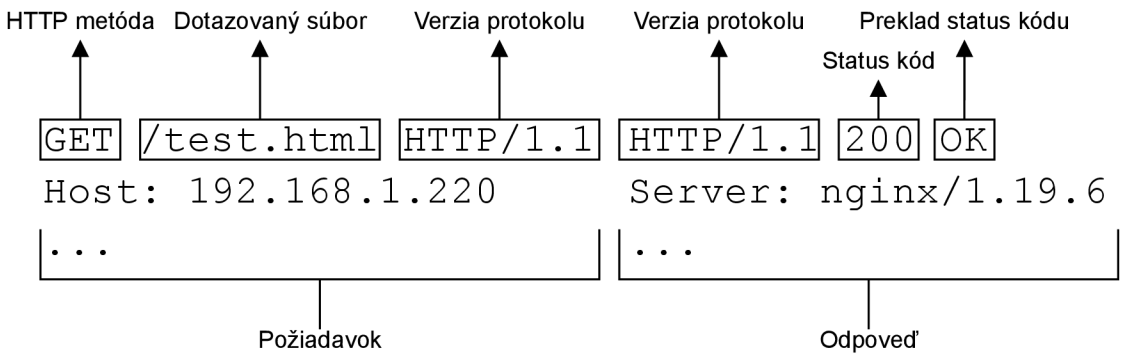
- **Let's Encrypt** (`www.letsencrypt.org`)
- **IdenTrust** (`www.identrust.com`)
- **GlobalSign** (`www.globalsign.com`)
- **digicert** (`www.digicert.com`)
- **ComodoCibersecuirty** (`www.comodo.com`)
- **Entrust** (`www.entrust.com`).

4 Protokol HTTP a HTTPS

HTTP (Hypertext Transfer Protocol) je protokol, ktorý pracuje na aplikačnej vrstve OSI modelu (sekcia 1.1). HTTP protokol definuje komunikáciu medzi klientom a serverom a umožňuje zdieľanie hypertextových dokumentov a na informačnom systéme WWW (World Wide Web). Inak povedané HTTP slúži ako protokol, pomocou ktorého môžeme pristupovať na internetové stránky.

4.1 HTTP metódy a status kódy

HTTP protokol definuje niekoľko metód a status kódov, ktoré klient a server používajú na výmenu správ medzi sebou. Správy majú definovaný formát a každá správa obsahuje buď odpoveď, alebo požiadavku. Metódy definujú typ požiadavky a status kódy určujú odpoveď. Tento princíp je ilustrovaný na obrázku 4.1. Metódy sa ďalej rozdeľujú na dve skupiny. Bezpečné metódy sú definované tak, že nemôžu vytvoriť žiadny dodatočný efekt na systém, na ktorý sú posielané (GET, HEAD, OPTIONS, TRACE). Do druhej skupiny patria metódy, ktoré nespĺňajú túto požiadavku [26].



Obr. 4.1: Požiadavka a odpoveď HTTP protokolu

4.1.1 GET a HEAD

Metódy GET a HEAD patria do skupiny bezpečných metód. Správa s týmito metódami má za účel odosielateľovi poslať dokument alebo iný zdroj, na ktorý sa dokazuje. Keďže server iba odošle kópiu požadovaného zdroja, nenastane žiadna radikálna zmena na serveri. Z toho vyplýva že práve tieto dve metódy patria do skupiny bezpečných metód. Metóda GET umožňuje odosielateľovi dotaz na zdroj, ktorý mu server vie poskytnúť, pričom metóda HEAD umožňuje dotazovanie iba na hlavičku odpovede (Response header).

4.1.2 POST, PUT a DELETE

Tieto tri metódy nepatria do typu bezpečných metód, keďže vyvolajú zmenu na serveri. Metóda PUT vytvorí nový obsah súboru alebo zmení, pričom klient, ktorý odosiela požiadavku si volí, aký súbor sa zmení a ako. Metóda POST je podobná tým, že vytvára nový obsah ale nie je určené, v akom súbore alebo databáze, keďže sa využíva v HTML dotazníkoch. Metóda DELETE, ako vyplýva z jej mena, sa používa na vymazanie súborov alebo zdrojov. Implementované metódy PUT a DELETE vo väčšine prípadoch naviac vyžadujú aby, sa klient autentifikoval.

4.1.3 Status kódy

Status kódy slúžia ako odpoveď serveru na požiadavku, keďže nie vždy bude požiadavka spracovaná úspešne. Každý status kód je vyjadrený číselne pre účel automatického spracovávania programov a slovne pre čitateľnosť ľudí. Celkovo je definovaných 5 kategórií status kódov ako je možné vidieť v tabuľke 4.1.

Tab. 4.1: Kategórie status kódov protokolu HTTP [26]

Číselný rozsah	Slovný popis	Príklad
100-199	Informácia	100 Continue
200-299	Úspech	201 Created
300-399	Presmerovanie	301 Moved Permanetly
400-499	Chyba klienta	404 Not found
500-599	Chyba servera	503 Service Unavailable

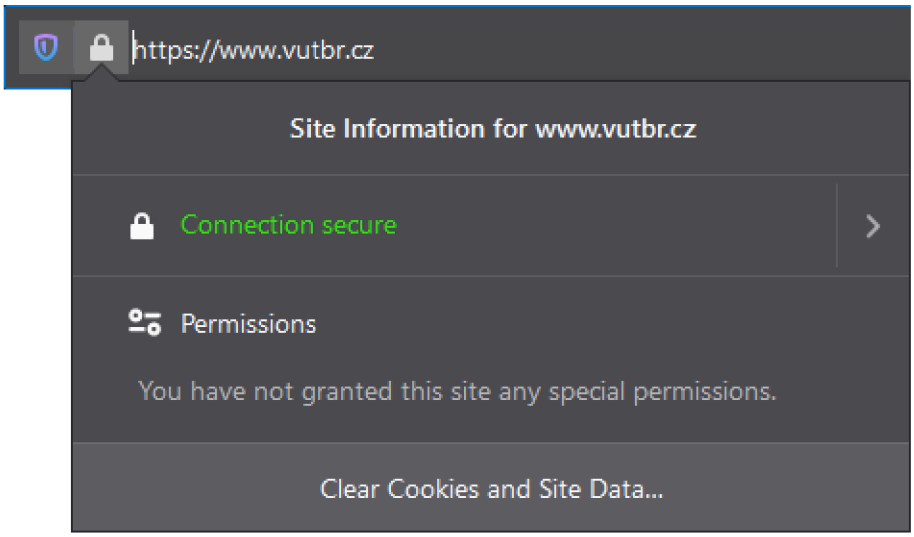
4.2 HTTPS

Keďže protokol HTTP komunikuje v otvorenej forme znamená, to že všetky správy, ktoré si klient a server vymieňajú, budú čitateľné, ak niekto odpočúva sieťový prenos. Protokol taktiež nezaisťuje autentifikáciu serveru ani klienta a nezaisťuje ani integritu posielaných správ. Preto vznikla potreba zabezpečiť protokol HTTP. Z tejto potreby vznikol protokol HTTPS alebo *Hypertext Transfer Protocol Secure* alebo aj *HTTP over TLS*. Protokol je veľmi podobný ako HTTP protokol, až na to, že všetka komunikácia prebieha cez TLS.

Predtým, ako sa vytvorí HTTPS spojenie, ako prvé prebehne TCP 3-way handshake, ktoré ustanoví spojenie s klientom a serverom. Ako ďalej prebehne TLS

handshake popísané v sekcii 2.2 ktoré zaistí dôveryhodnosť, integritu a autentifikáciu komunikačných strán pomocou PKI. Následne už prebieha samotná komunikácia cez protokol HTTPS pomocou metód.

V dnešnej dobe už všetky moderné a udržované internetové prehliadače podporujú protokol HTTPS. Ak je prehliadač pripojený na webový server pomocou protokolu HTTPS, väčšinou to býva znázornené ikonou zámku pred webovou adresou. Príklad HTTPS spojenia s web serverom pre prehliadač Mozilla Firefox je možné vidieť na obrázku 4.2.



Obr. 4.2: Protokol HTTPS na prehliadači Mozilla Firefox

5 Zraniteľnosti protokolu SSL/TLS

Táto kapitola skúma a popisuje zraniteľnosti, ktoré je nástroj popisovaný v tejto práci schopný analyzovať. Zraniteľnosti, ktoré sú skúmané, je možné rozdeliť do dvoch skupín:

- Zraniteľnosti ktoré vznikli vďaka chybnej implementácii protokolu SSL/TLS (sekcia 2.6).
- Zraniteľnosti ktoré vznikajú z chýb v definícii protokolu SSL/TLS.

5.1 Heartbleed (CVE-2014-0160)

Heartbleed je veľmi dobrý príklad zraniteľnosti, ktorá vznikla pomocou chybnej implementácie. Konkrétnejšie chyba v knižnici OpenSSL. Zraniteľnosť bola zverejnená verejnosti v apríli 2014 a využíva chybu v implementácii protokolu rozšírenia (sekcia 2.4) Heartbeat. Protokol heartbeat umožňuje odosielateľovi kontrolovať, ak je príjemca stále dostupný pomocou poslania Hearbeat Request správy. Ak je druhá strana dostupná, pošle správu Hearbeat Resposne. Tieto správy majú nasledovný formát [27][3]:

```
struct {
    HeartbeatMessageType type;
    uint16 payload_length;
    opaque payload[HeartbeatMessage.payload_length];
    opaque padding[padding_length];
} HeartbeatMessage;
```

- **HeartbeatMessageType** – typ hearbeat správy. Môže nabiť hodnotu Request(1) alebo Response(2).
- **payload_length** – dĺžka obsahu správy.
- **payload** – Samotný obsah správy. Odosielateľ správy hearbeat posiela náhodný obsah v tomto poli. Ak je príjemca správy dostupný pošle kópiu týchto dát.
- **padding** – Výplň dát.

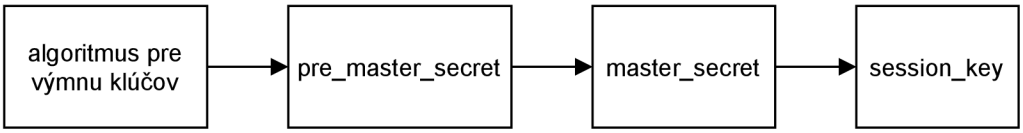
Zraniteľnosť umožňuje útočníkovi čítať bloky dát o veľkosti 64 KB z pamäte operačného systému, na ktorom je využívaná zraniteľná verzia knižnice OpenSSL. Keď útočník pošle hearbeat request s dĺžkou obsahu väčšou, ako je samotná dĺžka obsahu, tak knižnica nekontroluje, či je táto dĺžka väčšia ako samotné dáta. Toto spôsobí, že príjemca správy pošle v hearbeat response rovnaké dáta, ktoré prijal, plus ďalšie dáta z pamäte operačného systému, tak aby sa súčet týchto dát rovnal určenej dĺžke.

Často sa v pamäti operačných systémov nachádzajú nešifrované informácie, ako napríklad privátne kľúče alebo heslá. Útočník je schopný pomocou tejto zraniteľnosti získať tieto nešifrované informácie a využiť ich v ďalších útokoch.

5.2 CCS injection (CVE-2014-0224)

Podobne ako zraniteľnosť heartbleed CCS (CangeCipherSpec) injection využíva chybu v implementácii knižnice OpenSSL. Keď odosielateľ odošle CCS správu, indikuje tým, že všetka ďalšia komunikácia bude šifrovaná. Aby odosielateľ mohol zahájiť šifrovanú komunikáciu potrebuje, aby obidve strany vytvorili rovnaký tajný kľúč. Ako bolo spomínané v kapitole 2, protokol SSL/TLS toto umožňuje algoritmami na výmenu kľúčov.

