VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

BEZPEČNÉ A BEZEŠVÉ SDÍLENÍ DAT

SAFE AND SEAMLESS DATA SHARING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MICHAL VIŠŇOVSKÝ**
AUTHOR

VEDOUCÍ PRÁCE — **Doc. Dr. Ing. DUŠAN KOLÁŘ**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Višňovský Michal**
Program: Informační technologie
Název: **Bezpečné a bezešvé sdílení dat**
Safe and Seamless Data Sharing
Kategorie: Bezpečnost

Zadání:

1. Seznamte se s administrací dat a prostředky, které je možné využít pro sdílení dat. Zaměřte se na bezpečnost sdílení dat a ochranu proti možným útokům.
2. Na základě technologických možností a po konzultaci s vedoucím navrhněte sdílení dat založené na "Norman Sample Sharing framework" společnosti Avira. Zaměřte se na bezešvou implementaci vzhledem k existujícím nástrojům a použití daného protokolu, aby klienti nepoznali změnu.
3. Návrh implementujte, včetně serverové strany, API, komunikačního protokolu a webového rozhraní.
4. Přes webové rozhraní umožněte plnou správu uživatelů, sledování balíků dat, jejich správu včetně přístupových dat a statistik.
5. Řešení otestujte na kompletní funkčnost a odolnost vůči bezpečnostním útokům.
6. Zhodnoťte vaši práci, diskutujte možná rozšíření a další rozvoj systému.

Literatura:

- Avira, Virex: https://github.com/Avira/virex [online, navštíveno 5. 10. 2021].
- Dle doporučení vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- První dva body a rozpracovaný bod 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Kolář Dušan, doc. Dr. Ing.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 18. října 2021

Abstrakt

Antivírové spoločnosti tvoria medzi sebou komunitnú sieť zdieľaných vzoriek. Zdroje informácií nie sú unifikované a existuje viacero typov princípu zdieľania. Jedným z nich je systém Sampleshare podľa Norman Sample Sharing Frameworku. Jeho prevedenie dnes používa už zastaralé technológie a obsahuje bezpečnostné diery. Cieľom práce je vytvoriť reinterpretáciu tohto systému, bez toho aby si odberateľ všimol zmenu a musel uskutočniť rozsiahlu rekonfiguráciu odberateľského skriptu. Zámerom bolo využiť aj najnovšie technológie na vylepšenie bezpečnosti aplikácie a prenosového protokolu. Prítomná webová aplikácia umožňuje administráciu užívateľov, zdieľaných balíkov a sledovanie zdrojov prevádzkového stroja.

Abstract

Antivirus companies together create a community network of sample sharing. Data sources are not unified and there exist many types of sharing principles. One of them is the system of Sampleshare, working on basis of the Norman Sample Sharing Framework. The current version is using deprecated technologies and is open to network threats. The main goal of the thesis is to create a reinterpretation of this system, without the clients noticing any changes and having to reconfigure their feeder scripts in a larger scale. The focus is also set to use the newest technologies in means of improving the overall safety of the application and its transfer protocol. The included web application provides user and sample package administration as well as monitoring of the host machine resources.

Kľúčové slová

autentizácia, autorizácia, API, ReactJS, Node.js, express.js, bezpečnosť, GPG, REST, MariaDB, Selenium, Postman, NSSF, Norman

Keywords

authentization, authorization, API, ReactJS, Node.js, express.js, safety, GPG, REST, MariaDB, Selenium, Postman, NSSF, Norman

Citácia

VIŠŇOVSKÝ, Michal. *Bezpečné a bezešvé sdílení dat.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Dr. Ing. Dušan Kolář

Bezpečné a bezešvé sdílení dat

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Koláře. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Michal Višňovský
8. mája 2022

Poďakovanie

Chcel by som poďakovať vedúcemu práce Doc. Dr. Ing. Dušanovi Kolářovi za odborné vedenie práce a poskytnutí mnoho cenných rád pri implementovaní. Taktiež by som rád poďakoval kolegom z práce za konzultovanie problémov a mojej rodine a priateľom za podporu počas písania tejto práce.

Obsah

Kapitola 1

Úvod

Prenos dát je v dobe informatizácie každodenným procesom, nad ktorým sa človek už ani nepozastavuje. Či sa jedná o sťahovanie aplikácie do mobilného zariadenia alebo zdieľaním fotografie na sociálnej sieti, dôraz na bezpečný prenos dát sa v očiach užívateľa dostáva do úzadia. Užívateľ naslepo vkladá dôveru poskytovateľovi služby alebo samotného internetového pripojenia. Riziko útokov tretej strany a zneužívanie citlivých údajov stále rastie a treba voči útočníkom čoraz nastavovať silnejšie ochranné prostriedky.

Antivírové spoločnosti si v rámci komunitnej spolupráce posielajú medzi sebou buď vzorky čisté (neškodné pre počítače) alebo infikované (malware, ransomware). Je v záujme všetkých zainteresovaných strán, aby prenos bol chránený. Cieľom práce je vytvoriť systém zdieľania dát, ktorý na základe bezpečnostných technológií umožní prenos citlivých informácií bez rizika vonkajšieho napadnutia. Výsledkom je informačný systém pre správu týchto zdieľaných súborov a samotný komunikačný server, ktorý tieto vzorky poskytuje odberateľom.

Technická správa je zložená so siedmich kapitol. Prvá obsahová kapitola 2 zoznamuje čitateľa so známymi rizikami a nebezpečím číhajúcim na internete a samozrejme ako sa voči nemu brániť. Text sa zaoberá aj problematikou, ktorá opodstatnila túto bakalársku prácu.

V kapitole 3 sa čitateľ oboznámi s chránenými prenosovými protokolmi a nástrojmi, ktoré umožňujú prenášané dáta zabezpečovať pomocou šifrovania ako aj praktiky, ako odhaliť bezpečnostné diery v implementácií.

V ďalšej kapitole 4 je opísaná forma a podoba systému, ktorý bude vyhovovať špecifikácii. Nachádza sa tu návrh riešenia problému a predstavenie vývojového prostredia a použitých technológií, v ktorom je aplikácia implementovaná. Aplikácia je tvorená pomocou vývojových prostredí ReactJS a NodeJS, ktoré poskytujú modulárnu a flexibilnú kompozíciu zdrojového kódu a jednoduchú manipuláciu a pridávanie nových funkcionalít v budúcnosti.

Postupný vývoj a problémy počas implementácie sú podrobne popísané v kapitole 5. Nachádza sa tu opis implementovania jednotlivých častí a zvolených vývojových techník použitých v práci. Je tu popísaná implementácia NSSF serveru a úprava jeho nedostatkov. Taktiež tu rozoberám a vysvetľujem ako webová aplikácia funguje a ako je štrukturálne poskladaná, metódy obsluhujúce požiadavky o vzorky od odberateľov a API backendového serveru.

Kapitola 6 sa venuje testovaniu výslednej aplikácie a jej koncových bodov. Čitateľ sa dozvie o priebehu riadneho testovania ako aj o plánovaných bezpečnostných penetračných testoch. Je tu opísaný projekt Selenium aj jeho využitie v bakalárskej práci. Ako verifikáciu

integrity je nad webovou aplikáciou vytvorený testovací klikací skript práve s pomocou modulov Selenia (počítačový skript, ktorý overuje správne chovanie webovej aplikácie).

V poslednej kapitole 7 som predstavil ďalší vývoj a plánované zmeny, ktorými systém obohatím a vylepším pre všetkých užívateľov.

Kapitola 2

Bezpečnosť webovej aplikácie

2.1 Úvod do problematiky

Táto bakalárska práca sa zaoberá systémom zdieľania vzoriek externým užívateľom s názvom Sampleshare pre spoločnosť ESET[1], o ich administráciu a jednotlivé spravovanie posielaných balíkov dát či samotných vzoriek. Hlavný zámer má byť však bezpečnosť komunikácie so serverom a odolnosť voči útokom na informačný systém. Návrh preto treba konštruovať tak, aby vývoj už od počiatku dbal na implementáciu obranných mechanizmov.

2.1.1 Súčasná aplikácia a jej zabezpečenie

Jedným z hlavných dôvodov prečo Sampleshare potrebuje vybudovať od základov, sú jeho nedostatky v bezpečnosti. Zdrojom týchto nedostatkov bol samotný NSSF (Norman Sample Sharing Framework) [8], preberaný od inej antivírovej spoločnosti a implementácia aplikácie využívajúca už dnes zastaralé a nepodporované frameworky pre PhP. Značné chyby sa nachádzajú aj v databázi aplikácie, ktorá je celá upravená tak, aby nevytvárala komplikácie pri mazaní záznamov. Viaceré bezpečnostné nedostatky boli nachádzané až počas produkcie a ich oprava bola vždy nepríjemnou prácou. Niekedy išlo až o kritické bezpečnostné závady, ktoré mohli viesť k poškodeniu systému. Preto je treba zostaviť aplikáciu tak, aby bola modulárna, flexibilná a prípadné závady bolo jednoduché lokalizovať a opraviť.

2.2 Vonkajšie hrozby a ich prevencia

Forma hrozieb pre informačné systémy je veľmi rozličná a napadnuté môžu byť viaceré časti aplikácie. Podľa zoznamu najpoužívanejších metód na prelomenie webových aplikácií z roku 2021 od združenia OWASP (The Open Web Application Security Project)[2] bližšie popíšem niektoré z nich a ako sa vyhnúť bezpečnostným prelomeniam. List sa nachádza v prílohe B.

2.2.1 Broken Access Control

Predstavuje široký pojem značnej nesprávnosti implementácie, kedy neautorizovaný užívateľ vie použiť funkcionality systému mimo jeho práva.

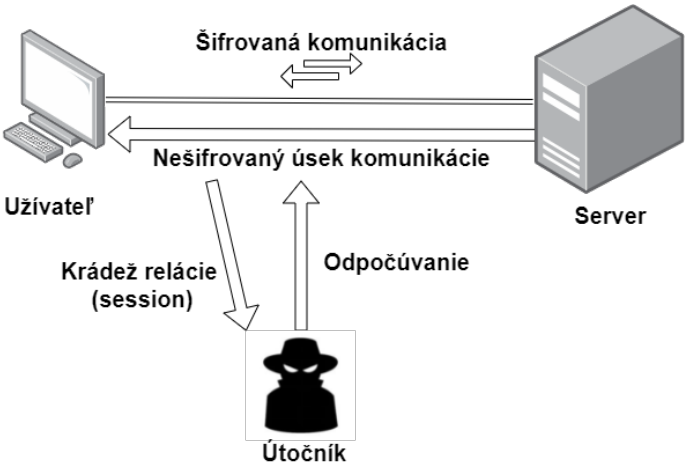
Konkrétne sa jedná o

[1] `https://www.eset.com`
[2] `https://owasp.org/Top10/`

- možnosť obísť bezpečnostné opatrenia stránky zmenou URL a jej parametrov (force browsing), vnútorného stavu aplikácie alebo stránky či zmeny API požiadavky
- povolené zobrazenie a editácia účtu iného užívateľa pomocou ich ID
- neoprávnený prístup k akciám a stránkam na webe, ku ktorým sú povolení len niektorí užívatelia alebo role
- manipuláciu metadát (JWT, Cookies) webovej aplikácie a následné povýšenie role užívateľa
- možnosť komunikácie s API z neautorizovaného alebo neverifikovaného zariadenia (následok zlej konfigurácie CORS – Cross-Origin Resource Sharing)

Prevencia a možné opatrenia [3]

- Odoprieť prístup k zdrojom, ktoré nie sú verejné (sú dostupné aj pre neautentizovaného užívateľa)
- Dôkladná implementácia overovacieho systému, modulárne aplikovanie v rámci služby
- Limitovať počet dotazov na API alebo nesprávnych pokusov o prihlásenie
- Zabezpečenie metadát, aby ich žiaden užívateľ nemohol manipulovať a meniť



Obr. 2.1: Jednoduchá ukážka krádeže relácie

2.2.2 Závady v šifrovaní

Informačné systémy majú variabilný rozsah, svojou narastajúcou veľkosťou a počtom komplikovaných transakcií narastá aj pravdepodobnosť, že aspoň jedna časť je zabezpečená slabo. Väčšinou to bývajú nešifrované protokoly FTP, HTTP, SMTP, ktoré nepoužívajú TLS (Transport Layer Security), teda ich sledovanie vie poskytnúť viac než základné informácie o oboch stranách komunikácie. Útočník tak jednoducho získa reláciu (session),

[3] `https://owasp.org/Top10/A01_2021-Broken_Access_Control/`

prihlasovacie údaje a a súkromné informácie obr.2.1. Takéto útoky sa veľmi ťažko sledujú, pretože sa z pohľadu stránky nič podozrivého nedeje.

Nedostatky sa môžu objaviť aj pri použití starých knižníc či použití zastaralého šifrovacieho algoritmu. Zabezpečenie a predchádzanie týmto rizikám sa dá uskutočniť výhradne použitím správnych a dôkladne implementovaných šifrovacích algoritmov. Správnym prístupom je použiť najnovšie technológie na ukladanie citlivých informácií a pokiaľ to ide, citlivé informácie neukladať. Je doporučené implementovať autentizované šifrovanie (AE), teda asymetrické šifrovanie prenosu, ktoré navyše využíva kľúčové slovo alebo reťazec, pomocou ktorého je šifrovanie obsahu jedinečné a dá sa pomocou nej overiť autenticita odosielateľa a integrita dát (v Sampleshare použité PGP) [2].

