VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

# BAKALÁŘSKÁ PRÁCE

Brno, 2020 Slavomíra Dzadíková


# 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


# WEBOVÁ APLIKACE PRO PŘENOS DAT S VYUŽITÍM TLS PROTOKOLU

SEMESTRÁLNÍ PRÁCE
SEMESTRAL THESIS


AUTOR PRÁCE
AUTHOR
Slavomíra Dzadíková

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. David Smékal


BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

# Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

***Studentka:*** Slavomíra Dzadíková ***ID:*** 203170
***Ročník:*** 3 ***Akademický rok:*** 2019/20

**NÁZEV TÉMATU:**

## Webová aplikace pro přenos dat s využitím TLS protokolu

**POKYNY PRO VYPRACOVÁNÍ:**

Cílem bakalářské práce je návrh a implementace webové aplikace realizující přenos dat mezi klientem a serverem pomocí protokolu TLS včetně bezpečné autentizace uživatele.
V rámci práce bude navrženo uživatelské rozhraní aplikace (GUI), zprovoznění webové prezentace, TLS protokolu a implementace vícefaktorového autentizačního schématu.
Aplikace bude podporovat logování událostí a uživatel bude mít možnost volby z více kryptografických sad pro šifrování uložených dat na serveru. V bakalářské práci bude realizováno experimentální prostředí pro prezentaci funkčnosti autentizace, bezpečného navázání spojení klient-server a přenosu dat.
Závěrem otestujte bezpečnost webové aplikace, zhodnoťte dosažené výsledky a diskutujte další pokračování práce.

**DOPORUČENÁ LITERATURA:**

[1] CASTRO, Elizabeth a Bruce HYSLOP. HTML5 a CSS3: názorný průvodce tvorbou WWW stránek. Brno: Computer Press, 2012, 439 s.ISBN 978-80-251-3733-8.

[2] RESCORLA E. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, 2018, DOI: 10.17487/RFC8446. Dostupné z: https://www.rfc-editor.org/info/rfc8446

***Termín zadání:*** 3.2.2020 ***Termín odevzdání:*** 8.6.2020

***Vedoucí práce:*** Ing. David Smékal
***Konzultant:*** Ing. Tomáš Mácha, Ph.D. (ANECT a.s.)

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

## ABSTRAKT

Práca sa zaoberá tvorbou webových aplikácií, možnosťami implementácie webových aplikácií, bezpečnou komunikáciou medzi klientskou a serverovou časťou. Bližšie sú popísané protokoly HTTPS (Hyper Text Transfer Protocol Secure) a protokol TLS (Transport Layer Secure) s čím súvisí aj problematika infraštruktúry verejných kľúčov (PKI). V práci sú rozobraté aj možnosti autentizácie a autorizácie vo webových aplikáciach a tiež najčastejšie útoky podľa OWASP TOP 10. Použité technológie a programovacie jazyky, prostredia: Python, Flask, Bootstrap, OpenSSL, Nginx, Nessus, JMeter, Lighthouse.



## KĽÚČOVÉ SLOVÁ

webová aplikácia, webový server, HTTPS, TLS, infraštruktúra verejných kľúčov, Python, Flask, Bootstrap, OpenSSL, Nginx, Nessus, JMeter, Lighthouse



## ABSTRACT

The work deals with web application development, implementation possibilities of web application, secure communication between server part and client part. Protocol HTTPS (Hyper Text Transfer Protocol) and TLS (Transport Layer Protocol) are described in more detail way, also the issue of PKI (Public Key Infrastructure). The work also covers authentication and authorization methods which are used in web applications, and the most common attacks according OWASP TOP 10. Technologies, programming languages and environments, which have been used: Python, Flask, Bootstrap, OpenSSL, Nginx, Nessus, JMeter, Lighthouse.



## KEYWORDS

web application, web server, HTTPS, TLS, PKI, Python, Flask, Bootstrap, OpenSSL, Nginx, Nessus, JMeter, Lighthouse

DZADÍKOVÁ, Slavomíra. *Webová aplikace pro přenos dat s využitím TLS protokolu*. Brno, 2020, 61 s. Bakalárska práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedúci práce: Ing. David Smékal

# VYHLÁSENIE

Vyhlasujem, že som svoju bakalársku prácu na tému „Webová aplikace pro přenos dat s využitím TLS protokolu“ vypracovala samostatne pod vedením vedúceho bakalárskej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autorka uvedenej bakalárskej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto bakalárskej práce som neporušila autorské práva tretích osôb, najmä som nezasiahla nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomá 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 zákoníka Českej republiky č. 40/2009 Sb.

Brno ..... .....

podpis autorky

# POĎAKOVANIE

Rada by som poďakovala vedúcemu práce pánovi Ing. Davidovi Smékalovi za odborné vedenie, konzultácie, trpezlivosť a podnetné návrhy k práci taktiež za odborné konzultácie pánovi Ing. Jaroslavovi Hájekovi.


Brno . . . . . .

. . . . . .

podpis autorky

# Obsah

# Zoznam obrázkov

# Zoznam výpisov

# Úvod

Webové aplikácie nevedome nahrádzajú staré desktopové aplikácie, sú pohodlnejšie pre užívanie, ľahko sa aktualizujú a nie sú viazané na jedno konkrétne zariadenie.

Bakalárska práca sa zaoberá problematikou webových aplikácií a ich bezpečnosťou. V dnešnej dobe vidíme stúpajúci trend v nájdených zraniteľnostiach vo webových aplikáciách. Podľa poskytovateľa dátových bezpečnostných a aplikačných bezpečnostných riešení, Imperva, v roku 2018 tento trend vzrástol o 23 % oproti roku 2017, čo je oproti roku 2016 o 126 % viac (z literatúry [1]). Aj kvôli týmto dôvodom sa bude bakalárska práca podrobnejšie zaoberať bezpečnosťou webových aplikácií a ich správnou implementáciou.

Čitateľ bude oboznámený s potrebnými technológiami pre tvorbu webových aplikácií, ako napríklad HTML (Hypertext Markup Language), CSS (Cascading Style Sheets), JavaScript a tiež procesom vývoja. Predstavené budú aj možnosti architektúry pre webovú aplikáciu napr. REST (Representational State Transfer) rozhranie.

Neoddeliteľnou súčasťou webovej aplikácie je jej serverová časť, preto je dôležité zvážiť použitý operačný softvér a druh webového serveru.

V ďalších častiach je rozobratá bezpečná komunikácia medzi klientom a serverom protokolom HTTPS (Hypertext Transfer Protocol Secure). HTTPS, komunikačný protokol, na šifrovanie komunikácie využíva TLS (Transport Layer Security) protokol, ktorý je v práci podrobne rozobratý - funkcionalita, dostupné verzie, možná implementácia. Bezpečná komunikácia medzi klientom a serverom úzko súvisí aj s oblasťou infraštruktúry verejných kľúčov, preto je tejto téme venovaná jedna celá podkapitola, kde sú podrobne popísané certifikačné autority, reťazec dôvery a self-signed certifikát (certifikát podpísaný od samého sebou).

Možnosti rôznych autentizácií vo webových aplikáciach sú rozobraté taktiež v samostatnej kapitole.

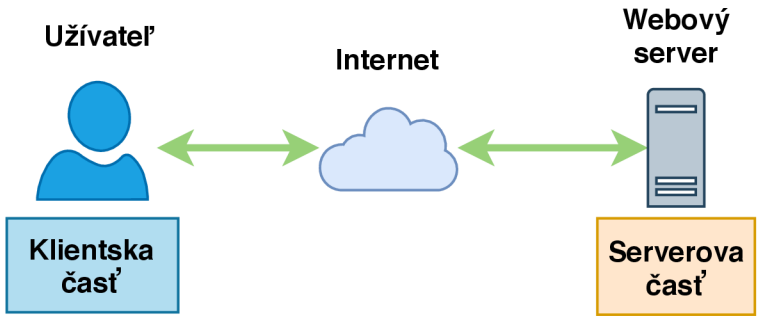
Mechanizmami autorizácie, ktoré sú vo webových aplikáciach často využívané, sa zaoberá nasledujúca kapitola.

Zraniteľnosti webových aplikácií a ich možné nápravy sú náplňou ďalšej kapitoly. Informácie v kapitole vychádzajú najmä z dokumentu OWASP TOP 10, kde je podrobne popísaných desať najčastejších útokov na webové aplikácie.

Poslednú kapitolu tvorí popis samotnej implementácie zadania. Je predstavené prostredie, v ktorom bola aplikácia vyvíjaná, postup konfigurácie webového serveru, nastavenie TLS komunikácie a ďalšie bezpečnostné opatrenia. Na záver bude čitateľ zoznámený s aktuálnym vzhľadom webovej aplikácie.

# 1 Webová aplikácia

Webová aplikácia je počítačový program, ktorý sa nachádza na vzdialenom serveri. Koncový užívateľ obdrží aplikáciu pomocou webového prehliadača (klientska časť), keďže je prístupná prostredníctvom siete, čiže aplikáciu nie je nutné sťahovať. Ako už bolo naznačené webová aplikácia je tvorená klientskou a serverovou časťou a je na tvorcovi, aby zaistil spoluprácu medzi týmito dvoma entitami (informácie čerpané z [2]). Komunikácia medzi klientom a serverom je znázornená na obrázku 1.1, prebieha pomocou nezabezpečeného protokolu HTTP (Hypertext Transfer Protocol) alebo zabezpečeného protkolu HTTPS (Hypertext Transfer Protocol Secure) – bližšie popísané v podkapitole 3.1.




Obr. 1.1: Ukážka komunikácie medzi webovým serverom a užívateľom.

## 1.1 Tvorba webovej aplikácie

Tvorba webovej aplikácie je zvyčajne rozdelená na dve časti: front-end a back-end vývoj. Back-end vývoj je na strane servera, definuje funkcionalitu webu a je zodpovedný za veci ako výpočty, obchodná logika, interakcie s databázou a výkon. Serverová logika môže byť realizovaná pomocou programovacích jazykov, napríklad: Java, C#, Rust, Go alebo pomocou skriptovacích jazykov Python, JavaScript, Ruby, PHP – viac v literatúre [3].

Front-end vývoj sa týka časti aplikácie, s ktorou používatelia web prichádzajú do styku – zvyčajne ide o kombináciu HTML (Hypertext Markup Language), CSS (Cascading Style Sheets) a JavaScriptu, ktoré riadia prehliadač. Pri tvorbe sa môžu využiť dostupné vývojové prostredia, a to Bootsrap, Semantic UI, UI Kit, Material UI ...

Počas vývoja sa často využíva Web Application Framework alebo „web framework“, čo je software, ktorý je navrhnutý na podporu vývoja webových aplikácií vrátane webových služieb, webových zdrojov a webových API (Application Programming Interface) – rozhranie pre programovanie aplikácií. Framework je knižnica,

ktorá pomôže vyvinúť aplikáciu rýchlejšie a inteligentnejšie. Medzi často používané frameworky patria napr. Node.js (JavaScript), ASP.NET Web (C#), Spring (Java), Django REST framework (Python), Flask (Python).

V nasledujúcich podkapitolách (1.1.1, 1.1.2, 1.1.3) budú podrobnejšie rozobraté technológie, ktoré sú využité pri praktickej implementácií práce.

### 1.1.1 Python

Python je univerzálny interaktívny objektovo orientovaný vysoko-úrovňový programovací jazyk, ktorý Vytvoril Guido van Rossum v rokoch 1985 – 1990, viac v dokumentácií [4]. Python je pod licenciou Python Software Foundation Version 2, podľa ktorej je možné software používať pre komerčné účely, modifikovať a distribuovať ho za podmienok zverejnenia popisu autorských práv, plného znenia licencie a uvedenia významných zmien. Licencia nepovoľuje použitie ochrannej známky a tiež zbavuje autorov držania zodpovednosti. Licencia je kompatibilná s GPL (General Public License), čo znamená, že je možné Python kombinovať s iným softvérom, ktorý je pod licenciou GPL, s ostatnými nie.

Pre implementáciu webovej aplikácie bola vybratá verzie 3.7., kde existujú dva rozšírené open-source frameworky pre tvorbu webových aplikácií, a to Flask a Django.

### 1.1.2 Flask

Mikroframework vhodný pre začiatočníkov na tvorbu menších projektov. V porovnaní s Django frameworkom je rýchlejší, flexibilnejší, voľnejší na implementáciu, viac v literatúre [5].

Flask je šírený pod licenciou BSD 3-clause (Berkeley Software Distribution), rovnaká ako BSD 2-clause s výnimkou treťej vety, ktorá zakazuje zneužívať meno držiteľa autorských práv a mená prispievateľov na podporu alebo propagáciu produktov odvodených z tohto softvéru bez osobitného predchádzajúceho písomného súhlasu. Podľa licencie je povolené komerčné využitie, modifikácia, distribúcia, súkromné užitie a úprava. Je potrebné uviesť autorské práva a plný text licencie. Táto licencia obsahuje obmedzenia zodpovednosti a neposkytuje žiadnu záruku.

### 1.1.3 Bootstrap

Bootstrap verzie 4 je najobľúbenejší HTML, CSS a JavaScript framework na vývoj webových stránok zameraných najmä na mobilné zariadenia. Obsahuje návrhové šablóny pre typografiu, formuláre, tlačidlá, navigáciu a ďalšie komponenty rozhrania. Pre viac informácií literatúra [6].

Bootstrap je voľne dostupný softvér, vydaný pod licenciou MIT (Massachusetts Institute of Technology). Ak používame CSS a JavaScript súbory z Bootstrap softvéru je nutné v nich ponechať zmienku o licencii a autorských právach. Autori a vlastníci licencie nie sú zodpovední za škody, pretože Bootstrap je poskytovaný bez záruky. Podľa licencie nie je zakázané používať ochranné známky Twitteru v zmysle, že podporuje danú distribúciu alebo že dané dielo vytvoril Twitter. Licencia umožňuje teda využívať Bootstrap úplne alebo iba čiastočne, na osobné alebo komerčné účely, daný zdrojový kód sa môže upravovať, tiež umožňuje sublicencovanie tretím stranám.

HTML je štandardným značkovacím jazykom pre dizajn dokumentov zobrazovaných vo webovom prehliadači. Webový server posiela webovému prehliadaču HTML kód, na základe ktorého je stránka prispôsobená. HTML poskytuje prostriedky na vytváranie štruktúrovaných dokumentov pomocou štrukturálnej sémantiky pre text, ako sú nadpisy, odseky, zoznamy, odkazy, citácie a ďalšie položky. Kód HTML je často previazaný s CSS a skriptovacím jazykom JavaScript.

CSS je kaskádový jazyk pre štýly, ktorý sa používa na opis prezentácie dokumentu v značkovacom jazyku, HTML. CSS je navrhnutý tak, aby umožňoval oddelenie prezentácie a obsahu vrátane rozloženia, farieb a typu písma.

JavaScript je vysoko-úrovňový skritiptovací jazyk, ktorý sa používa pri tvorbe webových aplikácií. Pomocou JavaScriptu sa vytvárajú najmä elementy, ktoré fungujú samé, ako napr. živé hodiny alebo „vyskakovacie“ správy. JavaScript nijako nesúvisí s programovacím jazykom Java.

## 1.2 Architektúra webových aplikácií

Architektúra webových aplikácií je mechanizmus, ktorý určuje, ako komponenty aplikácie medzi sebou navzájom komunikujú. Spôsob komunikácie a spojenia klienta a servera je stanovený architektúrou webových aplikácií.

### 1.2.1 REST

REST (Representational State Transfer) je štýl architektúry, ktorý po prvýkrát vo svojej dizertácií zverejnil Roy Thomas Fielding roku 2000 [7]. REST definuje pravidlá pre webovú službu, tie ktoré spĺňajú všetky pravidlá sa nazývajú „RESTful“ služby. Medzi 6 pravidiel RESTful webovej aplikácie patrí:

- **klient − server**, potreba rozdelenia zodpovednosti,
- **bezstavovosť**, každý požiadavok musí obsahovať všetky potrebné informácie na jeho vybavenie,

- **zálohovateľnosť**, požiadavok môže byť označený ako zálohovateľný alebo nie, ak je odpoveď označená ako zálohovateľná, potom klient má právo tieto údaje znovu použiť na neskoršie ekvivalentné žiadosti,
- **jednotné rozhranie**, základom pre akýkoľvek RESTful systém, zjednodušuje a oddeľuje architektúru, čo umožňuje každej časti sa vyvíjať nezávisle,
- **vrstevnatosť**, skladanie vrstiev poskytujúcich služby za účelom zvýšenia variabilnosti,
- **kód na vyžiadanie**, funkcionalita klienta môže byť rozšírená na základe kódu, ktorý zašle server.

REST môže byť chápaný ako transport reprezentácie zdrojov, pričom zdrojom môže byť takmer čokoľvek napr. obrázok, video, kniha, komentáre, posty ... Každý zdroj má svoj špecifický identifikátor URL (Uniform Resource Locator), na základe ktorého posielame svoje požiadavky serveru, server v odpovedi posiela reprezentáciu žiadaného zdroja. Pod slovom reprezentácia je myslený formát ľudsky čitateľný – HTML, obrázok, JSON (JavaScript Object Notation) alebo XML (Extensible Markup Language). REST definuje štyri základné metódy, známe pod skratkou CRUD (Create Retrieve Update Delete) pre prístup k zdrojom – Create (vytvoriť), Retrieve (získať), Update (obnoviť), Delete (vymazať), môžu byť implementované pomocou odpovedajúcich HTTP metód (POST, GET, PUT, DELETE).

