Mendelova univerzita v Brně
Provozně ekonomická fakulta

Webová aplikace pro analýzu útoku na vybrané servery MENDELU

Bakalářská práce

Vedoucí práce:
Ing. Oldřich Faldík

Aleš Lerch

Brno 2016

Děkuji panu Ing. Oldřichu Faldíkovi za to, že mi umožnil vypracovat tuto práci, dale za jeho cenné rady a připomínky v průběhu tvorby. Děkuji i panu Ing. Stratosu Zerdaloglu a panu Ing. Jiřímu Passingerovi za možnost tvořit aplikaci pro Ústav informačních technologií PEF MENDELU a za poskytnutí důležitých informací.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Webová aplikace pro analýzu útoku na vybrané servery MENDELU**
vypracoval samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*
Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.
Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

Brno 2016 .....

Abstract

Lerch, Aleš. Security, web application for web attacks control and analysis on chosen servers MENDELU. 2016

Thesis focuses on web application creation for attack analysis aimed on specific servers MENDELU. Target was achieved with creating web application made by Flask framework and application for getting data for another analysis. Whole solution is using NoSQL database MongoDB for data writing and reading. Data are acquired by controlling with rules every single input HTTP request. For result data presentation are used interactive graphs. Application is programmed with Pyhton 3. Main thesis result is application, which could run on server a control every single HTTP request on port 80 from client.

Key words:
Security, Python3, MongoDB, Flask, Exploits, NoSQL, Javascript, D3js, Web application

Abstrakt

Lerch, Aleš. Bezpečností, webová aplikace pro kontrolu a analýzu webových útoků na vybrané servery MENDELU. 2016

Cílem práce je vytvoření aplikace pro analýzu webových útoků cílených na dané servery MENDELU. Cíle bylo dosaženo pomocí vytvoření webové aplikace tvořené pomocí framerworku Flask a aplikace, které získává data pro další analýzu. Celé řešení využívá NoSQL databázi MongoDB pro zápis a čtení dat. Data jsou získána tak, že jsou kontrolovány pravidly jednotlivé HTTP požadavky. Pro prezentaci výsledků a analýzy jsou použiti interaktivní grafy. Aplikace je programována v jazyce Python 3. Hlavním výsledkem práce je aplikace, která může běžet na serveru a kontrolovat jeho vstupní HTTP požadavky od klienta na portu 80.

Klíčová slova:
Bezpečnost, Python3, MongoDB, Flask, Exploits, NoSQL, Javascript, D3js, Webová aplikace

Obsah

Seznam obrázků

1 Úvod a cíl práce

1.1 Úvod

Vývoj webových technologií se za posledních několik let rychle posunul. Dnes je vývoj webových aplikací a stránek každodenní záležitostí a během dne jich vznikne mnoho nových. Rychlý vývoj tohoto informačního odvětví má i negativní dopad. A to cílené webové útoky s cílem škodit nebo ukrást citlivá data. Skrze naše pohodlí dnes ukládáme na Internet vše. Většina těchto informací je zpřístupněna právě skrze webové stránky. Pro získaní dat za cílem škodit nemusí útočník útočit přímo nadatabázi, ale pro jeho využití stačí prolomit případnou slabou obranu zabezpečení webu nebo webové aplikace. Další nové hrozby se objevují na internetu každý den a právě napadených uživatelů je stále více. Názorným příkladem může být vytváření online dokumentů, tabulek, prezentací, fotografií, e-mailů a jiných. To vše je možné ukládat a kontrolovat pomocí webového prohlížeče. Tím si sami zjednodušíme práci a zpohodlníme, ale vystavujeme se většímu riziku, než kdybychom tyto operace uskutečnily lokálně na osobním počítači. Sami tak nabádáme útočníky, aby tak jednali skrze webové služby a mohli nás poškodit odcizením dat. To platí i pro budoucnost, naše závislost na online službách se neustále zvyšuje. Raději využíváme cloudové a jiné webové aplikace. Raději zálohujeme data online na cloud než na externí disk. Stejně jako uživatelé, tak i útočníci využívají stejný protokol, a to HTTP nebo HTTPS protokol. Skrze tyto protokoly tvoříme velké množství požadavků na server. Veřejné porty 80 (HTTP) a 443 (HTTPS), které jsou otevřeny dynamickému přenosu dat, mohou být součástí bezpečnostního rizika, pokud nejsou neustále monitorovány. I přes jejich neustále monitorovaní může dojít poškození uživatelů (Google Inc., 2015).