2.2.3 SQL injection

Informačný systém sa dnes nezaobíde bez úložiska dát, hrubá väčšina využíva databázové systémy využívajúce SQL jazyk (viď [7]). S databázou sa komunikuje pomocou SQL dotazov, ktoré sa môžu na základe typu úložiska syntakticky líšiť. Ak dotazy vykonávané nad databázou nie sú implementované s dbaním na možný injection útok, celý informačný systém alebo aplikácia je vystavená vážnemu riziku a môže dôjsť až k stratení dát o užívateľoch a odhaleniu citlivých informácií. Útočník nedostatok systému bez obtiažnosti vie zistiť a zneužiť slepým testovaním vstupov aplikácie. Klasický SQL dotaz, ku ktorému má neprihlásený užívateľ (útočník) prístup, je súčasťou registrácie alebo prihlasovania. Existuje mnoho variácii injection útokov, SQL je najčastejší.

```
SELECT userName, password
FROM Users
WHERE userName = 'user' AND password = 'pwd'; DROP TABLE Users;
```

Výpis 2.1: Ukážka dotazu vytvoreného pri SQL injection

Ak systém nevaliduje vstupy užívateľa, krátka ale efektívna časť SQL kódu vie znemožniť celú aplikáciu obr.2.1. Útočník vie do prihlasovacieho formulára vložiť časť škodlivého SQL kódu do poľa pre prihlasovacie heslo. Samotná validácia vstupov nestačí. Na ochranu pred injection útokom slúži sada pravidiel pre detekciu škodlivých SQL príkazov (regulárne výrazy) alebo zostavenie šablóny SQL príkazov, do ktorej sa len vkladajú vstupy od užívateľa. Medzi zaužívané praktiky tiež patrí užitie napr. SQL konštrukcie LIMIT na konci dotazu alebo string escaping špecifický pre systém. Tieto opatrenia by zneškodnili nebezpečný vstup potencionálneho záškodníka.

2.2.4 DDoS útoky

Populárnym útokom na server je Distributed Denial of Service (DDoS). Všeobecne ide o snahu zamietnutia prístupu k aplikácii preťažením. Útočník si najprv pripraví dostatočné množstvo strojov, ktoré útok budú uskutočňovať, tzv. botnety. V množstve prípadov stroje získava pomocou svojho malware. Škodlivé procesy na základe príkazu začnú na server posielať správy za zámienkou zaneprázdniť aplikáciu do takej miery, že nebude na reálne požiadavky vedieť odpovedať.

Najrozšírenejší typ útokov je TCP flood, kde stroje na server posielajú neúplné TCP správy. Samotný TCP protokol vyžaduje od oboch strán komunikácie určité chovanie, takzvaný TCP handshake. Klient začína transakciu SYN správou (synchronize), server na požiadavku odpovedá dvoma správami SYN a ACK (acknowledgement) a očakáva od klienta

ACK správu. Pokiaľ však klient ACK neodošle, server stále na slepo očakáva od klienta správu. Pri enormnom množstve strojov súčasne posielajúcich nekompletné TCP správy vie server prísť o všetky svoje zdroje a prakticky prestať fungovať [6].

Existuje viacero možností ako detekovať a reagovať na DDoS útoky. Zaujímavou stratégiou je zostavenie systému tak, aby mal čo najmenej vonkajších prístupov, čím je dosiahnutá menšia „plocha“, kde môže útok nastať. Tieto miesta sa dajú potom zabezpečiť firewallom, či inými ochrannými prostriedkami [4].

Implementáciou WAF alebo Web Application Firewall sa dá predísť viacerým formám útokov. Slúži ako ďalšia ochranná vrstva voči vonkajším útokom. Najpoužívanejším druhom nasadenia WAF je priamo v komunikácií medzi klientom a serverom, kedy požiadavky najprv musia prejsť cez implementovaný firewall. WAF sa riadi pomocou definovaných pravidiel, ktoré určujú jeho chovanie a chránia server od škodlivého prenosu dát. Existujú tri druhy WAF podľa jeho lokalizácie [5]:

- Network based – väčšinou hardvérové nasadenia WAF, nevýhodou je vysoká réžia a cena produktu
- Host based – WAF je aplikované priamo v kóde, má nízke nároky na cenu ale často vyžaduje viaceré závislosti na knižniciach a na zdrojoch hosťovského serveru
- Cloud based – WAF je sprostredkovávaný spoločnosťou tretej strany, výhodou je jednoduchá implementácia a nízke požiadavky na údržbu

Meranie počtu transakcií a ich objemu vie predčasne predísť úplnému výpadku aplikácie, pomocou algoritmov a porovnaním s obvyklým prenosom dát, na čo je potrebné ale vedieť rozoznať obvyklú návštevnosť stránky od zvýšenej (čo nie je veľmi jednoduché). Prostým riešením býva limitovanie počtu požiadaviek na server na hodnotu, ktorú zvládne obslúžiť.

2.2.5 Man in the Middle

Forma aktívneho odpočúvania, kedy útočník zachytáva a mení prenášané dáta medzi účastníkmi komunikácie za zámerom vydávania sa za jedno alebo viac zariadení zapojených do komunikácie. Podľa názvu útoku sa útočník umiestni v komunikácií medzi koncové body a získava prístup k informáciám prenášanými kanálom. Útočník tak získava kontrolu nad komunikáciou, vie posielať podvrhnuté dáta a vie získať citlivé informácie posielané v komunikácii. Obete útoku nemajú informáciu o prítomnosti votrelca, čo vytvára mienku, že komunikácia je zabezpečená. Zabezpečením proti takýmto útokom je naviazanie bezpečnej komunikácie medzi serverom a klientom, čiže komunikácia cez zabezpečený kanál a použitie autentifikácie pomocou kľúčov (v prípade Sampleshare využitie PGP kľúčov).

2.2.6 Cross Site Scripting (CSS/XSS)

Jedná sa o útoky mierené na klientov aplikácie. Útočník sa zmocní obsahu HTML, ktorý je posielaný užívateľom aplikácie. Vkladá do zdrojového kódu aplikácie svoje škodlivé úryvky kódu v Javascripte. Užívateľ nevedome pri zobrazení obsahu stránky spúšťa aj nebezpečný kód, ktorý môže šíriť malware, získavať citlivé informácie alebo kontrolu nad samotnou stránkou [4].

[4] `https://aws.amazon.com/shield/ddos-attack-protection/`

[5] `https://www.techtarget.com/searchsecurity/definition/Web-application-firewall-WAF`

Ako obrana voči týmto útokom je podľa spoločnosti OWASP riadenie sa pomocou nasledujúcich pravidiel:

- Neukladať neoverené dáta do databázy
- Zakódovať a striktne overovať dáta pred ich vkladaním do HTML, CSS, JS
- Použitie najnovších a aktuálnych vývojových prostredí, ktoré vrámci svojej implementácie poskytujú ochranu voči XSS útokom (React, Angular, Vue)

Kapitola 3

Bezpečný prenos dát

Hlavnou službou systému je prenos a zdieľanie vzoriek. Podľa aktuálnej štatistiky, terajší systém obsluhuje najviac požiadavky tohto typu. Preto treba tento endpoint dôkladne zabezpečiť a prenos a samotné vzorky zašifrovať. Predstavím najpoužívanejšie prenosové protokoly pri posielaní súborov. Taktiež sú v tejto kapitole popísané známe šifrovacie algoritmy a nástroje na odhalenie bezpečnostných dier.

3.1 FTP

Protokol FTP (File Transfer Protocol) slúži na posielanie súborov medzi dvoma zariadeniami (server a klient) [1]. Je treba vytvoriť poskytovateľa súborov – teda FTP server, na ktorý sa klienti budú vedieť dotazovať. Samotný protokol FTP obsahuje len základné zabezpečenie a jeho obsah nie je šifrovaný.

3.1.1 SSL/TLS

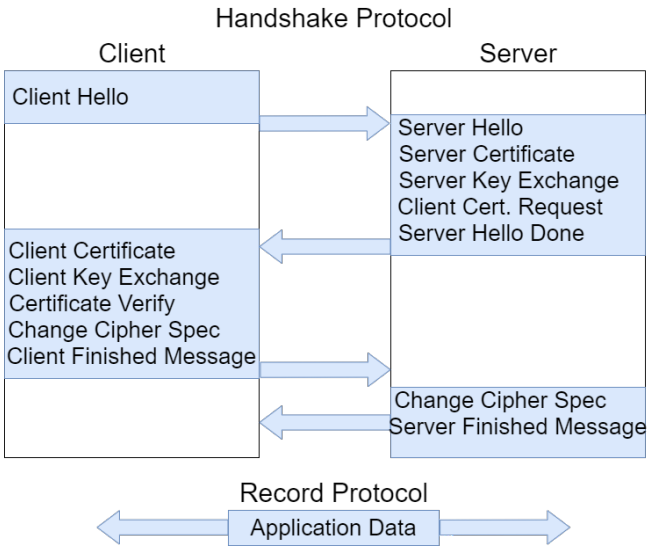
Transport Layer Security (TLS) je protokol slúžiaci primárne na udržiavanie diskrétnosti komunikácie medzi zariadeniami [5]. Je postavený na základoch svojho zastaralého predchodcu SSL. TLS prebieha v dvoch stavoch – TLS handshake obr.3.1 a Record protocol. Prvá správa je odoslaná klientom na server.

Server odpovedá a posiela výzvu k preukázaniu SSL certifikátu. Klient musí dodať certifikát, ktorý je dôveryhodný zo strany serveru. Taktiež prebehne výmena kľúčov, ktoré slúžia na šifrovanie obsahu ďalších správ medzi klientom a serverom. Ak tento postup prebehne bezchybne, komunikácia sa môže posunúť do stavu Record protocol, kde dáta samotnej správy sú fragmentované alebo znovu zložené naspäť podľa toho, na ktorej strane komunikácie sme[1].

3.1.2 SFTP

Jedna z možností, ako bezpečne preniesť súbor cez internet je SFTP (Secure File Transfer Protocol). Využíva SSH (Secure Shell Protocol) pre nerušený prenos prihlasovacích údajov. Klient po sérií handshake-ov pomocou SSH vytvorí zabezpečený tunel na server, v ktorom bude prebiehať celá transakcia. Protokol ponúka nielen prenos súborov ale aj široké

[1] `https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc785811(v=ws.10)`



Obr. 3.1: TLS Handshake Protocol

spektrum operácií nad súbormi a súborovým systémom. Patrí sem napríklad vzdialené odstránenie súborov, zobrazenie zoznamu adresárovej štruktúry alebo obnovenie prerušených prenosov súborov.

3.1.3 FTPS

Nadstavba protokolu FTP, zabezpečená pomocou TLS. Je zabezpečené aj prihlasovanie aj prenášané dáta. Po TLS handshake je celá komunikácia symetricky šifrovaná a nikto okrem transakčných zariadení nevie dáta čítať. Pre Sampleshare bude použitý tento protokol na správu a odosielanie súborov externým užívateľom, keďže je prioritou zachovať chovanie serveru voči klientovi s čo najmenšími zmenami (súčasná verzia používa protokol FTPS).

Vlastnosti	**SFTP**	**FTPS**
Využíva silné šifrovacie metódy	Áno	Áno
Šifruje prohlasovacie údaje	Áno	Áno
Podporuje autentizáciu pomocou kľúčov	Áno	Nie
Podporuje certifikáty	Nie	Áno

Tabuľka 3.1: Porovnanie vlastností SFTP a FTPS

3.2 HTTPS

HTTPS alebo HTTP over SSL/TLS je zabezpečený komunikačný protokol využívaný najmä webovými prehliadačmi. Zaisťuje autenticitu a dôveryhodnosť prenášaného obsahu a taktiež kontroluje jeho integritu [2]. Aby bol protokol správne zabezpečený a správne fungoval, musí obsahovať certifikát verifikovaný treťou stranou. Prenos je zabezpečený symetrickou šifrou a integrita obsahu je riešená pomocou hashovacieho algoritmu. Aplikácia je navrhnutá tak

[2] `https://cs.wikipedia.org/wiki/HTTPS`

aby podporovala beh pomocou protokolu HTTPS a v produkčnom nasadení bude musieť týmto protokolom komunikovať. V rámci vývoja aplikácie je možné pracovať s nešifrovanou verziou komunikačného protokolu, keďže ich chovanie je identické.

3.3 Šifrovacie algoritmy

3.3.1 Symetrické šifry

3DES

3DES je najrozšírenejší a najpoužívanejší blokový šifrovací algoritmus, ktorý pochádza z Data Encryption Standard (DES) algoritmu. DES predstavoval symetrické šifrovanie vyvinuté v 70. rokoch 20. storočia a krátko po jeho vzniku bolo označené za prelomiteľné z hľadiska nízkej komplexity. V roku 1999 bolo demonštrované získanie šifrovacieho kľúča za menej ako 24 hodín[3]. Dnes je tento algoritmus prelomiteľný v rámci dní verejne dostupným hardvérom. Jeho nastúpiteľom sa stal štandard 3DES. Šifrovanie používa tri rôzne kľúče o 56 bitoch (DES využíval iba jeden) a funguje na princípe viacnásobného šifrovania. Vstupný text je zašifrovaný kľúčom 1, dešifrovaný kľúčom 2 a znovu zašifrovaný kľúčom 3.

$$\text{šifrovaný\_text} = E_{K3}(D_{K2}(E_{K1}(\text{nešifrovaný\_text})))$$

Momentálne je aj táto šifrovacia metóda označovaná ako nedostatočná a je nahrádzaná algoritmom AES.