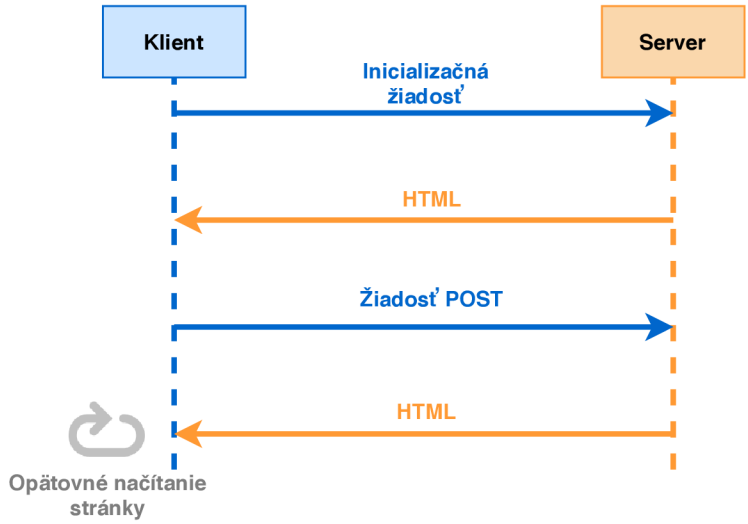
## 1.3 Návrhové vzory webových aplikácií

Webové aplikácie môžu byť jednoducho rozdelené do dvoch skupín, a to viac-stránkové aplikácie 1.3.1 a jedno-stránkové aplikácie 1.3.2. Oba prístupy majú svoje výhody a nevýhody, preto vznikli tzv. hybridné aplikácie, ktoré sú založené na výhodách oboch prístupov a snažia sa o minimalizáciu nevýhod. Informácie sú čerpané zo zdrojov [8] a [9].

### 1.3.1 Viac-stránkové webové aplikácie

V skratke MPA (Multi-Page Application) pracujú v tradičnom režime. Každá zmena v dátach alebo potvrdenie dát a ich zaslanie na server vyžaduje vytvorenie (preklad) novej stránky, ktorá je naspäť zaslaná na server – znázornené na obrázku 1.2. MPA webové aplikácie bývajú väčšie ako jedno-stránkové aplikácie kvôli množstvu obsahu a tiež komplikovanejšie na vývoj.

Výhodami sú zabezpečenie veľmi podrobnej navigácie vo webovej aplikácií, ktorá je možná iba pri viacvrstvových aplikáciach, tiež veľmi ľahká správa SEO (Search Engine Optimization vo voľnom preklade optimalizácia vyhľadávania) – poskytujú




Obr. 1.2: Ukážka životného cyklu viac-stránkových aplikácií.

lepšie šance na hodnotenie rôznych kľúčových slov, pretože aplikáciu je možné optimalizovať pre jedno kľúčové slovo na stránku.

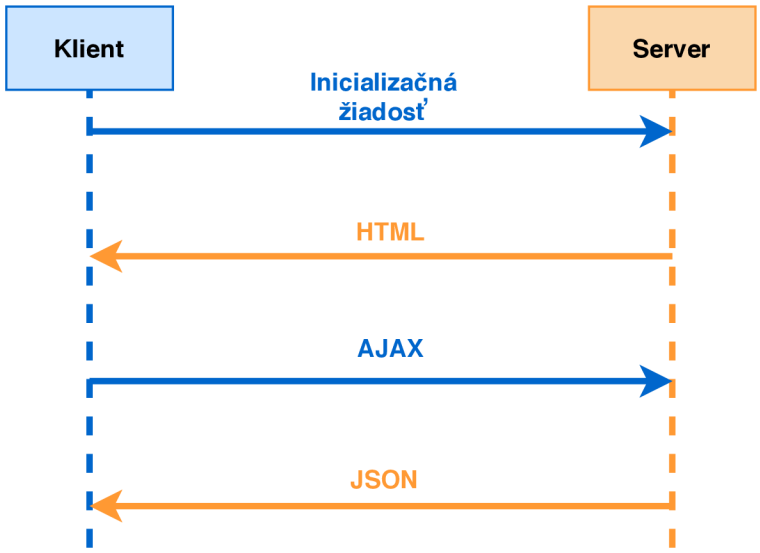
Medzi nevýhodou patrí slabšia kompatibilita s mobilnými zariadeniami, vývoj je omnoho zložitejší.

### 1.3.2 Jedno-stránkové webové aplikácie

Aplikácie nazývané aj skratkou SPA (Single-Page Application) pracujú v prehliadači a nepotrebujú počas používania dodatočné spracovávanie (načítavanie údajov zo serveru), čím je maximálne znížený čas čakania. Aplikácie teda obsahujú iba jednu stránku, ktorá načíta všetok obsah pomocou AJAX (Asynchronous JavaScript and XML) požiadavku na server. Server nevytvára novú stránku, ale posiela iba dáta zvyčajne vo formáte JSON. Aplikácia následne využije dáta na načítanie niektorých častí priamo v prehliadači. Proces práce SPA aplikácie je znázornený na obrázku 1.3.

Medzi výhody SPA aplikácií patrí rýchlosť, keďže HTML, CSS súbory a skripty sú načítané iba raz za životný cyklus, vývoj aplikácie je jednoduchší, majú výbornú adaptabilitu s mobilnými zariadeniam, ukladanie dát do lokálnej pamäte, čo umožňuje aplikácií fungovať aj pri zlyhaní internetového pripojenia.

Nevýhodami sú SEO (Search Engine Optimalization) problémy – dlhú dobu mali webové vyhľadávače problémy s indexovaním SPA aplikácií, čo malo za následok, že SPA aplikácie nikdy neboli na vrchole vyhľadávania (problémy s indexovaním Google v posledných rokoch celkom úspešne vyriešil), avšak, ďalším problémom môže byť obmedzené jadro, čo spôsobuje nemožnosť uloženia dostatočného počtu kľúčových




Obr. 1.3: Ukážka životného cyklu jedno-stránkovej aplikácie.

slov na jednu stránku. SPA aplikácie môžu tiež spôsobiť zahltenie pamäte, čo vedie ku značnému spomaleniu. Aplikácie sú náchylnejšie na Cross-Site Scripting (XSS).

# 2 Serverová časť webovej aplikácie

Serverová časť je neoddeliteľnou súčasťou webovej aplikácie, ako už bolo naznačené v kapitole 1. V nasledujúcich častiach bude písané o operačnom softvére pre server 2.1 a tiež možnostiach webového serveru 2.2.

## 2.1 Operačný softvér serveru

Pri výbere operačného softvéru pre server bolo hľadanie zamerané na distribúcie operačného systému Linux. Medzi kandidátov patrila distribúcia Debian a Ubuntu, pričom Debian v porovnaní s Ubuntu je stabilnejšia, bezpečnejšia a tiež menej náročná distribúcia Linuxu na hardvér (viac v literatúre [10]).

Za operačný software serveru bol zvolený Debian GNU/Linux, open source softvér (slobodný softvér) šíriteľný pod licenciou GNU GPL, ktorá zaručuje slobodu v spúšťaní, študovaní, modifikovaní a zdieľaní softvéru.

## 2.2 Webový server

Webový server slúži na predávanie súborov prehliadaču (klientovi), funguje ako prostredník medzi serverom a klientom, i keď sa volá webový „server“ v skutočnosti sa jedná o softvér, ktorý môže byť na fyzickom či virtuálnom serveri (z literatúry [11]). Apache a Nginx sú najpoužívanejšie a najznámejšie open source webové servery, spolu poháňajú viac ako 55 % webových serverov sveta podľa údajov z apríla 2019, podľa štatistiky z [12].

### 2.2.1 Apache

Webový server Apache je distribuovaný pod Apache License 2.0. Licencia, ktorej hlavné podmienky si vyžadujú zachovanie autorských práv a upozornenie o licenciách. Prispievatelia poskytujú výslovné udelenie patentových práv. Licencované diela, úpravy a väčšie diela sa môžu distribuovať za rôznych podmienok a bez zdrojového kódu, viac v [11].

Umožňuje softvér využívať pre komerčné účely, modifikovať, distribuovať, súkromné užitie a úpravy, práva na uplatnenie patentových nárokov prispievateľov do kódu, umiestenie záruky na licencovaný softvér. Pri tejto licencií je nutné dodržať určité podmienky, a to zahrnúť autorské práva, plný text licencie, popis zmien v softvére, ak je obsiahnutý aj súbor „NOTICE“ s poznámkami o pridelení, v distribúcií musí byť tento súbor tiež obsiahnutý. Pri danej licencii sa neudeľujú žiadne ochranné známky, obsahuje obmedzenia zodpovednosti a neposkytuje žiadnu záruku.

Nevýhodou webového serveru Apache je, že pre spojenie s klientom vytvára procesy tzv. moduly, ktoré sú zodpovedné za prijímanie požiadaviek a následné priradenie pod-procesov na ich spracovanie. Táto činnosť si zaťažuje pamäť RAM[1] a tiež CPU[2], čo môže byť problém pri webových stránkach s vysokou návštevnosťou. Výhoda Apachu – vie dynamicky spracovávať obsah v rámci svojich procesov.

### 2.2.2 Nginx

Nginx je distribuovaný pod licenciou 2-clauses BSD (licencia s dvomi doložkami). Licencia umožňuje takmer neobmedzenú slobodu so softvérom, pokiaľ v nej je oznámenie o autorských právach.

Podľa licencie je povolené komerčné využitie, modifikácia, distribúcia, súkromné užitie a úprava. Je potrebné uviesť autorské práva a plný text licencie. Táto licencia obsahuje obmedzenia zodpovednosti a neposkytuje žiadnu záruku.

Nginx dokáže pri veľkom množstve požiadavok pracovať efektívnejšie ako Apache. Narozdiel od Apachu vytvára pre každé nové spojenie proces, ktorý môže spracovať tisícky nových spojení. Nginx dynamicky spracovávať obsah nevie a musí sa spoliehať na externé komponenty, ale pri spracovaní statických súborov vie byť dokonca 2,5-krát rýchlejší ako Apache, podľa literatúry [11].

Apache a Nginx môžu fungovať spolu na jednom serveri, kde sa využijú výhody oboch webových serverov.

[1]Random Access Memory – forma počítačovej pamäte, ktorú je možné čítať a robiť zmeny v akomkoľvek poradí

[2]Central Processing Unit – procesor počítača, vykonáva strojové inštrukcie

# 3 Bezpečný prenos medzi klientom a serverom

Bezpečný prenos medzi klientom a serverom je zabezpečený pomocou HTTPS protokolu, o ktorom je viac napísané v časti 3.1.

## 3.1 Protokol HTTPS

Protokol HTTPS je komunikačný protokol, ktorý pre šifrovanie používa protokol TLS – takže HTTPS nie je nič iné ako HTTP so šifrovaním TLS (Transport Layer Security) protokolu. Protokol vytvára bezpečné spojenie od klienta k serveru v ináč nezabezpečenej sieti, počas komunikácie sa autentizuje server klientovi alebo môže byť umožnená obojstranná autentizácia, aby sa predišlo útokom man-in-the-middle (mužom uprostred) a odpočúvaniu. HTTPS vytvára spojenie na porte 443, pričom protokol HTTP pracuje s portom 80, informácie čerpané z literatúry [13], [14]. Viac o funkcionalite a procese komunikácie bude písané v časti 3.2.

## 3.2 Protokol TLS

TLS je najpoužívanejší kryptografický protokol na poskytovanie bezpečnej komunikácie, zabraňuje odpočúvaniu komunikácie či falošným správam. V modele TCP/IP sa nachádza medzi aplikačnou a transportnou vrstvou, kde zaobstaráva bezpečnú komunikáciu pre mnohé aplikácie, ako webové prehliadače, elektronické maily, VPN pripojenia a ďalšie ... Pôvodný protokol TLS bol navrhnutý v roku 1999, pričom sa jedná o náhradu za protokol SSL (Secure Sockets Layer). Slúži na zabezpečenie troch základných služieb: šifrovania, autentizácie a integrity. Podľa literatúry [13], [14].

TLS sa skladá z troch časti – TLS Handshake Protokol, TLS Record Protokol, Alert Protokol, ktoré sú ďalej bližšie rozobraté.

### TLS Handshake Protokol

TLS Handshake Protokol zaisťuje vyjednanie všetkých služieb, ktoré sú pre spojenie TLS potrebné. Handshake (v preklade „potrasenie rúk“, výmena, dohoda) vieme rozdeliť na tri základné fázy:

- dohoda účastníkov na podporovaných algoritmoch,
- výmena kľúčov založená na šifrovaní s verejným kľúčom a autentizácia vychádzajúca z certifikátov,

- šifrovanie dát symetrickou šifrou.

Pri použití protokolu TLS je zvyčajne autentizovaný iba server, koncový užívateľ si môže byť istý s kým komunikuje. Vzájomná autentizácia účastníkov komunikácie je tiež možná, ale vyžaduje nasadenie infraštruktúry verejných kľúčov (Public Key Infrastacture - PKI). Komunikácia medzi klientom a serverom prebieha následovne:

1. klient pošle správu **ClientHello**, v ktorej oznamuje najvyššiu podporovanú verziu TLS, náhodne vygenerované číslo (používa sa pri výpočte kľúčov) a zoznam podporovaných algoritmov.
2. Server odpovie správou **ServerHello**, kde uvádza zvolenú verziu protokolu, náhodné číslo a ustanovený šifrovací a kompresný algoritmus.
3. Server pošle svoj certifikát klientovi a tiež môže poslať správu **CertificateRequest**, ktorou žiada certifikát od klienta, aby došlo k vzájomnej autentizácií.
4. Správa **ServerHelloDone** od serveru inicializuje ukončenie prvotnej dohody nad použitými mechanizmami.
5. Klient odpovedá správou **ClientKeyExchange**, v ktorej je zašifrované náhodné číslo verejným kľúčom serveru (server je schopný dešifrovať pomocou súkromného kľúča), a to slúži ako pred-zdieľané tajomstvo tzv. *PreMasterSecret*.
6. Klient a server na základe náhodných čísel a *PreMaserSecret* vypočítajú pomocou pseudonáhodnej funkcie kľúč „master key“, všetky ostatné kľúče sú z neho odvodené.
7. Klient pošle správu **ClientCipherSpecification**, ktorou oznamuje, že následujúca komunikácia bude už šifrovaná. Klient odosiela správu **Finished**, ktorá je už šifrovaná a obsahuje haš a MAC predošlých správ.
8. Server sa pokúsi dešifrovať **Finished** od klienta, ak dešifrovanie zlyhá, inicializácia je považovaná za neúspešnú a spojenie sa preruší.
9. Na záver server pošle správu **ChangeCipherSpecification** a svoju správu **Finished**, klient prevedie analogické dešifrovanie. V tomto okamihu je inicializačný proces ukončený a komunikácia je šifrovaná.

Existuje aj zjednodušená verzia handshaku (tzv. podanie rúk), ktorá sa využíva v prípade, že klient so serverom chce pokračovať už v predošlom naviazanom spojení alebo spojenie chce duplikovať. Klient pošle správu **ClientHello** s identifikátorom spojenia, v ktorom chce pokračovať. Ak ho server nenájde proces handshaku sa vykoná v plnej forme.

Handshake protokol sa môže líšiť vzhľadom na použitú verziu autentizácie a kryptobalíčku. Predošlý popis Hanshake protokolu nezahrňuje použitie **Perfect Forward Secrecy** (perfektné predzdieľané tajomstvo) kvôli jednoduchosti a prehľadnosti popisu protokolu. Problémom jeho nepoužitia je, že sa súkromný kľúč serveru používa aj pre autentizovanie a aj pre šifrovanie, čo môže spôsobiť problémy do

budúcna, pri získaní súkromného kľúča serveru útočníkom (útočník by bol schopný dešifrovať správu s *PreMasterSecret* a následne odvodiť *Master Key*). Z tohoto dôvodu je dobré pri implementácií zahrnúť algoritmy, ako DHE (Diffie–Hellman Ephemeral v preklade DH s dočasnými kľúčmi) alebo verziu na základe eliptických kriviek ECDHE (Eliptic Curve Diffie–Hellman Ephemeral), ktoré poskytujú využitie Perfect Forward Secrecy, viac v literatúre [15]. Viac o popise najnovšej verzie TLS protokolu 1.3 a jeho bezpečnostných opatreniach v sekcií 3.2.1.

#### TLS Record Protokol

Protokol obalí protokoly z vyšších vrstiev. Zaoberá sa fragmentáciou, komprimáciou, vypočítaním kontrolnej informácie MAC a šifrovaním dát.

#### Alert Protokol

Množina chybových správ pre vyššie vrstvy. Ak dôjde ku fatálnej chybe, spojenie sa ukončí napr. prijatie chybných dát, ktoré nie je možné rozšifrovať.

### 3.2.1 Verzie TLS protokolu

TLS Protokol má aktuálne štyri verzie: TLS 1.0, 1.1, 1.2 a najnovšiu verziu TLS 1.3, ktorá bola vydané v marci 2018. Jednotlivé verzie sa líšia podporovanými kryptografickými balíčkami alebo zmenami v Hadshake protokole. Verzie TLS 1.1 a TLS 1.2 podporujú nasledujúcu kryptografiu:

- pre výmenu kľúčov: RSA[1], DH, ECDH;
- pre autentizáciu: RSA, ECDSA[2], DSA;
- pre symetrické šifrovanie: RC4, IDEA[3], DES[4], Triple DES, AES[5], Camellia;
- pre hašovanie: Message-Digest algorithm (MD5), Secure Hash Algorithm (SHA-1, SHA-2, SHA-3).