Ako prvé si účastníci vymenia tkz. `pre_master_secret`, ktorý sa použije na výpočet `master_secret`, z ktorého sa ďalej vypočíta `session_key`, ktorý sa používa na šifrovanie komunikácie. Tento proces je znázornený na obrázku 5.1.



Obr. 5.1: Generovanie kľúčov

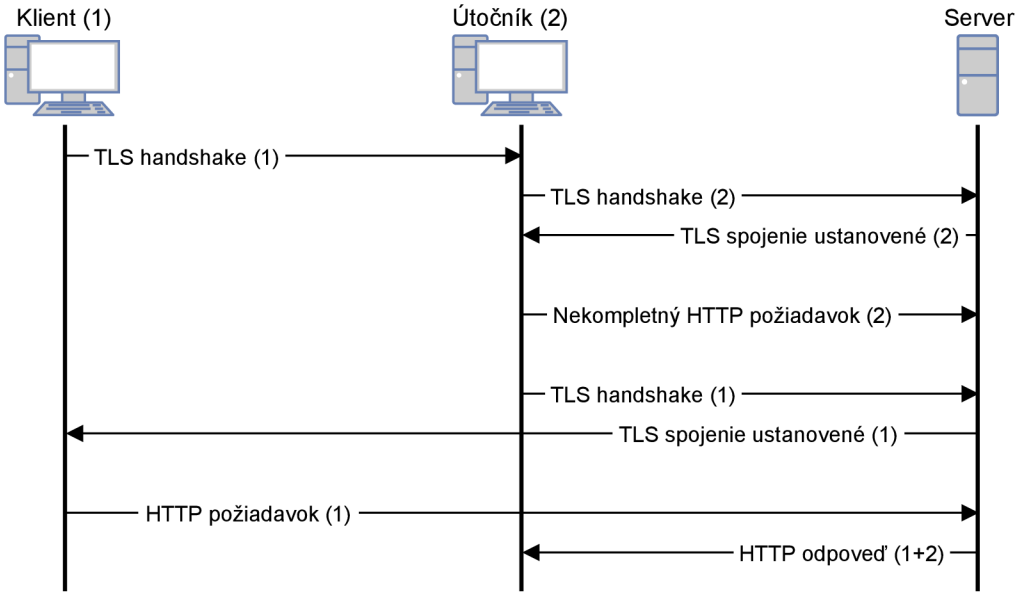
Následne je možné vidieť ako sa presne vypočíta `master_secret`. Na výpočet sa používajú 3 hodnoty. Náhodné čísla ustanovené z ClientHello a ServerHello správ a spomínaný `pre_master_secret` [25].

```
master_secret = PRF(pre_master_secret, "master secret",
                    ClientHello.random + ServerHello.random);
```

Aby bolo možné využiť CCS injection zraniteľnosť, útočník musí zachytávať komunikáciu a vytvoriť MitM (Man in the Middle) útok. Najprv útočník naslúcha a preposiela správy, aby zistil náhodné čísla obidvoch strán z hello správ. Nasledovne pošle správu CCS pred ustanovením `pre_master_secret` a tým oklame komunikujúce strany, že `pre_master_secret` kľúč je 0. Účastníci komunikácie vygenerujú `master_secret` iba z náhodných hodnôt, ktoré útočník taktiež pozná. Toto umožňuje útočníkovi dešifrovať komunikáciu oboch strán a získať citlivé informácie alebo meniť obsah správ[28].

5.3 Insecure renegotiation (CVE-2009-3555)

Insecure renegotation využíva zraniteľnosť v mechanizme renegotiation, ktorý slúži na znovu ustanovenie kryptografických parametrov spojenia. Na rozdiel od predošlých spomínaných zraniteľností, táto zraniteľnosť využíva chybu v definícii protokolu SSL/TLS a nie chybu v jeho implementácii. Princíp spočíva vo vytvorení novej handshake komunikácie, pomocou ktorej sú účastníci schopní dohodnúť sa na nových parametroch spojenia ako napríklad kryptografická sada.



Obr. 5.2: Využitie inscure renegotiation zraniteľnosti

Princíp zraniteľnosti spočíva v MitM útoku, ako je možné vidieť na obrázku 5.2. Útočník zachytí klientov pokus o spojenie so serverom a nepošle ho ďalej. Namiesto toho ustanoví spojenie ako prvý so serverom. Po ustanovení spojenia útočník pošle nekompletnú HTTP požiadavku a prepošle klientov zastavený pokus o vytvorenie spojenia so serverom. Z pohľadu klienta vytvára spojenie on, ale z pohľadu serveru sa klient iba pokúša o renegotiation. Keďže bola HTTP požiadavka poslaná pred klientovým začatím spojenia, tak server si myslí že klient ešte neposlal celú požiadavvku a čaká na klienta. Klient pošle svoju HTTP požiadavku spolu s identifikátorom ako napríklad session-id. Vďaka tomu je útočník schopný vydávať sa ako klient, pretože z pohľadu serveru mu celú požiadavku posiela klient [3].

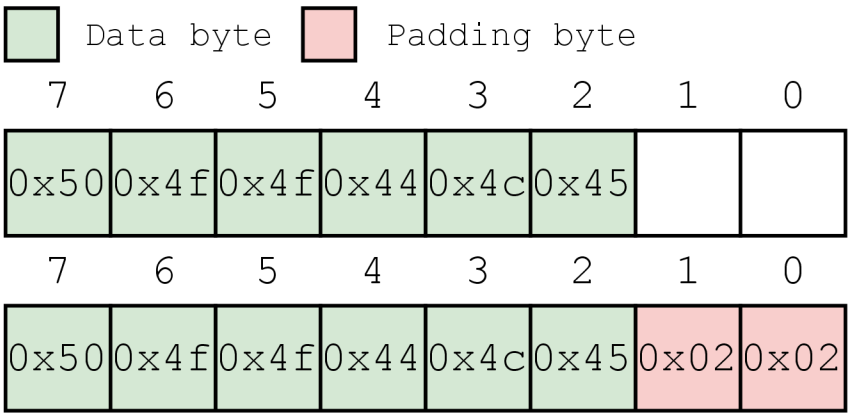
Princíp je ilustrovaný obrázkom 5.3 kde červene zvýraznený text je prvotná požiadavka, ktorú útočník poslal. Text čiernou farbou je klientova požiadavka. Vďaka značke `X-Ignore` bude druhý riadok HTTP požiadavky ignorovaný a server rozpozná iba 1. a 3. riadok. Vďaka tomu je útočník schopný získať zadaný súbor s použitím užívateľovho session-id.

```
GET /path/to/file HTTP/1.0
X-Ignore: GET /index HTTP/1.0
Cookie: PHPSESSID=...
```

Obr. 5.3: Obsah konečnej HTTP požiadavky

5.4 Zombie POODLE a GOLDENDOODLE

Zombie POODLE a GOLDENDOODLE sú v princípe rovnaké zraniteľnosti. Obidve využívajú útok typu POODLE (Padding Oracle On Dowgraded Legacy Ecryption). Tento typ útoku využíva chybu v móde blokových symetrických šifrách typu CBC (Cipher Block Chaining). CBC využíva padding (sekcia 1.3.2) na vyplnenie blokov na potrebnú dĺžku. Hodnoty na vyplnenie volí podľa dĺžky potrebného paddingu. To znamená, že ak blok má veľkosť 8 B a posledný blok 6 B, tak padding bude mať hodnotu 2. Príklad je možné vidieť na nasledujúcom obrázku 5.4.



Obr. 5.4: Príklad CBC padding

Zraniteľnosť POODLE funguje na princípe MitM útoku. Útočník sleduje komunikáciu a odchytí zašifrované bloky správy. Zachytené bloky sa bude snažiť dešifrovať pomocou tkz. padding oracle. Padding oracle je entita, ktorá vlastní súkromný kľúč, pomocou ktorého dokáže dešifrovať zašifrované bloky. V tomto prípade to je server, ktorému klient posiela správy. CBC mód dešifruje zašifrované bloky C na bloky čitateľného textu P pomocou algoritmu dešifrovania $D()$ nasledovne

$$P_n = D(C_n) \oplus C_{n-1} \tag{5.1}$$

a šifruje algoritmom šifrovania $E()$ nasledovne

$$C_n = E(P_n \oplus C_{n-1}). \tag{5.2}$$

Útočník vytvorí náhodných 16 B ako falošný zašifrovaný text C', ktorý bude slúžiť ako C_{n-1} (vstup pre XOR operáciu v dešifrovaní). Falošný zašifrovaný text pošle útočník na server, ktorý ho dešifruje a vznikne P'_n. Po doplnení do rovnice vznikne

$$P'_n = D(C_n) \oplus C'. \tag{5.3}$$

Substitúciou C_n z rovnice 5.2 do rovnice 5.1 vznikne rovnica

$$P'_n = D(E(P_n \oplus C_{n-1})) \oplus C'. \tag{5.4}$$

Operácie šifrovania a dešifrovania sú opačné operácie takže sa vykrátia a vznikne rovnica

$$P'_n = P_n \oplus C_{n-1} \oplus C'. \tag{5.5}$$

Keďže útočník potrebuje zistiť hodnotu P_n stačí prehodiť hodnoty P'_n a P_n a vznikne rovnica

$$P_n = P'_n \oplus C_{n-1} \oplus C'. \tag{5.6}$$

Vznikla rovnica ktorá obsahuje 4 hodnoty:

- P_n – Neznámy dešifrovaný text originálnej správy.
- P'_n – Neznámy dešifrovaný text, ktorý patrí zašifrovanému textu C'.
- C_{n-1} – Známy zašifrovaný text originálnej správy.
- C' – Známa náhodná hodnota, ktorú útočník posiela na server.

Útočníkovi stačí získať hodnotu P'_n a dokáže vypočítať hodnotu P_n. Začne teda posielať náhodne zvolené zašifrované texty C' na server (padding oracle) spolu so zašifrovaným textom C_{n-1}. Server dešifruje blok C_{n-1} a XORuje ho s blokom C'. Ako prvé bude útočník skúšať meniť hodnotu na pozícii nultého byte. Ak server odpovie, že padding hodnota je neplatná, útočník skúsi ďalšiu hodnotu. Ak server neodpovie, že padding hodnota je neplatná, tak z toho vyplýva, že padding hodnota je platná a útočník vie, že posledný byte výsledného textu P'_n má hodnotu `0x01`. Útočník pozná, že nultý byte má hodnotu `0x01`, pretože dĺžka paddingu v tomto prípade je 1 ako bolo spomínané v úvode CBC volí padding hodnotu podľa dĺžky paddingu. Toto znamená, že keď server dešifroval text C', neohlásil útočníkovi že padding je chybný, pretože dešifrovaný text P'_n má na konci byte `0x01` a server to považuje za správny padding. V najhoršom prípade útočník vystrieda 256 hodnôt, keďže 1 B má 8 b ($2^8 = 256$) [29][30].

Keď útočník pozná 3 hodnoty zo 4, vie pomocou rovnice 5.6 získať 1 B jedného bloku. Tento proces môže opakovať na viacej bytov a dešifrovať celú správu bez znalosti súkromného kľúča.

5.5 Prevencia zraniteľností

Heartbleed

Zraniteľné verzie OpenSSL knižnice sú od 1.0.1 do 1.0.1f. Z toho vyplýva, že k zabráneniu využitia zraniteľnosti stačí aktualizovať verziu OpenSSL knižnice, ktoré implementujú kontrolovanie dĺžky obsahu v závislosti na samotnom obsahu.

CCS injection

Pre zneužitie tejto zraniteľnosti je potrebné, aby obidvaja účastníci komunikácie mali zraniteľnú verziu knižnice OpenSSL. K zabráneniu využitia zraniteľnosti stačí aktualizovať knižnicu OpenSSL. Zraniteľné verzie OpenSSL knižnice sú [31]:

- všetky verzie pred 0.9.8za,
- verzie od 1.0.0 do 1.0.0m,
- verzie od 1.0.1 do 1.0.1h.