AES

Je blokovou šifrou, ktorá spracováva vstup v pevných blokoch o 128 bitoch a používa kľúče z dĺžkového rozmedzia 128, 192 alebo 256 bitov. AES rozloží spracovávaný blok na menšie časti po bajtoch a uloží ich do tabuľky podobnej matici (4x4). Nad tabuľkou sú prevedené matematické operácie posuvu a výmeny riadkov a stĺpcov. Šifrovací algoritmus je veľmi silný a jeho prelomenie je pomocou brute force nemožné.

3.3.2 Asymetrické šifry

RSA

Jedná sa o jeden z prvých kryptografických systémov využívajúci verejné kľúče. Kľúč je vytvorený z dvoch veľkých prvočísiel a publikovaný na verejnosť. Šifrovať pomocou tohto kľúča môže hocijaká osoba, ale na čítanie jeho zašifrovaného obsahu je možné len osobou, ktorá pozná originálny pár prvočísiel teda vlastní súkromný kľúč. Výhodou RSA systému je jeho komplexnosť a neprelomiteľnosť. Algoritmus výpočtu je ale relatívne pomalý a pre jeho funkčnosť je predpokladaná distribúcia kľúčov.

Diffie-Hellman

Jedná sa o nekonvenčnú šifrovaciu metódu, ktorá využíva spoločne zdieľaného tajomstva medzi oboma stranami komunikácie. Dve strany sa dohodnú na použití rovnakého číselného základu a modulového čísla, jednotlivé strany si zvolia svoje tajné čísla, vypočítajú mocninu

[3] `https://www.schneier.com/blog/archives/2004/10/the_legacy_of_d.html`

základu a svojho tajného čísla a navzájom si vymenia medzi sebou výsledok modula[3]. Opätovným umocnením zdieľaných čísel tajným číslom získajú skryté číslo. Pri použití prvočísiel vysokého rádu sa stáva šifrovanie týmto jednoduchým algoritmom neprelomiteľným.

3.4 PGP

Nejedná sa o prenosový protokol, ale o šifrovací program, ktorého využitie je nevyhnutné pri implementovaní Sampleshare. PGP sa používa na podpisovanie, šifrovanie a dešifrovanie textov, e-mailov, súborov a celých adresárových stromov [4]. Proces prebieha kombináciou hashovania, dátovej komprimácie, symetrickej kryptografie a nakoniec asymetrickej kryptografie. PGP využíva mnoho z predstavených šifrovacích algoritmov ako RSA a Diffie-Hellman.
Každý užívateľ bude musieť poskytnúť svoj verejný PGP kľúč, aby zdieľanie vzoriek podľa NSSF protokolu mohlo prebiehať. Kľúč je viazaný na užívateľské meno a e-mailovú adresu. Zašifrovaný obsah a dáta verejným kľúčom sú nečitateľné pre osobu, ktorá nevlastní súkromný kľúč generovaný párovo spolu s tým verejným. Výhodami použitia šifrovacieho programu PGP je praktická neprelomiteľnosť šifrovacích algoritmov. Napriek tomu jeho štruktúra a nasadenie je niekedy obtiažna a obsiahla. Vyžaduje vynaloženie námahy aj od klienta aj poskytovateľa služby, ktorá obsahuje šifrovanie PGP. Proces šifrovania a dešifrovania je popísaný obrázkom 3.2.

OpenPGP je distribúcia programu PGP, ktorá vznikla z rastúcich obáv o patentoch v PGP. Problémy s licencovanými algoritmami boli vyriešené a OpenPGP bolo vydané ako nezávislý produkt. Z hľadiska autora OpenPGP to bol dôležitý krok pre rozšírenie využitia v kryptografickej komunite. Verejne dostupné šifrovanie by celkovo zvýšilo bezpečnosť aplikácií.

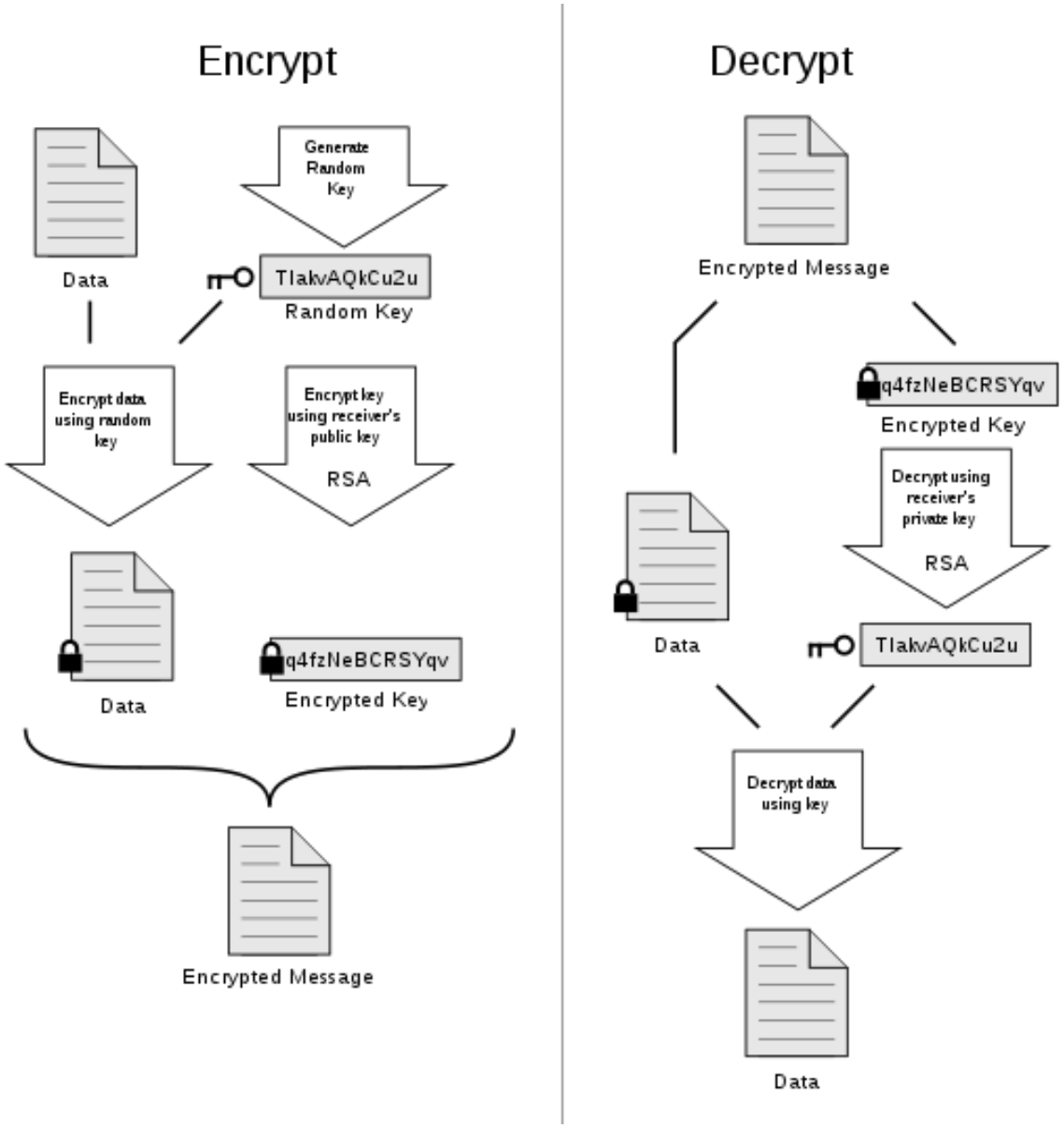
GnuPG alebo označovaná aj ako GPG, je kompletná implementácia OpenPGP štandardu, ktorá je dostupná pre verejnosť. Poskytuje všetky elementárne prípady užitia ako systém PGP a ponúka aj službu pre správu súkromných aj verejných kľúčov užívateľov. Nástroj je vyvinutý pre príkazový riadok a umožňuje jednoduchú integráciu do iných aplikácií.

3.5 Nástroje na odhalenie bezpečnostných dier

Testovacia sada pre klikacie skripty
Klikacie skripty sa dajú použiť na veľa užitočných vecí, či už na automatické sťahovanie súborov z webu alebo na overovanie základných funkcií webovej aplikácie. Pre tieto programy je možné spísať sadu testovacích požiadaviek, ktoré sa automaticky vykonajú a overia správne chovanie webu. Framework Selenium poskytuje možnosť tieto skripty implementovať vo forme knižnice pre programovací jazyk Python. Skript je tým pádom jednoducho spustiteľným programom, ktorý preverí každý testovací prípad v sade. Pre produkčný server, kde bude výsledná aplikácia nasadená, môže byť program naplánovaný (pomocou plánovača úloh) skontrolovať integritu webovej aplikácie napríklad každý deň. Nájdené chyby vieme tak jednoducho odhaliť bez ľudskej námahy.

Penetračné testy

[4] `https://www.philzimmermann.com/EN/essays/WhyIWrotePGP.html`



Obr. 3.2: PGP šifrovanie a dešifrovanie obsahu. Prevzaté z [10]

Jedná sa o test, ktorý simuluje útok cielený na web s využitím podobných alebo rovnakých nástrojov, ktoré by použili útočníci v reálnom svete. Test sa zameriava na najčastejšie chyby a zraniteľnosti podľa OWASP Top 10 či už z pohľadu anonymného alebo prihláseného užívateľa. Pre správnosť testovania sú výsledky penetračných testov overované manuálne a z nich sú vytvorené posudky a presné body zraniteľnosti systému[5]. Využitím penetračných testov aplikácie je možné včasne predísť fatálnym chybám a optimalizovať zdroje na vývoj bezpečnostných opatrení.

[5] `https://www.eset.com/sk/firemna-it-bezpecnost/bezpecnostne-sluzby/services/penetracne-testy/`

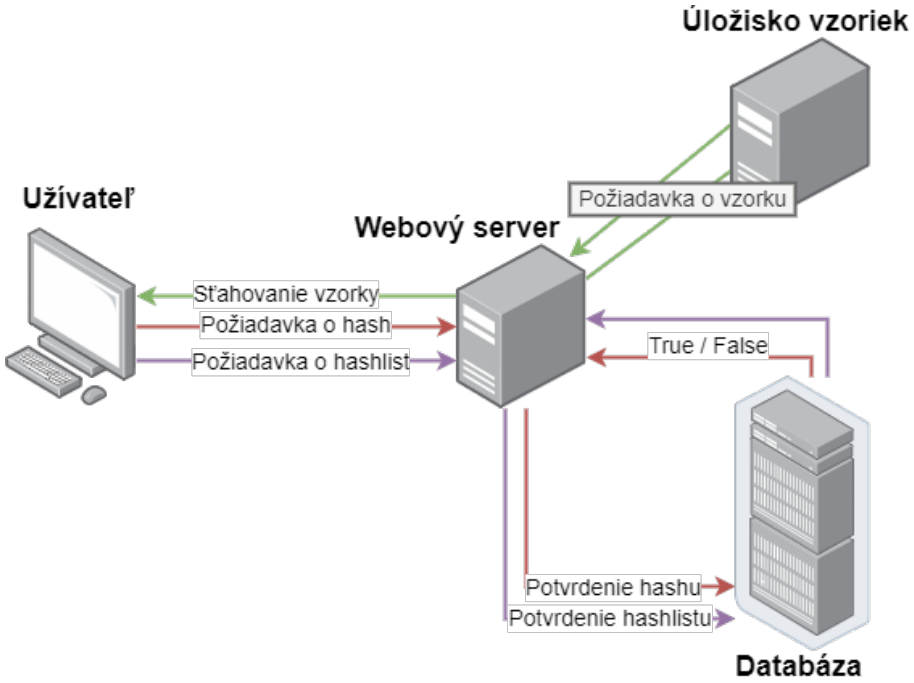
Nové prevedenie ESET Sampleshare bude podrobené penetračnými testami, ktoré majú za účelom nájsť a objaviť nedostatky v implementácií a bezpečnostných opatreniach v aplikácií.

Kapitola 4

Návrh implementácie

4.1 Norman Sample Sharing Framework

Podmienkou implementácie je prakticky neporovnateľné chovanie zdieľania vzoriek voči predošlej verzií Sampleshare [8]. Jedná sa hlavne o spôsob zdieľania vzoriek, ich administráciu a bezpečnosť ich prenosu. Užívateľ komunikuje pomocou HTTPS s backendom aplikácie. Server overuje autenticitu užívateľa a či patrí do skupiny externých užívateľov, ktorí majú prístup k vzorkám. Klient zadáva požiadavku v tvare GET Requestu, kde špecifikuje žiadanú akciu od servera a vo väčšine prípadov nevyhnutný údaj od akého dátumu očakáva výsledné dáta od serveru. Vzorky sú kompresované a šifrovanie je podmienené pgp kľúčom klienta[1]. Aktuálnu verziu momentálne používa vyše 40 jednotlivých klientov na celom svete, ktorí svojím odberateľským skriptom posielajú tisíce požiadaviek každý deň.