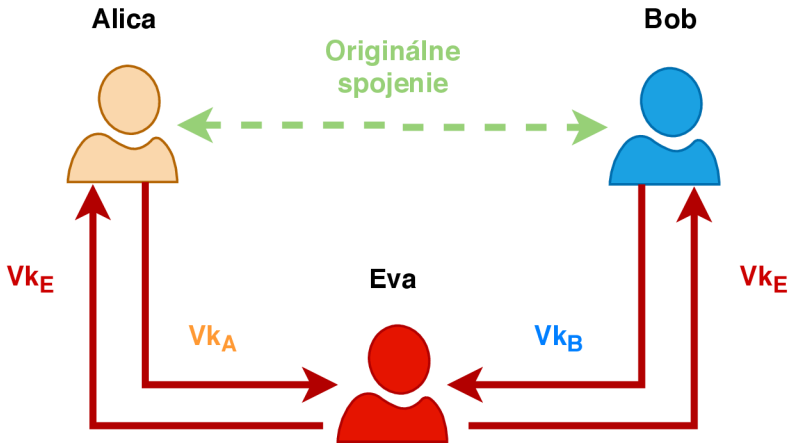
Medzi hlavné rozdiely verzie 1.3 oproti verzii 1.2 patrí odstránenie všetkých symetrických algoritmov zo zoznamu podporovaných algoritmov, ktoré sa považujú za staré, napr. RC4, DH, MD5 ... Algoritmy, ktoré v zozname zostali sa radia medzi algoritmy autentizovaného šifrovania s pridruženými dátami. Koncepcia šifrovania bola zmenená tak, aby oddeľovala mechanizmy autentizácie a výmeny kľúčov od algoritmu ochrany záznamu a hašu. Statické šifry ako RSA a DH boli odstránené. V novej verzií všetky mechanizmy na výmenu kľúčov založené na verejnom kľúči

[1] Rivest, Shamir, Adleman
[2] Eliptic Curve Digital Signature Algorithm
[3] International Data Encrypt Algorithm
[4] Data Encryption Standard
[5] Advanced Encryption Standard

poskytujú utajenie vopred a všetky spravy po „ServerHello“ sú šifrované. Handshake protokol bol tiež významne pozmenený, aby bol konzistentnejší a aby odstránil všetky zbytočné správy, ako napr. ChangeCipherSpecification, zmeny urýchľujú celý proces spojenia. Boli pridané nové algoritmy pre podpis napr. ed25519 a ed488. Vyjednávanie formátov bodov pre algoritmy nad eliptickými krivkami bolo odstránené v prospech jediného formátu bodov pre každú krivku. Medzi ďalšie kryptografické zmeny patrí odstránenie kompresie, vlastné skupiny DHE, zmena RSA dopĺňania na RSASSA-PSS[6] a odstránenie DSA. Viac v literatúre [16].

## 3.3 Infraštruktúra verejných kľúčov

Využitie zabezpečenia pomocou asymetrickej kryptografie prináša problém s distribúciou verejného kľúču. Komunikácia medzi dvomi subjektami, ktorí si vymieňajú svoje verejné kľúče, môže byť napadnutá útokom man-in-the-middle tak, ako je znázornené na obrázku 3.1. Z tohoto dôvodu pre bezpečnú distribúciu verejných kľúčov sa využívajú certifikáty (certifikát bližšie v časti 3.3.1).




Obr. 3.1: Ukážka výmeny verejných kľúčov pri napadnutí komunikácie útokom man-in-the-middle.

Infraštruktúru verejných kľúčov teda možno chápať, ako súhrn technických a organizačných prostriedkov spojených s vydávaním, správou, používaním a odvolávaním platnosti kryptografických kľúčov a certifikátov. Mechanizmus zabraňuje používaniu falošnej identity, keďže verejný kľúč je platný len vtedy, ak je potvrdený dôveryhodnou entitou napr. certifikačnou autoritou (v časti 3.3.2) – informácie podľa literatúry [13], [14].

[6]RSA Signature Scheme with Appendix – the Probabilistic Signature Scheme je vylepšená podpisovacia schéma s dodatkom

### 3.3.1 Digitálny certifikát

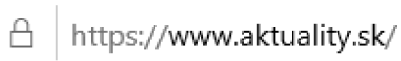
Údaje týkajúce sa jedného subjektu, ktorých pravosť je potvrdená iným subjektom (certifikačnou autoritou). Certifikát obsahuje verejný kľúč subjektu, ale môže obsahovať aj ďalšie údaje. Všetky informácie sú v certifikáte podpísané súkromným kľúčom autority. Najpoužívanejším štandardom, podľa ktorého sa vytvárajú certifikáty je X.509 aktuálne v tretej verzií. Formát certifikátu vytvorený podľa štandardu X.509 obsahuje položky: verzia, sériové číslo, identifikátor algoritmu, identifikátor certifikačnej autority, doba platnosti, subjekt (vlastník certifikátu), verejný kľúč, podpis autority. Dôveryhodnosť certifikátu sa odvíja od dôveryhodnosti certifikačnej autority a od spôsobu, akým autorita získava a potvrdzuje údaje o subjektoch, z literatúry [13].

Certifikáty majú mnoho využití, ale v súvislosti s témou práce sa text bude zameriavať na tzv. serverové certifikáty pre bezpečný prístup na server pomocou TLS protokolu.

### 3.3.2 Certifikačná autorita (CA)

Subjekt, ktorý vydáva a overuje certifikáty. Najväčším aktívami, ktoré musí certifikačná autorita chrániť: súkromný kľúč, uložený na bezpečnom hardvére (väčšinou bez prístupu k internetu), databázu užívateľov, archív súkromných kľúčov užívateľov (ak autorita danú službu poskytuje), podľa literatúry [14].

Certifikačná autorita môže overovať na základe doménového mena (tzv. **Domain Validation** certifikát – DV) alebo organizačného mena (tzv. **Organization Validation** certifikát – OV) – v týchto prípadoch sa nekontrolujú žiadne iné informácie o totožnosti užívateľa, zobrazenie v prehliadači na obrázku 3.2.



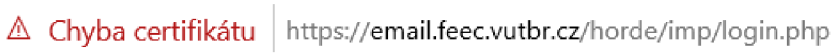

Obr. 3.2: Ukážka DV alebo OV certifikátu v prehliadači.

**Exetended Validation** certifikát – EV, certifikát s rozšíreným overením, overuje sa nie len doménové meno, ale aj ostatné informácie o totožnosti užívateľa. Prehliadače zvyčajne zobrazujú dodatočné informácie zelenou farbou v poli pre URL adresu, na obrázku 3.3.


Obr. 3.3: Ukážka EV certifikátu v prehliadači.

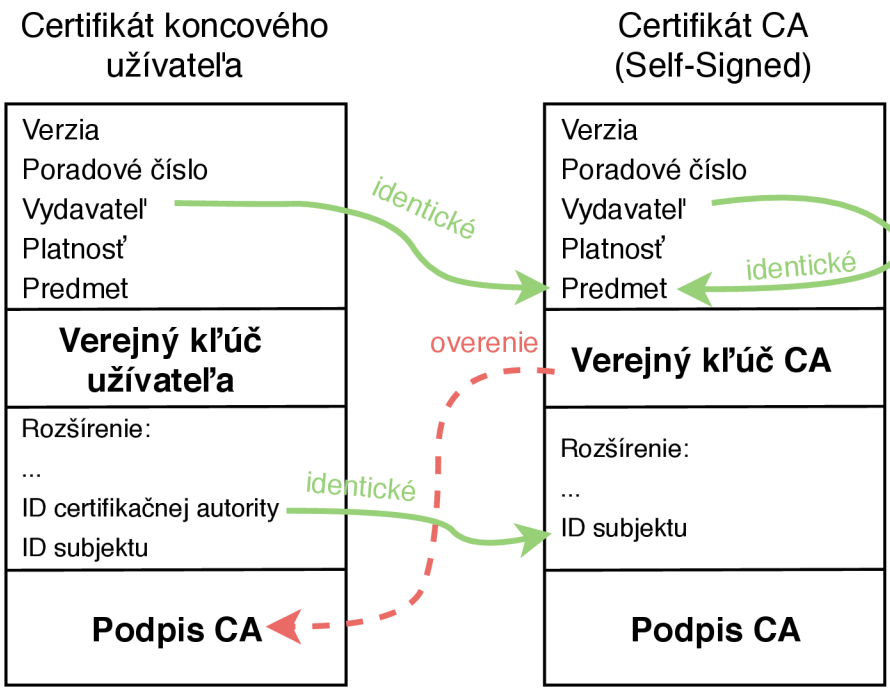
Trh pre certifikačné autority je veľmi obmedzený kvôli vysokým technickým nárokom na niekoľko organizácií. Dlhodobo sa na prvých miestach v rebríčku drží IdenTrust, Sectigo alebo DigiCert Group, zo štatistiky [17]. Let's Encryp je nezisková organizácia, ktorá vytvára certifikáty na základe štandardu X.509, kde je možné si vytvoriť certifikát typu Domain Validation zdarma (jediné, čo je nutné mať vlastnú doménu). Certifikát podpísaný autoritou Let's Encrypt sa v prehliadačoch nachádza v zozname dôveryhodných certifikačných autorít.

Posledný typ certifikátu je **Self-Signed** certifikát (certifikát podpísaný samým sebou, polia vydavateľ a predmet sa zhodujú). Certifikát nie je pre prehliadače dôveryhodný, keďže nie je podpísaný žiadnou autoritou, užívateľ sa musí sám rozhodnúť či je certifikát dôveryhodný alebo nie – môže sa jednať o man-in-the-middle útok, ukážka na obrázku 3.4. Organizácie často využívajú Self-Signed certifikát v prípade čakania na vydanie certifikátu od CA. Tento typ certifikátu sa môže nazývať aj koreňový certifikát, využívajú ho certifikačné autority pre svoje vlastné certifikáty, svoj verejný kľúč si podpíšu svojim súkromným kľúčom.




Obr. 3.4: Ukážka Self-Signed certifikátu v prehliadači.

Overenie certifikátu koncového užívateľa pozostáva v porovnaní identifikátoru certifikačnej autority alebo položky vydavateľ (z certifikátu koncového užívateľa) a identifikátoru subjektu alebo položky predmet (z koreňového certifikátu certifikačnej autority), naznačené na obrázku 3.5 – z literatúry [13].




Obr. 3.5: Ukážka overovanie certifikátu koncového užívateľa.

# 4 Možnosti autentizácie vo webových aplikáciach

Kapitola rozoberá rôzne typy autentizačných schém podľa mechanizmu a tiež najčastejšie používané spôsoby vo webových aplikáciach. Autentizácia slúži pre overenie totožnosti užívateľa na základe faktorov (dôkazov). Pojem autentizácia je niekedy zamieňaný s autorizáciou, čo je proces, pri ktorom je overované či má autentizovaný užívateľ dostatočné práva pre prístup k chránenému zdroju. Počas implementácie tieto dva procesy môžu splývať.

Dôkazy sa delia na tri typy: **dôkaz vedomosťou** – užívateľ má nejakú vedomosť (heslo, PIN, odpovedá na výzvy), patria medzi najčastejšie využívané. Druhým typom je **dôkaz vlastníctvom**, užívateľ vlastní nejaký nosič, na ktorom je uložený kľúč napr. ID karta alebo mobilné zariadenie so zabudovaným hardvérovým či softvérovým tokenom (nosičom). **Dôkaz vlastnosťou** je posledným typom, zahŕňa tzv. biometrické dôkazy – užívateľ sa môže identifikovať otlačkom prsta alebo skenovaním sietnice či tváre. Ďalšie informácie ohľadom delenia atď. v literatúre [18], [19].

## 4.1 Autentizácia menom a heslom

Užívateľ sa do aplikácie autentizuje pomocou užívateľských údajov, ktoré sú porovnávané s uloženými hodnotami na strane serveru. Heslo by malo byť uložené v zašifrovanej podobe alebo vo forme hašu (prípadne s využitím techniky solenia – pridanie náhodnej hodnoty môže byť zamedzené útokom s predpočítanými tabuľkami – rainbow table attack).

## 4.2 Autentizácia typu výzva – odpoveď

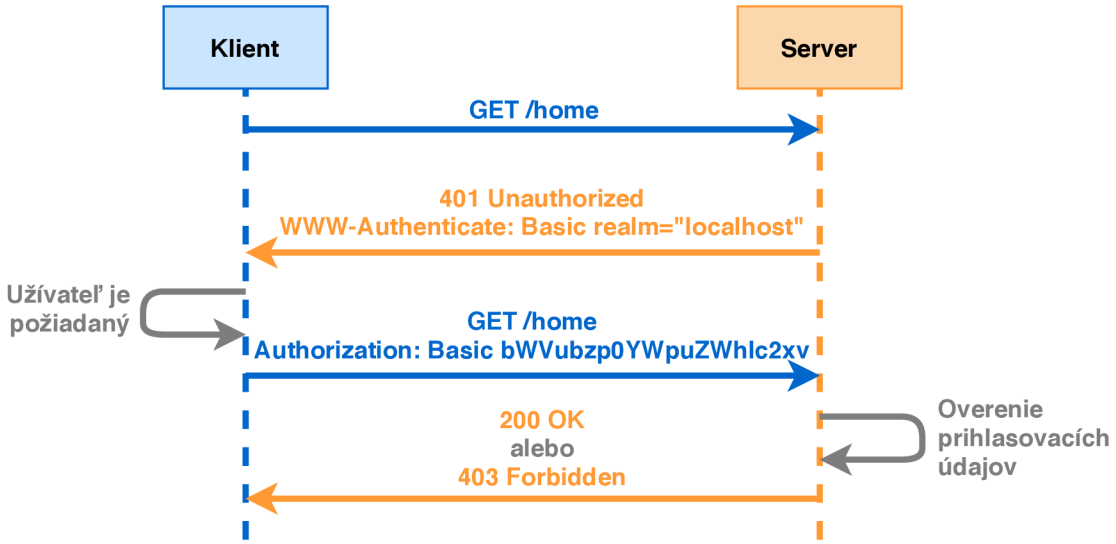
V podkapitole sú predstavené autentizačné schémy založené na mechanizme výzva – odpoveď, ktoré sú často využívané v prostredí webových aplikácií a sú definované protokolom HTTP: 4.2.1, 4.2.2.

### 4.2.1 Basic – základná autentizácia

Patrí medzi najjednoduchší druh autentizácie, pretože využíva pole štandardnej HTTP hlavičky a nevyžaduje súbory cookie, relácie alebo čokoľvek iné. Podrobnejšie informácie v literatúre [20].

Klient poskytuje na vyžiadanie serveru svoje prihlasovacie meno a heslo – väčšinou sú oddelené dvojbodkou „:“. Prihlasovacie meno a heslo sú následne kódované

do formátu Base64 (metóda pre prevedenie znakov na 64 bitový reťazec využívaná na bezpečný prenos – bezchybný) a vložené do HTTP hlavičky. V prípade, ak server požaduje od klienta prihlásenie pošle správu `HTTP 401 Unauthorized` s poľom v hlavičke `WWW-Authenticate`. Komunikácia počas autentizácie je znázornená na obrázku 4.1.




Obr. 4.1: Schéma komunikácie počas Basic autentizácie.

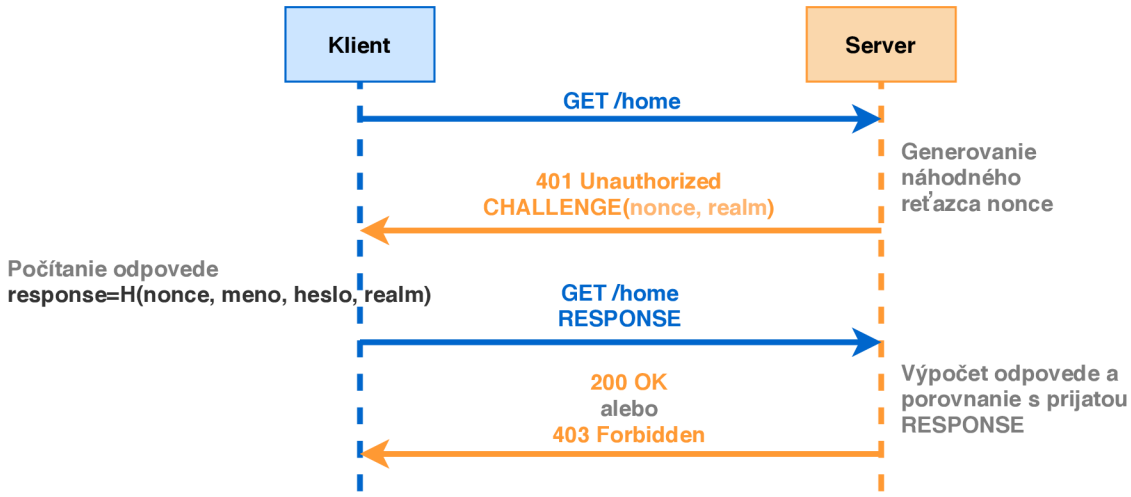
Nevýhodou je, že prihlasovacie údaje sa posielajú v hlavičke každej žiadosti (správe), server musí uchovávať v pamäti po vhodnú dobu údaje, aby sa zamedzilo opakovanému prihlasovaniu užívateľa. Tiež je vhodné používať Basic autentizáciu v kombinácií s TLS protokolom, keďže schéma autentizácie sama o sebe neobsahuje žiadnu formu šifrovania alebo hašovania.

### 4.2.2 Digest – overovanie totožnosti

Využíva hašovanie prihlasovacieho mena, hesla, HTTP metódy, požadovanej URL adresy pomocou funkcie MD5 (od roku 2015 je možné využiť aj iné hašovacie funkcie, ako napr. SHA-256, SHA-512), aby sa zabránilo posielaniu nešifrovanému textu, z literatúry [20].

Pri autentizácií najskôr klient pošle žiadosť serveru – bežne pomocou navštívenia webovej stránky na to určenej, následne server odpovedá správou `401 Unauthorized` spolu s náhodne zvoleným číslom nonce (number used only once – číslo použité len raz), tiež posiela reťazec realm (zvyčajne popis systému, do ktorého sa pristupuje) a žiada klienta o autentizovanie. Klient odpovedá pomocou nonce a zašifrovaného prihlasovacieho mena, hesla a realmu. Server na koniec porovnáva haš, ktorý má

uložený s hašom poslaným klientom počas komunikácie. Ukážka komunikácie na obrázku 4.2.




Obr. 4.2: Schéma komunikácie počas Digest autentizácie.

## 4.3 Autentizácia pomocou digitálneho podpisu

Užívateľ podpisuje každú svoju správu svojim súkromným kľúčom následne server správu overuje pomocou verejného kľúča užívateľa, čo je značne nevýhodné pri webových aplikáciách – je nutná znalosť verejného kľúča užívateľa (problém s distribúciou spomínané v kapitole 3.3).