Insecure renegotiation

Zraniteľnosti sa dá predísť použitím TLS rozšírenia s názvom Secure Renegotiation. Rozšírenie dovoľuje klientovi dokázať serveru, že si je vedomý predošlej komunikácie pomocou poslania verifikačných dát z Finished správy z prvej handshake komunikácie. Ďalší spôsob prevencie tejto zraniteľnosti je vypnutie podpory mechanizmu renegotiation na webovom serveri.

ZombiePOODLE a GOLDENDOODLE

Padding oracle útok je uskutočniteľný, iba ak obidve komunikujúce strany podporujú kryptografické sady s blokovým módom CBC. Z toho plynie, že prevencia tejto zraniteľnosti je v použití iných blokových módov ako napríklad GCM (Galois/Counter Mode).

5.6 Ďalšie podporované zraniteľnosti

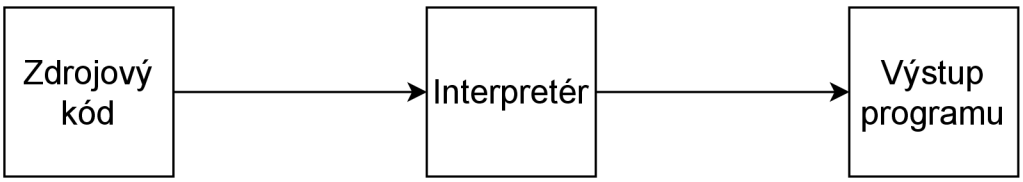
Nástroj ktorý je popísaný v kapitole 7 dokáže skenovať aj ďalšie zraniteľnosti:

- **Session ticket** podpora – ak je útočník schopný získať kľúč spojený z ticketom, vie spätne dešifrovať celú zachytenú komunikáciu [32].
- **CRIME** – zraniteľnosť v kompresnej metóde DEFLATE TLS protokolu.
- Podpora **RC4** šifry – RC4 je prúdová symetrická šifra, ktorá je považovaná za prelomenú, ale ešte neexistuje praktický útok, ktorý by oslabil bezpečnosť protokolu TLS. Keďže ale existuje mnoho bezpečnejších alternatív podpora RC4 je považovaná za zraniteľnoť [3].

6 Programovací jazyk Python

Python je vysoko orientovaný programovací jazyk [33]. Vysoko orientovaný jazyk znamená že kód v jazyku sa ďalej prekladá na iný programovací jazyk, ktorému už môže rozumieť iný jazyk na nižšej úrovni alebo už samotný hardware počítača. Python je ešte aj objektovo orientovaný, takže je možné v ňom vytvárať objekty podľa definovaných tried, ktoré podporujú bežné vlastnosti objektovo orientovaného jazyka ako napríklad dedičnosť.

Preklad jazyka Python na menšiu úroveň programovacieho jazyka prebieha pomocou interpretéra, ako je možné vidieť na obrázku 6.1. Interpretér je program, ktorý prekladá kód po častiach a potom tieto preložené časti spustí. Programy, ktoré používajú jazyk Python, sa nazývajú skripty, ktoré majú príponu .py. Pri spustení súborov s príponou .py sa použije interpretér ktorý vykoná kód ktorý sa nachádza v skripte.



Obr. 6.1: Python interpretér

Ako už bolo spomínané v úvode, Python je multiplatformový, takže dokáže spustiť rovnaký kód na rôznych operačných systémoch, až na niektoré malé zmeny ako napríklad prístupové cesty k súborom. Python je taktiež navrhnutý, aby sa v kóde jednoducho orientovalo a obsahuje mnoho pomocných funkcií, ktoré uľahčujú vývoj v tomto programovacom jazyku.

7 Základné informácie o nástroju

Ako bolo spomínané v úvode, praktická časť práce je naprogramovaná v programovacom jazyku Python verzie 3.7. Nástroj je schopný textového výpisu a výstupu do json súboru. Viacej o výstupu nástroja nasleduje v sekcii 8.8.

7.1 Spustenie nástroja a výstup v textovom rozhraní

Nástroj sa spúšťa pomocou prepínačov, ktoré sa dajú zobraziť pomocou prepínača `-h`. Výstup tohto prepínača je možné vidieť v prílohe B.1. Jediný prepínač, ktorý nástroj potrebuje pre spustenie je, `-u`, pomocou ktorého sa zadáva url adresa webového serveru. Východzie nastavenie portu, na ktorom nástroj analyzuje server, je 443, ale pomocou prepínača `-p` sa dá zvoliť iný port. V prílohe B.2 je znázornený výstup nástroja, ktorý bol spustený príkazom:

```
./tlstest.py -u vutbr.cz -ns -t 1 2
```

Výstup sa skladá z 5 častí, pričom každá časť má vlastný názov a prvky.

- **Kryptografické parametre** – bezpečnostné hodnotenie kryptografických parametrov. Do tejto kategórie patria parametre a algoritmy vyčítané z použitej kryptografickej sady a z certifikátu. Na záver časti je vypísané konečné ohodnotenie všetkých parametrov. Aký význam majú hodnotenia a ako sa určujú bude vysvetlené v nasledujúcej sekcii 7.2.
- **Podporované a nepodporované verzie protokolov** – v tejto časti sú vymenované verzie protokolu TLS, ktoré web server podporuje a nepodporuje. Verzie protokolu sú hodnotené rovnako ako v prvej časti.
- **Ostatné informácie o certifikáte** – nachádzajú sa tu ostatné informácie o certifikáte, ktorý server poslal užívateľovi počas TLS handshake. Tieto parametre nie je možné ohodnotiť.
- **Typ a verzia webového serveru** – vypísané konkrétne implementácie webových serverov, ktoré server používa. Viac o tom ako ich nástroj analyzuje, v sekcii 8.5.2.
- **Analyzované zraniteľnosti** – výsledky testov zraniteľností, ktoré si užívateľ zadal prepínačom `-t`. Testy nie sú ohodnotené pretože, výsledky testov môžu byť buď hodnoty `Yes`, alebo `No`.

7.2 Ohodnocovanie parametrov

Nástroj udáva kryptografickým parametrom úroveň bezpečnosti. Existuje 5 úrovní, ktoré je možné vidieť v nasledujúcej tabulke 7.1. Jednotlivé úrovne bezpečnosti boli navrhnuté pomocou inšpirácie z odporúčaní od organizácie NIST presnejšie, NIST SP 800-52[34] a NIST SP 800-131A[35].

Tab. 7.1: Úroveň kryptografických parametrov

Číselná hodnota	Krátky slovný popis
0	nezistené/chyba
1	bezpečné
2	neodporúčané
3	zastarané
4	zakázané

Dlhšie vysvetlenie parametrov:

- **nezistené/chyba** – predvolená hodnota parametrov, objaví, sa iba ak nastane chyba v nástroji. Napríklad ak by nástroj nemohol nájsť daný parameter alebo ho nevedel ohodnotiť.
- **bezpečné** – znamená, to že parameter je bezpečný na použitie v praxi a zatiaľ sa nenašla žiadna bezpečnostná diera ktorá by ohrozovala daný parameter.
- **neodporúčané** – parameter je použiteľný v praxi, ale ak existuje možnosť, odporúča sa použiť parameter úrovne 1. Väčšinou sú algoritmy ktoré nezabezpečujú tkz. forward secrecy[1].
- **zastarané** – parameter stále môže byť použitý ale užívateľ musí počítať so zvýšeným rizikom útoku. V mnohých prípadoch sú to algoritmy, ktoré budú v blízkej dobe zakázané.
- **zakázané** – zakázané parametre, ktoré sa už nemôžu používať v praxi.

Zhrnutie týchto parametrov znie nasledovne. Ak je možné, tak sa silne odporúča používať parametre 1. úrovne pre najväčšiu bezpečnosť, ale ak sa použijú parametre 2. alebo 3. úrovne zvyšuje sa riziko útoku. Pri 4. úrovni je riziko najväčšie.

V tabuľke 7.2 je možné vidieť odporúčané hodnoty parametrov. V treťom stĺpci sú hraničné hodnoty parametrov pre dosiahnutie bezpečnosti 1. úrovne. V prvom stĺpci tabuľky sa nachádza názov algoritmu, pre ktorý je určená minimálna hodnota algoritmu. V druhom stĺpci tabuľky sa nachádza odkaz na číslo kapitoly, v ktorom je popísaný daný parameter. Ak sa v tabuľke nachádzajú číselné hodnoty,

[1] pri komprimácií jedného súkromného kľúča sa nekomprimujú všetky

platia pre algoritmus predošlého riadku v takom istom poradí. Parametre jednotlivých bezpečnostných úrovní boli určené autorom pomocou inšpirácie z odporúčaní od organizácie NIST, presnejšie NIST SP 800-52[34] a NIST SP 800-131A[35].

Tab. 7.2: Minimálna úroveň kryptografických parametrov 1. úrovne

Názov parametru	Teoretická časť	Minimálna hodnota
Typ a verzia protokolu	2	`TLSv1.2`
Algoritmus výmenu kľúčov	1.5.3	`DHE, ECDHE`
Algoritmus verejného kľúča	1.5	`ECDSA, RSA`
Veľkosť verejného kľúča	1.5	$\geq$`224, `$\geq$`2048`
Algoritmus symetrickej šifry	1.3	`AES`
Veľkosť kľúča symetrickej šifry	1.3	$\geq$`128`
Blokový mód symetrickej šifry	1.3.2	`GCM, CCM`
Hash funkcia	1.4.2	`SHA223, SHA256`
Algoritmus podpisu certifikátu	1.5.1	`ECDSA, RSA`
Hash funkcia pre podpis certifikátu	1.4.2	`SHA223, SHA256`

Dodatočné informácie o parametroch 1. úrovne a ostatných úrovní, podľa čoho sa nástroj riadi, sa dajú nájsť v súbore `resources/security_levels.json`. Akým spôsobom nástroj vyčítava tieto parametre, bude nasledovať v sekcií 8.4

7.3 Grafické rozhranie nástroja

Nástroj je možné ovládať aj pomocou GUI (Graphical User Interface). GUI funguje na základe webovej aplikácie a umožňuje ovládanie a čítanie výstupu nástroja pomocou webovej stránky. Tento prístup umožňuje používať GUI nástroja na akomkoľvek OS, ktorý podporuje webový prehliadač.

Vstup do nástroja je možné vidieť v prílohe B.3. Stránka umožňuje zadávať všetky možnosti testovania ktoré sú relevantné k formátu grafického rozhrania. Znamená to, že prepínače ako `-i` a `-v` nie sú dostupné, keďže upravujú textový výstup nástroja.

Skrátený výstup nástroja je možné vidieť v prílohe B.4, kde sú aj znázornené hodnotené parametre skenovaného serveru. Hodnotenia môžu byť vyznačené troma farbami ktoré, korelujú s danými hodnotami bezpečnostnými úrovňami:

- úroveň **1** – zelená farba,
- úroveň **2** – oranžová farba,
- úroveň **3** a **4** – červená farba.

Keďže GUI aplikácie funguje na základe webového serveru, je ako prvé potrebné spustiť daný server. Server sa skladá z dvoch komponentov:

- **REST API** (Representational State Transfer) – tento komponent má za úlohu volať nástroj, ktorý prevádza skenovanie zadaného webového serveru a predať výsledok skenovania v JSON formáte.
- **Webová aplikácia** – úlohou webového servera je predať užívateľovu požiadavku na skenovanie pomocou REST API a následne spracovať a zobraziť výsledok užívateľovi v čitateľnej podobe.