Obr. 4.1: NSSF Transakcia medzi užívateľom a v rámci serveru

[1] `https://github.com/Avira/virex`

Server má podporovať viacero príkazov z URL (query string):

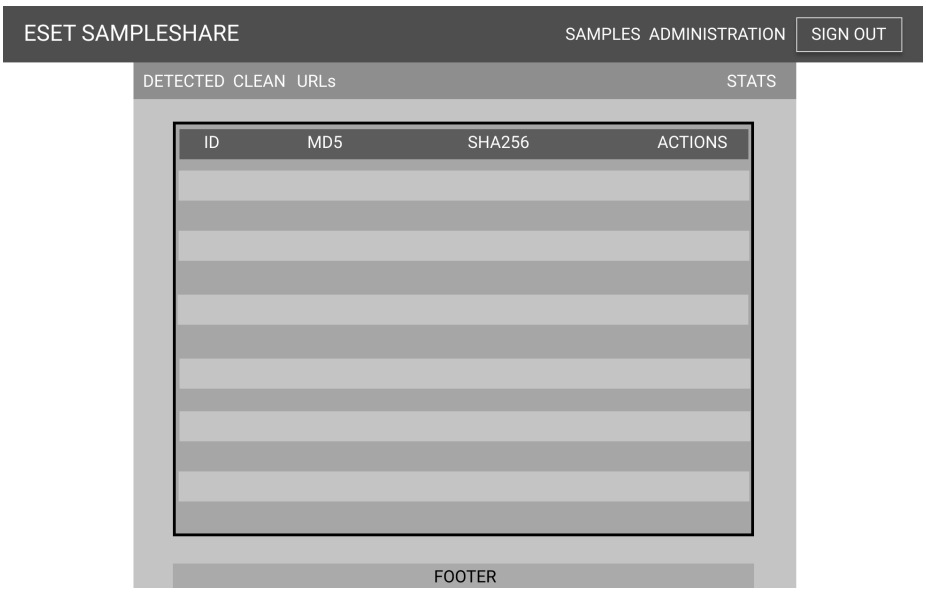
- getlist – vráti klientovi list hashov pre stiahnuteľné vzorky
- getmetadata – vráti klientovi metadáta o súboroch
- geturls – vráti klientovi list URL záznamov zo Sampleshare
- get_supported_compression – vráti klientovi možné druhy kompresie vzoriek
- get_supported_hashes – vráti klientovi podporované hashe, ktorými vie vzorky vyhľadávať
- getfile_by_list – vráti klientovi vzorky podľa listu hashov
- getfile – vráti klientovi vzorku podľa hashu

Samotná aplikácia bude musieť zvládnuť veľké množstvo požiadaviek v jednom momente a bude musieť byť dostatočne optimalizovaná, aby obsluha požiadavky bola čo najmenej náročná na zdroje počítača.

4.2 Návrh webovej aplikácie

Cieľom je vytvoriť prostredie, ktoré bude pôsobiť elegantne, ľahko sa bude v ňom orientovať a bude poskytovať dôležité informácie vhodným spôsobom pre externých a interných užívateľov.

Pre externých užívateľov bude možné na stránke vyhľadávať súbory podľa hashu, meniť a upravovať svoje údaje (mailovú adresu, heslo, PGP kľúč) a stiahnúť si klienta na komunikáciu NSSF protokolom.



Obr. 4.2: Wireframe pohľadu na tabuľku vo webovej aplikácií

Interní užívatelia spravujú vzorky systému, vedia jednotlivé balíky mazať či obmedzovať ich sťahovanie. Taktiež riadia povolenia užívateľov – ktoré vzorky môžu odoberať, všeobecnú administráciu a sledovanie štatistík. Ako rozšírenia systému je plánované zavedenie identifikačného hashu pre každého dodávateľa vzoriek a sledovanie uploadu/downloadu databáze.

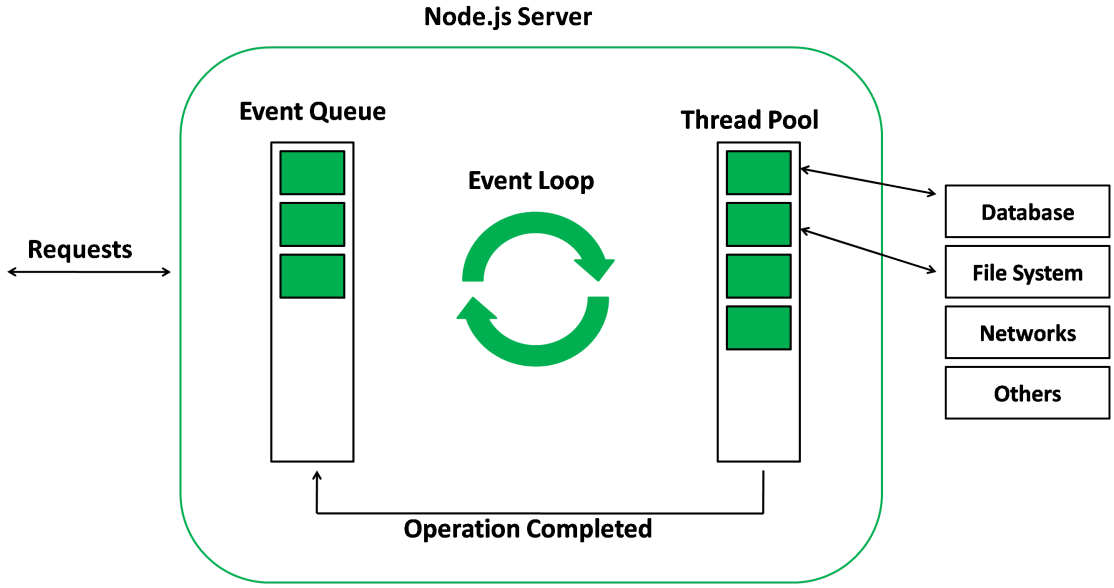
Štatistiky

Súčasťou implementácie budú taktiež rozsiahle štatistické analýzy pre interných užívateľov. Budú zobrazovať široké spektrum sledovateľných parametrov Sampleshare. Zámerom je taktiež navrhnúť zobrazenie, ktoré bude pre užívateľa prívetivejšie (v súčasnej verzií treba po zmene atribútov manuálne obnoviť graf). Grafy štatistík sa budú generovať dynamicky a v rámci zobrazenia dobového úseku si taktiež program bude ukladať okolie grafu, aby zbytočne nemusel zaťažovať server veľkými dotazmi. Postupne si ďalšie oblasti bude dotazovať a sťahovať zo serveru ak sa užívateľ bude posúvať na osi bližšie k úsekom, ktoré si program ešte nezaobstaral. Výsledkom bude plynulé zobrazenie štatistických dát.

4.3 Vývojové prostredie

4.3.1 NodeJS

Zámer implementácie leží hlavne v backende informačného systému, ktorý užívateľom bude poskytovať vzorky vírov a dát zdieľaných v systéme Sampleshare pomocou GET požiadavkov. Pre realizáciu nemenného chovania NSSF a implementácie všetkých backend funkcionalít som zvolil vývojové prostredie NodeJS. Jedná sa o interpretáciu a simuláciu webového prostredia, ktorá umožňuje písať zdrojový kód pomocou Javascriptu ako iný skriptovací jazyk (Python, PHP). Výhodami NodeJS[2] je jeho modulárnosť a široké využitie , či už na správu HTTP požiadavkov od klienta alebo výpočet a riešenie komplexnejších problémov. Architektúru spracovávania požiadaviek a kódu v NodeJS je vidieť na obrázku 4.3.



Obr. 4.3: Architektúra NodeJS serveru. Prevzaté z [9]

[2] https://nodejs.org/en/docs/

ExpressJS

Pomocou využitia knižnice ExpressJS je možné z jednoduchého skriptu v JavaScripte vytvoriť responzívny a flexibilný základ pre backend a tým vytvoriť REST API. REST (Representational state transfer) je princíp tvorby API (Application programming interface), kedy sú dodržiavané isté postupy implementácie, ktoré spolu vytvárajú jednu z najpoužívanejších architektúr pre vývoj webových aplikácií.

Základné princípy REST API[3]:

- Unifikované rozhranie – všetky požiadavky pre určitý zdroj API musia mať rovnaký formát a obsahovať rovnaké typy informácií.
- Rozdelenie serveru a klienta – obidve aplikácie musia byť od seba oddelené a kompletne nezávislé, avšak klient väčšinou zobrazuje dáta vymáhané od serveru, čiže sú prepojené iba na miestach, ktoré sa nazývajú endpointy. Klient pozná len URI (Uniform Resource Identifier – v našom prípade prípona /api), na ktoré posiela požiadavky s očakávaním odpovede, ktorá obsahuje dáta nezbytné pre chod a funkčnosť webového klienta.
- Bezstavovosť – v tomto prípade to znamená, že každá požiadavka musí obsahovať všetky dôležité informácie pre spracovanie požiadavky, to predstavuje autentizačné a autorizačné informácie, bez ktorých server nevie požiadavku obslúžiť. Server si nemôže ukladať dáta ohľadom požiadaviek klienta.
- Využitie pamäte – pokiaľ je možné, niektoré zdroje je výhodné ukladať už na serverovej či klientskej strane, umožňuje to dosiahnuť ľahkosť klienta alebo škálovateľnosť serveru.
- Vrstvovaná architektúra – klient väčšinou nekomunikuje priamo s API serverom, požiadavka prechádza cez viacero zachytávačov (interceptor) alebo middleware. ktoré napríklad slúžia na verifikáciu užívateľa posielajúceho požiadavku.

Server verifikuje a spracováva požiadavky webového klienta. Taktiež server obsluhuje požiadavky NSSF protokolu a posiela žiadané vzorky užívateľovi. Na verifikáciu je použitá aplikačná logika, cez ktorú musí požiadavka prejsť, aby sa mohlo pracovať s dátami uloženými na serveri. Webový klient vždy (okrem požiadaviek, pri ktorých nemusí byť autentizovaný tj. prihlásenie alebo registrácia) posiela v hlavičke alebo tele správy (záleží na type HTTP dotazu) autentizačný a zároveň autorizačný token typu JWT, ktorý je následne kontrolovaný v parametroch expirácie a validity.

JWT

JWT alebo JSON Web Token je spôsob ukladania relácie prihláseného užívateľa do objektu, ktorý má v sebe uložené informácie o užívateľovi a všeobecné informácie o tokene samotnom. Objekt je rozdelený na tri časti:

- Hlavička – väčšinou obsahuje dva údaje a to typ objektu, v tomto prípade vždy JWT a hashovací algoritmus použitý na šifrovanie podpisu.

[3] `https://www.ibm.com/cloud/learn/rest-apis`

- Telo – v tele sa nachádzajú všetky informácie o relácií užívateľa, ktoré sú zvolené už implementáciou a časové razítka vytvorenia tokenu ako aj čas jeho expirácie a vypršania. Neodporúča sa v tele ukladať citlivé dáta napriek tomu, že je telo celé šifrované, ale je stále jednoducho čitateľné, čo vytvára hrozbu napadnutia aplikácie.
- Podpis – každý token obsahuje podpis na overenie nemennosti obsahu tela alebo hlavičky. Vytvára sa šifrovaním reťazca zloženého z hlavičky, tela a tajného kľúča, ktorý je uložený na serverovej strane a pri každej verifikácií užívateľa je pomocou tohto podpisu overená integrita tokenu.

Celý token je poskladaný z troch častí do reťazca v tvare hlavička.telo.podpis a posielaný naspäť na webového klienta pri úspešnom prihlásení. Klient si ho ukladá a posiela spolu s požiadavkami na server až kým nevyprší jeho platnosť a užívateľ nie je nútený vytvoriť nový autentizačný token opätovným prihlásením. Nepríjemnému a niekedy častému odhláseniu vieme predísť viacerými princípmi implementácie.

4.3.2 ReactJS

Frontend aplikácie využíva framework a knižnicu ReactJS, ktorá patrí medzi jednu z najpoužívanejších frontend vývojových prostredí. Generovanie HTML dokumentu je flexibilné a dynamické, ponúka viacero riadiacich konštrukcii, ktoré umožňujú prekresľovanie a zmenu obsahu na základe chovania a akcii užívateľa. Podporuje tvorbu tried a komponent, ktoré môžu byť procedurálne použité vo viacerých častiach webovej aplikácie[4].

React Router

Jedná sa o modul pre aplikácie využívajúce ReactJS, ktorý bol vytvorený komunitne a slúži na kontrolu a zobrazovanie správneho obsahu na webovej stránke odvíjajúcej sa od cesty. Samotný balík poskytuje mnoho komponent vo frameworku React a hooks (metódy, ktoré umožňujú čerpať informácie zo stavov aplikácie, prípadne pri zmene stavu reagovať a uskutočniť bloky kódu). Všetky tieto funkcionality umožňujú jednoduchšie a intuitívnejšie vytvárať základ navigácie a zobrazovania obsahu. Komponenty `Router` a `Route` sú základom pri odlišovaní obsahu podľa cesty, vykresľujú a volajú funkčnú komponentu s vizuálnym obsahom.