1.2 Cíl práce

Cílem práce je najít možné řešení, jak chránit vybraný server MENDELU před webovými útoky. Ověřit si existující nástroje na trhu, které autor porovná a zvolí optimální řešení pro lepší bezpečnost. Po zjištění optimální kombinace požadavků a nástrojů je třeba vytvořit návrh aplikace. Po návrhu aplikace musí být aplikace naimplementována. Implementace je třeba podrobně popsat s detaily. Po úspěšné tvorbě aplikace musí být řešení řádně otestováno. Po testech sestavit popis, jak vytvořenou aplikaci nasadit na server a spustit ji. Na závěr celou funkčnost aplikace zrekapitulovat a přidat popis možného budoucího rozšíření.

2 Současný stav

V současné době patří bezpečnost mezi klíčové prvky každé fungující webové aplikace. Ochrana proti cizímu vniknutí a ochrana dat je povinností každého správce webu. Protože je kladen takový důraz na bezpečnost, je v této oblasti mnoho firem a vývojářů, kteří se na tuto problematiku soustředí. Jednotlivé nástroje pro zabezpečení se mohou lišit přístupy k řešení problému i jejich postupy. Může se jednat o open source s jednoduchými nástroji nebo se jedná o velkou mezinárodní společnost nabízející své služby, nástroje a konzultace s odborníky právě pro odhalení zranitelných míst (RedTeam Secure Consulting, 2016).

2.1 Webové útoky

Cílem webových aplikací je poskytovat svá data a své služby uživatelům. Ti mají potřebu neustálého přístupu, aby mohli svá data kontrolovat nebo upravovat. Každé aplikaci hrozí nebezpečí odhalení chyb, které mohou vést k poškození uživatele. Pokud webová aplikace není často aktualizována, může se tento proces jenom zrychlit. V nejhorších případech dochází k odcizení citlivých dat nebo případným útokům na uživatele ze samotné strany aplikace. Příkladem může být scénář, kdy útočník změní obsah stránky tak, aby uživatele odvedl na jiné pochybné stránky za cílem získání cizích dat, anglicky řečeno *phising*. Nezabezpečené webové aplikace a webové služby poskytují útočníkům přístup databázím nebo jim dovolují spáchat nelegální operace při použití kompromitovaných stránek (Acunetix Company, 2016).

Aplikační vrstva je pro ochranu tou nejtěžší. Potíž je ve zranitelnosti v závislosti na vstupu uživatele. Tato vrstva je vystavena celému světu. Pro připojení k této vrstvě stačí uživateli připojit se přes port 80 (HTTP), anebo port 443 (HTTPS). Kyberkriminálníci se v dnešní době soustředí více na objevování slabých míst ve webových aplikací, jako jsou eCommerce platformy, blogy, přihlašovací stránky a další dynamický obsah (Acunetix Company, 2016).

SQL Injection

SQL Injection (dále SQLi) je způsob napadení databázové vrstvy aplikace vsunutím (injection) kódu přes neošetřený vstup a vykonání upraveného SQL dotazu. To vede k útoku s propojením aplikační vrstvy s databázovou vrstvou. Další známou technikou je použití UNION SQL operátoru, což dovoluje útočníkovi kombinovat více příkazů SELECT v jeden výsledek. Možné je i využití neexistujících hodnot, jako je hodnota indexu -1. Tato metoda se používá často i k prolomení autorizace, kdy útočník prolomí přihlášení do systému za pomoci SQL. Zabránit tomuto útoku pomáhá jednoduchá metoda *escapování* potenciálně nebezpečných znaků. Například každý výskyt uvozovky (') v parametru musí být nahrazen dvěma uvozovkami pro vytvoření úplného validního řetězce. Každá stránka nebo webová aplikace, která využívá databáze založené na SQL, je zranitelná proti tomuto útoku. Zároveň se jedná

o jednu z nejstarších metod webového útoku, přesto je tato metoda i dnes neustále používána a velice nebezpečná. V případě úspěšného napadení se může útoč-ník dostat k celé databázi, ve které může upravit obsah. Například přidat falešné záznamy, některé změnit a jiné smazat. Aby mohl být útok úspěšně aplikován na databázi serveru, útočník musí nejprve najít vstup ve webové aplikaci, na který se vztahuje SQL příkaz k databázi (Acunetix Company, 2016).

Názorná ukázka historie použití tohoto druhu útoku v praxi ukáže, jak účinný tento útok je.