Návod ako spustiť webový server, sa nachádza v prílohe C. Po spustení webového serveru je možné na neho pristúpiť na adrese `http://localhost:5000` (samozrejme tento prístup bude fungovať iba na počítači, na ktorom je spustený webový server).

8 Popis funkcie nástroja

Program nástroja je rozdelený na niekoľko logických častí, ktoré majú nastarosti chod programu. V nasledujúcich sekciách budú tieto logické časti detailnejšie popísané. Zoznam všetkých súborov a adresárov, ktoré korešpondujú k logickým častiam programu, sú vypísané v prílohe A.

8.1 Spustiteľný súbor tlstest.py

Nástroj začína v súbore `tlstest.py`. Keď užívateľ spustí nástroje, ako prvé sa spustí metóda `tlstest()`. Výpis tejto metódy je možné vidieť na výpise 8.1. Ako prvé sa zavolá metóda `parse_options()`, ktorá má na starosti definíciu prepínačov a triedenie vstupných argumentov nástroja podĺa definovaných pravidiel. Na triedenie je použitá knižnica `argprase`, ktorá sa natívne nachádza v Pythone. Všetky parametre sú potom uložené do premene `args`. Nasledovne ak sa v url adrese nachádza znak „`/`“ metóda `fix_url()` ju upraví do formátu koreňovej adresy. Príklad úpravy:

`https://www.vutbr.cz/login/intra` ⟶ `www.vutbr.cz`

Ďalšie metódy, ktoré sa volajú na riadkoch 20 a 21 kontrolujú, ak boli zadané prepínače pre rozšírený výstup a skenovanie webového serveru pre ďalšie HTTPS porty. Ak boli, tak sa zavolajú potrebné metódy. Na riadku 22 prebieha samotná analýza serveru na portoch pre:

- parametre z kryptografickej sady (sekcia 8.4.1),
- parameter z certifikátu (sekcia 8.4.2),
- podporované verzie TLS protokolu serverom (sekcia 8.5.1),
- konkrétne implementácie webového serveru (sekcia 8.5.2).

Po dokončení analýzy sa všetky zistené parametre uložia do premennej `output_data`. Tieto dáta sa predávajú metóde `json_options()`, ktorá má na starosti výstup nástroja podla zvolených parametrov (sekcia 8.8).

Výpis 8.1: `tlstest.py` – funkcia `tls_test()`

```
16 def tls_test():
17     args = parse_options(program_args)
18     fix_conf_option(args)
19     if '/' in args.url: args.url = fix_url(args.url)
20     info_report_option(args)
21     nmap_discover_option(args)
22     output_data = scan_all_ports(args)
23     return json_option(args, output_data)
```

8.2 Získanie potrebných objektov

Na výpise 8.2 je možné vidieť funkciu `get_website_info()`, ktorá enkapsuluje všetky ostatné funkcie v súbore `session_info.py`. Funkcie sú zodpovedné za vytvorenie HTTPS spojenia a získanie potrebných informácií o servere. Funkcia zodpovedná za vytvorenie spojenia je `create_session()`. Ako prvé vytvorí spojenie so zadaným webovým serverom a nasledovne vracia objekt `ssl_socket`, ktorý slúži na komunikáciu so serverom.

Pomocou objektu `ssl_socket` a funkcie na riadku 17 nástroj zistí ustanovenú kryptografickú sadu, typ TLS protokolu a verziu. V tejto funkcii nástroj taktiež musí kontrolovať, či je kryptografická sada v IANA formáte. Ak nie je v tom správnom formáte skonvertuje ju na správny formát.

Výpis 8.2: `connection_utils.py` – získanie objektov servera

```
14 def get_website_info(url: str, port: int):
15     ssl_socket = create_session(url, port)
16     cipher_suite, protocol =
17     get_cipher_suite_and_protocol(ssl_socket)
18     certificate = get_certificate(ssl_socket)
19     ssl_socket.close()
20     return certificate, cipher_suite, protocol
```

Ďalej nástroj zistí pomocou funkcie `get_certificate()` vrátený certifikát serveru počas handshake komunikácie. Funkcia vráti certifikát v binárnom formáte DER, ktorý je následne konvertovaný do pythonového objektu. Z neho je možné vyčítať potrebné informácie neskoršie v programe nástroja. Funkcia uzatvorí vytvorené spojenie a vráti všetky 3 zistené objekty.

8.3 Typy parametrov

Aby mohol nástroj triediť parametre, potrebuje mať vytvorené typy parametrov, ktoré sú definované v triede `PType` (Parameter Type) v súbore `PType.py`. Trieda má typ `enum` a vytvára množinu hodnôt, ktoré potom môžu byť použité ako indexy v objektoch, ktoré sa nazývajú `dictionary`. Tento typ je podobný obyčajnému poľu až na jeden rozdiel. Prvky sa môžu indexovať nielen celými číslami, ale aj ostatnými objektami, ako sú `string` alebo spomínaný `enum`. Tieto indexy sa nazývajú kľúče (`key`) a hodnota, ktorá patrí danému kľúču sa nazýva `value`. V ostatných programovacích jazykoch sa takýto objekt nazýva Mapa.

Trieda `PType` obsahuje 22 hodnôt. Na výpise 8.3 je možné vidieť prvých 8. Každá hodnota vyjadruje 1 parameter, ktorý bude zapracovaný a následne ohodnotený. Vý-

raz `auto()` umožňuje automatické číslovanie `enum` hodnôt. Táto vlastnosť je užitočná pri zmene počtu prvkov alebo pri prehodení poradia prvkov.

Trieda taktiež obsahuje 4 ďalšie vlastnosti, ktoré sa označujú pomocou anotácie `@property` nad funkciou, ktorá definuje danú vlastnosť. Vlastnosť objektu je možné získať pomocou operátora "." a názvom danej vlastnosti. Príklad môže vyzerať nasledovne pri získaní vlastnosti `key_pair`:

```
PType.hash_function.key_pair
```

Výpis 8.3: `PType.py` – typy parametrov

```
4  class PType(Enum):
5      protocol = auto()
6      kex_algorithm = auto()
7      cert_pub_key_algorithm = auto()
8      cert_pub_key_length = auto()
9      sym_enc_algorithm = auto()
10     sym_enc_algorithm_key_length = auto()
11     sym_enc_algorithm_block_mode = auto()
12     sym_ecn_algorithm_block_mode_number = auto()
```

Vlastnosti ktoré trieda obsahuje sú nasledovné:

- **`key_pair`** – vracia hodnotu algoritmu ktorý patrí k hodnote kľúča. Obsah vlastnosti je možné vidieť na výpise 8.4.
- **`is_cipher_suite`** – vráti hodnotu True, ak sa hodnota dá vyčítať z kryptografickej sady.
- **`is_certificate`** – vráti hodnotu True, ak sa hodnota dá vyčítať z certifikátu.
- **`is_ratable`** – vráti hodnotu True, ak sa hodnota dá ohodnotiť.

Výpis 8.4: `PType.py` – príklad vlastnosti

```
25 @property
26 def key_pair(self):
27     pairs = {
28         self.sym_enc_algorithm_key_length:
29             self.sym_enc_algorithm,
30         self.cert_pub_key_length:
31             self.cert_pub_key_algorithm,
32         self.sym_ecn_algorithm_block_mode_number:
33             self.sym_enc_algorithm_block_mode
34     }
35     return pairs[self]
```

8.4 Získavanie a ohodnocovanie parametrov

Keď už má nástroj zabezpečené objekty, z ktorých je možné získať parametre, začne ich triediť do kategórií definovaných v triede `PType` (sekcia 8.3). Keď sú parametre roztriedené, začne ich ohodnocovať. Tieto funkcionality sú spolu definované v 3 súboroch, ktoré budú v nasledujúcich sekciách popísané.

8.4.1 Trieda CipherSuite

Keďže sada je získaná v textovom formáte a pozície parametrov nie sú striktne, určené musia sa parametre triediť špeciálnym spôsobom. Napríklad v jednej sade sa môže nachádzať algoritmus symetrickej kryptografie na 4. pozícii (pod pojmom pozícia sa myslia časti oddelené "_". V inej sade sa môže nachádzať na 3. pozícii, ak sada neurčuje algoritmus verejného kľúča.

V tejto triede sa nachádzajú všetky potrebné metódy na triedenie parametrov sady. Hlavná metóda, ktorá triedi tieto parametre, je `parse_cipher_suite()` ktorú je možné vidieť na výpise 8.5.

Metóda ako prvé načíta súbor `cipher_parameter.json` pomocou funkcie zo súboru `utils.py`. Súbor sa nachádza v adresári `resources` a pre každý parameter, ktorý je možné vyčítať z kryptografickej sady, obsahuje jeden riadok. Po definícii typu parametru sú vypísané všetky možné hodnoty, ktoré daný parameter môže obsahovať. Formát súboru je nasledovný:

```
"TYP_PARAMETRU": "ALGORITMY,ODDELENÉ,ČIARKAMI"
```

Výpis 8.5: `CipherSuite.py` – algoritmus triedenia parametrov

```
15 def parse_cipher_suite(self):
16     json_data = read_json('cipher_parameters.json')
17     raw_parameters = self.cipher_suite.split('_')
18     raw_parameters.remove('TLS')
19     parameter_types = list(self.parameters.keys())
20     for p_raw in raw_parameters:
21         for p_type in parameter_types:
22             if p_raw in json_data[p_type.name].split(','):
23                 parameter_types.remove(p_type)
24                 self.parameters[p_type] = {p_raw: 0}
25                 break
26     for p_type in list(self.parameters.keys()):
27         if not self.parameters[p_type]:
28             self.parameters[p_type] = {'N/A': 0}
```

Následne metóda rozdelí parametre kryptografickej sady pomocou rozdeľovača "_" do listu `raw_parameters`. Do ďalšieho listu `parameter_types` zapíše všetky možné typy parametrov sady. Pre každý parameter neznámeho typu (`p_raw`) nástroj prejde list všetkých možných typov parametru (`parameter_types`). Hodnotu neznámeho typu parametru porovnáva s obsahom každého riadku súboru json, až pokým nenájde zhodu s riadkom hodnôt. Keď nájde zhodu a zistí kategóriu parametra, uloží jeho hodnotu spolu so zistenou kategóriou do premennej `self.parameters` a priradí mu hodnotenie 0. Taktiež vymaže typ nájdeného parametra z listu `parameter_types`. Nakoniec metóda pre každý nenájdený typ parametra nastaví hodnotu N/A.

8.4.2 Trieda Certificate

Podobne ako v triede `CipherSuite` v tejto triede sa nachádzajú metódy na triedenia parametrov z certifikátu. Avšak na rozdiel od triedenie parametrov kryptografickej sady sa parametre z certifikátu triedia oveľa jednoduchšie, keďže certifikát je uložený vo formáte jednoduchého pythonového objektu, v ktorom sú parametre uložené ako vlastnosti objektu. Parametre sú ale rozdelené na 2 skupiny:

- Parametre, ktoré sa **dajú** hodnotiť – parametre ako napríklad algoritmus verejného kľúča alebo veľkosť kľúča verejného kľúča. Parametre sú uložené v premennej `self.parameters`.
- Parametre, ktoré sa **nedajú** hodnotiť – tieto parametre sa ukladajú do premennej `self.non_parameters`. Sú to napríklad informácie typu verzie certifikátu alebo alternatívne mená, pre ktoré je certifikát platný.

Ako je možné vidieť na výpise 8.6, tieto premenné sa inicializujú v konštruktore triedy a k triedeniu sa využívajú vlastnosti triedy `PType` popisované v sekcii 8.3.