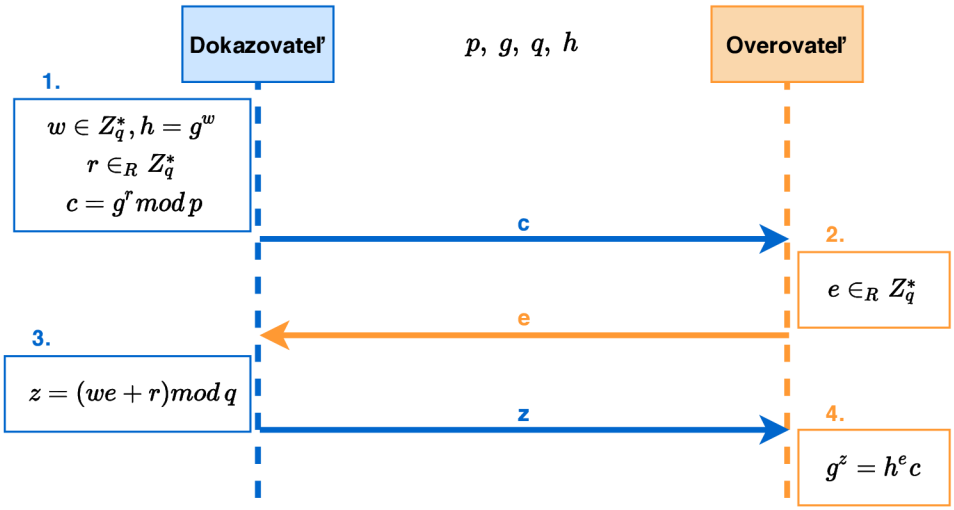
## 4.4 Autentizácia pomocou protokolov s nulovou znalosťou

Pri autentizačnom procese je väčšinou veľmi namáhavé kontrolovať, koľko presne informácií je zdieľaných. Protokoly s nulovou znalosťou zverejňujú iba jeden bit informácie. Protokol prebieha medzi dvoma stranami zvyčajne nazývanými ako dokazovateľ a overovateľ a sú definované nasledujúcimi vlastnosťami:

- **úplnosť** hovorí o tom, že každé pravdivé tvrdenie odoslané dokazovateľom by mala byť takmer vždy prijatá overovateľom,
- **spoľahlivosť** znázorňuje fakt, že každé nepravdivé tvrdenie zaslané dokazovateľom by malo byť takmer vždy, zamietnuté overovateľom.
- **nulová znalosť** – overovateľ nemá žiadne informácie o vstupe, ktorý dokazovateľ chce overiť.

Praktickejšou formou protokolov s nulovou znalosťou sú Sigma protokoly. Sigma protokol je založený na troj-cestnom vzore (sú zasielané iba tri správy) a je nutné, aby mál dôveryhodného overovateľa (čím sa o čosi znižuje bezpečnosť oproti klasickým protokolom s nulovou znalosťou). Jedným z najpoužívanejších Sigma protokolov je práve Schnorrov protokol, je založený na znalosti diskrétneho logaritmu – $w : h = g^w$ v tzv. DSA (Digital Signature Algorithm) grupe – multiplikatívna grupa $\mathbb{Z}_p^*$ (kde $p$ je prvočíslo) s generátorom $g$ o ráde $q$.

Protokol je zobrazený na obrázku 4.3, v prvom kroku je vypočítaný verejný parameter $h$ na základe tajného parametru $w$, ktorý pozná iba dokazovateľ. Overovateľ náhodne vygeneruje parameter $r$ a vypočíta správu $c$, ktorú posiela overovateľovi. V druhom kroku overovateľ náhodne vygeneruje parameter $e$ a zasiela ho dokazovateľovi. Následne v treťom kroku dokazovateľ vypočíta $z$ a posiela ho overovateľovi. V poslednom kroku overovateľ zistí či daná rovnica platí alebo nie, ak platí dokazovateľ preukázal znalosť tajomstva a je prijatý.




Obr. 4.3: Schéma Schnorrovho protokolu.

## 4.5 Multi-faktorová autentizácia

Multi-faktorová autentizácia je založená na overení viacerých dôkazov, ktorými sa užívateľ preukazuje. Ako už bolo spomenuté na začiatku kapitoly 4, užívateľ je buď držiteľom daného dôkazu alebo má určitú znalosť, alebo sa preukazuje svojou vlastnosťou (dedičnosťou). Na základe týchto možností sú rôzne zostavené schémy viac-faktorovej autentizácie. Príklad pre dvoj-faktorovú autentizáciu je výber z bankomatu pomocou kreditnej karty – je to kombinácia karty (niečoho, čo užívateľ vlastní) a štvorciferného PINu (niečo, čo užívateľ vie).

Dôkazy vlastníctva sú často využívané pri viac-faktorovej autentizácií bývajú uložené na nosičoch (taktiež nazývaných tokenoch), ktoré môžu mať rôznu formu:

- nosiče sú samostatné zariadenia a nijakým spôsobom sa nepripájajú k počítaču užívateľa – zvyčajne majú zabudovanú obrazovku, aby zobrazili kľúč (dôkaz);
- nosiče s možnosťou pripojenia k počítaču napr. čítačky kariet – automaticky prevádzajú dáta;
- softvérové nosiče – dáta nie sú uložené na špeciálnom zariadení, môžu byť duplikované.

### 4.5.1 Jednorázové heslo

Autentizačná schéma s jednorázovým heslom generuje heslo na základe zdieľaného tajomstva a aktuálneho času alebo pomocou počítadla, ktoré je aktívne len po dobu jednej relácie, transakcie. Metóda je využívaná pri dvoj-faktorovej autentizácií, kde sa najskôr užívateľ pošle serveru meno a heslo a následne klient aj server vygenerujú jednorázové heslo, viac v literatúre [21].

Jednorázové heslo môže byť doručené viacerými spôsobmi. Zaslanie pomocou SMS správy je najbežnejším spôsobom, avšak podľa odporúčaní NIST (National Institute of Standards and Technology) nie je najvhodnejší z dôvodu možného zachytenia SMS správ alebo podvodu presmerovania správ z daného telefónneho čísla na inú SIM kartu, z literatúry [22].

Ďalším spôsobom doručenia jednorázového hesla je skrz aplikáciu - kde je vygenerované jednorázové heslo na určitú dobu, a to je následne použité pri autentizácií.

V prípade webovej aplikácie je možné nechať užívateľa si pri prvom prihlásení vybrať obrázky z viacerých kategórií poskladaných v mriežke, kde každému obrázku prislúcha alfanumerický znak, pri ďalších prihláseniach je poradie obrázkov (aj alfanumerické znaky) zmenené pričom stále prislúchajú predom daným kategóriám. Užívateľ vyhľadá obrázky, ktoré zodpovedajú jeho predom vybraným kategóriám a pomocou príslušných alfanumerických znakov vytvorí jednorázové heslo.

Viac-faktorová autentizácia s využitím jednorázového hesla ponúka viaceré výhody: vyššia zložitosť pre útočníka kvôli kompromitovaniu SMS/emailu okrem samotnej aplikácie – autentizácia závisí od viacerých systémov, obmedzená platnosť hesla, rozšírený prístup pre útočníka je obmedzený, keďže jednorázové heslo sa môže použiť pre konkrétnu transakciu iba raz (zamedzenie útoku opakovaním - replay attack), čo je výhoda oproti statickým heslám.

Medzi nevýhody patrí možné oneskorenie pri vybavovaní požiadavok.

# 5 Možnosti autorizácie vo webových aplikáciach

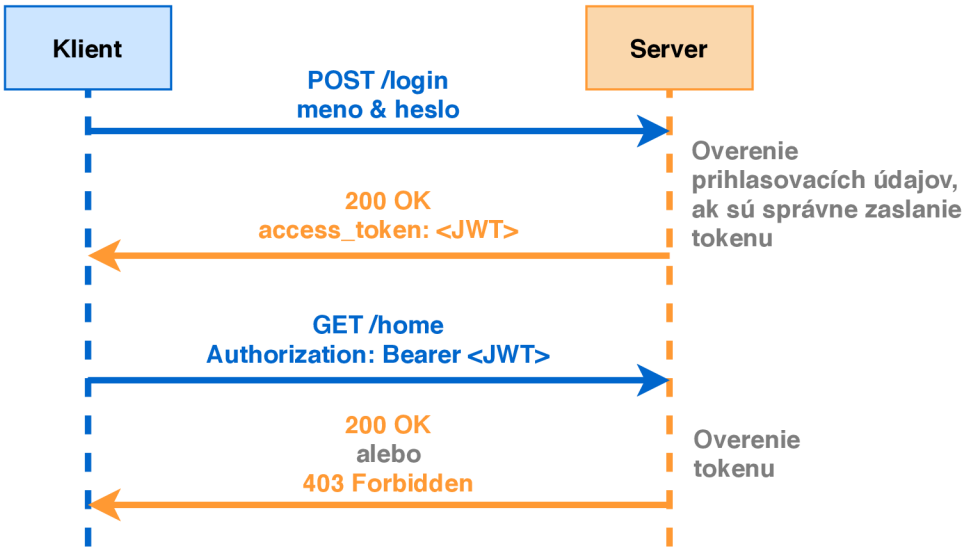
Kapitola predstavuje najčastejšie používané autorizačné mechanizmy vo webových aplikáciách. Autorizácia je proces pri ktorom sa kontrolujú práva a privilégia autentizovaných užívateľov. Pri niektorých schémach môže rozdiel medzi autentizáciou a autorizáciou splývať v jeden ucelený proces.

## 5.1 Bearer – autorizácia na základe nositeľa

Mechanizmus je založený na tokenoch a bol vytvorený ako súčasť OAuth 2.0 autorizácie, ale je používaný aj samostatne. Počas procesu server neuchováva informácie o vydaných tokenoch alebo o prihlásených užívateľoch. Viac informácií v literatúre [20] a [23].

Server klientovi vygeneruje token – náhodný reťazec rôznej dĺžky napr. v hexadecimálnom tvare alebo v tvare štruktúrovaného token JWT (JSON Web Token). JWT obsahuje tri časti: hlavičku s typom tokenu a hašovacím algoritmom, payload obsahujúci nároky a na záver podpis ako je znázornený vo výpise 5.1. Klient musí následne použiť token v poli `Authorization` pre prístup k chráneným zdrojom. Server pri prístupe klienta k chráneným zdrojom overuje token poslaný v poli autorizácie, tým že je token dekódovaný, a pri úspešnom overení je klientovi obsah zdroju sprístupnený. Proces autorizácie je zobrazený na obrázku 5.1.

Autorizačnú schému je nutné používať v súčinnosti s HTTPS protokolom.




Obr. 5.1: Schéma komunikácie počas Bearer autorizácie.

Výpis 5.1: Formát podpisu v JWT tokene.

```
HMACSHA256(base64UrlEncode(header) + "." +
    + base64UrlEncode(payload), secret)
```

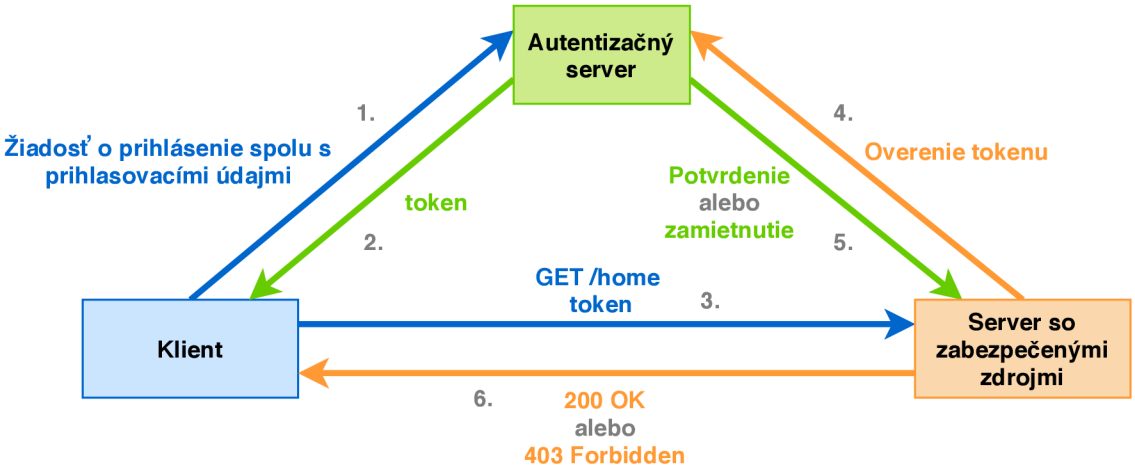
## 5.2 Autorizácia OAuth 2.0

Oproti predošlej verzií OAuth 1.0 nie je nutné podpisovať každú správu (žiadosť) kľúčom. Pri autorizačnom procese sa využíva prístupový token a obnovovací token, je možné využiť jeden alebo aj oba. Informácie informácie čerpané z [23].

Prístupový token povoľuje aplikácií pristupovať k dátam užívateľa. Obnovovací token obnoví prístupoví token, ak jeho platnosť vypršala.

OAuth 2.0 patrí medzi najlepšie voľby pre identifikáciu osobných účtov užívateľov a pridelenie správnych prístupov. Komunikácia prebieha medzi troma entitami, a to medzi klientom (užívateľom), autentizačným serverom a serverom so zdrojmi. Jedná sa o autentizáciu a následnú autorizáciu pomocou tretej strany.

Pri autentizácií sa najskôr užívateľ prihlási do systému (aplikácie), požiadavka o autentizáciu je presmerovaná na autentizačný server, kde sú od užívateľa vyžiadané prihlasovacie údaje. Autentizačný server požiadavku zamietne alebo povoli a vygeneruje prístupový alebo obnovovací token, na základe ktorého môže užívateľ pristúpiť k zabezpečeným zdrojom – pri poslaní tokenu užívateľom do aplikácie, aplikácia komunikuje s autentizačným serverom, ktorý token overuje. Zdroje sú nakoniec sprístupnené užívateľovi. Popis komunikácie je schématicky znázornený na obrázku 5.2.

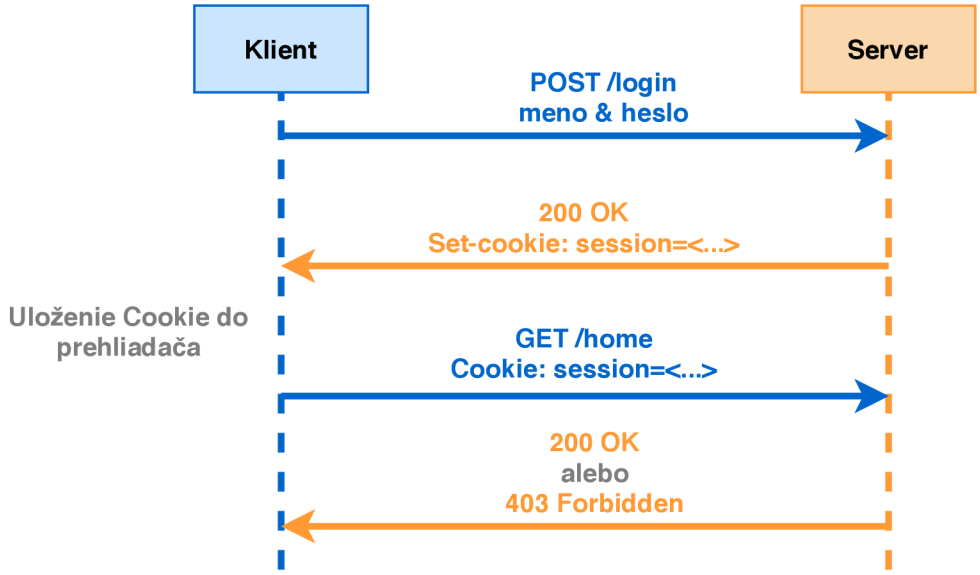



Obr. 5.2: Schéma komunikácie počas OAuth 2.0 autorizácie.

## 5.3 Autorizácia pomocou cookies

Metóda osvedčená pre autorizáciu užívateľa na dlhší čas. Záznam o autentizácií a reláciach je na oboch stranách komunikácie – na strane klienta a aj serveru. Server neustále kontroluje aktívne relácie v databáze, pričom sa vytvára reťazec cookie, ktorý udržiava informáciu o identifikačnom čísle relácie, viac v literatúre [24].

Pri komunikácií najskôr klient posiela prihlasovacie údaje, server údaje overuje a vytvára reláciu, ktorú je uložená v databáze relácií. Do prehliadača na klientskej strane je uložený reťazec cookie. Pri prístupe klienta k zabezpečeným zdrojom je reťazec cookie kontrolovaný a porovnávaný s dátami z databáze relácií. Pri odhlásení klienta sú údaje o reláciach vymazané z oboch strán komunikácie. Komunikácia je znázornená na obrázku 5.3.




Obr. 5.3: Schéma komunikácie počas Cookie autorizácie.

## 5.4 Zhrnutie

V novodobých webových aplikáciach sú využívané najmä autorizačné schémy na základe tokenov, poprípade autentizácia s následnou autorizáciou pomocou tretích strán OAuth 2.0

Tokenová autorizácia je uprednostňovaná voči autorizácií na základe cookie z dôvodu nepotrebonosti uchovávania informácií o jednotlivých tokenoch. Každý token je samostatný a obsahuje všetky potrebné informácie k overeniu jeho platnosti a taktiež obsahuje informácie o užívateľovi.

# 6 Najčastejšie útoky na webové aplikácie a možnosti ochrany

Kapitola vychádza zo zoznamu OWASP TOP 10 z roku 2017, popisuje najčastejšie útoky alebo bezpečnostné chyby pri webových aplikáciach a možnosti nápravy.

OWASP (The Open Web Application Project) je medzinárodná neprofitujúca organizácia, ktorej cieľom je zvýšiť bezpečnosť webových aplikácií. Všetky ich dokumenty, videá a odporúčania sú voľne dostupné na ich stránke pre každého. Medzi najznámejšie projekty organizácie patrí OWASP TOP 10.