Ďalšie moduly

Počas vývoja v projekte pribudlo viacero modulov, s ktorých použitím som pri návrhu nerátal. V krátkosti popísané:

- `Axios` – modul spravujúci prenos HTTP požiadaviek s backendovým serverom
- `ag-grid-react` – modul na tvorbu mapovateľných tabuliek s podpornými funkciami
- `jwt-decode` – modul na dekonštrukciu JWT tokenov a čítanie dát a príznakov z nich

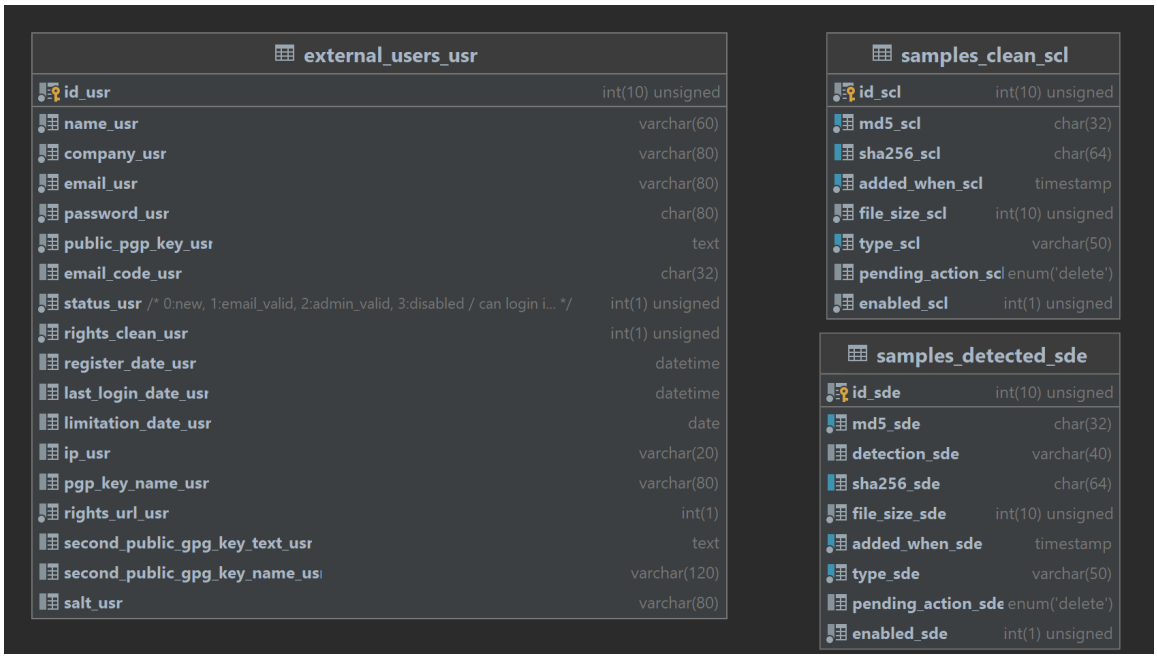
4.3.3 Databáza

Pre aplikačný server som pred samotnou implementáciou zvolil databázu typu PostgreSQL, ktorá ponúkala novšie prostredie s viacerými dátovými typmi ako klasické MySQL, tvoril

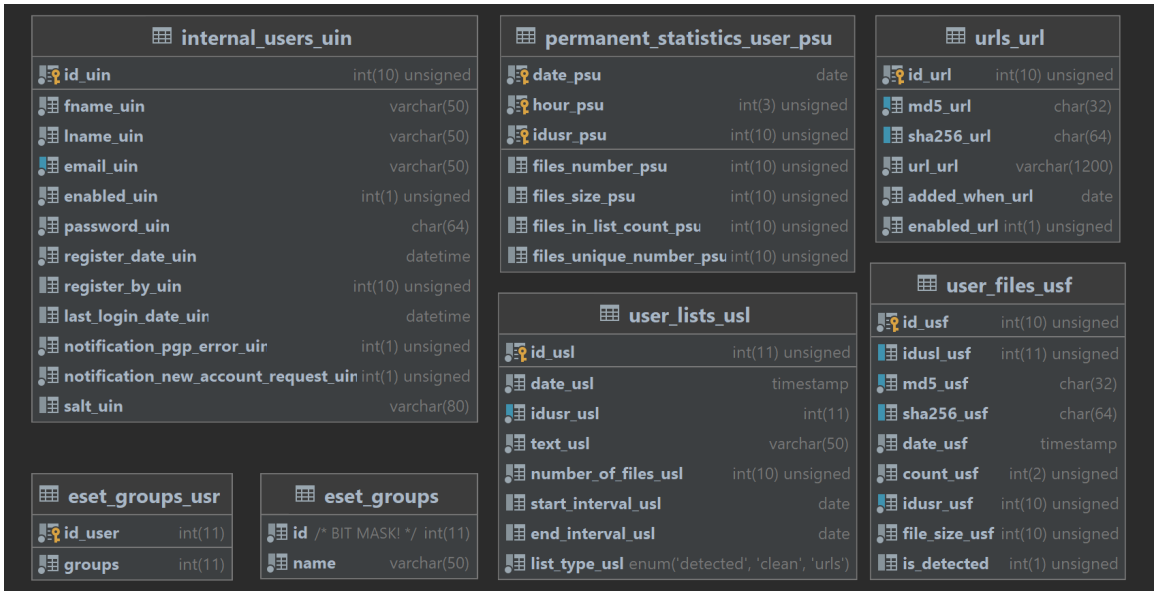
[4] `https://reactjs.org/`

som v nej základné databázové schémy využité v backendovom serveri. Počas tvorby som však narazil na nejednoznačnú a nedostatočnú dokumentáciu tohto SQL dialektu, ktoré PostgreSQL využíva. Rozhodol som sa preto databázový systém zmeniť, keďže som narazil na viacero problémov pri implementovaní schém a dotazovania sa na dáta uložené v tabuľkách. Po radách s kolegami v práci mi bol odporučený relačný databázový systém MariaDB, ktorý vznikol zo základov MySQL a je vo viacerých smeroch s databázami MySQL kompatibilný. MariaDB poskytuje dialekt podobný až identický MySQL, nové databázové stroje a jednoduché nasadenie spolu s nízkou záťažou na prevádzkujúci stroj.

Cieľom bakalárskej práce je aj získanie znalostí tvorby informačného systému pomocou týchto vývojových prostredí a ich kompletné chápanie a expertíza.



(a)



(b)

Obr. 4.4: Tabuľky databázy

Kapitola 5

Aplikovanie návrhu a implementácia

Projekt je rozdelený na tri hlavné funkčné celky. Jedná sa o server, slúžiaci na obsluhu požiadaviek typu NSSF protokolu, webovej aplikácie a k nej príslušnému backend REST API, ktoré komunikuje s databázou. Pri implementácií som sa rozhodol, že vytvoriť identickú obsluhu požiadaviek NSSF je prioritou a mala by byť vytvorená skôr, než webová aplikácia.

5.1 Metóda vývoja

Pri implementácií som zvolil iteratívny prístup, kedy sú najprv tvorené jednotlivé časti projektu od základov a následne pridávané nadstavby a doplňujúca funkcionalita. Pri ukončení každého celku bolo prevedené testovanie, či produkt bol správne implementovaný. Tento prístup viedol k tomu, že dokončené časti nepotrebovali veľa úprav po spomínanom testovaní.

Následovne som postavil celú aplikáciu podľa troch hlavných celkov, ktoré som delil na menšie ciele.

5.2 Štruktúra projektu

Pre jednotný backend, čiže API webového klienta a obsluhu NSSF správ sa rozvíja zo zdrojového súboru `server.js` v adresári `backend`, samotná implementácia obsluhy NSSF správ sa nachádza v súboroch `router.js` a `includes.js`, kde sa nachádza hlavný rozcestník pre aplikačnú logiku a dve triedy objektov s obslužnými a pomocnými metódami pre NSSF protokol. Všetky koncové body (endpointy) sú obsiahnuté v súbore `server.js`.

Zdrojové súbory frontendu sa nachádzajú v adresári `frontend`, hierarchicky najvyššie zdrojové súbory sa nachádzajú priamo tu. Komponenty a podstránky sú v adresári `components`, kde sa nachádzajú komponenty využívané vo viacerých stránkach. Tie sú všetky v adresári `pages`, kde je rozdelenie `external` a `internal` pre stránky špecifické pre týchto užívateľov, ostatné stránky sa nachádzajú v tomto adresári.

5.3 Obsluha protokolu NSSF

V tejto časti bolo hlavným cieľom vytvoriť čisto novú aplikačnú logiku, ktorá však mala identické vlastnosti a chovanie, ako jej predchodca. Jediným zdrojom informácií, ako pre-

došlá verzia pracovala na pozadí, existuje iba zdrojový súbor jej implementácie [8]. Popis systému a funkčnosti je veľmi nedostačujúci a dokumentácia kódu nie je prítomná. Dodržanie špecifikácie a implementácia bola preto veľmi obtiažna. Predošlý zdrojový kód bol písaný v jazyku PHP, čo vytvorilo možnosť odlišnej interpretácie a prístupov k implementácií.

HTTP požiadavka typu GET je zachytená serverovým API a prebehne verifikácia užívateľa, vytvorenie dvoch inštancií `userObject` a `serverObject`, kde prebehne inicializácia databázového pripojenia, uloženie údajov z požiadavky a prípadné doplnenie dát z databázy. Užívateľ posielajúci požiadavku musí byť verifikovaný a oprávnený interným užívateľom na odber hocijakej informácie z NSSF systému a taktiež musí mať udelené práva pre skupinu informácií, na ktoré sa chce dotazovať. Tento stav je pre uživateľa neoprávnený a predčasne sa ukončuje komunikácia. Je dôležité, aby externý užívateľ dbal na správne zadanie vstupných parametrov požiadavky, pretože môže jednoducho žiadať o informácie z určitého časového okna, kedy ešte nebol na túto akciu autorizovaný.

5.3.1 Metóda getList

Užívateľ pri sťahovaní vzoriek potrebuje zistiť identifikáciu súborov, ktoré chce stiahnuť, čo v našom prípade je hash súboru v MD5 alebo SHA256. Vyhľadávanie sa odlišuje podľa výberu z typu vzorky, či už sa jedná o vzorku typu `clean` (neškodná pre užívateľa) alebo `detected` (obsahujúca škodlivý software). Táto metóda už podľa názvu slúži na stiahnutie zoznamu (list) prístupných súborov pre daného užívateľa v danom časovom rozmedzí podľa zadaných parametrov `from` a `to`. Dôležitou súčasťou procesu je záznam o tejto požiadavke, ktorý sa ukladá do SQL databázy so zámerom na neskoršie použitie pri tvorbe štatistík a sledovania ruchu (traffic) u klientov.

5.3.2 Metóda getFile

Metóda vracia jeden konkrétny súbor alebo vzorku definovanú hashom. Súbor je preverený voči databázi a fyzickému úložisku, následne je zašifrovaný pomocou PGP verejným kľúčom užívateľa a poslaný cez HTTPS naspäť klientovi v blob formáte. V tomto kroku záleží na viacerých parametroch sťahovania vzorky. Užívateľ musí byť oprávnený tento súbor stiahnuť, súbor samotný musí byť v databázi označený ako voľný na stiahnutie a súbor musí samozrejme existovať.

5.3.3 Metóda getFileByList

S poskytnutým zoznamom z metódy getList užívateľ žiada o vzorky hromadne, jedná sa o iteratívne volanie funkcií metódy getFile.

Funkcia encrypt

Funkcia vracia cestu zašifrovaného súboru, ktorého cesta je vstupným parametrom. Vytvorí sa súbor s jedinečným názvom, aby sa predišlo možným konfliktom pri obsluhe množstva požiadaviek naraz. Názov je tvorený funkciou `tempnam` z rovnomenného modulu, ktorý ako parameter vyžaduje kľúčové slovo, ku ktorému pridá určitý počet znakov tak, aby sa meno so žiadnym podobným súborom nezhodovalo. Do novo vytvoreného súboru sa nakopíruje obsah vzorky, ktorá sa šifruje. Pomocou verejného PGP kľúča sa súbor zašifruje príkazom gpg. Funkciou exec sa vykoná príkaz cez príkazový riadok a výstup z neho je zachytený do premennej v programe 5.1. Vznikne súbor vzorky so zašifrovaným obsahom, ktorého čítanie

je možné len pomocou dešifrovania súkromným kľúčom, ktorý vlastní užívateľ. Dočasný súbor `tempfile` je vymazaný a šifrovaná vzorka môže byt poslaná užívateľovi. Obdobná funkcia `encrypt_buffer()` pracuje s dátami v premennej. Jediným rozdielom je že dáta z bufferu sa vpíšu do súboru a ten sa zašifruje.

```
await exec(‘gpg --batch --no-tty --homedir=${path} --always-trust
    --no-secmem-warning -e -r ${key} ${tempfile}‘)
```

Výpis 5.1: Príkaz šifrovania súboru. Parameter `key` predstavuje názov používaného kľúča a `tempfile` je cesta k súboru ktorý sa má šifrovať

Doplňujúce informácie

Pri posielaní hocijakej citlivej informácie serverom prebieha šifrovanie obsahu. Dáta sú nakopírované do dočasného súboru a jeho obsah zašifrovaný. Tento súbor je v bajtovom slede poslaný klientovi. Pri akejkoľvek nekonzistencií bolo treba ošetriť stavy, aby takto vytvorené súbory boli vždy korektne odstránené z hosťovského stroja, aby sa predišlo nadmernému zahlteniu pamäte. Pri sťahovaní viacerých súborov alebo súboru zaberajúceho veľké miesto v pamäti užívateľ vie požiadať o kompresiu súborov alebo súboru, čo zníži náklady aj serveru ale aj klienta.

Medzi ďalšie informácie o ktoré klient môže žiadať sú napríklad podporované typy hashov, podľa ktorých vie vyhľadávať vzorky, list detekovaných URL z databázy, metadáta o súboroch ktoré môže stiahnuť a podporované typy kompresií, pomocou ktorých server vie komprimovať posielané vzorky.

Hlavným problémom implementácie tejto časti projektu bola nedostupnosť informácií, na čo niektoré sekcie programu slúžili a či sa v predošlej verzií vôbec používali. Príprava na implementáciu a pochopenie fungovania predchodcu predstavovala veľa reverse engineering (vyhľadávaním infromácií o chode a správaní aplikácie), čo však v konečnom dôsledku umožnilo mnoho oblastí vylepšiť alebo systém okresať o nepotrebné funkcie a zvýšiť tak prehľadnosť a odstrániť slepé miesta. Jeden z prvých cieľov tejto refaktorizácie bola kompletná rekonštrukcia databázy, z ktorej aplikácia zbierala informácie. Postavenie nového schématu databázy značne uľahčilo prácu so zdrojmi a ušetrilo značnú časť využívanej pamäte. Obsahuje v konečnom dôsledku menej tabuliek a menej stĺpcov v tabuľkách čo zvýšilo rýchlosť operácií nad databázou.