Výpis 8.6: `Certificate.py` – konštruktor triedy

```
8  def __init__(self, certificate: x509.Certificate):
9      super().__init__()
10     self.parameters = {enum: {} for enum in PType
11         if enum.is_certificate and enum.is_ratable}
12     self.non_parameters = {enum: [] for enum in PType
13         if enum.is_certificate and not enum.is_ratable}
14     self.certificate = certificate
```

8.4.3 Trieda Parameters

Nástroj po roztriedení ohodnotí všetky parametre z certifikátu a z kryptografickej sady. Tento proces je definovaný v triede `Parameters`. Keďže bol nástroj navrhnutý tak, aby mali premenné, ktoré obsahujú všetky získané dáta, rovnakú štruktúru,

tak je možné zovšeobecniť proces hodnotenia. To znamená že, z triedy `Parameters` dedia spomínané triedy `CipherSuite` a `Certificate` metódu `rate_parameters()`, ktorá slúži na hodnotenie parametrov. Triedy taktiež dedia vlastnosť `parameters`, do ktorej ukladajú hodnoty parametrov.

Ako je možné vidieť na výpise 8.7, metóda `rate_parameters` má 2 vstupné argumenty:

- **`ratable_parameters`** – typy parametrov, ktoré sa budú hodnotiť.
- **`key_types`** – typy parametrov, ktoré sa musia hodnotiť ako číselné parametere.

Metóda začne ohodnocovať každý typ parameteru z prvého vstupného argumentu. Ak je parameter nie je číselného typu, zavolá sa funkcia `rate_parameter`.

Výpis 8.7: `Parameters.py` – algoritmus hodnotenia parametrov

```
10 def rate_parameters(self, rateable_parameters, key_types):
11     for p_type in rateable_parameters:
12         parameter = key(self.parameters[p_type])
13         if p_type in key_types:
14             self.parameters[p_type][parameter] =
15             rate_key_length_parameter(
16                 key(self.parameters[p_type.key_pair]),
17                 parameter, p_type
18             )
19             continue
20         self.parameters[p_type][parameter] =
21         rate_parameter(p_type, parameter)
22     self.rating = self.get_max_rating()
```

Funkcia začína z načítaním súboru `security_levels.json` ktorý má nasledovný formát:

```
"TYP_PARAMETRU": {
    "1": "ALGORITMY,ODDELENÉ,ČIARKAMI",
    "2": "ALGORITMY,ODDELENÉ,ČIARKAMI",
    "3": "ALGORITMY,ODDELENÉ,ČIARKAMI",
    "4": "ALGORITMY,ODDELENÉ,ČIARKAMI"
}
```

Jednotlivé čísla vyjadrujú už popísané úrovne bezpečnosti parametrov v sekcii 7.2. Každá úroveň bezpečnosti obsahuje zoznam konkrétnych parametrov. Keď nástroje ide ohodnotiť parameter, načíta ho a prejde všetky bezpečnostné úrovne parametru, až kým nenájde zhodu s názvom analyzovaného parametra s niektorou úrovňou bezpečnosti. Úroveň, na ktorej nastala zhoda, funkcia vráti a zapíše sa.

Môže ale nastať prípad, keď ohodnocovaný parameter je číselného typu a nachádza sa vo vstupnom argumente `key_types`. V tomto prípade sa volá funkcia `rate_key_length_parameter()`. Keďže sa číselné parametre nedajú hodnotiť samostatne a sú závislé na algoritme, ktorému určujú dĺžku kľúča, posiela sa ako vstupný argument aj typ algoritmu, ktorému číselný parameter určuje hodnotu. Vďaka tomu sú úrovne bezpečnosti v json súbore definované inak. Pre každý pár veľkosti kľúča a algoritmu je definovaná jedna hodnota, ktorá sa skladá z názvu algoritmu, znamienka porovnania a číselnej hodnoty algoritmu:

```
"TYP_ČÍSELNÉHO_PARAMETRU": {
    "1": "NÁZOV_ALGORITMU,{znamienko_porovnania}ČÍSLO"
    ...
}
```

Príklad tohto zápisu pre algoritmus RSA s dĺžkou kľúča by vyzeral nasledovne:

```
"cert_pub_key_length": {
    "1": "RSA,>=2048,ECDSA,>=224,EC,>=224",
    ...
}
```

Proces hodnotenia funguje práve na základe porovnávacieho znamienka. Podobne ako pri hodnotení obyčajných parametrov nástroj prechádza všetky bezpečnostné úrovne parametru a hľadá zhodu. Najprv program nástroja hľadá zhodu nečíselného parametru (RSA, ECDSA). Potom začne porovnávať parameter na základe porovnávacieho znamienka (`>=`), ktoré nasleduje za názvom nečíselného parametru. Číselný parameter potom porovnáva s číselnou hodnotou (2048) ktorú nájde za znamienkom. Ak je porovnanie pravdivé, tak funkcia vráti rovnako ako v predošlej funkcii bezpečnostnú úroveň číselného parametru. Znamienka porovnania: `>=`(viac alebo rovné), `>>`(viac), `<=`(menej alebo rovné), `<<`(menej), `==`(rovná sa).

Nakoniec je zavolaná metóda `get_max_rating()`, ktorá vyhodnotí najhoršie hodnotenie parametrov.

8.5 Skenovanie ostatných informácií

Spolu s analýzou parametrov webového serveru nástroj poskytuje aj ďalšie funkcionality:

- Analýza podporovaných verzií protokolu TLS.
- Skenovanie implementácie webového serveru.
- Odhaľovanie ostatných portov protokolu HTTPS na danej url adrese.

Tieto ostatné funkcionality sú popísané v nasledujúcich sekciách.

8.5.1 Podporované verzie protokolu TLS

Všetky potrebné metódy na analýzu podporovaných verzií protokolu TLS sa nachádzajú v triede `ProtocolSupport`. Trieda obsahuje 2 metódy.

Nástroj využíva prvú metódu `scan_protocols()` na samotnú analýzu serveru. V tejto metóde sa vytvorí spojenie na verzii protokolu, ktorú práve testuje. Ak server nenaviaže spojenie, tak server nepodporuje túto verziu a nástroj vyskúša ďalšiu metódu. Ak práve podporuje danú verziu, pridá sa táto verzia do listu, ktorý sa nasledovne vracia. Metóda vrazia druhý list v ktorom sú zapísané verzie TLS protokolu ktoré niesu podporované serverom.

Druhá metóda vypísaná nižšie (výpis 8.8) slúži na hodnotenie analyzovaných protokolov. Každá zistená verzia je ohodnotená funkciou `rate_parameter()`. Proces hodnotenia je rovnaký ako v prípade parametrov u kryptografickej sady alebo certifikátu. Následne je zistené najhoršie hodnotenie a zapísané do premennej triedy.

Výpis 8.8: `ProtocolSupport.py` – odhaľovanie podporovaných verzií

```
39 def rate_protocols(self):
40     supported_protocols = self.scan_protocols()
41     for protocol in supported_protocols:
42         self.versions[protocol] =
43         rate_parameter(PType.protocol, protocol)
44     if not self.versions:
45         return
46     self.rating = max(self.versions.values())
```

8.5.2 Implementácia web serveru

Metódy na skenovanie implementácií analyzovaných webových serverov sa nachádzajú v triede `WebServerSoft`. Trieda obsahuje metódu `scan_server_software()`, ktorá je zodpovedná za volanie všetkých ostatných metód triedy. Túto metódu je možné vidieť na výpise 8.9.

Výpis 8.9: `WebServerSoft.py` – odhaľovanie implementácií web serveru

```
55 def scan_server_software(self):
56     scans = []
57     scans.append(self.scan_software_http)
58     if self.scan_nmap:
59         scans.append(self.scan_software_nmap)
60     for scan in scans:
61         scan()
```

V metóde sa ako prvé vytvorí list funkcií. Do tohto listu sa potom pridávajú metódy odpovedajúce typu skenovania podĺa toho, aké vstupné prepínače boli zadané. Po rozhodovaní, aké metódy sa pridajú do listu, sú všetky zavolané. Vďaka tomuto prístupu je možné v budúcnosti pridať viac metód skenovania implementácie webového serveru bez výraznej zmeny. Momentálne podporuje nástroj skenovanie implementácie webového serveru 2 metódami:

- Pomocou programu namp – využíva sa obálka programu nmap v pythone, ktorá umožňuje získať informácie o webovom serveri.
- Pomocou HTTP response hlavičky – na server sa posiela HEAD request, na ktorý server odpovedá HTTP hlavičkou, v ktorej sú uložené informácie o web serveri.

8.5.3 Odhaľovanie ostatných portov

Tento proces je implementovaný vo funkcii `discover_ports()`, ktorá sa nachádza v súbore `port_discovery.py`. Funkcia opätovne využíva obálku programu nmap, pomocou ktorého sú najprv zistené všetky otvorené porty na analyzovanom serveri. Pre každý nájdený port nástroj posiela HTTP HEAD požiadavku aby zistil, či sa na tomto porte nachádza spustená inštancia webového serveru. List validných portov potom funkcia vráti.

8.6 Analýza zraniteľnosti

Nástroj ako posledný krok analyzuje, či je webový server zraniteľný na užívateľom zvolené zraniteľnosti. Implementácia analýzy zraniteľností záleží na povahe samotnej zraniteľnosti. Preto je pre každý test vytvorený súbor, v ktorom sa nachádzajú všetky potrebné funkcie na otestovanie danej zraniteľnosti. Súbory sa nachádzajú v zložke `scan_vulnerabilities`, ktorej obsah je možné vidieť v prílohe A. Všetky zraniteľnosti ktoré nástroj dokáže testovať sú popísané v kapitole 5.

Každý test môže trvať niekoľko sekúnd, keďže nástroj musí čakať na odpovede od servera. Testy preto nie sú spúšťané sekvenčne, ale paralelne pomocou python vstavanej knižnice `concurrent.futures` [36]. Knižnica umožňuje vytvorenie buď nového procesu, alebo nového vlákna pre každé zavolanie funkcie. Nástroj využíva túto knižnicu vo funkcii `scan_vulnerabilities()`, ktorú je možné vidieť na výpise 8.10. Vo funkcií je využitá trieda `ThreadPoolExecutor`, ktorá vytvára nové vlákno pre každú zavolanú funkciu.

Ďalej nástroj pre každý zvolený test vytvorí nové vlákno pomocou `executor` objektu. Pre každé vlákno (`future`) sa vyhodnotí návratová hodnota testu a zapíše sa do premennej `output`, ktorá obsahuje výsledky všetkých testov. Táto premenná je potom ďalej spracovávaná na výstupe nástroju.

Výpis 8.10: `multitheard_scan.py` – paralelné testovanie

```
4  def scan_vulnerabilities(tests: list, address: tuple):
5      output = {}
6      futures = {}
7      with cf.ThreadPoolExecutor(max_workers=len(tests))
8          as executor:
9          for test in tests:
10             futures.update(
11                 {executor.submit(test[0], address): test[1]}
12             )
13         for future in cf.as_completed(futures):
14             test_name = futures[future]
15             data = future.result()
16             output.update({test_name: data})
17     return output
```

8.7 Konfigurácia OpenSSL knižnice

Keď nástroj analyzuje webové servery, ktoré podporujú iba protokol TLS 1.0 alebo TLS 1.1, tak je možné, že počas behu programu nástroja nastane nasledovná chyba:

```
SSL: UNSUPPORTED_PROTOCOL
```

Táto chyba oznamuje, že daný SSL/TLS protokol nie je momentálne podporovaný. Toto je spôsobené tým, že keď python vytvára zabezpečené spojenie pomocou protokolu TLS, používa pritom knižnicu OpenSSL. Táto knižnica má v niektorých prípadoch v konfiguračnom súbore nastavený minimálny podporovaný protokol a vďaka tomu nie je schopná vytvoriť spojenie z nižšími verziami protokolu TLS ako je zadané v konfiguračnom súbore.