Dokument OWASP TOP 10 sumarizuje desať najčastejších útokov na webové aplikácie, má slúžiť na zvýšenie povedomia pre jednotlivých tvorcov webových aplikácií, vychádza každé 3 roky a je zostavený odborníkmi z celého sveta.

Webové aplikácie môžu obsahovať viacero bezpečnostných zraniteľností, skrz ktoré je útočník schopný sa určitou cestou (cestou sa rozumie využitie jedného z útočných vektorov, jednej zraniteľnosti) dostať do systému a narušiť ho. Každá takáto zraniteľnosť predstavuje určité riziko jej zneužitia. Niektoré zraniteľnosti sú závažnejšie ako ostatné, z dôvodu že majú vážne následky na funkciu systému a následne dopad aj na podnikanie, práve tieto zraniteľnosti sú popísané v dokumente organizácie OWASP [25].

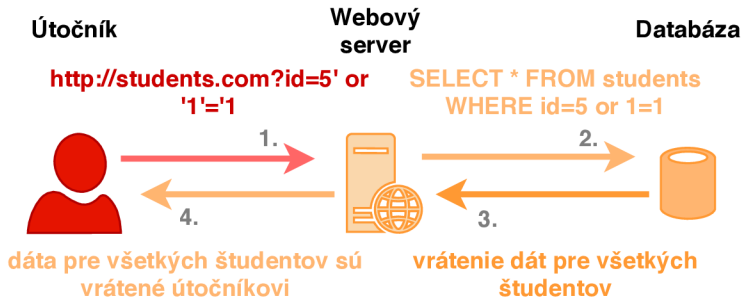
## 6.1 Injection útoky

Injection útoky (voľný preklad útoky pomocou vkladania - Injektovanie) patria medzi najstaršie typy útokov, môžu viesť k strate dát, strate identity údajov alebo k nedostupnosti služby. Trieda útokov Injection obsahuje širokú škálu útočných vektorov (napr. SQL, NoSQL, OS, LDAP), vďaka ktorým útočník vkladá nedôveryhodný vstup do programu, ktorý je následne spracovávaný serverom ako súčasť príkazu alebo dotazu, počas spracovania sa kvôli nedôveryhodnému vstupu mení priebeh vykonávania tohoto programu. Útok SQL Injection je znázornený na obrázky 6.1.

Možno predísť overením alebo tzv. dezinfikovaním dát, ktoré užívateľ posiela. Overenie znamená odmietnutie podozrivých dát, dezinfekcia znamená vyčistenie podozrivých častí z dát.

## 6.2 Nefunkčná autentizácia

Autentizácia je často nesprávne implementovaná – je nefunkčná alebo nedostatočná, čo môže viesť k odcudzeniu prihlasovacích údajov od užívateľského účtu alebo k prístupu do celého systému. Typickým príkladom je útok hrubou silou, konkrétne slov-




Obr. 6.1: Ukážka priebehu útoku SQL Injection.

níkový útok, kde útočník pomocou jednoduchého skriptu hľadá správne prihlasovacie údaje – zostavený skript skúša rôzne kombinácie prihlasovacích mien a hesiel z predom zostaveného zoznamu.

Medzi opatrenia proti útokom hrubou silou je využitie viac-faktorovej autentizácie, ktorá je viac rozobratá v časti 4.5, tiež nastavenie limitov a oneskorenia po niekoľkých neúspešných prihláseniach.

## 6.3 Nezabezpečenie citlivých dát

Ak webová aplikácia nechráni citlivé dáta typu finančných informácií alebo hesiel, útočníci môžu získať dáta a zneužiť ich pre nekalé účely napr. útokom man-in-the-middle (útok mužom uprostred).

Ochranou môže byť šifrovanie všetkých citlivých informácií, deaktivácia ukladania dát do vyrovnávacej pamäte (prax pre ukladanie údajov, ktoré sa budú opakovane využívať) a minimalizácia uchovávania citlivých údajov.

## 6.4 XML externé entity

Veľa starších a zle nakonfigurovaných webových stránok, ktoré analyzujú vstupy XML (Extensible Markup Language), môžu podporovať odkazovanie na externú entitu z daného vstupu. Analyzátor XML môže teda priamo pre-posielať údaje externej entite (externá entita môže byť reprezentovaný pevným diskom, ku ktorému pristupuje útočník).

Najlepším spôsobom ako zabrániť webovej aplikácií nechcenú komunikáciu s externými entitami pomocou vstupných XML dát je podpora iba jednoduchších dátových typov pre vstup ako je JSON. Jazyk XML parí medzi značkovacie jazyky a je určený na čítanie pre človeka, ale aj stroj, vzhľadom na jeho zložitosť a viaceré chyby sa vyraďuje z používania na webových aplikáciach.

## 6.5 Nefunkčná kontrola prístupu

Kontrola prístupu predstavuje možnosť povoliť alebo odoprieť použitie určitého zdroja určitému subjektu, riadenie prístupu k materiálom, logickým alebo digitálnym zdrojom. Nefunkčný mechanizmus kontroly prístupu umožňuje útočníkom vykonávať úlohy v mene privilegovaných užívateľov (správcov).

Zabezpečenie vedie k používaniu autorizačných tokenov a nastaveniu prísnych kontrol.

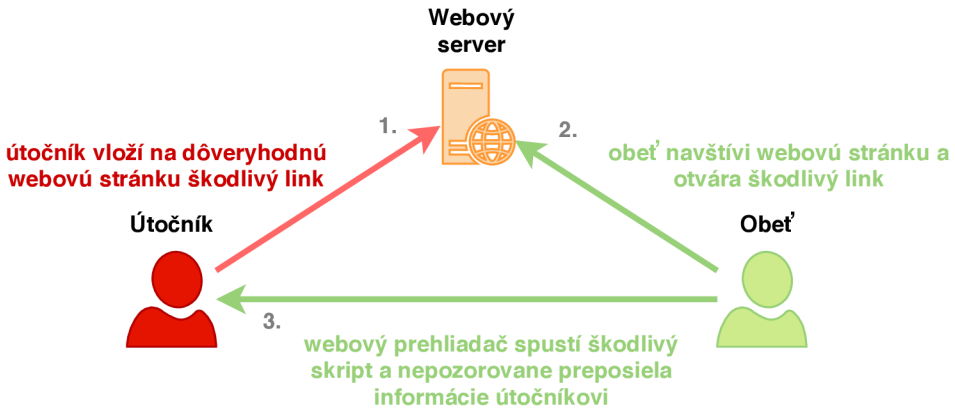
## 6.6 Chybná konfigurácia

Najčastejšia zraniteľnosť, ktorá pramení hlavne z využitia predvolených nastavení a veľmi podrobne opísaných chýb. Zraniteľné miesta môžu byť odhalené kvôli podrobnému opisu chyby, ktorá je zobrazená užívateľovi.

Opatreniami, ako odstránenie všetkých nevyužívaných funkcií z kódu, nastavenie všeobecné znenie hlášky chybovosti bez podrobných popisov, je možné zmierniť vzniknuté zraniteľnosti, ktoré sú zapríčinené chybnou konfiguráciou.

## 6.7 Cross Site Scripting XSS

Zraniteľnosť XSS nastáva vtedy, keď webová aplikácia umožňuje užívateľom pridať vlastný kód (škodlivý kód JavaScript, ktorý môže byť spustený v prehliadači obete) do URL adresy alebo na webovú stránku, ktorú uvidia ostatní užívatelia. Pomocou XSS môžu byť unesené relácie užívateľov, zničené webové stránky alebo presmerovaný užívatelia na škodlivé stránky. Priebeh útoku je znázornený na obrázku 6.2.




Obr. 6.2: Ukážka priebehu útoku Cross Site Scripting XSS.

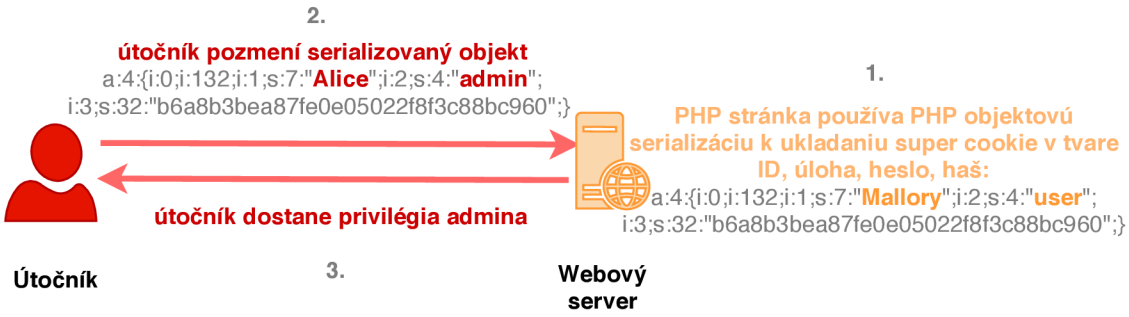
Jednou metódou zabezpečenia je zahadzovanie nedôveryhodných HTTP požiadavok a ďalšia metóda je využitie moderných frameworkov, ktoré poskytujú určitú ochranu proti skriptovaniu medzi stránkami.

## 6.8 Nezabezpečená deserializácia

Serializácia znamená prevzatie objektov z kódu a ich konverzia na iný dátový typ pre iný účel (uloženie na disk, streamovanie, ...). Deserializácia je opačný proces, kedy sa z dát stávajú opäť objekty, ktoré môže aplikácia použiť.

Zraniteľné môžu byť stránky, ktoré často serializujú a deserializujú dáta, deserializačné útoky sa zameriavajú na manipulovanie dát predtým než sa z nich stanú objekty – útoky sú výsledkom deserializácie dát z nedôveryhodných zdrojov, čo môže viesť k útokom, ako DDoS (Distributed Denial of Service – distribuovaný útok odopretia služieb). Príklad útoku PHP deserializácie je znázornený na obrázku 6.3.

Jediným istým opatrením proti útokom na deserializáciu je zabezpečenie deserializácie jedine z dôveryhodných zdrojov.




Obr. 6.3: Ukážka priebehu útoku PHP deserializácie.

## 6.9 Použitie komponentov so známymi zraniteľnosťami

Pri tvorbe webových aplikácií vývojári väčšinou využívajú už vytvorené knižnice, aby sa vyhli zbytočnej práci navyše. Niektorí útočníci vo využívaných knižniciach a často používaných komponentoch hľadajú zraniteľnosti, vďaka ktorým by sa stalo mnoho webových aplikácií zraniteľnými.

Pre minimalizovanie tohoto druhu zraniteľnosti je nutné, aby vývojári webových aplikácií používali len overené komponenty od overených zdrojov a udržiavali

ich aktuálne, v prípade závažných bezpečnostných chýb je nutné tieto komponenty z projektu vymazať.

## 6.10 Nedostatočné logovanie a monitorovanie

Pri vývoji webovej aplikácie je veľmi podstatnou časťou aj implementácia efektívnej detekcie útokov za pomoci logovania a monitorovania, mnoho webový aplikácií detekuje útok až po 200 dňoch, čo ponúka obrovský priestor útočníkom.

Vo webovej aplikácií je nutné zaznamenávať informácie o neúspešných prihláseniach, o chybných riadeniach prístupu a vstupov na strane serveru, o podozrivých účtoch, ktoré môžu byť nebezpečné. Treba zaistiť, aby logy boli v rovnakom formáte a bolo možné ich hneď spracovávať, tiež treba zabezpečiť zálohovanie logov na externé úložisko a zaistiť ich integritu. Na základe zaznamenaných udalostí je možné zostaviť monitorovanie, ktoré pri podozrivej udalosti vydá upozornenie.

# 7 Praktická implementácia

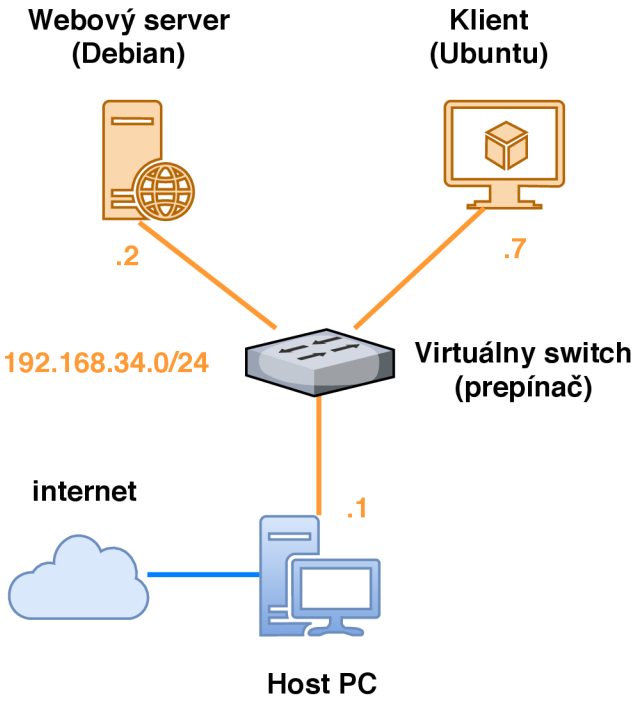
Podľa zadania bakalárskej práce sa mala vytvoriť webová aplikácia, ktorá by bola schopná spracovávať súbory, podľa potreby ich šifrovať, od klienta a posielať ich na server pomocou bezpečného spojenia TLS.

V kapitole bude rozobratá praktická implementácia, a to prostredie, na ktorom bola aplikácia nasadená, vývoj webovej aplikácie, konfigurácia webového serveru, bezpečnostné opatrenia. Všetky vytvorené zdrojové súbory webovej aplikácie a skripty, ktoré boli vytvorené alebo upravené, sa nachádzajú na priloženom médiu, popísané v prílohe A.

## 7.1 Prostredie

Vývojové prostredie sa skladá z dvoch virtuálnych strojov: server s operačným systémom Debian bez grafického rozhrania, klient s operačným systémom Ubuntu.

Medzi virtuálnymi strojmi a hostujúcim počítačom je vytvorená izolovaná sieť, pripojenie do vonkajšej siete k internetu by nebol nutný, ale vo veľkej miere uľahčuje inštaláciu všetkých potrebných balíkov. Nasledujúci obrázok znázorňuje prepojenie virtuálnych strojov 7.1.

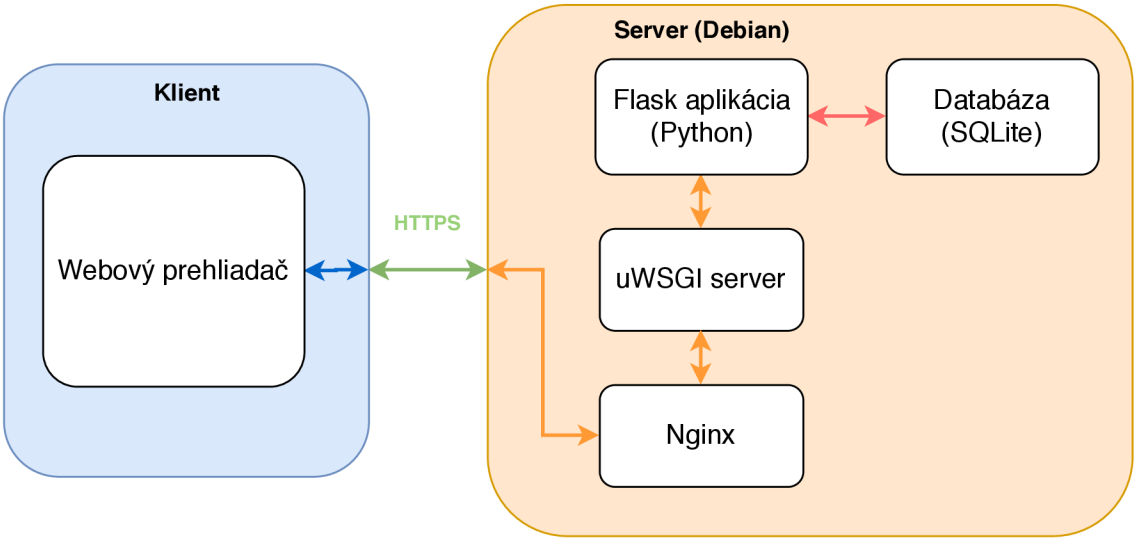



Obr. 7.1: Ukážka prepojenia virtuálnych strojov v sieti.

## 7.2 Koncepčný návrh webovej aplikácie

Návrh webovej aplikácie znázornený na obrázku 7.2 popisuje klientsku a serverovú stranu a ich jednotlivé súčasti. Na strane klienta sa nachádza webový prehliadač, pomocou ktorého klient komunikuje so serverovou stranou kanálom, ktorý je zabezpečený protokolom HTTPS.

Serverová strana obsahuje samotnú Flask aplikáciu napísanú v jazyku Python, aplikácia komunikuje s databázou, kde sú uložené údaje o účtoch užívateľov. Webový server Nginx neumožňuje komunikáciu s aplikáciami napísanými v jazyku Python, preto je využitý WSGI (Web Server Gateway Interface) server – je využitá implementácia uWSGI, ktorá aplikáciu transformuje do zrozumiteľnej podoby pre webový server. Webový server následne spätne komunikuje s klientom.




Obr. 7.2: Koncepčný návrh webovej aplikácie.

## 7.3 Vývoj webovej aplikácie

Webová aplikácia bola vytvorená ako viac-stránková aplikácia (viac v podkapitole 1.3.1), samotný vývoj webovej aplikácie prebiehal na hostujúcom počítači v prostredí PyCharm. V časti 1.1 už bolo spomenuté, že aplikácia je vyvíjaná v Pythone (časť 1.1.1) vo frameworku Flask (časť 1.1.2) a pre HTML a CSS súbory bol použitý framework Bootstrap, opísané v časti 1.1.3.

Vývoj webovej aplikácie je možné rozdeliť na časť autentizačnú a časť tvorby funkcionality. Aplikácia disponuje dvoj-faktorovou autentizáciou (podrobnejšie popísané v kapitole 4.5), kde prvým faktorom je znalosť hesla a druhým faktorom je

znalosť vybratých kategórií, na základe ktorých vytvorí jednorázové heslo.