5.4 React frontend

Manipulácia s balíkom frameworku začína príkazom `npx create-react-app`, kedy sa vygeneruje aplikačná štruktúra pre webovú aplikáciu a proces implementácie štartuje v súbore `App.js`, automaticky sa vytvorí funkčná komponenta, ktorá je vykresľovaná v koreňovom súbore `Index.js`. Každá funkčná komponenta je povinne ukončená príkazom `return` obsahujúci vykresľované komponenty. Pohľad sa už na koreňovej adrese mení podľa autentizácie a autorizácie užívateľa. Čiže už v tomto stave sa zobrazuje odlišný obsah pre prihláseného, prihláseného s nadriadenými právami a neprihláseného. Každý z týchto pohľadov obsahuje komponentu `Router`, ktorá striktne definuje chovanie a obmedzuje dostupné stránky a ich obsah. Pre príklad neprihlásený užívateľ má k dispozícií len stránku na prihlásenie, registráciu a odhlásenie. Prihlásený užívateľ sa ale k formuláru prihlásenia nedostane, keďže sa táto cesta a komponenta nenachádza v jeho „Routes“. Interní a externí užívatelia zdieľajú

len malé množstvo rovnakého obsahu, väčšina ich akcií je diametrálne odlišných a preto je `react-router-dom` flexibilným navigačným nástrojom vo webovej aplikácií.

```
<Router>
    <Navbar/>
    <Routes>
        <Route path='/search_file' exact element={<SearchByHash/>}/>
        <Route path='/download_client' exact element={<DownloadCl/>} />
        <Route path='/profile' exact element={<ProfileExternal/>} />
        <Route path='/change_password' exact element={<ChangePassword}/>}/>
        <Route path='/logout' exact element={<Logout setUser={setUser}/>}/>
        <Route
            path="*"
            element={<Navigate to="/search_file" replace/>}
        />
    </Routes>
</Router>
```

Výpis 5.2: Ukážka kódu Routeru pre externého užívateľa

Na výpise je vidieť štruktúru komponenty `Router` z modulu `react-router-dom` v rámci funkcie `render()`. Navbar sa nachádza mimo `Routes`, čiže sa vykreslí vždy bez ohľadu na adresu URL a zmena adresy nespôsobuje jeho prekreslenie. Docieli to plynulý prechod medzi zmenami obsahu stránky.

Kontext

Pre jednoduchú prácu s aktuálnymi premennými bol implementovaný kontextový wrapper (obaľovacia komponenta), ktorý poskytuje prístup k aktuálne uloženým objektom v rámci celej aplikácie. Hlavným zdieľaným objektom je `user`, obsahujúci identifikátor užívateľa, jeho aktívny token, cieľový identifikátor (pri operáciach nad inými užívateľmi), príznak autentifikácie a príznak admina. Wrapper taktiež slúži na šírenie inštancie objektu modulu `axios`, ktorý je nakonfigurovaný pracovať s JSON web tokenmi. Kontext sa jednoducho ovláda v rámci všetkých komponent. Funkciou `useContext()` sa umožní čítať informácie z kontextu. Do zdieľaného objeku je jednoduché pridať ďalšie dátové štruktúry pri rozširovaní webovej aplikácie.

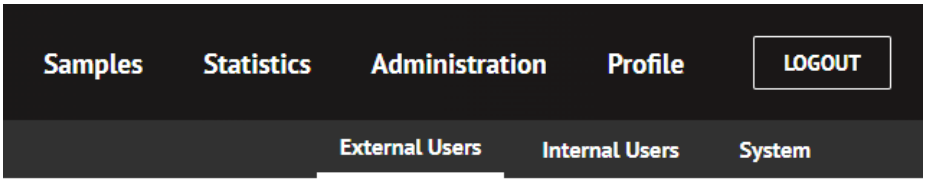
5.4.1 Navbar

Pre prehľadné premostenie medzi nesúvislými stránkami slúži navigačný panel. Rozhodol som sa implementovať panel flexibilne tak, že mení svoj obsah na základe stavu aplikácie a na akej adrese sa v tom momente nachádza, čo umožnilo jednoduchú manipuláciu a priamočiare pridávanie nových pohľadov pre navbar. Je jednoducho modifikovateľný a pomocou modulu react-router sa zobrazuje na každej stránke bez ohľadu na obsah a adresu URL.

Pre interných užívateľov slúži aj doplnkový navigačný panel, ktorý predstavuje orientáciu v menších podcelkoch. Využitie a jeho aplikovanie je vidieť v pohľadoch `administration` a `samples` obr. 5.1.

5.4.2 Login a Registrácia

Užívateľ bez prebiehajúcej relácie je presmerovaný na úvodnú stránku určenú na prihlásenie. Úspešným prihlásením sa vytvorí sada JWT tokenov. Aby bolo uľahčené udržanie



Obr. 5.1: Náhľad navigačného panelu z aplikácie pre interného užívateľa

relácie, je vytvorený obnovovací token s nastavením expirácie na dlhší časový úsek (v aplikácií nastavený na jeden deň). Tento token slúži na obnovovanie aktívneho tokenu vždy, keď jeho platnosť vyprší. Operácia, ktorá obnovuje identifikačné dáta užívateľa, je volaná aktívne pri vyslaní hocijakej požiadavky na backendový server. Zistí sa, či bol prekročený expiračný čas aktívneho tokenu a skôr než sa zašle pôvodná požiadavka, klient očakáva od servernu novú sadu tokenov. Refresh token je uložený v cookie súboroch webového prehliadača a jeho obsah je nečitateľný vďaka nastaveniu príznaku `HTTPOnly`. Nasadenie tejto formy autentifikácie a správy relácie umožnilo vytvoriť prostredie pre aplikáciu, kde je relácia užívateľa bezprostredná a chránená voči vonkajším útokom, keďže aktívna relácia má veľmi krátku dobu platnosti a je prenášaná v zabezpečenom a šifrovanom kanále (HTTPS) a k údajom obnovovacieho tokenu sa útočník tretej strany nedostane, čiže session spoofing je nedosiahnuteľný.

Registrácia v tomto pohľade je prítomná len pre užívateľov z externých spoločností. Jedná sa o jednoduchý formulár, kde sa zadávajú klasické údaje o registrovanom užívateľovi. Pre jednoznačnosť formy vstupných údajov a bezpečnosť aplikácie som sa rozhodol nasadiť sadu regulárnych výrazov na porovnanie vstupných hodnôt, aby nedošlo k vloženiu nesprávnych hodnôt či útoku typu SQL injection do databázy. Každou zmenou vstupu sú aktivované metódy `useEffect`, ktoré vykonávajú tieto testy. Ak niektorá z podmienok nie je splnená, formulár nejde odoslať. Vonkajší užívateľ musí dodať taktiež platný verejný PGP kľúč, podľa ktorého sú šifrované posielané vzorky a dáta z NSSF serveru. Kľúč je potrebné overiť na serverovej strane v try catch bloku s využitím gpg príkazu v príkazovom riadku. Ak je konto registrovaného užívateľa úspešne vytvorené, je poslaný verifikačný mail na poskytnutú mailovú adresu. Bez verifikácie nejde účet použiť na sťahovanie vzoriek. Aj po verifikácií užívateľa je potrebné od jedného z adminovských účtov, aby potvrdili legitímnosť a oprávnenie na sprístupnenie všetkých služieb aplikácie.

5.4.3 Pohľad externého užívateľa

Hlavnou službou, ktorá je poskytovaná externému užívateľovi je práve komunikácia s NSSF serverom, ktorá ale nepredstavuje žiadnu vizuálnu reprezentáciu vo webovom klientovi pre Sampleshare, so serverom sa komunikuje pomocou klientského skriptu. Skript si užívateľ vie stiahnuť práve pomocou webovej stránky, ktorá užívateľovi poskytne skomprimovaný adresár obsahujúci skript na komunikáciu s NSSF serverom v jazyku Python a možnosť sťahovať vzorky aj pomocou staršieho PHP programu (Pozn. klienti na sťahovanie vzoriek nie sú súčasťou implementácie bakalárskej práce, ich reinterpretácia nebola nutná). Pred spúšťaním treba stiahnuté skripty nakonfigurovať s užívateľovým verejným PGP kľúčom.

Pre verifikovaného užívateľa je umožnené sťahovať jednotlivé vzorky podľa hashu aj cez webového klienta. Webový klient pošle požiadavku spolu s hashom súboru na endpoint NSSF serveru a prebehne získanie súboru na backende v rámci funkcie `/getfile`, ak užívateľ má udelené práva na sťahovanie vzoriek z danej skupiny. Dôležitou súčasťou webu je aj

samotná možnosť zmeniť svoj PGP kľúč a ostatné údaje a nastavenia aplikácie. Predošlá verzia niektoré z týchto zmien nepodporovala.

5.4.4 Pohľad interného užívateľa

V tejto sekcií je obsiahnutá najväčšia časť implementácie frontendu webovej aplikácie. Je zostavená z modulov a komponent, ktoré sú navrhnuté tak, aby čo najprehľadnejšie umožnili adminom aplikácie získať aktuálne informácie o stroji, ktorý prevádzkuje webovú stránku a backendový server a spravovať entity v databázi a aj na fyzickom úložisku. Väčšina funkcionality webového klienta sa nachádza v celkoch:

- `/administration` – poskytuje tabuľky s prehľadom interných aj externých užívateľov a k nim príslušné operácie
- `/samples` – taktiež obsahuje tabuľkový prehliadač aktívnych balíkov, ktoré sú v aplikácií rozdelené na Clean (čisté, neškodné), Detected (s detekovanou hrozbou), URLs a operácie nad konkrétnymi balíkmi
- `/statistics` – pohľad webovej aplikácie, kde sa nachádzajú štatistiky

Tabuľková šablóna

Prehľad a zobrazenie informácií o rôznych dátach som zahrnul do jednej flexibilnej komponenty `Grid`. Tabuľky budú rozhodne najpoužívanejším obsahom webovej aplikácie, preto bola vytvorená šablóna, aby tabuľky boli jednoduchšie exportovateľné do ľubovolnej komponenty. Aplikovanie myšlienky prebehlo s využitím modulu `ag-grid`, ktorý poskytuje výkonnú komponentu s množstvom parametrov formátovania a chovania tabuliek.

Hlavička tabuliek je zadefinovaná v parametri `columnDefs`, obsahuje širokú množinu využiteľných polí, či už estetických alebo funkcionálnych (viď definíciu stĺpca pre dátum5.3, kedy bol zdieľaný balík pridaný do databázy). `Field` predstavuje hodnotu, na ktorú sa majú namapovať dáta v objektoch riadkov, `valueFormatter` slúži na preformátovanie dátumu na čitateľnú hodnotu, `headerName` je alias, ktorý sa zobrazí v hlavičke stĺpca (nezadaním by sa v tabuľke objavilo „Date“, čo by bolo zavádzajúce), `filter` predstavuje ktorý typ filtrovania tabuľky sa má použiť a `filterParams` obsahuje logiku filtrovania.

Ostatné stĺpce sú definované podobným spôsobom, vždy obsahujú povinný parameter `field` a rozličné parametre odvíjajúce sa od typu dát v stĺpci a ich potrebnej úpravy.

```
{
    headerName: 'Added on',
    field: "Date",
    valueFormatter: convertDate,
    filter: 'agDateColumnFilter',
    filterParams: dateFilter
}
```

Výpis 5.3: Ukážka definície jedného stĺpca v tabuľke

Po nastavení definícií stĺpcov sú žiadané informácie z backendového serveru o dáta, ktorými sa tabuľka naplní. Množstvo riadkov vrátených SQL dotazom môže byť početné a tabuľka sa ocitne v stave `Loading`, až kým sa v stave `rowData` neuložia jednotlivé riadky z požiadavky na backend. Pred samotným vykreslením sa ešte na základe obsahu tabuľky

celá formátuje parametrom `onFirstDataRendered` komponenty `<AgGridReact/>`, ktorá zavolá funkciu `formatGrid()`. Funkcia berie ohľad na hodnoty predstavujúce najdlhší reťazec a podľa nich responzívne upraví stĺpce tak, aby zaberali čo najmenej miesta, ale aby vyplnili celú plochu tabuľky. Prípady, ktoré spôsobovali vizuálne chyby boli stĺpce `SHA256` a `MD5`, ktorých reťazce sú príliš dlhé a dáta boli pre užívateľa nečitateľné. Šírka stĺpcov sa prispôsobí a všetky zobrazované informácie sú čitateľné. Aby sa predišlo zbytočnému posúvaniu v riadkoch tabuľky (scroll-u), nastavenie `pagination` vytvorí stránky, cez ktoré je možné sa preklikať a obmedzí počet zobrazených položiek na výšku webovej stránky.

Tabuľky užívateľov

Zobrazujú všetkých užívateľov a operácie, ktoré sa nad nimi dajú previesť. Pomocou dotazovania na endpoint adresu backendu `/api/get_users` sa pripravia riadky informácií do `rowData` a zobrazia sa v tabuľke. Admin má pre externých užívateľov umožnené udelenie práv na odber vzoriek a sťahovanie balíkov z jednotlivých skupín, úpravu ich PGP kľúča a celkové odstránenie užívateľa z aplikácie. Interným užívateľom vedia adminovské účty meniť všetky údaje, nastaviť mailové notifikácie a účty odstraňovať. Každá z operácií komunikuje s backendovým serverom a upravuje dáta v databázových tabuľkách, ktoré sa premietajú aj do chovania NSSF serveru.