Tento problém je možné riešiť zmenou minimálnej podporovanej hodnoty protokolu TLS v zmienenom konfiguračnom súbore `/etc/ssl/openssl.cnf` (iba v Linuxových OS). Zmena sa deje automaticky pomocou skriptu, ktorý sa nachádza v koreňovom adresári praktickej časti s názvom `fix_openssl_config.py`. Skript je možné spustiť samostatne ako python skript alebo pomocou prepínača `-fc` pri

spúštaní hlavného programu. Skript je taktiež možné spustiť pri spúštaní skriptu `start.sh` pomocou prepínača `-c`. Nakoniec sa skript automaticky spúšťa aj pri zostavovaní docker kontajneru pre webovú aplikáciu.

8.8 Výstup nástroja

Keďže všetky informácie, ktoré nástroj zistil, boli ukladané do pythonových objektov typu `dictionary`, je jednoduché ich skonvertovať do formátu json. Najprv sú json dáta uložené v dlhom reťazci. Ak bol pri spustení nástroju pridaný prepínač -j, program nástroja uloží tento reťazec do json súboru, ktorý bol zvolený spolu s prepínačom alebo ho vypíše ak nebol zadaný názov súboru.

Ak ale prepínač `-j` nebol špecifikovaný pri štarte, tak nástroj postupuje vypísaním všetkých zistených informácií na štandardný výstup (stdout). Tento proces je definovaný v metóde `text_output()` v triede `TextOutput`. Metódu je možné vidieť vo výpise 8.11. Pri vytvorení objektu triedy sa pomocou konštruktu uložia json dáta do premennej `self.data`. Json dáta sú nasledovne v metóde prekonvertované naspäť na dictionary objektov. Pre každý port analyzovaného web serveru sú vypísané údaje pomocou ďalších metód v triede, keďže každá kategória zistených informácií má odlišný formát.

Výpis 8.11: `TextOutput.py` – textový výstup

```
19 def text_output(self):
20     json_data = json.loads(self.data)
21     for key, value in list(json_data.items()):
22         print(f'Result␣for␣{key}')
23         self.current_data = value
24         self.print_parameters(
25             self.current_data['parameters'])
26         self.print_supported_versions(
27             self.current_data['protocol_support'])
28         self.print_certificate_info(
29             self.current_data['certificate_info'])
30         self.print_versions(
31             self.current_data['web_server_versions'])
32         self.print_vulnerabilities(
33             self.current_data['vulnerabilities'])
```

Nástroj taktiež podporuje prepínač `-v`, pomocou ktorého je nástroj prepnutý do verbose režimu. V tomto režime nástroj vypisuje ladiace informácie, ako napríklad s kým vytvára spojenie a či bolo úspešné. Dodatočné informácie sa ale neukladajú do json súboru, sú dostupné iba na štandardnom chybovom výstupe (stderr).

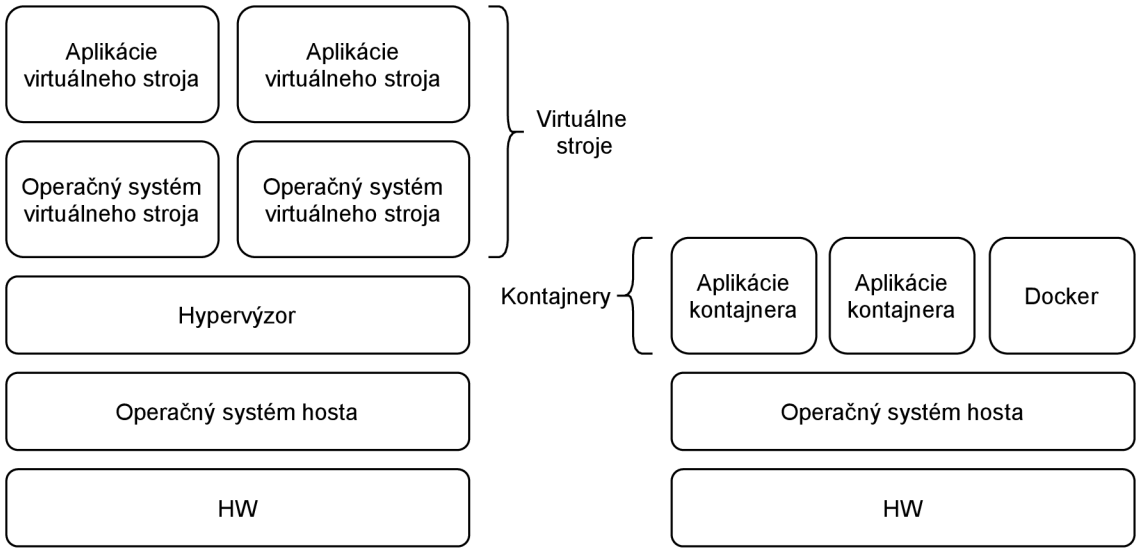
9 Testovanie nástroja

V tejto kapitole je popísané, ako sa nástroj testoval počas vývoja na zaistenie správnej funkcionality a spôsob a výsledky testovania programu v konečnom stave.

9.1 Testovanie počas vývoja

Testovanie je dôležitá časť vývoja akéhokoľvek softvéru, pretože zaisťuje, že aplikácia alebo program, ktorý je vyvíjaný, funguje podľa očakávaní a bez chýb. Preto bol aj nástroj popisovaný v tejto práci testovaný. Aby mohol byť nástroj testovaný, bola potreba ako prvé vytvoriť testovacie prostredie, v tomto prípade webový server. Je dôležité aby sa tento webový server dal ľahko modifikovať, aby bolo možné otestovať nástroj vo viacerých situáciách.

Týmto požiadavkám vyhovoval virtualizačný software Docker. Docker umožňuje vytvárať izolované prostredia (kontajnery), ktoré sú spustené nad kernelom operačného systému (viď obrázok 9.1) na rozdiel od bežných virtualizačných nástrojov ako hypervízory, ktoré majú svoj vlastný virtuálny operačný systém. Tento princíp umožňuje vytváranie a spúšťanie kontajnerov v krátkom čase a taktiež umožňuje spúšťanie mnohých kontajnerov naraz.



Obr. 9.1: Rozdielné virtualizačné architektúry

Docker ďalej využíva mechanizmus copy-on-write, vďaka ktorému je možné kontajnery vytvárať a meniť v minútach času. Ak by bola potreba napríklad zmeniť verziu OpenSSL knižnice kontajneru na otestovanie zraniteľnosti, tak stačí len zmeniť konfiguračný súbor, ktorý vytvára kontajner a Docker namiesto toho, aby znova

vytváral celé prostredie od začiatku jednoducho využije už vytvorený kontajner a zmení len nainštalovanú verziu OpenSSL knižnice vďaka spomínanému copy-on-write mechanizmu [39].

Pomocou Dockeru je vytvorený zraniteľný webový server, ktorý je hostovaný na samostatnom serveri, aby bolo možné nástroj testovať z akékoľvek operačného systému a aby sa jednoducho dala meniť konfigurácia webového serveru pre účely testovania. Použitá implementácia webového servera je nginx na operačnom systéme Linux debian. Nástroj je počas vývoja testovaný na operačných systémoch:

- Linux distribúcia **Mint** nainštalovaná na fyzickom HW,
- Linux distribúcia **Debian** na dockeru (samotný webový server),
- Linux distribúcia **Ubuntu** na WSL (Windows Subsystem for Linux).

9.2 Praktické testovanie

Aby bola overená správna funkčnosť nástroju, nástroj je testovaný aj na reálnych webových serverov. Výsledky týchto testov je možné vidieť v tabuľke 9.1. V prvom stĺpci sa nachádza url adresa analyzovaného webového serveru a v ďalších 2 stĺpcoch tabuľky sa nachádzajú hodnotenia jednotlivých kategórií.

Tab. 9.1: Testovanie webových serverov

Webový server	Kryptografické parametre	Podpora protokolu TLS
vutbr.cz	2	2
fekt.vutbr.cz	1	2
www.linux.cz	4	4
github.com	1	1

Nižšie je možné vidieť, prečo boli dané webové servery ohodnotené danou hodnotou.

vutbr.cz

- Kryptografické parametre:
  - Algoritmus pre výmenu kľúč: RSA(2),
  - Blokový mód symetrickej šifry: CBC(2),
  - Hash funkcia: SHA(2).
- Podpora protokolu TLS: TLS 1.0(2), TLS 1.1(2), TLS 1.2(1).

fekt.vutbr.cz

- Kryptografické parametre: všetky parametre boli úrovne 1.
- Podpora protokolu TLS: TLS 1.0(2), TLS 1.1(2), TLS 1.2(1), TLS 1.3(1).

www.linux.cz

- Kryptografické parametre:
  - Niektoré parametre boli úrovne 2 ako napríklad SHA hash funkcia.
  - Neplatný certifikát(4), momentálne možno už bude platný ale v čase skenovania nebol certifikát platný.
- Podpora protokolu TLS: nie je podporovaný protokol TLS 1.2 ani TLS 1.3(4).

github.com

- Všetky parametre boli bezpečnostnej úrovne 1. Dobre zabezpečený server.

10 Záver

Ako bolo spomínané v úvode práca mala za účel zoznámiť čitateľa so základmi protokolu TLS. Na začiatku práca popisovala kryptografické základy ako hašovacie funkcie, symetrické a asymetrické šifry, ktoré sú potrebné k pochopeniu bezpečnosti na internete ako vcelku. Ďalej bol popísaný už samotný protokol TLS a jeho štruktúra. Ako posledný bol popísaný spôsob autentifikácie pomocou infraštruktúry PKI a certifikátov. Nakoniec sa všetky tieto koncepty združili pod protokol HTTPS, ktorý využíva každý z týchto konceptov na vytvorenie bezpečných spojení s webovými servermi.

Ďalej práca popisovala konkrétne zraniteľnosti, ktoré je nástroj schopný analyzovať. V nasledujúcich kapitolách bol popísaný už samotný nástroj. Boli popísané funkcionality nástroja, bezpečnostné úrovne, ktoré program využíva a GUI nástroja. Práca ďalej popisovala, ako nástroj funguje a boli popísané jednotlivé logické časti, z ktorých sa program nástroja skladá. Nástroj je teda schopný analyzovať použité algoritmy a parametre použité pri zostavovaní zabezpečeného spojenia so serverom, podporované a nepodporované verzie TLS protokolu webovým serverom a dokáže zisťovať, akú konkrétnu implementáciu webového serveru používa webová stránka.

Na záver práce sú popísané programy, ktoré boli využívané na testovanie nástroja a akým spôsobom boli využívané. Jeden z týchto nástrojov bol vizualizačný sowftware Docker. Na koniec sú popísané konkrétne výsledky testovania niektorých webových serverov.

Literatúra

[1] ISO/IEC 7498-1:1994. *Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model*. ISO, 1994, 59 s. Dostupné také z: https://www.iso.org/standard/20269.html

[2] The CIA triad defined. *Forcepoint* [online]. 2018 [cit. 2020-11-13]. Dostupné z: https://www.forcepoint.com/cyber-edu/cia-triad

[3] RISTIĆ, Ivan. *Bulletproof SSL and TLS: Understanding and Deploying SSL/TLS and PKI to Secure Servers and Web Applications Ivan Ristic*. 6 Acantha Court, Montpelier Road, London W5 2QP, United Kingdom: Feisty Duck, 2014. ISBN 978-1-907117-04-6.