- V únoru 2002 Jeremiah Jacks objevil, že Guess.com byla zranitelná proti SQLi útoku po vytvoření precizně zabaleného URL k stažení záznamu více jak 200.000 jmen, čísel kreditních karet a jejich data vypršení v databázi pro uživatele (SecurityFocus, 2002).
- 1. září 2005 mladý hacker použil SQLi k prolomení stránek Taiwanské společnosti pro bezpečnost a následnému odcizení zákaznických dat (PCWorld Company, 2005).
- V květnu 2008 serverová farma v Číně používala automatické dotazy k Google vyhledávacím enginu k identifikaci stránek, které byly zranitelné proti automatickému nástroji pro SQL útok (Bloombit, 2008).
- 27. dubna 2011 byla mysql.com, oficiální domovská stránka pro MySQL, byla prolomena útokem, který použil SQL blind injection (Sucuri Blog, 2011).
- V červenci 2012 hackerská skupina ohlásila ukradení 450.000 přihlašovacích údajů ze stránek Yahoo!. Přihlašovací údaje byly ukládány jako text, k útoku byla použita technika SQL union-based injection (ZDnet Company, 2012).
- V červnu 27 2013 hackerská skupina "RedHack" prolomila administrativní stránky Istanbulu. Prohlásili, že měli možnost vymazat záznamy o účtech občanů za vodu, plyn, Internet nebo elektřinu. Dodatečně zveřejnili administrátorské jméno a heslo pro občany, kteří si tak mohli své účty promazat. Zprávu zveřejnili na Twitteru (Twitter Corporation, 2016).
- V říjnu 2015 byl SQLi útok použit k ukradení 156.959 osobních údajů zákazníků britské telekomunikační společnosti Talk Talk. Jejich servery byly napadeny díky zranitelnosti webového portálu společnosti, který byl napojen na databáze (Information Commissioner's Office Company, 2016).

Boj proti tomuto druhu útoků je rozsáhlý. Například v PHP jazyku existuje mnoho funkcí, které se snaží zabránit SQLi pomocí escapovaní znaků jako například *pg_escape_string()* pro PostgreSQL. Funkce, které slouží k ochraně databází, neobsahují funkce pro escapování v SQL. Funkce se nazývá *addslashes* (string $str). Nevrací string se zpětnými lomítky před znaky, které musí v dotazech na databázi být v uvozovkách. Mezi tyto znaky patří jednoduchá uvozovka (’), dvojitá uvozovka (“), zpětné lomítko (”") a NULL. Běžné předávání escapovaných řetězců SQL data-

bázi je náchylné k chybám, protože je snadné zapomenout daný řetězec escapovat. Vytvoření transparentní vrstvy za cílem ošetření dat ze vstupu může snížit náchylnost k chybám, dokonce ji eliminovat úplně (Owasp Organization, 2016).

Blind SQL Injection

Blind SQL Injection se liší od normálního tím, že útočník nespoléhá na chybný výpis database, jako je to u normálního SQLi útoku. To je důvod, proč se nazývá Blind SQL Injection (slepý). Pokud má databáze vypnuté hlášení chyb, pak může útočník stejně tento útok použít.

Podoba chybové hlášky při použití SQLi může vypadat následovně.

```
Microsoft SQL Native Client error \'80040e14\'
Unclosed quotation mark after the character string ''.
target.asp, line 9
```

Administrátoři rychle přišli na to, že ukazovat výpis chybových hlášek veřejnosti není moudrá věc, a proto začali zjednodušovat detaily z jednotlivých chybových zpráv. To je samozřejmě špatné řešení, protože neřeší základní problém – vstup uživatele může ještě být součástí příkazu SQL. Takto vznikl útok Blind SQL Injection. Útočníkům stačí vědět, kdy je vstup interpretován jako SQL příkaz. S tímto útokem jsou spojeny dvě techniky, které jsou často používány. Jsou to**Content-based Blind SQL Injection** a **Time-based Blind SQL Injection**.

Content-based Blind SQL Injection

Pro tento typ útoku, se útočník snaží ověřit, zda je databáze náchylná k útoku porovnáním obsahu výsledků jiných dotazů, které vrací Pravda nebo Nepravda. Následuje příklad zobrazení produktu s hodnotou id 34. Odkaz provede příkaz v databázi. Druhý příkaz vrací hodnotu Nepravda.

```
http://www.shop.local/item.php?id=34
SELECT name, description, price FROM Store_table WHERE ID = 34

http://www.shop.local/item.php?id=34 and 1=2
SELECT name, description, price FROM Store_table WHERE ID = 34
and 1=2
```

Pokud útočník změní konec odkazu na 1=1, pak je navrácena hodnota Pravda spolu s dalšími detaily položky 34. To dokazuje zranitelnost proti tomuto útoku.