Hlavnou úlohou webovej aplikácie je prenášať súbory od klienta na server, stránka s touto funkcionalitou je chránená, iba užívateľ, ktorý sa nachádza v databáze, môže využívať funkciu posielania súborov na server. Užívateľ má možnosť pri nahrávaní súborov na server zvoliť či sa má súbor ukladať v šifrovanej alebo v otvorenej podobe.

Medzi dodatočné funkcie webovej aplikácie patrí aj možnosť dešifrovania súborov a opätovného stiahnutia nahratého súboru zo serveru.

Flask aplikácia má následujúcu štruktúru 7.1, štruktúra daná dokumentáciou Flasku. V zložke `database` sa nachádzajú skripty napísané v Pythone, skript `createDB.py` vytvára databázu o dvoch tabuľkách (užívateľov a súborov), ktoré majú medzi sebou vzťah typu one-to-many (jeden k viacerým) – jeden užívateľ môže mať viacero súborov. Tabuľka užívateľov sa vytvára na základe modelu **User** s položkami **id** (identifikačné číslo), **name** (prihlasovacie meno), **password** (heslo), **file** (súbor) a položkami jednotlivých deviatich kategórií (potrebné pre vytvorenie jednorázového hesla), ktoré nadobúdajú hodnoty True alebo False. Položka file vytvára vzťah medzi modelom User a medzi modelom **File**. Model File obsahuje položky **id**, **name**, **time** (čas), kedy bol súbor pridaný, a **user_id**, identifikačné číslo užívateľa, ktorému súbor patrí. K vytvoreniu databázy sa využívajú metódy z knižnice *SqlAlchemy*, knižnica dokáže pracovať s rôznymi druhmi databáz, v prípade danej webovej aplikácie je vytvorená SqLite databáza. Ďalej je v skripte využitá knižnica *werkzeug.security*, metóda pre generovanie hašu je následne implementovaná tak, aby heslo v databáze bolo uložené v tvare hašu, presnejšie PBKDF2:SHA256[1]. Pomocou skriptu `insertData.py` sú pridaní užívatelia do databázy.

V zložke `static` sú umiestené súbory CSS z Bootsrtrap frameworku, ale taktiež `myCssFile.css`, ktorý upravuje priamo dizajn danej webovej aplikácie – farby, umiestnenie komponentov, veľkosť tlačidiel.

HTML šablóny pre jednotlivé stránky obsahuje zložka `templates`. Súbor `errorTemplate.html` sa využíva ak dôjde k chybe, aby na základe defaultnej stránky webového serveru nemohlo byť zistené, o aký server sa presne jedná. Šablóna `base.html` je rodičom pre ostatné šablóny, potomkovia dedia štruktúru a pomocou blokov, ako `{% block title %}`, `{% block header %}`, `{% block content %}` vkladajú obsah.

Skript `main.py` spúšťa celú webovú aplikáciu. Definuje jednotlivé koncové body aplikácie pomocou dekorátoru `@app.route()`, ako je znázornené vo výpise 7.2,

[1]Password-Based Key Derivation Function 2 – funkcia odvodenia kľúča, vyznačuje sa posuvným výpočtom, aby znížila zraniteľnosť na brute-force útok. Funkcia PBKDF2 aplikuje pseudonáhodnou funkciu (SHA256), k vstupnému heslu pridá náhodnú informáciu napr. soľ a tento proces zopakuje v niekoľkých iteráciách, výsledkom je odvodený kryptografický kľúč.

Výpis 7.1: Štrúktúra Flask webovej aplikácie.

```
webApp
    \database
        \createDB.py
        \insertData.py
    \static
        \images # zložka obsahuje obrázky kategórií
                # k vytvoreniu jednorázového hesla
        \bootstrap.min.css
        \bootstrap.min.css.map
        \myCssFile.css
    \templates
        \errors
            \errorTemplate.html
        \base.html
        \contact.html
        \file.html
        \login.html
        \otp.html
        \upload.html
    \main.py
    \otp_func.py
```

podobným spôsobom upravuje chybové stránky pomocou dekorátoru `@app.errorhandler()`. Aplikácia má nasledujúce koncové body:

- `/` – upravuje prihlasovanie užívateľa, overuje prihlasovacie meno a heslo s databázou, ak je užívateľ úspešne nájdený v databáze, udalosti prihlásenia prvého stupňa je priradená hodnota `True` (prvý dôkaz znalosti), stránka sa presmeruje na koncový bod `/otp`.
- `/otp` – na stránke sa zobrazuje deväť kategórií (obrázkov), ktoré menia priradené alfanumerické znaky každých 30 sekúnd, užívateľ musí na základe svojich zvolených kategórií (ktoré sú uložené v databáze) zložiť svoje jednorázové heslo, ak je správne – udalosti prihlásenia druhého stupňa je priradená hodnota `True` (druhý dôkaz znalosti) a užívateľ je presmerovaný na koncový bod `/upload`.
- `/logout` – obe udalosti prihlásenia sú ukončené (priradí sa hodnota `False`) a stránka sa presmeruje na úvodné prihlasovanie.

- `/upload` – hlavná stránka aplikácie, užívateľ môže nahrať súbory v otvorenej alebo šifrovanej podobe. Ak zvolí šifrovanie súboru je možné naraz nahrať iba jeden súbor a je nutné v dialógovom okne zadať heslo pre šifrovanie (z ktorého je následne vyrátaný kľúč). Užívateľ má možnosť vybrať algoritmus pre šifrovanie: blokovú symetrickú šifru AES s dĺžkou kľúča 256 bitov v móde CBC (implementované pomocou knižnice `Crypto.Cipher`) alebo blokovú symetrickú šifru Camellia[2] taktiež s 256-bitovým kľúčom v móde CBC (implementované pomocou knižnice `cryptography.hazmat`). Súbory sa ukladajú do zvlášť zložky pre každého užívateľa `..\uploadFiles\<username>`. Stránka ponúka aj možnosť dešifrovania, v tomto prípade je nutné nahrať súbor, zvoliť algoritmus a zadať heslo, aplikácia následne zašle dešifrovaný súbor do prehliadača užívateľa. Počas dešifrovania sa súbor uloží iba dočasne do `..\tmp` zložky, z ktorej je ihneď po dokončení funkcie vymazaný.
- `/file` – zobrazuje v tabuľke všetky súbory, ktoré užívateľ nahral na server, tieto súbory je možné opätovne stiahnuť.
- `/contact` – informácie o autorovi a kontakt.

Výpis 7.2: Tvorba koncového bodu pomocou frameworku Flask.

```
@app.route('/')
def index():
    return 'Hello world!'
```

Všetky koncové body majú ošetrený prístup, ak nie užívateľ prihlásený je automaticky presmerovaný na úvodnú stránku prihlasovania.

### 7.3.1 Záznam udalostí

Známe pod názvom ako „logovanie“ je podstatnou súčasťou aplikácií, ktorá pomáha pri riešení problémov, chýb v samotnej aplikácií, ale aj pri detekcií rôznych anomálií napr. chyby pri autentizácií, nadmerné užitie systému a podobne, na základe ktorých je možné detekovať nepriaznivé udalosti.

Jednotlivé záznamy obsahujú zvyčajne informácie o čase, zdroji, užívateľovi a udalosti, poprípade závažnosti. Informácie takéhoto typu zaznamenáva aj vytvorená webová aplikácia do samostatného súboru (ukážka vo výpise 7.3) pri udalostiach: úspešného/neúspešného prihlásenia, nahrania súboru, dešifrovania súboru a stiahnutia súborov.

[2]Šifrovací algoritmus Camellia má porovnateľnú úroveň zabezpečenia a schopnosti spracovania ako AES, algoritmus bol schválený ISO/IEC, spoločná technická komisia organizácií ISO (International Organization for Standardization) a IEC (International Electrotechnical Comission).

Výpis 7.3: Ukážka zo súboru pre záznam udalostí.

```
2020-05-20 13:27:07,283 DESKTOP-NNMH0OU 192.168.34.1
    INFO: slavka logged in (first factor).
2020-05-20 13:27:18,629 DESKTOP-NNMH0OU 192.168.34.1
    INFO: slavka logged in (second factor).
2020-05-20 16:18:13,108 DESKTOP-NNMH0OU 192.168.34.1
    INFO: slavka uploaded file Bakalarska_praca.pdf
2020-05-20 16:19:53,694 DESKTOP-NNMH0OU 192.168.34.1
    WARNING: Failed log in (first factor).
```

## 7.4 Konfigurácia serverovej časti

Po inštalácií operačného softvéru Debian bolo potrebné aktualizovať lokálny index balíkov a nainštalovať potrebné balíčky, výpis 7.4. Za pomoci programu WinSCP bola zložka skriptov prenesená z hostového počítača na virtuálny stroj serveru, kde bola umiestnená v domovskej zložke užívateľa `student` (obyčajný užívateľ bez práv super užívateľa `root`) – `\home\student\webTestApp`.

Výpis 7.4: Inštalácia potrebných balíkov.

```
apt-get update
apt-get install python3, python3-pip, python3-dev nginx
pip3 install virtualenv
```

V zložke `webTestApp` bolo vytvorené virtuálne prostredie `venv` (s kópiou použitého Pythonu 3.7) pomocou príkazu na prvom riadku vo výpise 7.5. Po aktivovaní virtuálneho prostredia (druhý riadok výpisu) je nutné naištalovať potrebné balíčky (tretí riadok výpisu). Keďže webové servery, ako napr. Nginx alebo Apache nevedia komunikovať s kódom napísaným v Pythone, preto je nutné vytvoriť WSGI server (bola použitá uWSGI implementácia), ktorý aplikáciu podá webovému serveru v zrozumiteľnej forme. K správnej interakcií uWSGI serveru bol vytvorený skript `wsgi.py` taktiež v zložke `webTestApp`, ktorý určuje vstupný bod aplikácie. Ďalej bol vytvorený konfiguračný súbor, `webTestApp.ini`, pre `wsgi` skript, kde sa nastavuje názov `wsgi` skriptu v položke modul, spustenie v master móde, počet procesov, je definované vytvorenie socketu (pre komunikáciu medzi Nginxom a uWSGI), sú nastavené práva pre socket, pomocou príkazu `vacuum = true` je nastavené vyčistenie socketu pri zastavení procesu, poslednou položkou je `die-on-term = true`, ktorá zaručí rovnaké správanie systémov init.d a uWSGI – pôvodne jednotlivé procesné signály spracúvajú rôznym spôsobom.

Výpis 7.5: Tvorba virtuálneho prostredia.

```
1 virtualenv venv
2 source venv\bin\activate
3 pip3 install flask uwsgi
```

Podstatným krokom je vytvorenie init.d skriptu – `webTestApp.conf`, zaručí, že vždy pri zapnutí Debianu, systém init.d automaticky zapne server uWSGI, ktorý aplikáciu predá webovému serveru Nginx. Skript je umiestnený v `\etc\ini.d`, definuje spustenie pri úrovniach behu systému s číslami 2, 3, 4, 5[3], podobne je nastavená aj podmienka ukončenia, následne je určený užívateľ a skupina pre spustenie uWSGI serveru (je nutné, ako skupinu nastaviť `www-data`, keďže pod ňou sa spúšťa Nginx), pre spustenie je dôležité nastaviť cestu k virtuálnemu prostrediu, cestu k zložke aplikácie a príkaz, ktorým sa spúšťa konfiguračný súbor uWSGI serveru.

### 7.4.1 Implementácia TLS protokolu

Protokol TLS je podrobne teoreticky vysvetlený v častiach 3.1, 3.2. Na zabezpečenie komunikácie je využitý certifikát typu Self-Signed.

Dvojica certifikátu a súkromného kľúča sa vytvorí pomocou `openssl` knižnice, príkaz 7.6. Certifikát bude vytvorený podľa štandardu X.509 na dobu jedného roku (pri bežnom certifikáte je vhodné voliť kratšie obdobie napr. 3 mesiace), prepínač `nodes` určí, aby sa nepoužívalo žiadne heslo pri otváraní súboru (Nginx musú byť schopný pristupovať k súboru bez akejkoľvek interakcie užívateľa), kľúč bude vytvorený RSA algoritmom o dĺžke 2048 bitov. Po zadaní príkazu 7.6 je nutné upresniť informácie o servere, kvôli certifikátu.

Výpis 7.6: Tvorba certifikátu.

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048
 -keyout /etc/ssl/private/nginx-selfsigned.key -out
 /etc/ssl/certs/nginx-selfsigned.crt
```

Pre vytvorenie Perfect Forward Secrecy je nutné vytvoriť silnú grupu o dĺžke 4096 bitov pre algoritmus EDH alebo ECDHE pomocou príkazu 7.7, prečo používať Perfect Forward Secrecy je popísané v časti TLS Handshake Protokol 3.2.

[3]Úroveň behu systému je stav init.d a celý systém, hovorí o tom, ktoré systémové služby fungujú. Úrovne behu sú označené jednotlivými číslami, Linux kernel definuje 7 úrovni. Pri zapnutí je najskôr spustený systém init.d, zistí príslušnú úroveň behu a na základe toho spustí príslušné skripty.

Výpis 7.7: Tvorba grupy pre algoritmus EDH alebo ECDH.

```
openssl dhparam -out /etc/nginx/dhparam.pem 4096
```

Pomocný súbor `\etc\nginx\snippets\self-signed.conf`, ktorý bude odkazovať na umiestnenia vygenerovaného súkromného kľúča a certifikátu, je vytvorený kvôli implementácií v Nginx servere. Ďalej je vytvorený súbor `\etc\nginx\snippets\ssl-params.conf`, ktorý obsahuje konfiguráciu samotného protokolu TLS (detailnejšie informácie o možnostiach nastavenia v literatúre [26]), upravuje:

- verziu – v prípade aktuálneho nastavenia Serveru sa jedná o verziu TLS 1.3,
- odkazuje na umiestnenie vygenerovanej grupy,
- kryptografické balíčky – `TLS13-AES-256-GCM-SHA384`,
- autentizačný algoritmus – ECDH alebo EDH,
- uprednostnenie serverových šifrovacích balíčkov,
- nastavenie veľkosti vyrovnávacej pamäte,
- zakázanie obnovenia relácií pomocou vstupeniek,
- povolenie OCSP staplingu (Online Certificate Status Protocol zabezpečuje lepší výkon vyjednávania TLS pri zachovaní súkromia užívateľov), viac v literatúre [27],
- pridanie DNS (Domain Name System) prekladaču,
- nastavenie automatického prechodu na HTTPS stránky (HSTS – HTTP Strict Transport Security, pomáha chrániť webovú aplikáciu pred útokmi, ako napr. odpočúvaniu alebo SSL Strip),
- nastavenie voči ukradnutiu kliknutia (click jacking),
- zabránenie MIME (Multipurpose Internet Mail Extensions) sniffingu (pri zlom nastavení Content-Type v hlavičke si prehliadač sám upraví typ obsahu podľa odpovede), MIME sniffing môže viesť až k XSS (Cross-Site Scripting) útoku, podrobnejšie informácie o útoku MIME sniffing v literatúre [28],
- zabránenie XSS útoku.

Pomocou skriptov `self-signed.conf` a `ssl-param.conf` bude v konfiguračnom súbore Nginxu zabezpečené TLS a tiež iné bezpečnostné opatrenia, viac v časti 7.4.2.

### 7.4.2 Konfigurácia serveru Nginx

Správanie Nginx serveru je nastavené predovšetkým pomocou skriptu `\etc\nginx\sites-available\webTestApp` 7.8. Nginx načúva na porte 443, pomocou príkazu `include` zahŕňa skripty ohľadom nastavenia TLS protokolu, ďalej je určené meno serveru a tiež sa určuje umiestnenie vytvoreného socketu pre prácu

s uWSGI serverom na zobrazenie Flask aplikácie.

Server takisto načúva aj na porte 80 a všetky nezabezpečené HTTP požiadavky presmerováva na port 443 pomocou status kódu 301 – Moved Permanently (Trvalo presťahovaná stránka).

Pre nastavenie zmeny názvu serveru v zobrazovanej hlavičke odpovedí je nutné doinštalovať balíček `nginx_extras`. V súbore `\etc\nginx\nginx.conf` v bloku `http` je nastavený prepínač `server_tokens off` (zaručuje, že o servere nebudú informácie o presnej verzií) a prepínač `more_set_headers 'Web Server'` (zobrazovaný názov serveru je „Web Server“).

V súbore `\etc\nginx\nginx.conf` je možné tiež konfigurovať kompresiu odpovedí, ktoré bude server posielať klientovi, aby sa zaistilo rýchlejšie načítanie stránok, viac v literatúre [29].

Výpis 7.8: Konfiguračný súbor pre Nginx server.

```
server {
    listen 443 ssl;
    listen [::]:443 ssl;
    include snippets/self-signed.conf;
    include snippets/ssl-params.conf;
    server_name 192.168.34.2;

    location / {
        include uwsgi_params;
        uwsgi_pass unix:/home/student/webTestApp/
                              /webTestApp.sock;
    }
}
server {
    listen 80;
    listen [::]:80;
    server_name 192.168.34.2;
    return 301 https://$server_name$request_uri;
}
```

Posledným krokom je nastavenie `ufw` firewallu (program pre správu netfilter firewallu), kde je zamietnuté užívanie profilu `Nginx HTTP` príkazom 7.9 a povolená zostane len verzia `Full`, ktorá podporuje prácu s HTTPS prevozom.

Výpis 7.9: Natavenie firewallu.