[4] PETITCOLAS, Fabien. Kerckhoffs' principles from La cryptographie militaire. *The information hiding homepage* [online]. [cit. 2020-10-17]. Dostupné z: https://petitcolas.net/kerckhoffs/index.html

[5] BARKER, Elaine a Nicky MOUHA. NIST SP 800-67. *Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher*. 2nd ed. NIST Pubs, 2017, 32 s. Dostupné také z: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-67r2.pdf

[6] DWORKIN, Morris, Elaine BARKER, James NECHVATAL, James FOTI, Lawrence BASSHAM, E. ROBACK a James DRAY. FIPS PUB 197. *Specification for the Advanced Encryption Standard*. NIST Pubs, 2001, 51 s. Dostupné také z: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

[7] NIR, Y. a A. LANGLEY. RFC 7539. *ChaCha20 and Poly1305 for IETF Protocols*. Internet Engineering Task Force, 2015, 45 s. Dostupné také z: https://tools.ietf.org/html/rfc7539

[8] FIPS PUB 198-1. *The Keyed-Hash Message Authentication Code*. Information Technology Laboratory National Institute of Standards and Technology Gaithersburg, MD 20899-8900: NIST Pubs, 2008, 13 s. Dostupné také z: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.198-1.pdf

[9] RIVEST, Ronald. RFC 1321. *The MD5 Message-Digest Algorithm*. MIT Laboratory for Computer and RSA Data Security, Inc., 1992, 21 s. Dostupné také z: https://datatracker.ietf.org/doc/rfc1321/?include_text=1

[10] OPPLIGER, Rolf. *SSL and TLS Theory and Practice*. 685 Canton Street Norwood, MA 02062: ARTECH HOUSE, 2009. ISBN 978-1-59693-447-4.

[11] WANG, Xiaoyun, Yiqun Lisa YIN a Hongbo YU. Finding Collisions in the Full SHA-1. *MIT Computer Science and Artificial Intelligence Lab* [online]. Shandong University, Jinan 250100, China, 2005, s. 20 [cit. 2020-10-17]. Dostupné z: https://people.csail.mit.edu/yiqun/SHA1AttackProceedingVersion.pdf

[12] FIPS PUB 180-4. *Secure Hash Standard*. Information Technology Laboratory National Institute of Standards and Technology Gaithersburg, MD 20899-8900: NIST, 2015, 36 s. Dostupné také z: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

[13] FIPS PUB 202. *SHA-3 standard: permutation-based hash and extendable output functions*. Information Technology Laboratory National Institute of Standards and Technology Gaithersburg, MD 20899-8900: NIST, 2015, 37 s. Dostupné také z: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

[14] FIPS PUB 186-4. *Digital Signature Standard*. Information Technology Laboratory National Institute of Standards and Technology Gaithersburg, MD 20899-8900, 2013, 130 s. Dostupné také z: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

[15] SMART, Nigel. *Cryptography: An Introduction* [online]. 3rd. ed. McGraw-Hill College, 2004 [cit. 2020-10-18]. ISBN 978-0077099879. Dostupné z: https://www.cs.umd.edu/~waa/414-F11/IntroToCrypto.pdf

[16] RESCORLA, Eric. RFC 8446. *The Transport Layer Security (TLS) Protocol Version 1.3*. Internet Engineering Task Force, 2018, 160 s. Dostupné také z: https://tools.ietf.org/html/rfc8446

[17] NIR, Yoav, Rich SALZ a Nick SULLIVAN. Transport Layer Security (TLS) Parameters. *Internet Assigned Numbers Authority* [online]. 12025 Waterfront Drive, Suite 300, Los Angeles CA 90094, USA: IANA, 2005 [cit. 2020-10-30]. Dostupné z: https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml

[18] SALZ, Rich, Nick NIR a Yoav SULLIVAN. Transport Layer Security (TLS) Extensions. *IANA* [online]. 12025 Waterfront Drive, Suite 300, Los Angeles CA 90094, USA: IANA, 2005 [cit. 2020-10-31]. Dostupné z: https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml

[19] COOPER, D., S. SANTESSON, S. FARRELL, S. BOEYEN, R. HOUSLEY a W. POLK. RFC 5280. *Internet X.509 Public Key Infrastructure Certificate*

and Certificate Revocation List (CRL) Profile. Internet Engineering Task Force, 2008, 151 s. Dostupné také z: https://tools.ietf.org/html/rfc5280

[20] TURNER, S. a T. POLK. RFC 6176. *Prohibiting Secure Sockets Layer (SSL) Version 2.0*. Internet Engineering Task Force, 2011, 7 s. Dostupné také z: https://tools.ietf.org/html/rfc6176

[21] BARNES, R., M. THOMSON, A. THOMSON a A. LANGLEY. RFC 7568. *Deprecating Secure Sockets Layer Version 3.0*. Internet Engineering Task Force, 2015, 7 s. Dostupné také z: https://tools.ietf.org/html/rfc7568

[22] CVE-2014-3566. *Common Vulnerabilities and Exposures* [online]. The MITRE Corporation, 2014 [cit. 2020-11-26]. Dostupné z: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-3566

[23] DIERKS, T. a E. RESCORLA. RFC 4346. *The Transport Layer Security (TLS) Protocol Version 1.1*. Internet Engineering Task Force, 2006, 86 s. Dostupné také z: https://tools.ietf.org/html/rfc4346

[24] CVE-2011-3389. *Common Vulnerabilities and Exposures* [online]. The MITRE Corporation, 2011 [cit. 2020-11-26]. Dostupné z: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-3389

[25] DIERKS, T. a E. RESCORLA. RFC 5246. *The Transport Layer Security (TLS) Protocol Version 1.2*. Internet Engineering Task Force, 2008, 104 s. Dostupné také z: https://tools.ietf.org/html/rfc5246

[26] GOURLEY, David, Brian TOTTY, Marjorie SAYER, Anshu AGGARWAL a Sailu REDDY. *HTTP: The Definite Guide: Understanding Web Internals*. O'Reilly Media, 2002, 658 s. ISBN 9781565925090.

[27] SEGGELMANN, R., M. TUEXEN a M. WILLIAMS. 6520. *Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension*. Internet Engineering Task Force, 2012, 9 s. Dostupné také z: https://tools.ietf.org/html/rfc6520

[28] CCS Injection Vulnerability. *Lepidium* [online]. Lepidum Co., 2014 [cit. 2021-03-27]. Dostupné z: http://ccsinjection.lepidum.co.jp/

[29] BOWES, Ron. A padding oracle example. *Skullsecurity* [online]. 2013 [cit. 2021-02-07]. Dostupné z: https://blog.skullsecurity.org/2013/a-padding-oracle-example

[30] BOWES, Ron. Padding oracle attacks: in depth. *Skullsecurity* [online]. 2013 [cit. 2021-02-07]. Dostupné z: https://blog.skullsecurity.org/2013/padding-oracle-attacks-in-depth

[31] CVE-2014-0224. *Common Vulnerabilities and Exposures* [online]. The MITRE Corporation, 1999 [cit. 2020-11-26]. Dostupné z: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0224

[32] VALSORDA, Filippo. We need to talk about Session Tickets. *Filippo.io* [online]. 2017 [cit. 2021-5-20]. Dostupné z: https://blog.filippo.io/we-need-to-talk-about-session-tickets/

[33] DOWNEY, Allen. *Think Python*. Needham, Massachusetts: Green Tea Press, 2012, 240 s. ISBN 144933072X.

[34] MCKAY, Kerry A. a David A. COOPER. NIST SP 800-52. *Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations*. 2nd ed. NIST, 2019, 72 s. Dostupné také z: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf

[35] BARKER, Elaine a Allen ROGINSKY. NIST SP 800-131A. *Transitioning the Use of Cryptographic Algorithms and Key Lengths*. Computer Security Division, Information Technology Laboratory 100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930: NIST Pubs, 2019, 33 s. Dostupné také z: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar2.pdf

[36] concurrent.futures – Launching parallel tasks. *Python* [online]. Python software foundation, 2001 [cit. 2021-03-28]. Dostupné z: https://docs.python.org/3/library/concurrent.futures.html

[37] GitHub-pyopenssl. *GitHub* [online]. [cit. 2020-11-18]. Dostupné z: https://github.com/pyca/pyopenssl

[38] GAYNOR, Alex, Alex CHAN, Donald STUFFT, et al. GitHub-pyca. *GitHub* [online]. GitHub [cit. 2020-11-18]. Dostupné z: https://github.com/pyca

[39] TURNBULL, James. *The Docker Book*. 2018. ISBN 0988820234.

Zoznam symbolov, veličín a skratiek

AES	Advanced Encryption Standard
API	Application Programing Interface
ASN.1	Abstract Syntax Notation One
CBC	Cipher Block Chaining
CCS	CangeCipherSpec
CFB	Cipher FeedBack
CRC	Cyclic Redundancy Check
DER	Distinguished Encoding Rules
DES	Data Encryption Standard
DN	Distinguished name
DSA	Digital Signature Algorithm
ECB	Electronic Code Book
ECDSA	Eliptic Curve Digital Signature Algorithm
ECDH	Eliptic Curve Diffie-Hellman
GCM	Galois/Counter Mode
GUI	Graphical User Interface
HMAC	Hashed-based Message Authentication Code
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ISO	International Organization for Standardization
IV	Inicialization Vector
MAC	Messege Authentication Code
MitM	Man in the Middle
OSI	Open Systems Interconnection

PDU	Protocol Data Unit
PEM	Privacy-Enchanced Mail
PKI	Public Key Infrastructure
POODLE	Padding Oracle On Dowgraded Legacy Ecryption
REST	Representational State Transfer
RFC	Request For Comment
RSA	Rivest Shamir Adleman
SHA	Secure Hashing Algorithm
SSL	Secure Socket Layer
TDES	Tripple Data Encryption Standard
TLS	Transport Layer Security
WSL	Windows Subsystem for Linux
WWW	World Wide Web

A Obsah adresára praktickej časti

- BP
  - resources ..... všetky json súbory, ktoré nástroj využíva
    - cipher_parameters.json
    - iana_openssl_cipher_mapping.json
    - security_levels.json
    - security_levels_names.json
    - type_names.json
  - server_app 7.3
    - static/styles
      - style_result.css
      - style_query.css
    - templates
      - query_form.html
      - query_result.html
    - server.py
    - utils.py
  - scan_parameters
    - connection
      - connection_utils.py 8.2
    - exceptions ..... výnimky, ktoré nástroj vyhadzuje
      - ConnectionTimeoutError.py
      - DNSError.py
      - NoIanaPairFound.py
      - UnknownConnectionError.py
    - non_ratable
      - port_discovery.py 8.5.3
      - ProtocolSupport.py 8.5.1
      - WebServerSoft.py 8.5.2
    - ratable
      - Certificate.py 8.4.2
      - CipherSuite.py 8.4.1
      - Parameters.py 8.4.3
      - PType.py 8.3
    - utils.py ..... súbor s pomocnými funkciami
  - ...