Time-based Blind SQL Injection

Pro časové útoky potřebují útočníci nařídit databázi, aby vykonala časově náročnou operaci. Pokud stránka neodpoví okamžitě, webová aplikace je zranitelná proti

Time-based Blind SQL injection. Populární časová operace je spací. Může například vypadat následovně.

```
http://www.shop.local/item.php?id=34 and if(1=1, sleep(10), false)
```

Aplikace je zranitelná, pokud je odpověď zpožděna o deset sekund. Útok je často používán k vytvoření schématu databáze a získání všech dat v databázi. To se provádí pomocí techniky hrubé síly, což vyžaduje mnoho požadavků, které mají být odeslány na server (Acunetix Company, 2016).

XSS – Cross–site Scripting

XSS se váže na klientovu stranu, protože útočník může vložit útočný skript do webu nebo webové aplikace a následně je spuštěn v momentě, kdy daný web uživatel navštíví. Zranitelnost proti útoku XSS nastane, když webová aplikace využívá neověřený nebo neošetřený vstup od uživatele. Útočník se nesnaží na uživatele útočit přímo. Místo toho se snaží odhalit zranitelné místo, na které by cílená osoba mohla na webu narazit. Pak stačí, aby oběť sama spustila nevědomky útočný skript ve svém prohlížeči. Při útoku je nejčastěji používán Javascript, protože ho využívá snad každý prohlížeč (dále se používá VBScript, ActiveX nebo Flash). Tento útok připomíná SQLi, jelikož i zde útočník vkládá útočný kód do aplikace, který se ale spustí později. Tato past je silnou stránkou útoku, protože může lehce napadnou uživatele, který nečeká útok ze strany své oblíbené aplikace, které zcela důvěřuje. Útočník vloží script jako normální string, ale oběti se zobrazí jako kód (Acunetix Company, 2016).

XSS je stejně populární útok jako SQLi, protože i proti tomuto druhu útoku je mnoho webů a webových aplikací zranitelných. Následuje příklad XSS útoku.

```
<?php
echo $_GET['id'];
www.example.com/index.php?id=<h1>Vítej světe</h1>
?>
```

To způsobí vypsání v prohlížeči text ”Vítej světe”, který je ovšem naformátovaný jako nadpis první úrovně. Aplikace **nesmí** umožnit uživatelům zvenčí do stránky svévolně vkládat jakékoliv interpretovatelné HTML nebo jiné značky. Upravování vzhledu nemusí způsobit takové škody (záleží na schopnostech útočníka), ale v momentě použití tagu *<script>* se může jednat o silný nástroj pro útok. Například po tomto volání skriptu vyskočí na uživatele upozorňující okno:

```
www.example.com/index.php?id=<script>alert('Vítej světe');</script>
```

Škody, které útočník způsobí za pomoci Javascriptu, se nemusí ihned projevit. Od té doby co prohlížeče spouští Javascript ve velmi kontrolovaném prostředí, navíc javascript má omezený přístup do uživatelova operačního systému a dat, jsou prohlížeče daleko bezpečnější. Avšak po zváženi k čemu všemu mají útočníci přistup,

je snadnější pochopit, jak mohou být útočnici kreativní v tvorbě vlastních útoků. Útočný skripty mají přístup ke stejným objektům jako má zbytek webové stránky, včetně přístupu ke cookies. Ty jsou skoro vždy používány k ukládaní *session* relací a pokud útočník může získat uživatelův *session* cookie, pak se mohou vydávat za samotného uživatele. Javascript může číst a tvořit libovolné modifikace k uživatelově DOM (v rámci stránky, který je javascript spuštěn). Javascript může použít XML-HTTPReuest k poslaní HTTP požadavku s libovolným obsahem na libovolné místo. Javascript v moderních prohlížečích může mít vliv na HTML5 knihovny jako jsou pro přístup k uživatelovy geolokace, webové kamery, mikrofonu a dokonce ke speřickým souborům od uživatelova souborového systému. XSS spolu s některými dobře vymyšlenými částmi útoku s pomocí sociálního inženýrství, může dostat útočníka velmi daleko (ukradení cookies, keylogging, phishing a ukradení identity). Příklad útočného skriptu pro získání cookies.