```
#zobrazia sa aktívne profily s~číslom riadku
```

```
/usr/sbin/ufw status numbered
#odstráni sa profil 'Nginx HTTP'
/usr/sbin/ufw delete <číslo riadku pre 'Nginx HTTP'>
```

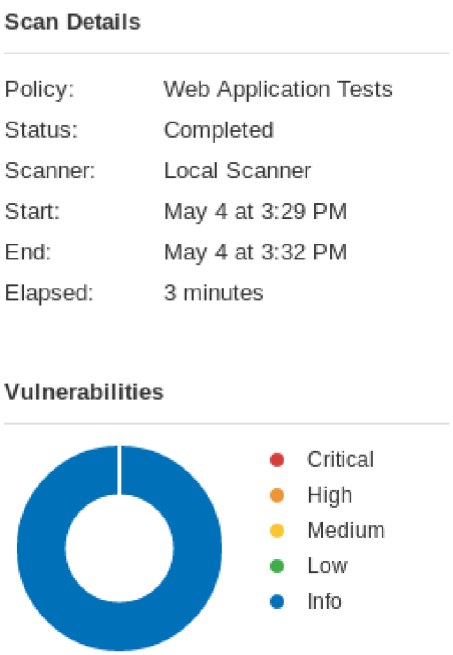
## 7.5 Testovanie bezpečnosti webovej aplikácie

Bezpečnosť webovej aplikácie je bola testovaná pomocou automatizovaných nástrojov, ako Nessus (literatúra [30]), JMeter (lit. [31]) a Lighthouse (lit. [32]).

Nessus je proprietárny skener zraniteľností vyvinutý spoločnosťou Tenable, Inc. Hlavnou funkciou je detekcia potenciálnych zraniteľností systému, medzi príklady nájdených zraniteľností patrí – nájdenie zraniteľných miest, ktoré by mohli umožniť neoprávnený prístup k citlivým dátam, nesprávna konfigurácia, predvolené heslá alebo chýbajúce heslá na systémových účtoch, zraniteľnosť odmietnutia služby.

Skenovanie pokrýva širokú škálu technológií vrátanie operačných systémov, sieťových zariadení, hypervizorov, databáz, webových serverov a kritickej infraštruktúry.

Nástroj je možné používať s istými obmedzeniami zdarma pre nekomerčné účely pod názvom Nessus Essentials, týmto nástrojom bola otestovaná finálna verzia vytvorenej webovej aplikácie. Výsledky sú prezentované na obrázku 7.3. z obrázku vyplýva, že pri komplexnom teste zraniteľností neboli nájdené žiadne podstatné, kritické chyby.



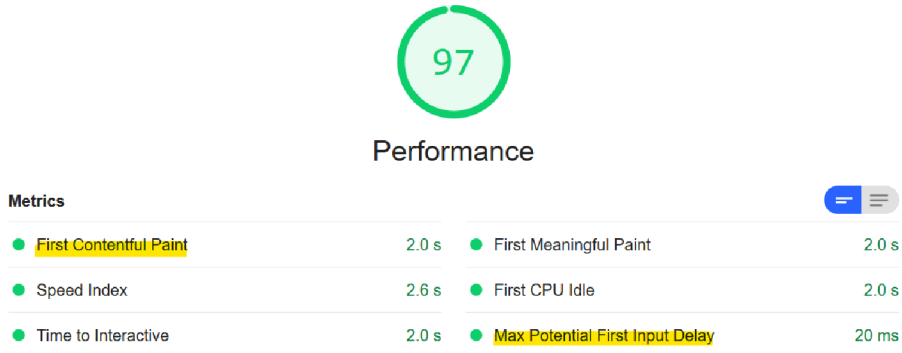

Obr. 7.3: Ukážka výsledkov testovania zraniteľností nástrojom Nessus Essentials.

Webová aplikácia bola testovaná aj pomocou open-source softvéru JMeter, ktorý slúži predovšetkým na záťažové testovanie webových aplikácií. Pri testovaní bola vytvorená skupina o 20-tich užívateľoch, ktorý posielajú HTTP požiadavky typu GET a POST na stránku prihlasovania. Oneskorenie medzi spustením všetkých užívateľov bolo nastavené na jednu sekundu, celý test bol opakovaný 3000-krát. Na server bolo celkovo zaslaných

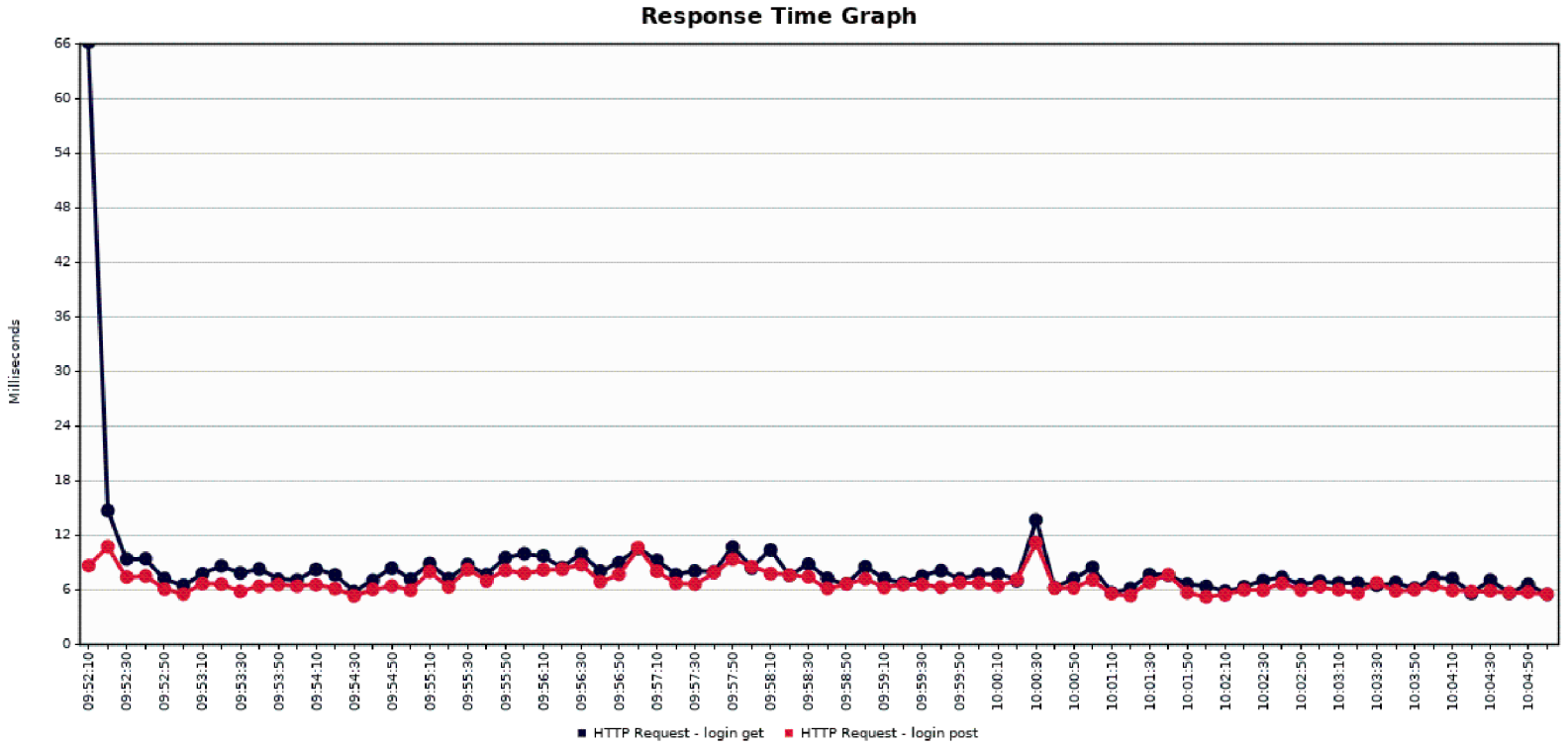
$$20 \text{ užívateľov} \times 2 \text{ požiadavky} \times 3000 \text{ opakovaní} = \mathbf{120000\ HTTP\ požiadavok.}$$

Výsledky testu sú znázornené na grafe odozvy 7.4, kde čiernou linkou sú znázornené HTTP požiadavky typu GET a červenou sú označené HTTP požiadavky POST. Čas odozvy servera je ovplyvnený množstvom prichádzajúcej prevádzky, zdrojmi, ktoré používa webová aplikácia, softvérom, ktorý server využíva a samozrejme hostiteľským riešením, ktoré je použité. Čas odozvy je možné definovať ako čas medzi užívateľovím vstupom v prehliadači po doručenie odpovede. Podľa odporúčaní spoločnosti Google (z literatúry [33]), maximálny čas odozvy by nemal presiahnuť 300 milisekúnd, inak je už stránka považovaná za nedostatočnú. Z grafu vyplýva, že prvý čas odozvy je rádovo vyšší ako ostatné, to je zapríčinené prvotným načítaním stránky a jej obsahu, napriek tomu je stále limit 300 milisekúnd splnený.

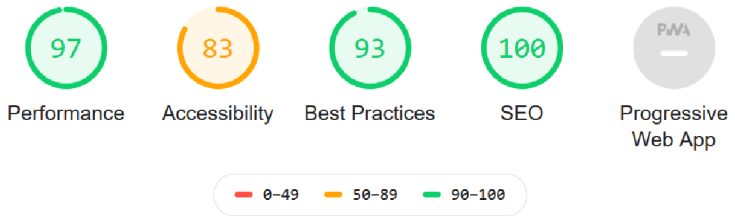
Spoločnosť Google taktiež rozlišuje aj pojem tzv. First Contentful Pain (FCP), čo v preklade znamená prvé obsahové vykreslenie (vyobrazenie), čo je možné chápať, ako čas medzi prvým kliknutím na URL adresu až po prvé zobrazovania obsahu na stránke. Čas FCP by nemal podľa odporučení presiahnuť 3 sekundy, viac v literatúre [34]. Časový údaj bol pre vytvorenú webovú aplikáciu zmeraný pomocou open-source automatizovaného nástroja Lighthouse vyvíjaným spoločnosťou Google. Nástroj okrem iného taktiež umožňuje ohodnotenie výkonnosti, prístupnosti, osvedčených postupov a SEO. Výsledky z hľadiska výkonnosti sú zobrazené na obrázku 7.5, ostatné vlastnosti na obrázku 7.6.




Obr. 7.5: Ukážka ohodnotenia výkonnosti nástrojom Lighthouse.




Obr. 7.4: Ukážka výsledkov záťažového testovania nástrojom Jmeter.




Obr. 7.6: Ohodnotenie vlastností webovej aplikácie nástrojom Lighthouse.

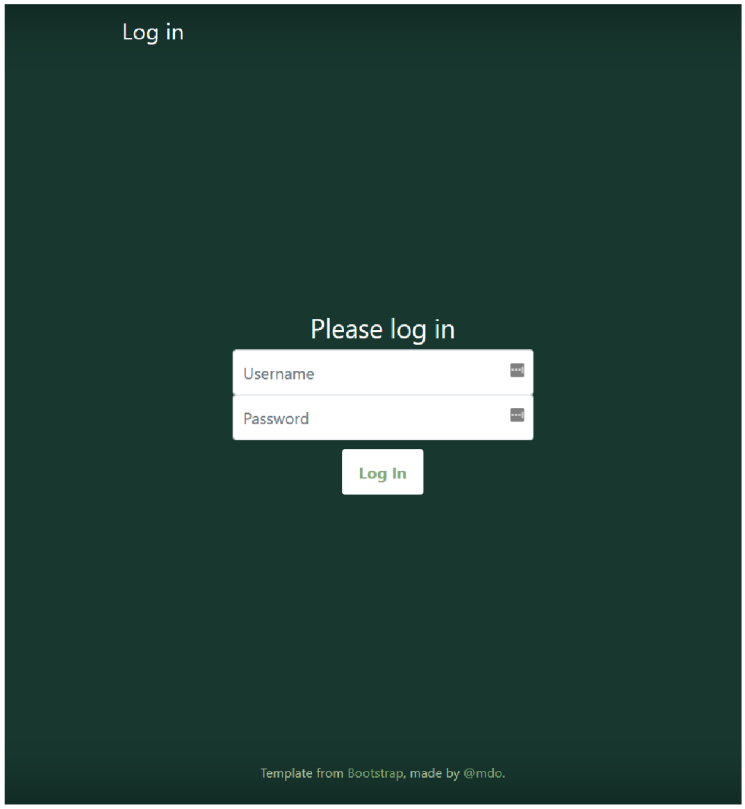
Webová aplikácia celkovo prešla prevedeným testovaním, pomocou nástroju Nessus neboli objavené žiadne potencionálne zraniteľnosti – je možné konštatovať, že mechanizmy, ktoré boli konfigurované (hlavne v podkapitole 7.4.1) plnia svoju funkciu podľa predpokladov. Aplikácia taktiež obstála počas testovania záťaže a spĺňa limity, dané spoločnosťou Google, maximálneho času odozvy (maximálne 300 milisekúnd) a prvého obsahového vykreslenia (maximálne 3 sekundy).

## 7.6 Konečný vzhľad webovej aplikácie

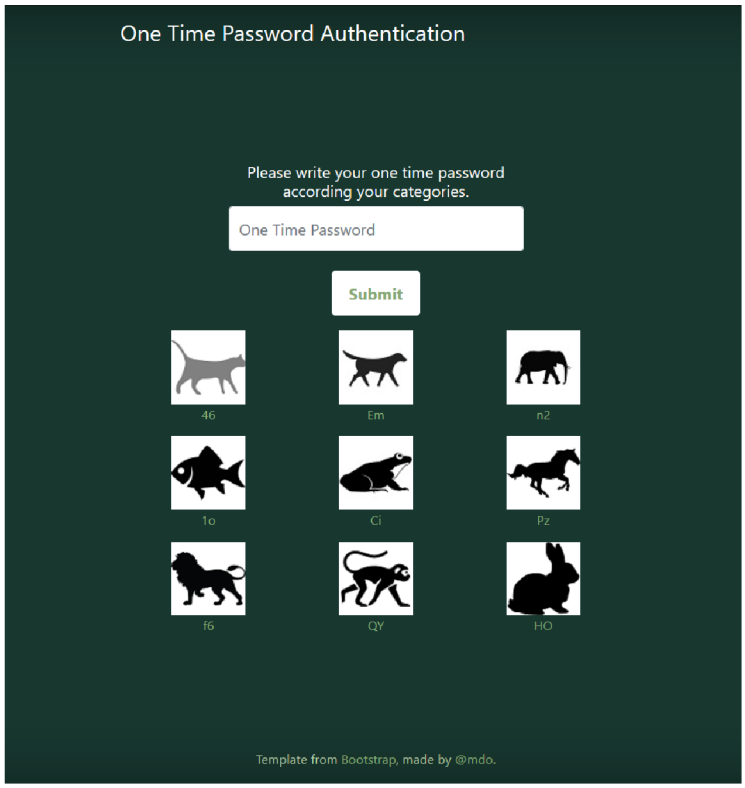
Sekcia popisuje finálnu verziu vzhľadu webovej aplikácie vytvorenej v bakalárskej práci, ktorý bol vytvorený pomocou frameworku Bootstrap.

Webová aplikácia obsahuje stránku pre prihlasovanie, zobrazenú na obrázku 7.7, užívateľ zadáva svoje heslo – prvý dôkaz. Po úspešnom dokázaní znalosti prvého faktoru je užívateľ presmerovaný na stránku so zadaním jedno-rázového hesla, na obrázku 7.8, užívateľ musí zostaviť jedno-rázové heslo na základe svojich kategórií[4]. Po úspešnom prihlásení je užívateľ presmerovaný na hlavnú stránku, kde je možné nahrávať súbory na server v otvorenej alebo v zašifrovanej podobe (AES-256-CBC alebo Camellia-256-CBC), kde je nutné zvoliť prístupovú frázu. Na danej stránke je možné súbory taktiež dešifrovať, a to spôsobom, že užívateľ nahrá súbor, zvolí algoritmus pre dešifrovanie, zadá prístupovú frázu – svoj výber nakoniec potvrdí, server odpovedá zaslaným dešifrovaný súbor do prehliadača, ktorý je možné stiahnuť, stránka je zobrazená na obrázku 7.9. V pravom hornom rohu sa nachádza navigačný panel s odkazmi na ďalšie stránky: stránka `/file` (obsahuje tabuľku so zoznamom súborov, ktoré užívateľ nahral na server, súbory je možné opätovne stiahnuť po kliknutí na tlačidlo `Save`, na obrázku 7.10), stránka `/contact` (informuje o autorovi aplikácie) a poslednou položkou je odkaz na odhlásenie z aplikácie.

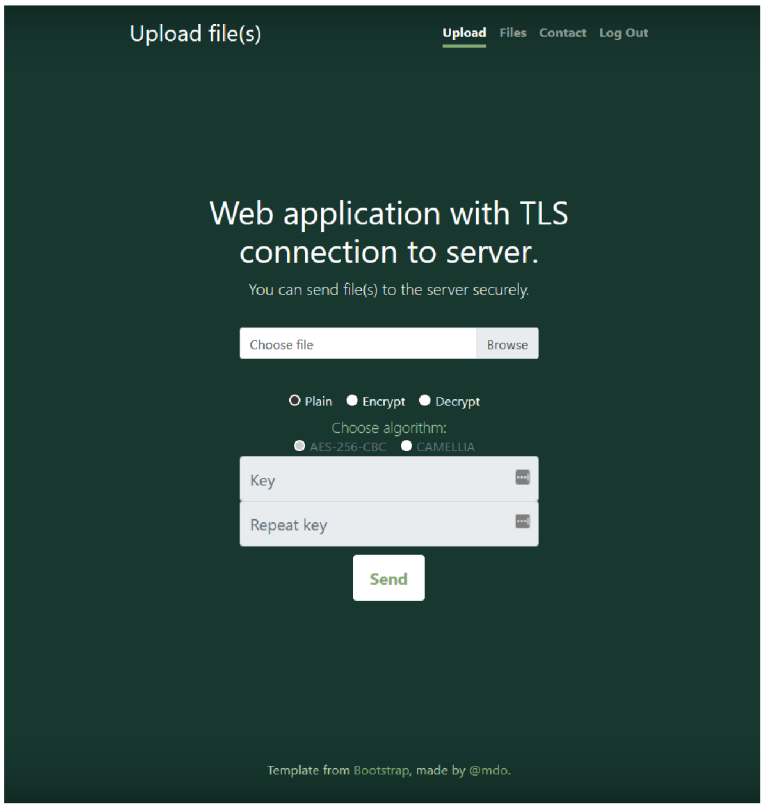
[4]Pojmom kategórie sa rozumejú jednotlivé druhy zvierat (mačka, pes, slon, ryba, žaba, kôň, lev, opica a zajac), ktoré sú na stránke zobrazené, vid obrázok 7.8.



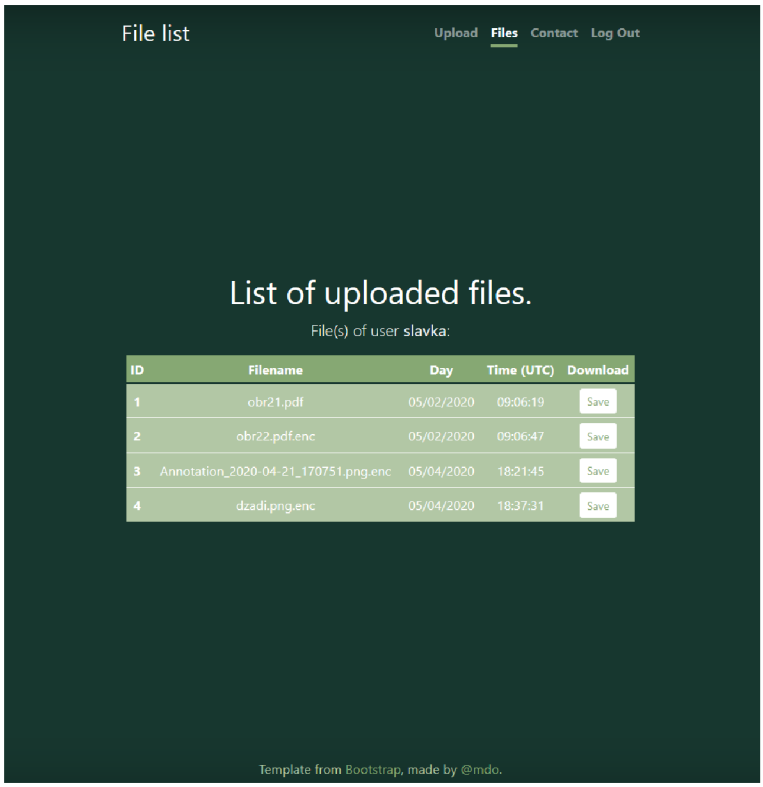

Obr. 7.7: Ukážka stránky pre prihlasovanie.




Obr. 7.8: Ukážka stránky pre jedno-rázové heslo.




Obr. 7.9: Ukážka stránky s nahrávaním súborov a dešifrovaním.




Obr. 7.10: Ukážka stránky so zoznamom nahratých súborov užívateľa.

# 8 Záver

Témou práce bola tvorba webovej aplikácie s funkciou nahrávania súborov na server po zabezpečenom spojení TLS. V prvých šiestich kapitolách je čitateľ oboznámený so súvisiacou teóriou – pojem webová aplikácia, proces tvorby, protokoly pre zabezpečenú komunikáciu klient – server, infraštruktúra verejných kľúčov, možnosti autentizácie a autorizácie a najčastejšie útoky na webové aplikácie.

Kapitola o praktickej implementácií popisuje samotný vývoj aplikácie, proces testovania a dosiahnuté výsledky. Pre tvorbu bol využitý framework Flask, ktorý je založený na jazyku Python a uľahčuje tvorbu webovej aplikácie, využitý bol aj framework Bootstrap k vývoju HTML a CSS súborov.

Webová aplikácia je nasadená na virtuálnom stroji s operačným systémom Debian, je plne funkčná, je možné nahrávať súbory na server, šifrovať ich a dešifrovať, prezerať históriu práce so súbormi a opätovne ich stiahnuť. Pre využívanie funkcií aplikácie je nutné, aby sa užívateľ prihlásil pomocou dvoj-faktorovej autentizácie, prvým faktorom je znalosť jeho mena a hesla, ktoré je overované pomocou databázy, druhým faktorom je znalosť kategórií pre zostavenie jedno-rázového hesla. Prístup je na základe porovnania s databázou povolený alebo zamietnutý. Databáza ukladá záznamy o prihlasovacom mene a hesle, ktoré je hašované pomocou funkcie PBKDF2.

Spojenie medzi klientskou a serverovou časťou je zabezpečené pomocou najnovšej verzie TLS protokolu 1.3. Serverová časť obsahuje aj ďalšie bezpečnostné opatrenia, ako napr. proti XSS útokom, MIME sniffingu, zákaz obnovenia relácií pomocou vstupeniek.

Webovú aplikáciu by bolo možné vylepšiť o implementáciu Schnorrovho protokolu pre autentizáciu, zhotovenie externého úložiska pre záznamy udalostí, aby nemohlo dôjsť k zmazaniu záznamov útočníkom, implementovanie IPS (Intrusion Prevention System) systému, ktorý dokáže detekovať nepriaznivé udalosti a reagovať – napr. jeden z najpoužívanejších systémov Suricata. Na základe výsledkov z testovania nástrojom Lighthouse by bolo vhodné zlepšiť prístupnosť aplikácie zväčšením kontrastného pomeru medzi pozadím a farbou písma na tlačidlách, súčasný vzhľad by mohol byť problematický pre ľudí s poruchami zraku, avšak text s nízkym kontrastom môže ovplyvniť aj bežných užívateľov napr. pri vysokom osvetlení vonkajšieho prostredia.

# Literatúra


[1] By the Numbers: Web Application Security Vulnerabilities. *Solutions Review* [online]. Woburn: Solutions Review, ©2012 – 2019 [cit. 2019-11-08]. Dostupné z: <https://solutionsreview.com/security-information-event-management/by-the-numbers-web-application-security-vulnerabilities/>

[2] What Exactly Is a Web Application? *Lifewire* [online]. New York, 2019 [cit. 2019-11-07]. Dostupné z: <https://www.lifewire.com/what-is-a-web-application-3486637>

[3] Frontend vs Backend: v čom je rozdiel? *Learn2Code* [online]. Bratislava: Learn2Code, 2017 [cit. 2019-11-07]. Dostupné z: <https://www.learn2code.sk/blog/front-end-vs-back-end>

[4] Python 3.7.5 documentation. *Python* [online]. Python Software Foundation, ©2001–2019 [cit. 2019-12-02]. Dostupné z: <https://docs.python.org/3.7/>

[5] Flask vs. Django: Why Flask Might Be Better. *Codementor* [online]. Codementor, ©2019 [cit. 2019-11-10]. Dostupné z: <https://www.codementor.io/garethdwyer/flask-vs-django-why-flask-might-be-better-4xs7mdf8v>

[6] Documentation. *Bootstrap* [online]. ©2019 [cit. 2019-12-02]. Dostupné z: <https://getbootstrap.com/docs/4.0/getting-started/introduction/>

[7] FIELDING, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures.* [online]. Irvine, 2000 [cit. 2019-11-08]. Dostupné z: <https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.Dizertačnápráca.UniversityofCalifornia.>

[8] Single-page application vs. multiple-page application. *Medium* [online]. A Medium Corporation, ©2020, Dec 2, 2016 [cit. 2020-05-10]. Dostupné z: <https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58>

[9] CHURYLOV, Maksym. SPA vs MPA: The definitive guide for decision makers. *Mindk* [online]. San Francisco [cit. 2020-05-10]. Dostupné z: <https://www.mindk.com/blog/single-page-applications-the-definitive-guide/>

[10] Debian vs Ubuntu: Compared as a Desktop and as a Server. *ThisHosting.Rocks* [online]. WordPress, ©2019 [cit. 2019-11-08]. Dostupné z: <https://thishosting.rocks/debian-vs-ubuntu/>

[11] Najpoužívanejšie webové servery – Apache vs. Nginx. *WebSupport* [online]. WordPress [cit. 2019-11-08]. Dostupné z: <https://www.websupport.sk/blog/2019/01/najpouzivanejsie-webove-servery-porovnanie-apache-vs-nginx/>

[12] April 2019 Web Server Survey. *Netcraft* [online]. Netcraft, ©1995-2019 [cit. 2019-11-08]. Dostupné z: <https://news.netcraft.com/archives/2019/04/22/april-2019-web-server-survey.html>

[13] DOSTÁLEK, Libor, Marta VOHNOUTOVÁ a Miroslav KNOTEK. *Velký průvodce infrastrukturou PKI a technologií elektronického podpisu*. 2., aktualiz. vyd. Brno: Computer Press, 2009. ISBN 978-80-251-2619-6.

[14] MALINA, Lukáš. *Kryptografie v informatice* [online]. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2014 [cit. 2019-11-08]. ISBN 978-80-214-5024-0. Dostupné z: <https://moodle-archiv.ro.vutbr.cz/pluginfile.php/422652/mod_page/content/7/skripta_OPVK_2012.pdf>

[15] BERNAT, Vincent. TLS & Perfect Forward Secrecy. *Vincent bernat* [online]. 2019 [cit. 2019-12-02]. Dostupné z: <https://vincent.bernat.ch/en/blog/2011-ssl-perfect-forward-secrecy>

[16] The Transport Layer Security (TLS) Protocol Version 1.3. *TLSWG* [online]. [cit. 2019-11-08]. Dostupné z: <https://tlswg.org/tls13-spec/draft-ietf-tls-tls13.html#rfc.section.1.3>

[17] Usage of SSL certificate authorities for websites. *W3Techs* [online]. Q-Success, ©2009 – 2019 [cit. 2019-11-08]. Dostupné z: <https://w3techs.com/technologies/overview/ssl_certificate/all>

[18] Authentication. *TechTarget* [online]. Grove Street Newton: TechTarget, 2020 [cit. 2020-05-10]. Dostupné z: <https://searchsecurity.techtarget.com/definition/authentication>

[19] RANI, Ch.Jhansi a SK.Shammi MUNNISA. A Survey on Web Authentication Methods for Web Applications. *International Journal of Computer Science and Information Technolog* [online]. 2016, 2016, (7), 1678 - 1680 [cit. 2020-05-10]. ISSN 0975-9646. Dostupné z: <http://ijcsit.com/docs/Volume%207/vol7issue4/ijcsit2016070406.pdf>

[20] Basic, Bearer, Digest Oh MY! So Many Auths! *Dev* [online]. DEV Community copyright, 2020, Sep 05, 2019 [cit. 2020-05-10]. Dostupné z: <https://dev.to/caffiendkitten/authentication-types-3984>

[21] PARMAR, Himika. Generation of Secure One-Time Password Based on Image Authentication. *Computer Science & Information Technology (CS & IT)* [online]. © CS & IT-CSCP, 2012, 2012-10-31, (07), 195 – 206 [cit. 2020-05-10]. DOI: 10.5121/csit.2012.2417. ISBN 9781921987052. Dostupné z: <http://www.airccj.org/CSCP/vol2/csit2417.pdf>

[22] MEYER, David. Time Is Running Out For This Popular Online Security Technique. *Fortune* [online]. Fortune Media IP Limited, ©2019 [cit. 2020-05-10]. Dostupné z: <https://fortune.com/2016/07/26/nist-sms-two-factor/>

[23] The OAuth 2.0 Authorization Framework: Bearer Token Usage. *IETF* [online]. Internet Engineering Task Force (IETF), 2012 [cit. 2020-05-10]. Dostupné z: <https://tools.ietf.org/html/rfc6750>

[24] KUKIC, Adnan. Cookies vs. Tokens: The Definitive Guide. *DZone* [online]. Jun. 02, 16 [cit. 2020-05-10]. Dostupné z: <https://dzone.com/articles/cookies-vs-tokens-the-definitive-guide>

[25] OWASP Top 10 - 2017: The Ten Most Critical Web Application Security Risks. In: *OWASP* [online]. Agora Drive, Bel Air: The OWASP Foundation, ©2003 – 2017 [cit. 2020-05-10]. Dostupné z: <https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_(en).pdf.pdf>

[26] Module ngx_http_ssl_module. *Nginx* [online]. [cit. 2019-12-02]. Dostupné z: <http://nginx.org/en/docs/http/ngx_http_ssl_module.html#ssl_stapling>

[27] How To Configure OCSP Stapling on Apache and Nginx. *DigitalOcean* [online]. DigitalOcean, ©2019 [cit. 2019-12-02]. Dostupné z: <https://www.digitalocean.com/community/tutorials/how-to-configure-ocsp-stapling-on-apache-and-nginx>

[28] MIME Sniffing in Browsers and the Security Implications. *Denim Group* [online]. Denim Group, ©2019 [cit. 2019-12-02]. Dostupné z: <https://www.denimgroup.com/resources/blog/2019/05/mime-sniffing-in-browsers-and-the-security-implications/>

[29] How to configure gzip compression with NGINX. *TechRepublic* [online]. CBS Interactive, ©2019 [cit. 2019-12-02]. Dostupné z: <https://www.techrepublic.com/article/how-to-configure-gzip-compression-with-nginx/>

[30] Get Started with Nessus. *Tenable* [online]. Tenable, ©2020 [cit. 2020-05-11]. Dostupné z: <https://docs.tenable.com/nessus/8_10/Content/GetStarted.htm>

[31] User's Manual. *APACHE JMeter* [online]. Wakefield, U.S.A.: Apache Software Foundation, ©1999-2019 [cit. 2020-05-11]. Dostupné z: <https://jmeter.apache.org/usermanual/index.html>

[32] Lighthouse. *Google Developers* [online]. Mountain View, California, United States: Google, 2020-03-10 [cit. 2020-05-11]. Dostupné z: <https://developers.google.com/web/tools/lighthouse>

[33] About PageSpeed Insights. *Google Developers* [online]. Mountain View, California, United States, 2020-05-05 [cit. 2020-05-10]. Dostupné z: <https://developers.google.com/speed/docs/insights/v5/about>

[34] WALTON, Philip. First Contentful Paint (FCP). *Google Developers* [online]. Mountain View, California, United States, May 4, 2020 [cit. 2020-05-10]. Dostupné z: <https://web.dev/fcp/>

# Zoznam symbolov, veličín a skratiek

| | |
|---|---|
| **AES** | Advanced Encryption Standard |
| **AJAX** | Asynchronous JavaScript and XML |
| **API** | rozhranie pre programovanie aplikácií |
| **BSD** | Berkeley Software Distribution |
| **CA** | Certification Authority |
| **CPU** | Central Processing Unit |
| **CRUD** | Create Retrieve Update Delete |
| **CSS** | Cascading Style Sheets |
| **DDoS** | Distributed Denial of Service |
| **DES** | Data Encryption Standard |
| **DH** | Diffie—Hellman |
| **DHE** | Diffie–Hellman Ephemeral |
| **DNS** | Domain Name System |
| **DSA** | Digital Signature Algorithm |
| **DV** | Domain Validation |
| **HTML** | Eliptic Curve Diffie–Hellman Ephemeral |
| **ECDSA** | Eliptic Curve Digital Signature Algorithm |
| **EV** | Extended Validation |
| **FCP** | First Contentful Pain |
| **GPL** | General Public License |
| **HSTS** | HTTP Strict Transport Security |
| **HTML** | Hypertext Markup Language |
| **HTTP** | Hypertext Transfer Protocol |
| **HTTPS** | Hypertext Transfer Protocol Secure |
| **IDEA** | International Data Encrypt Algorithm |
| **JSON** | JavaScript Object Notation |
| **JWT** | JSON Web Token |
| **MD5** | Message-Digest algorithm |
| **MIME** | Multipurpose Internet Mail Extensions |
| **MPA** | Multi-Page Application |
| **OCSP** | Online Certificate Status Protocol |
| **OV** | Organization Validation |
| **PBKDF** | Password-Based Key Derivation Function |
| **PKI** | Public Key Infrastucture |
| **RAM** | Random Access Memory |
| **REST** | Representational State Transfer |
| **RSA** | Rivest, Shamir, Adleman |

**RSASSA-PSS** RSA Signature Scheme with Appendix — the Probabilistic Signature Scheme

**SEO** Search Engine Optimization

**SHA** Secure Hash Algorithm

**SPA** Single-Page Application

**SSL** Secure Sockets Layer

**TLS** Transport Layer Security

**URL** Uniform Resource Locator

**WSGI** Web Server Gateway Interface

**XML** Extensible Markup Language

**XSS** Cross-Site Scripting

# A Obsah priloženého média

Na priloženom médiu sa nachádza vyexportovaný virtuálny stroj serveru, zdrojové kódy webovej aplikácie a konfiguračné súbory pre nastavenie serveru Nginx, WSGI serveru, TLS protokolu a iných bezpečnostných prvkov, READ_ME súbor s návodom na obsluhu a text samotnej bakalárskej práce.