Tabuľky vzoriek

Z API adresy `/api/get_samples` získava webový klient dáta do tabuliek o vzorkách a balíkoch, z ktorých admin vie prehľadávať všetky aktuálne zdieľané súbory NSSF serverom. Obsah si vie filtrovať na základe dátumu pridania vzorky, či je vzorka povolená na stiahnutie, podľa hashov, identifikátoru balíka a veľkosti súboru. Balíky sa dajú odstrániť a je možné zamedziť ich odber.

Tabuľky sú spravené tak, že ich znovupoužitie je jednoduché, stačí zadefinovať stĺpce a naplniť riadky zobrazovanými dátami.

Štatistiky

Admin si cez aplikáciu vie zobrazovať štatistiku vyjadrenú v počte stiahnutých vzoriek za určitý časový úsek. Dôraz na implementovanie bol kladený hlavne na jednoznačnú čitateľnosť a jednoduchú interakciu pre užívateľa, aby sa vedel dostať k relevantným informáciám z grafu.

5.5 Node.js backend API

Backendový server predstavuje aplikáciu, ktorá spravuje a čaká na prichádzajúce správy od klientov. Nastavenie serveru vzniká vytvorením objektu `app` z modulu `express.js`, následným nastavením vstupných parametrov sa určí, ako má server spracovávať prichádzajúce dáta použitím `app.use()`. Na čítanie správ sa využíva `express.json()`, keďže väčšina informácií sú poskladaných v JSON formáte. Využitie modulu `cookieParser` umožňuje jednoduchý prístup k Cookies a k ich manipulácií, v rámci handler funkcií sa dáta z cookies dekódujú a sú prístupné v `req` objekte. Server sa vytvorí volaním `https.createServer()` a spustenie jeho prevádzky sa uskutoční s `httpsServer.listen(443)`, čo je obvyklý port

pre HTTPS komunikáciu. Vytvorí sa pripojenie na databázu, ktoré je prítomné v celom serveri, ktorý ho využíva na všetky svoje dotazy.

Väčšinu zdrojového kódu tvoria endpointy, ktoré sú vymedzené na CRUD metódy (get, post, put a delete). Všetky validné URL adresy na backendové endpointy sú odchytávané načúvacími metódami. Na ukážke hlavičky funkcie je vidieť aj funkciu `verify`, ktorá verifikuje užívateľa a jeho JWT tokeny 5.4 a následne riadenie predané načúvacej funkcií. Funkcia, ktorá je volaná pred samotným prevedením cieľového kódu, sa nazýva middleware.

```
app.post("/api/logout", verify, (req,res) => {})
```

Výpis 5.4: Hlavička handleru cesty /api/logout

Backendový server obsluhuje viacero ciest s rozličnými funkcionalitami, ich krátky popis nájdete v prílohe C. Každá cesta je jedinečná, v snahe bolo endpointov vytvoriť čo najmenej, aby nevznikla redundantnosť a duplikácie.

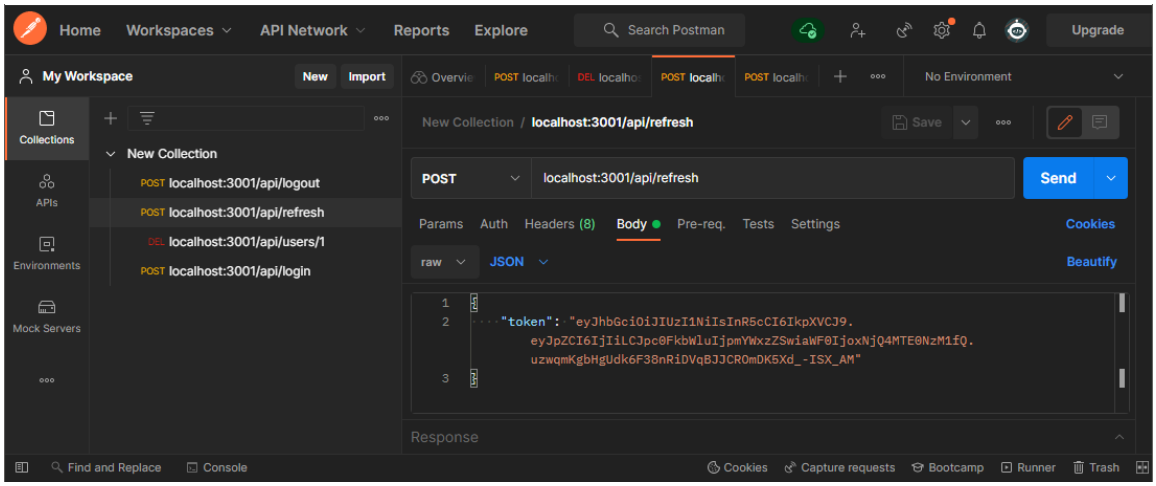
Kapitola 6

Testovanie a spätná väzba

Testovanie implementácie prebiehalo priebežne a odhalilo nepatrné chyby už počas vývoja. Cieľom bolo dosiahnuť náhradu za starú produkčnú verziu Sampleshare a zachovať rovnaké chovanie pri odpovediach od NSSF serveru.

Pri písaní kódu bol využitý program `Postman`. Program slúži na posielanie HTTP dotazov na endpointy API serveru a spracovávanie odpovedí. Pre NSSF protokol bola využitá funkcia použitia jednoduchého UI na pridanie query dotazov, ktoré sú pri dotazovaní protokolom nevyhnutné. Postman slúžil aj na dosiahnutie korektnej prevádzky middlewaru pracujúcim s JWT, pretože nasadenie užitia tokenov nebolo jednoduchou úlohou. Aplikácia podporuje aj modifikáciu a vytvorenie parametrov hlavičiek správ, v ktorých sa v konečnom dôsledku presúvali autentifikačné JWT tokeny. Zjednodušila sa implementácia endpointov a testovanie ich funkcionality. Postman ukázal, ako prenos dát funguje na pozadí a objasnil dátový prenos. Vzhľad aplikácie je vidieť na obrázku 6.1.

Testovanie NSSF serveru prebiehalo s postupným porovnávaním stavov oboch aplikácií pri dotazovaní sa na rovnakú akciu, rovnakým užívateľom a v rovnakom časovom okne. Vyžadovalo to komplexné porovnávanie všetkých detailov zdieľaných z dvoch odlišných serverov. V konečnom dôsledku sa reinterpretácia NSSF serveru ukázala ako dôveryhodná.



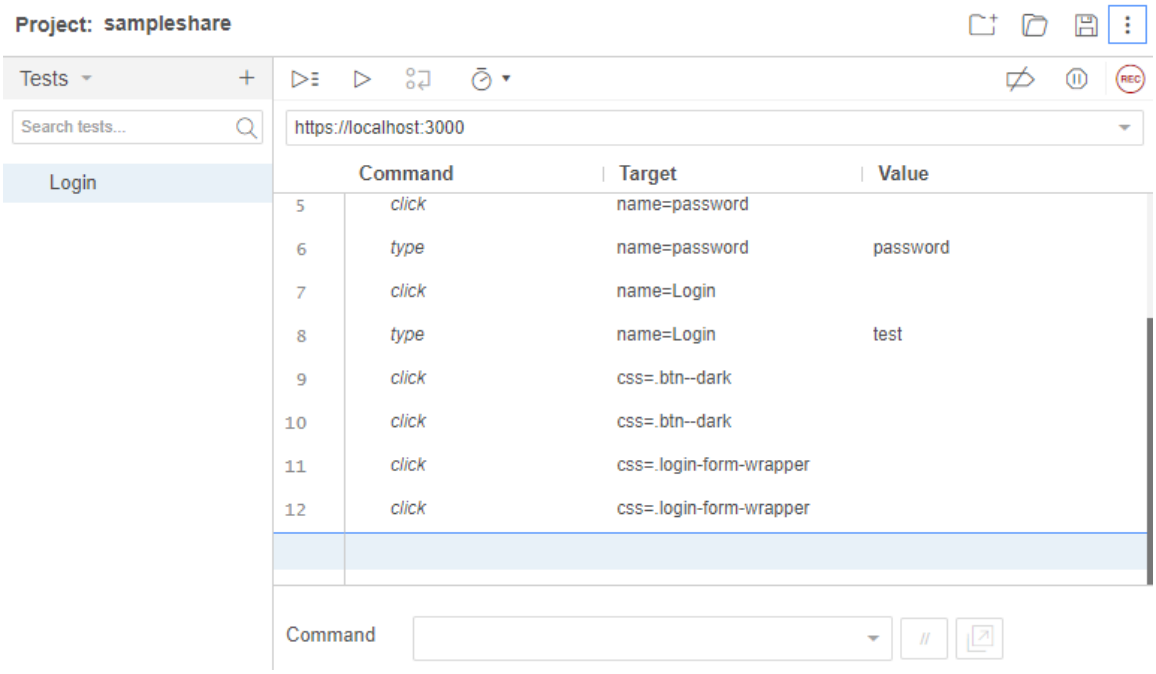
Obr. 6.1: Aplikácia Postman

6.1 Testovací skript

Rozhodol som sa implementovať aj automatické testy pre webovú aplikáciu. Základom je vytvorenie testovacej sady, v ktorej sa definuje čo, akým spôsobom alebo úkonmi chceme dosiahnuť cieľový stav, ktorý má dokazovať správny chod a chovanie webovej aplikácie. Dôležitým faktorom pri tvorení zoznamu testov je aj úsudok, ktoré funkcionality treba dôkladnejšie testovať, aby nevznikli zbytočné testy na triviálne úkony. Testy je možné konštruovať tak, že obsahujú viacero oblastí aplikácie, ktoré sa v danom test case overujú.

Vytvorením kvalitného testovacieho skriptu vznikne vývojové prostredie, v ktorom sa spustením testovania zistí, či nové zmeny v komponentách a na stránkach menia chovanie webu a či sa zmenou stavu napríklad v kontextovej premennej nestala aplikácia nepoužiteľnou. Tento typ testovania sa nazýva regresným.

Prvotné testy by sa mali zaberať správnym chovaním prihlasovacej a registračnej stránky. Zadávať treba nie len dáta, ktoré sú správne a systém ich kvalifikuje za akceptovateľné ale aj nesprávne hodnoty, ktoré vyvolávajú chyby a testujú okrajové stavy (edge cases).



Obr. 6.2: Selenium IDE so zaznamenanými klikacími vstupmi

6.1.1 Selenium

Jedná sa o open source projekt, ktorý sa stará a vyvíja moduly a aplikácie vo sfére automatizácie webových prehliadačov[1]. Komunita poskytuje viaceré softvérové nástroje ako napríklad `IDE`, `Grid`, `WebDriver` a integráciu funkcionality do balíkov a modulov pre programovacie jazyky. Pre jednoduchosť a skúsenosti s prácou som zvolil použiť distribúciu v jazyku Python, kde je prehliadač zakomponovaný v objekte `webdriver`. Driver je možné nakonfigurovať pre hocijaký webový prehliadač, v tomto projekte utilizujem Google Chrome. Zaujímavým parametrom je využitie takzvaného „headless“ módu, kedy skript prebehne bez

[1] `https://www.selenium.dev/about/`

vizuálneho otvorenia prehliadača. Po inicializácií driveru je možné na cieľovej URL imitovať užívateľa a uskutočniť všetky akcie, ktoré sú dostupné aj pri klikaní myšou na obrazovke v normálnom prehliadači.

Selenium IDE

Selenium IDE (integrované vývojové prostredie) je prídavná aplikácia do ľubovolného prehliadača, ktorá umožňuje zaznamenávať akcie užívateľa na webovej stránke a zapisovať ich do sledu interakcií. Aplikácia presne označuje elementy interakcie a ich typ, či už sa jedná o klik na tlačítko alebo vpísanie konkrétnej hodnoty do textového poľa. Rozoznávanie HTML elementov je realizované podľa identifikátorov, môžu nimi byť CSS triedy, jedinečné HTML atribúty alebo pomocou identifikačného reťazca xpath. Výhodou aplikácie je možný export úkonov a tak zjednodušene tvoriť interaktívne testy. Importovať sa dajú priamo do jazyka Python. Tvorbu testu je vidieť na obrázku 6.2.

6.2 Testovanie bezpečnosti systému

Bezpečnosť webovej aplikácie bola testovaná manuálne s využitím známych a obvyklých vstupov a akcií, ktoré ohrozovali integritu systému a vytvárali škody v aplikácií. Zameral som sa na dôkladnú implementáciu autentizačného systému, aby nedošlo k broken access control 2.2.1. Testovaním tohto rizika som dospel k záveru, že pri použití HTTPS sa útočník nevie dostať do pohľadu admina alebo aj iného užívateľa. Dáta sú prenášané šifrovaným kanálom a autentizačné tokeny sa nedajú replikovať.

Použitie brute force (prelomenie násilnou technikou) na cesty URL, ku ktorým nemá užívateľ autorizačné práva je vďaka funkcionalite `react-router` modulu nemožná, keďže sa užívateľ vie pohybovať len v URL cestách, ktoré má preddefinované a hocijaká žiadosť o prístup na nepovolenú stránku je riešená presmerovaním na default URL.

Proti SQL injection sú zavedené jednoduché opatrenia a to string escaping, ktorý celkovo zamedzuje vykonanie akéhokoľvek nebezpečného príkazu pri manipulovaním s databázou a jej dotazmi. Najobvyklejším bodom útoku je login alebo register stránka, ku ktorej má prístup ľubovolný užívateľ. Predurčený formát v registračnom formulári zamedzuje akýkoľvek vstup príkazov do backendu. Na oboch stránkach bol otestovaný SQL injection a z tohto ohľadu by mala byť aplikácia zabezpečená.