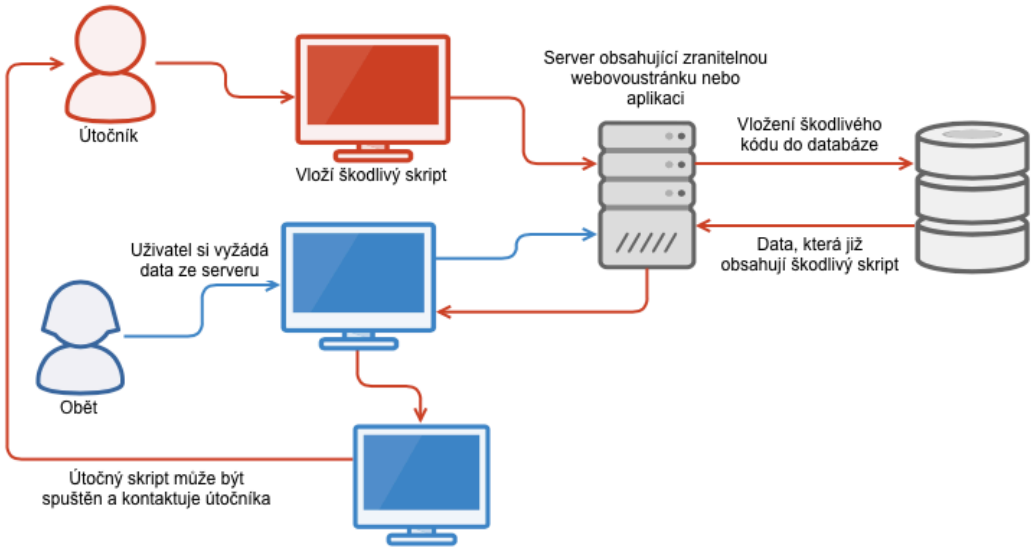
Škody, které útočník způsobí za pomoci Javascriptu, se nemusí ihned projevit. Od té doby, co prohlížeče spouští Javascript ve velmi kontrolovaném prostředí jsou daleko bezpečnější, protože Javascript má tak omezený přístup k uživatelově systému a datům. Avšak po zvážení, k čemu všemu mají útočníci přistup, je snadnější pochopit, jak mohou být útočnici kreativní v tvorbě vlastních útoků. Útočné skripty mají přístup ke stejným objektům, jako má zbytek webové stránky; včetně přístupu ke cookies. Ty jsou skoro vždy používány k ukládaní session relací, a pokud útočník může získat uživatelův *session* cookie, pak se mohou vydávat za samotného uživatele. Javascript může číst a tvořit libovolné modifikace k uživatelově DOM (v rámci stránky, který je javascript spuštěn). Javascript může použít *XML-HTTPReuest* k poslaní HTTP požadavku s libovolným obsahem na libovolné místo (Acunetix Company, 2016). Javascript v moderních prohlížečích může mít vliv na HTML5 knihovny, jako je přístup k uživatelovy geolokaci, webová kamera, mikrofonu, a dokonce ke specifickým souborům uživatelova souborového systému. XSS spolu s některými dobře vymyšlenými částmi útoku s pomocí sociálního inženýrství může útočníka dostat velmi daleko (ukradení cookies, keylogging, phishing a ukradení identity). Příklad útočného skriptu pro získání cookies:

```
1 <script>
2    window.location="http://site.com/?cookie=" + document.cookie
3 </script>
```

Perzistentní XSS je zdaleka nejpoužívanější variantou útoku. Jedná se o variantu, kdy se útočníkovi podaří umístit škodlivý kód do napadeného webu natrvalo. Typickým příkladem je vložení diskuzního příspěvku, do kterého se spolu s běž- ným textem vloží i Javascriptový škodlivý kód. Příspěvek se uloží do databáze a následně se zobrazí všem následujícím návštěvníkům webu, včetně spustitelného skriptu. Jsou zaznamenány případy použití XSS s dalšími útoky a tím pádem slouží XSS jen jako pomocný nástroj pro přípravu a provedení dalších druhů útoků (PHPGuru, 2008).