```
/ ..... koreňový adresár priloženého DVD
├── konfiguracia nginx ..... súbory pre nastavenie serveru Nginx
│   ├── nginx.conf
│   └── webTestApp
├── konfiguracia TLS a inych bezpecnostnych prvkov .... nastavenie bezpečnosti
│   ├── ssl-signed.conf
│   └── ssl-params.conf
├── konfigurácia wsgi ..... nastavenie komunikácie prostredníctvom WSGI serveru
│   ├── webTestApp.conf
│   ├── webTestApp.ini
│   └── wsgi.py
├── text BP ..... zložka s textom bakalárskej práce
│   └── Bakalarska_praca.pdf
├── video ukazka ..... obsahuje video ukážku funkcionality aplikácie
│   └── video_demo.mp4
├── virtualny stroj ..... zložka s virtuálnym obrazom serveru
│   └── Debian.vdi
├── webova aplikacia ..... zložka so zdrojovými kódmi aplikácie
│   └── webTestApp
│       ├── database ..... zložka so skriptami pre vytvorenie databázy
│       ├── static ..... zložka pre statické súbory – obrázky, CSS
│       ├── templates ..... zložka pre HTML šablóny
│       ├── uploadFiles ..... zložka pre nahraté súbory užívateľov
│       ├── venv ..... virtuálne prostredie – knižnice
│       ├── log.txt
│       ├── main.py
│       ├── otp_func.py
│       └── users.db
└── READ_ME.txt ..... inštrukcie pre spustenie
```