Manuálne testovanie však nikdy nie je dostatočné a preto je treba vykonať penetračné testy odborníkmi. Predať systém bezpečnostnému tímu v ESETe v rámci bakalárskej práce nebolo možné z dôvodu nedostatku času. Čakacia doba je príliš dlhá a aplikácia nebola hotová včas, aby výsledky testov boli dostupné v technickej správe. Bez podrobného testovania sa aplikácia nebude dať zaviesť do produkcie, takže penetračné testy budú musieť byť v budúcnosti vykonané.

6.3 Spätná väzba

Novo implementovaný systém bol predvedený zamestnancom spoločnosti ESET, ktorí s predošlým systémom Sampleshare pracovali a dlhšiu dobu udržovali v produkcií. Bol im ukázaný nový vzhľad užívateľského rozhrania a demonštrovanie všetkých dôležitých funkcionalít vo webovej aplikácií. Boli oboznámení aj s detailnými prvkami tabuliek a administráciou užívateľov, oboma hlavnými pohľadmi externých aj interných užívateľov. Vizuály a

funkcionalita bola prijatá s pozitívnymi ohlasmi od oboch vývojových tímov a nadriadených. Priestor na diskusiu bol zaplnený otázkami a dotazmi o riešeniach problematiky na backende a s NSSF protokolom. Bola vysvetlená aj štruktúra tabuliek a použitej bitovej masky na rozlíšenie práv externých užívateľov. Tento meeting bol prínosným aj pre moju stranu, bola odhalená chyba pri zobrazovaní v tabuľke, ktorá bola neskôr opravená. Bol dohodnutý ďalší postup na vývoji aplikácie a spomenuté možné rozšírenia systému.

Kapitola 7

Rozšírenia a ďalší rozvoj systému

Systém je navrhnutý tak, že pridanie nových modulárnych funkcionalít a webových rozhraní je jednoduché a aplikácia má vysokú škálovateľnosť. Pridanie novej položky do navigačného panelu je otázka pár riadkov kódu a pridanie funkcionálnej komponenty do react-routeru je veľmi jednoduché. Je otvorená k pridaniu nových pohľadov pre užívateľov a vylepšení pracujúcich na pozadí. Plánujem zaviesť pravidelnú údržbu databázy a fyzického úložiska, aby nevznikala prebytočná záťaž na prevádzkujúci stroj, využitím plánovaných spúšťaní čistiacich skriptov alebo pomocou databázových triggerov.

Pre jednotlivé vzorky je plánované vytvoriť sledovací systém, ktorý bude poskytovať informácie, z akého zdroja vzorka pochádza. Taktiež je predpokladaná implementácia rozšírených štatistík o vzorkách, kde sa bude dať sledovať sťahovanie každej vzorky osobitne a ktorý užívateľ ju odobral.

Prídavné moduly budú obsahovať prikladanie maskovaných súborov k zdielaným balíkom, podľa ktorých sa bude dať identifikovať, že zdieľaný balík je od spoločnosti ESET a bude jeho pôvod v interných nástrojoch rozoznateľný. Forma a typ implementovania tejto funkcionality nie je plne rozhodnutý.

Pred nasadením do produkcie budú ešte prebiehať stretnutia so zainteresovanými tímami vo firme, ktoré vyjadria svoje požiadavky a vylepšenia. Spracované špecifikácie sa budú realizovať počas roka. Predstavuje to aj zapojenie systému do monitorovacieho nástroja Zabbix. Jedná sa o softvér, ktorý aktívne zbiera informácie o sledovanej aplikácií, jeho parametroch a stave hosťovského stroja. Tieto informácie nástroj zobrazuje na webovej stránke v tabuľkovom prehľade. Najpoužívanejšou funkciou je však možnosť zasielania upozorňovacích mailov skupine užívateľov (v tomto prípade interným užívateľom Sampleshare), keď nastane nežiadaný stav alebo aplikácia nefunguje správne.

Kapitola 8

Záver

Cieľom bakalárskej práce bolo implementovať a reinterpretovať webovú aplikáciu, API server, autentifikáciu užívateľov a hlavne znovu vytvoriť identický server pre NSSF protokol. Prípravou na tvorbu zdrojového kódu bola analýza hrozieb pre informačné systémy a ochrana voči nim. Následne bolo treba preskúmať možnosti bezpečného prenosu dát cez internet. Zo špecifikácie vznikol návrh aplikácie a nového serveru.

Výsledkom práce je informačný systém pre adminovské účty, ktoré spravujú zdieľané balíky a odberateľov, kontrolujú stav hosťovského systému, kde webová aplikácia a server beží. Pre odberateľov stránka predstavuje vstupný bod k získaniu práv a možností čerpať dáta z NSSF serveru a upravovať ich profil. Obsiahle dátové tabuľky umožňujú rýchlo vyhľadať potrebné informácie o balíkoch, filtrovať obsah, jednotlivo popierať zdieľanie a odstraňovať nepotrebné vzorky. Prevedenie aplikácie je zrozumiteľné, jednoduché a elegantné, pretože jeho cieľová skupina nie je veľká a preferuje prístup k informáciám skôr ako hry na dojem. Orientácia vo webovom prostredí je jednoznačná vďaka navigačnému panelu prítomného vo všetkých pohľadoch aplikácie. Úsilie bolo vynaložené aj k bezprostrednému a bezpečnému zdieľaniu dát ako aj k celkovej bezpečnosti webovej stránky.

Webová aplikácia bola tvorená pomocou frameworku ReactJS s ktorou som nemal veľa skúseností a voľba tohto vývojového prostredia predstavovala výzvu ovládnuť technológiu a pochopiť jej silné aj slabé stránky. Postupným implementovaním funkcionálnych komponent som si objasnil využívané metódy a objekty.

NSSF server spolu s API boli vyvíjané v Node.js s pomocou modulu express, čo umožnilo prevádzkovať komunikáciu CRUD operáciami a spoľahlivo spracovávať požiadavky od webového klienta. Pri Node.js aj ReactJS som ocenil kvalitný balíčkový systém a živú komunitu developerov, ktorá riešila na fórach jednoduché ale aj netriviálne problémy, ktorá mi pomohla k dosiahnutiu mojich výsledkov.

Veľkou súčasťou práce bolo aj samostatné testovanie vizuálnych prvkov klikacími skriptami v Pythone, koncových bodov API, autentifikácie pomocou technológie JWT a hlavne Norman serveru a jeho zdieľania vzoriek. Výsledný systém je pripravený k škálovaniu a doplneniu nových modulov a funkcionalít.

Pri vývoji bol celý proces ukladaný vo verzovacom systéme Git pomocou nástroja Sourcetree, ktorý poskytol jednoduchú a prehľadnú kontrolu nad verziami zdrojového kódu.

Literatúra

[1] Abbay Bhushan, B. B. *The File Transfer Protocol* [RFC 265]. RFC Editor, november 1971. DOI: 10.17487/RFC0265. Dostupné z: https://rfc-editor.org/rfc/rfc265.txt.

[2] Black, J. *Authenticated encryption*. 1. vyd. Boston, MA: Springer US, 2005. 11–21 s. ISBN 978-0-387-23483-0. Dostupné z: https://doi.org/10.1007/0-387-23483-7_15.

[3] Diffie, W. a Hellman, M. E. *New Directions in Cryptography*. 1976.

[4] Dobeš, M. *Nástroj pro penetrační testování webových aplikací*. Vysoké učení technické v Brně. Fakulta informačních technologií, 2015.

[5] Freier, A. O., Karlton, P. a Kocher, P. C. *The Secure Sockets Layer (SSL) Protocol Version 3.0* [RFC 6101]. RFC Editor, august 2011. DOI: 10.17487/RFC6101. Dostupné z: https://rfc-editor.org/rfc/rfc6101.txt.

[6] Hurta, M. *Odvozování pravidel pro mitigaci DDoS*. Vysoké učení technické v Brně. Fakulta informačních technologií, 2016.

[7] Kutypa, M. *Nástroj pro detekci zranitelnosti SQL Injection*. Vysoké učení technické v Brně. Fakulta informačních technologií, 2013.

[8] Moise, R. *Norman Sample Sharing Framework*. Dostupné z: https://github.com/Avira/virex.

[9] Sen, A. *Node.js Server*. October 2021. Dostupné z: https://www.geeksforgeeks.org/node-js-event-loop/.

[10] xaedes, jfreax a Acdx. *PGP diagram*. May 2012. Dostupné z: https://en.wikipedia.org/wiki/Pretty_Good_Privacy#/media/File:PGP_diagram.svg.

Príloha A

Obsah priloženého pamäťového média

```
/
├──source/
│   ├──sampleshare/        - Zdrojové súbory práce
│   │   ├──frontend/       - Zdrojové frontendu
│   │   └──backend/        - Zdrojové backendu
│   │
│   └──latex.zip           - Zdrojové súbory textu (Latex)
└──README.txt              - Návod na spustenie projektu
```

Príloha B

OWASP TOP 10

List najčastejších bezpečnostných rizík webových aplikácii v roku 2021

- A01:2021-Broken Access Control moves up from the fifth position; 94 % of applications were tested for some form of broken access control. The 34 Common Weakness Enumerations (CWEs) mapped to Broken Access Control had more occurrences in applications than any other category.
- A02:2021-Cryptographic Failures shifts up one position to #2, previously known as Sensitive Data Exposure, which was broad symptom rather than a root cause. The renewed focus here is on failures related to cryptography which often leads to sensitive data exposure or system compromise.
- A03:2021-Injection slides down to the third position. 94% of the applications were tested for some form of injection, and the 33 CWEs mapped into this category have the second most occurrences in applications. Cross-site Scripting is now part of this category in this edition.
- A04:2021-Insecure Design is a new category for 2021, with a focus on risks related to design flaws. If we genuinely want to "move left" as an industry, it calls for more use of threat modeling, secure design patterns and principles, and reference architectures.
- A05:2021-Security Misconfiguration moves up from #6 in the previous edition; 90% of applications were tested for some form of misconfiguration. With more shifts into highly configurable software, it's not surprising to see this category move up. The former category for XML External Entities (XXE) is now part of this category.
- A06:2021-Vulnerable and Outdated Components was previously titled Using Components with Known Vulnerabilities and is #2 in the Top 10 community survey, but also had enough data to make the Top 10 via data analysis. This category moves up from #9 in 2017 and is a known issue that we struggle to test and assess risk. It is the only category not to have any Common Vulnerability and Exposures (CVEs) mapped to the included CWEs, so a default exploit and impact weights of 5.0 are factored into their scores.
- A07:2021-Identification and Authentication Failures was previously Broken Authentication and is sliding down from the second position, and now includes CWEs that are more related to identification failures. This category is still an integral part of the Top 10, but the increased availability of standardized frameworks seems to be helping.

- A08:2021-Software and Data Integrity Failures is a new category for 2021, focusing on making assumptions related to software updates, critical data, and CI/CD pipelines without verifying integrity. One of the highest weighted impacts from Common Vulnerability and Exposures/Common Vulnerability Scoring System (CVE/CVSS) data mapped to the 10 CWEs in this category. Insecure Deserialization from 2017 is now a part of this larger category.
- A09:2021-Security Logging and Monitoring Failures was previously Insufficient Logging & Monitoring and is added from the industry survey (#3), moving up from #10 previously. This category is expanded to include more types of failures, is challenging to test for, and isn't well represented in the CVE/CVSS data. However, failures in this category can directly impact visibility, incident alerting, and forensics.
- A10:2021-Server-Side Request Forgery is added from the Top 10 community survey (#1). The data shows a relatively low incidence rate with above average testing coverage, along with above-average ratings for Exploit and Impact potential. This category represents the scenario where the security community members are telling us this is important, even though it's not illustrated in the data at this time.

Príloha C

Koncové body API

- `/get_external_user/:id` - Získanie informácií o užívateľovi zo skupiny external.
- `/get_internal_user/:id` - Získanie informácií o užívateľovi zo skupiny internal.
- `/external_change_profile` - Zmena informácií o užívateľovi zo skupiny external.
- `/internal_change_profile` - Zmena informácií o užívateľovi zo skupiny internal.
- `/get_system_specs` - Získanie informácií o hosťovskom prevádzkovom stroji, zväčša pomocou modulu `systeminformation`.
- `/verify_user` - Verifikácia užívateľa v rámci používania aplikácie.
- `/get_samples` - Dotaz na všetky zdieľané vzorky, rozlišované na `clean`, `detected` a `urls`.
- `/download_client` - Vracia zazipovaný archív s klientskými skriptmi pre NSSF server.
- `/change_password` - Zmena hesla užívateľa.
- `/external_get_username/:userid` - Získanie mena užívateľa.
- `/get_users` - Dotaz na všetkých užívateľov, rozlišovaných na `external` a `internal`.
- `/disable_samples` - Zrušenie odberu vzorky.
- `/enable_samples` - Povolenie odberu vzorky.
- `/delete_samples` - Odstránenie vzorky.
- `/internal_create_profile` - Vytvorenie interného účtu.
- `/disable_users` - Zrušenie práv odberu pre externého užívateľa.
- `/enable_users` - Pridanie práv odberu pre externého užívateľa.
- `/delete_users` - Odstránenie užívateľského konta z aplikácie.
- `/login` - Prihlásenie do webovej aplikácie.
- `/register` - Registrácia externého užívateľa.

- `/refresh` - Obnova JWT tokenov, predĺženie relácie užívateľa.
- `/logout` - Odhlásenie z aplikácie, zrušenie všetkých JWT tokenov viazaných k tomuto účtu.
- `/server` - Cesta k NSSF serveru, je potrebné v query reťazci špecifikovať. parametre `action`, `user`, `from` a `to`.