2.2 Blind XSS

Stejně jako Blink SQL Injection, tak i Blind XSS útočí naslepo (anglicky řečeno *blind*) tak, že rozmístí sérii útočných skriptů na webové stránky, které jsou schopny je uložit natrvalo (do databáze nebo záznamového souboru). Potom bez znalosti o detailech, kde se script nachází nebo kdy bude spuštěn, útočník vyčká až se načtou a spustí oběti na stránce. Protože většina XSS útoků není persistentní, spoléhá na bezprostřední načtení stránky. Proto jsou tyto útoky vypuštěny ve velkém množství, aby měl útočník co největší šanci pro úspěch. Napadeny jsou nejčastěji přihlašovací formy, záznamy, aplikace, fóra, služby pro zprávy, feedback fóra, chatovací okna a jiné.



Obrázek 1: Schéma Blind XSS útoku.

Útoky mohou mířit i na administrátora aplikace, pokud je skript uložen do databáze anebo do záznamového souboru, který může později administrátor zkontrolovat a nebezpečný kód se mu spustí s cílem útočit. Útočníkům může ztížit přístup to, že se musí přihlásit. Stejně je nejlepší obranou kontrola každého vstupu aplikace. Z hlediska osob jsou nejvíce pod útokem moderátoři fóra, protože obsahují *session* s přihlašovacími údaji, což jsou cenná data pro útočníka. Platí zlaté pravidlo nikdy nevěřit uživateli, ať už zadává jakýkoliv vstup. Existující speciální nástroje, které kontrolují každý vstup ve webové aplikaci kontrolují, zda-li nebyl vložen vstup, jsou ale často tvořeny speciálně na zakázku a jsou drahé pro pronájem. I přes použití všech moderních technologií, jako je HTML 5, nebo kontroly vstupů, může být aplikace stále citlivá na XSS útok. Kromě toho Blind XSS útoky je ještě více obtížné odhalit, protože skript je spuštěn na zcela jiné webové aplikace než na té, ve které byl vložen (Acunetix Company, 2013).

Directory Traversal Attack

Directory traversal (někdy přezdíván *dot dot slash*) je druh útoku, který dovoluje útočníkovi přístup k souborům nebo adresářům, které potencionálně sdílí server navenek ve svém systémovém adresáři. Útočník může manipulovat s URL tako-vým způsobem, aby místo webové stránky nalezl náhodné soubory nebo spustil různé systémové příkazy. Jakékoliv zařízení, které zpřístupňuje rozhraní založené na protokolu HTTP, je potenciálně zranitelné proti tomuto druhu útoku. Většina webový stránek omezuje uživatelův přístup ke specifickým částem souborového systému. Tento adresář bývá typický nazýván *web content root* neboli webový kořenový adresář. Obsahem adresáře jsou soubory určené pro uživatele, jeho přístup k nim a soubory pro funkcionalitu webové aplikace. Jako základní nástroj útoku je používán souhrn, sekvence speciálních znaků ke změně lokace zdroje zadané v URL ("../"). I když většina populárních web serverů zabraňuje tuto techniku pro průnik mimo adresář aplikace, alternativní kódování sekvence znaků může obejít bezpečností filtry. Tyto metody obsahují validní a nevalidní Unicode kódování pro znak lomítka ("..%2216" nebo "..%c0%a"), zpětné lomítko znaky ("..\") na serverech se systémem Windows. URL zakódované znaky ("% 2e% 2e% 2F"), a kódování dvojité URL ("..% 255c") obráceného lomítka (The Web Application Security Consortium, 2010).

I přes omezení pokusu útoku tohoto typu může být aplikace stále zranitelná z důvodu špatného ovládání vstupu. To je častý problém webových aplikací, které používají šablonový mechanismus nebo nahrávají neomezenou délku textů ze souborů. V různých variantách útoku je originální URL parametr hodnota nahrazena názvem a cesty k souboru jednoho z dynamických skriptů. Následně může výsledek prozradit zdrojový kód, protože soubor je inpretován jako test místo spustitelného skriptu. Tyto techniky často používají dodatečné speciální znaky, jako je tečka (.) k odhalení vypsaní současného pracujícího adresáře, nebo "%00" NULL znaky k prolomení jednoduché kontroly přípony souborů. Příklady pro tento typ útoku proti webovému serveru:

```
Příklad útoku pro webovou stránku

http://priklad/../../../../../etc/passwd
http://priklad/..\%255c..\%255c..\%255cboot.ini
http://priklad/..\%u2216..\%u2216nejakydalsi/soubor

Příklad útoku pro webovou aplikaci

Původní: http://priklad/foo.cgi?home=index.htm
Útok: http://priklad/foo.cgi?home=foo.cgi
```