B Výstupy praktickej časti

B.1 Výstup prepínača -h

```
usage: use -h or --help for more information

optional arguments:
  -h, --help            show this help message and exit
  -ns, --nmap-scan      use nmap to scan the server version
  -nd, --nmap-discover  use nmap to discover web server ports
  -p port [port ...], --port port [port ...]
                        port or ports (separate with spaces)
                        to scan on (default: [443])
  -j [output_file], --json [output_file]
                        change output to json format, if a
                        file name is specified output is
                        written to the given file
  -t test_num [test_num ...], --test test_num [test_num ...]
                        test the server for a specified
                        vulnerability possible vulnerabilities
                        (separate with spaces):
                            1: Heartbleed
                            2: ChangeCipherSpec Injection
                            3: Insecure renegotiation
                            4: ZombiePOODLE/GOLDENDOODLE
                            5: Session ticket support
                            6: CRIME
                            7: RC4 support
  -fc, --fix-conf       allow the use of older versions of
                        TLS protocol (TLSv1 and TLSv1.1) in
                        order to scan a server which still
                        run on these versions.
                        !WARNING!: this may rewrite the
                        contents of a configuration file
                        located at /etc/ssl/openssl.cnf
                        backup is recommended, root
                        permission required
  -i, --information     output some information
  -v, --verbose         output more information

required arguments:
  -u url, --url url     url to scan
```

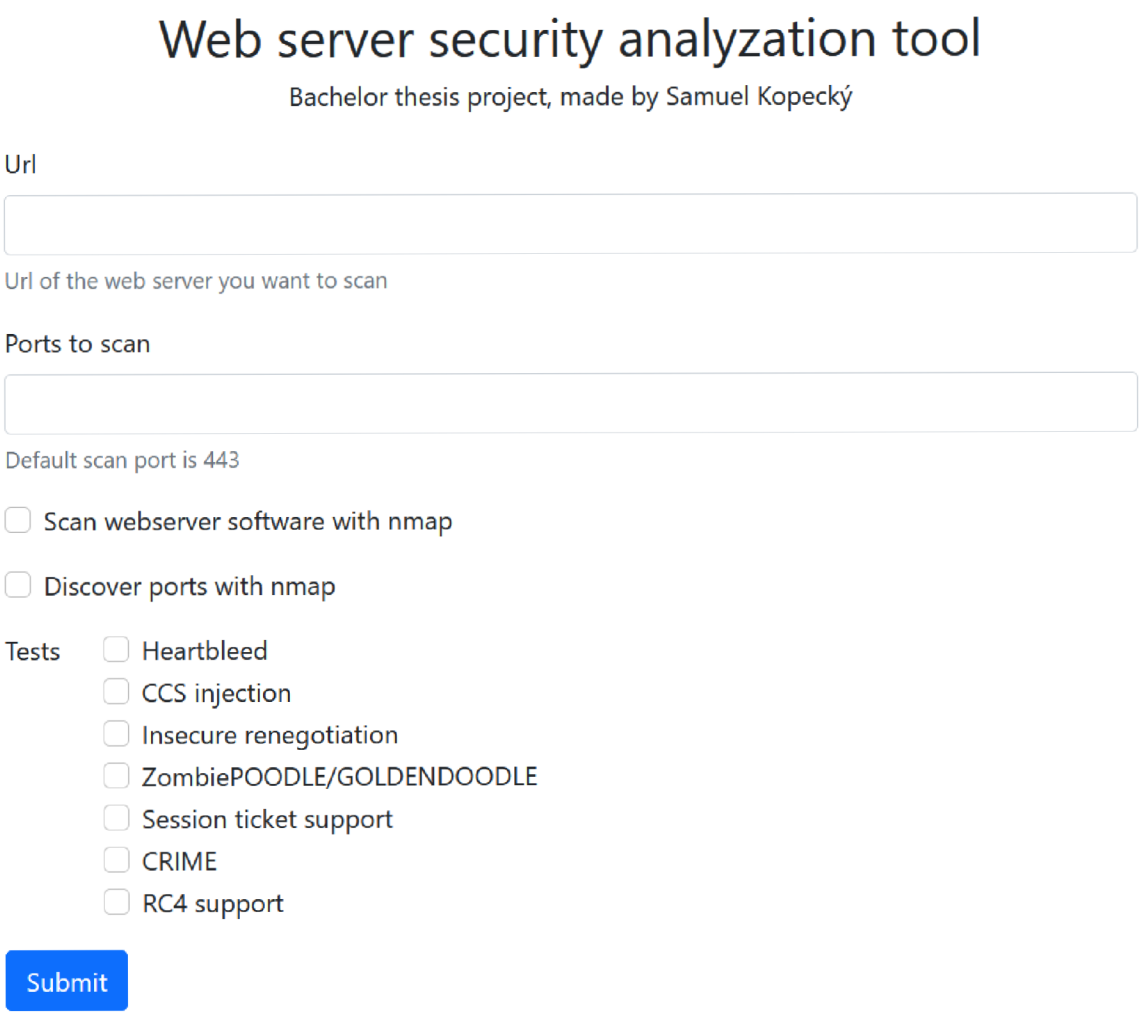
Obr. B.1: Výstup prepínača **-h**

B.2 Výstup nástroja

```
Cryptographic parameters:
   Key exchange algorithm: RSA->2
   Symmetric encryption algorithm: AES->1
   Symmetric encryption algorithm length: 128->1
   Symmetric encryption algorithm block mode: CBC->2
   Hash function: SHA->2
   Protocol: TLSv1.2->1
   Public key algorithm: RSA->1
   Public key length: 2048->1
   Certificate signature algorithm: RSA->1
   Certificate signature algorithm hash function: SHA256->1
   rating: 2
Protocol support:
   Supported protocols:
      TLSv1->2
      TLSv1.1->2
      TLSv1.2->1
   Unsupported protocols:
      TLSv1.3->2
   rating: 2
Certificate information:
   Certificate versions: 2
   Certificate serial number: 145211...
   Valid from: 2020-04-21
   Valid until: 2021-04-26
   Subject:
      countryName=CZ
      localityName=Brno
      organizationName=Vysoké učení technické v Brně
      commonName=www.vutbr.cz
   Issuer:
      countryName=NL
      stateOrProvinceName=Noord-Holland
      localityName=Amsterdam
      organizationName=TERENA
      commonName=TERENA SSL CA 3
   Alternative names:
      vut.cz
Web server software:
   Http header: Apache
   nmap: Apache httpd
Scanned vulnerabilities:
   CSS injection->No
   Heartbleed->No
```

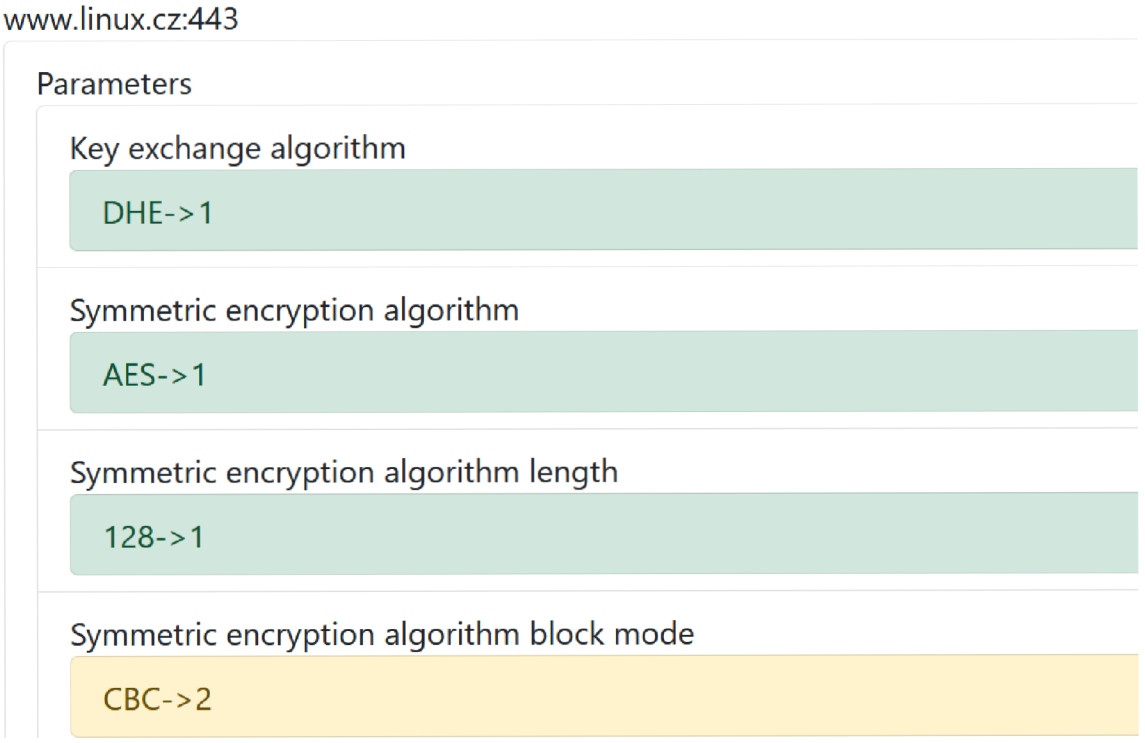
Obr. B.2: Výstup nástroja

B.3 GUI nástroja

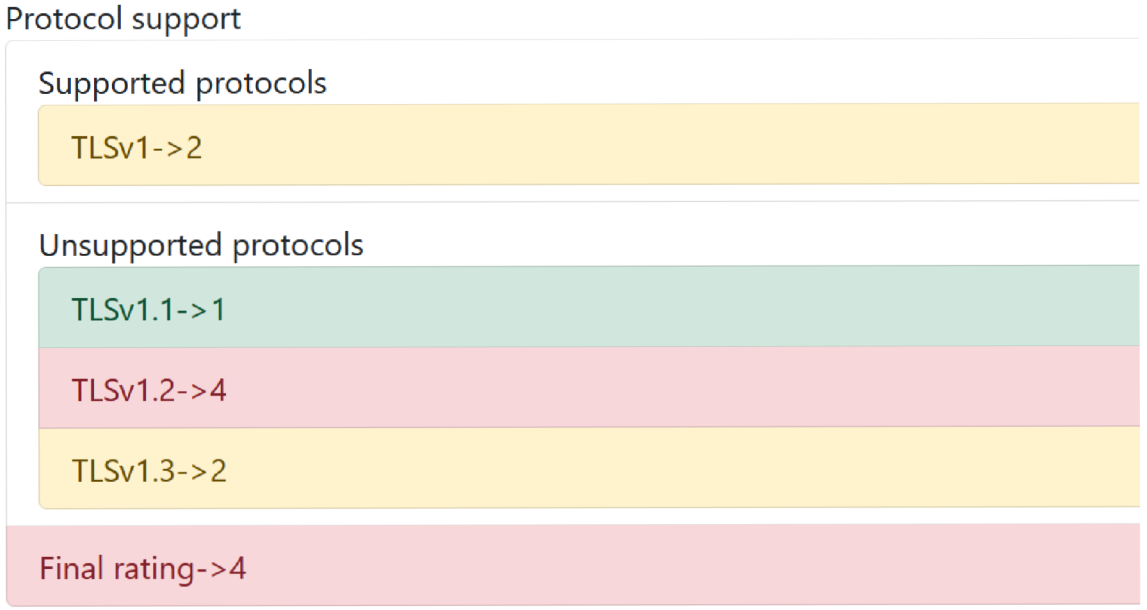


Obr. B.3: GUI nástroja

B.4 Príklad výstupu GUI nástroja



Obr. B.4: Ohodnotené parametre v GUI pre `www.linux.cz`



Obr. B.5: Ohodnotené podporované TLS verzie v GUI pre `www.linux.cz`

C Návod na nasadenie webového serveru

Webový server je možné nasadiť dvoma spôsobmi.

C.1 V prostredí hostovacieho OS Linux

Ako prvé je potreba nainštalovať potrebné knižnice, ktoré python potrebuje na spustenie webového serveru a skriptu. Ak sa užívateľ práve nachádza v koreňovom adresári projektu, je možné ich ľahko doinštalovať pomocou nasledujúceho príkazu `pip`:

- `pip3 install -r requirements.txt`

Taktiež je treba nainštalovať program nmap, ktorý je využívaný v niektorých funkciách nástroja pomocou príkazu (pre iné Linux distribúcie sa môže príkaz meniť):

- `apt install -y nmap`

Po nainštalovaní potrebných knižníc stačí spustiť shell skript:

- `start.sh`

Alebo spustiť nasledujúci príkaz pre zistenie informácií o dostupných prepínačoch:

- `start.sh -h`

C.2 Pomocou docker kontajneru pre ostatné OS

Webovú aplikácie je možné spustiť aj pomocou vytvorení docker kontajneru. Predtým je ale potrebné, aby užívateľ mal nainštalovaný:

- Docker engine – `https://docs.docker.com/engine/install/`
- Docker compose – `https://docs.docker.com/compose/install/`

Po nainštalovaní týchto dvoch programov stačí v koreňovom adresári projektu spustiť príkaz:

- `docker-compose up -d`