Výsledkem příkladu je to, že webová aplikace odhalí zdrojový kód souboru foo.cgi, protože hodnota domovské adresáře byla použita jako obsah. Útočník nemusí zadávat žádné invalidní znaky nebo znaky pro průchozí cestu, aby byl tento

útok úspěšný. Následuje útok toho typu pro webovou aplikaci za použití speciální sekvence znaků.

```
Původní: http://example/scripts/foo.cgi?page=menu.txt
Útok: http://example/scripts/foo.cgi?page=../scripts/foo.cgi\%00txt
```

Jak bylo předchozím příkladu řečeno, webová aplikace zobrazí zdrojový kód souboru foo.cgi. Sekvence znaků "../" nebo "%00" byla použita pro přesun o jeden adresář nahoru a vstoupit do adresáře skriptů (scripts). Tyto znaky byly použity prolomení kontroly přípony souboru. Při útoku může být použit i soubor s příponou zip, který obsahuje soubory se znaky pro útok **Directory Traversal**.

AJAX Bezpečnost

AJAX (asynchronní JavaScript a XML) technologie se stala ihned populární, protože mění obsah svých stránek bez nutnosti jejich kompletního znovu načítání. Za pomoci asynchronního zpracování mění obsah webový stránek a nepotřebuje aktualizovat stránku pro změnu obsahu. Na rozdíl od klasických webových aplikaci poskytuje uživatelsky příjemnější prostředí, protože nevyžaduje použití nejnovějších verzí webových prohlížečů. Díky AJAXu se sníží zátěž na webové servery a na síť obecně. Není potřeba sestavit při každém požadavku celý HTML dokument, ale pouze provedené změny. Výrazně nižší množství vyměňovaných dat může mít příznivý vliv na zátěž databázových serverů či dalších backendových systémů. Komunikace mezi klientem a serverem probíhá přes Javascript a data jsou přesouvána ve formátu XML, XSLT nebo JSON. Příkladem použití AJAXu může být online e-mailový klient od Googlu Gmail, uživatel nemusí otevřit nové okno, aby si přečetl nový e-mail. AJAX se zda být perfektní technologií pro webové služby, ale má svou bezpečnostní zranitelnost. Zvýšená interaktivita XML a generovaní HTML znamená zvýšení síťového provozu, což vede k zveřejnění backend aplikací, které nemusí být dostatečně chráněny. Tím je myšleno, aby nedávali neověřeným uživatelům možnost manipulovat s jejich konfigurací, přestože by uživatel neměl zasahovat do serverové části. Jelikož XML, HTTP požadavky používají stejný protokol jako všechny ostatní na webu (HTTP/S), jsou AJAX web aplikace stejně zranitelné stejnými metodami jako normální aplikace.

AJAX využívá Javascript k zachycení uživatelova příkazu a jejich následné přeměny na funkce. Ty jsou pak zaslány jako viditelný text k serveru a mohou být jednoduše objeveny v databázovém poli jako validní data, se kterými útočník může manipulovat. Tyto informace může útočník využít k vytváření specifických HTTP požadavků přímo na server. Při použití XSS může využít infekční skript s AJAX funkcionalitou, aby oklamal uživatele za cílem přesměrovaní na zcela jinou doménu (phising), nebo jeho monitorovaní. Vždy se jedná o taktiku, "nasbírej co nejvíce dat", které později můžeš použít proti své oběti (Acunetix Company, 2016).

Exsploit

Do výbavy útočníků patří části kódu, kterým se anglicky říká *exploits*. Tyto nástroje mají proniknout neboli násilně vniknout do cíle, na který se zaměřily. Může se jednat o různé útočné metody, z nichž některé cílí na webová rozhraní. Častou jsou označovány jako*zero day* nebo *0day*, zůstávají po nějakou dobu skryté v ilegalitě, ale nakonec se stanou veřejnými a vystaví se na diskuzních fórech nebo webových stránkách, aby je mohl použít kdokoliv. Takovéto nástroje pak mohou použít nezpůsobilí hackeři (black hats) pro vlastní účely a způsobilí hackeři (white hats), kteří použijí nástroje pro odhalení chyb. Nalezení různých exploitů je přitom jednoduché. Stačí například použít k jejich vyhledání Google (Long, 2005).

2.3 Analýza stavu zabezpečení serverů MENDELU

Při konzultacích s Ústavem Informačních Technologií (ÚIT) PEF MENDELU byly zjištěny informace o aktuálním zabezpečení vybraného serveru MENDELU. Kromě důležitých informací zaměřených pro bezpečnost byly zjištěny systémové údaje o stavu serveru. *Po dohodě s ÚIT nebudou tyto informace zveřejněny z bezpečnostních důvodů.*

2.4 Požadavky Ústavu Informačních Technologií MENDELU

Funkční požadavky

- Rozpoznání příchozích webových útoků
- Ukládání informací o příchozích útocích
- Webová aplikace pro zobrazení informačních dat
- Zasílání jednotýdenního reportu e-mailem
- Možná rozšířitelnost aplikace

Nefunkční požadavky

- Bezpečné uložení dat
- Aplikace nesmí zasahovat do soukromí uživatelů
- Zabezpečit webovou aplikaci
- Dodání návodu pro nasazení aplikace na server

2.5 Výběr existujcích služeb a nástrojů pro ochranu

Následující výpis zkoumaných produktů je zaměřenen na důležité i zajimavé nástroje pro zabepečení. Důraz je kladen na pozitivní a negativní prvky. V drtivé většině

případů se jedná o online nástroje, jako jsou *skenery* pro odhalení bezpečnostních děr v kódu, anebo se jedná o nástroje pro kontrolu nastavení firewallu. Informace týkající se všech kandidátů jsou aktuální a ověřené minimálně z jednoho oficiálního zdroje.

Pentest-Tools.com

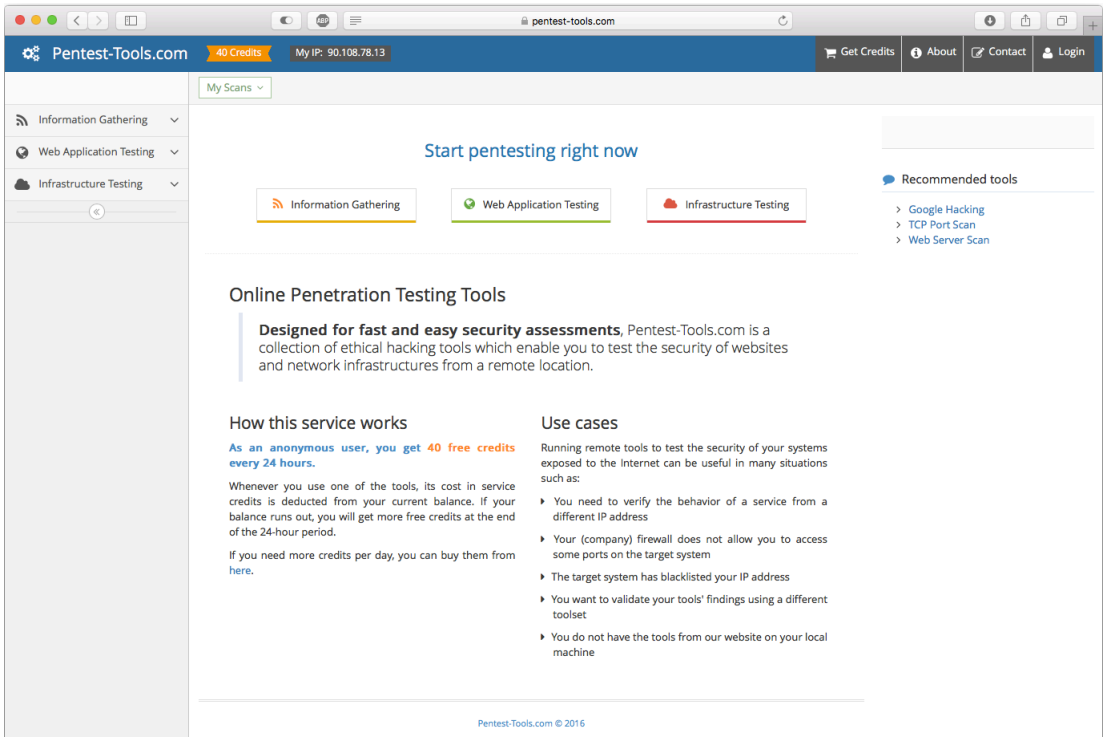
Pentest-Tools (dále jen PT) je služba, která se soustředí pouze na online nástroje. Tyto nástroje se dělí na tři různé kategorie. Tou první je získání veškerých informací za pomoci **Google Hacks**, kde by ne veškeré informace měly být veřejně dostupné. Testovaní hosta, pokud je dispozici skrze ICMP nebo Whoip protokolu. Druhým je webové aplikační testování. Pro odhalení slabých míst používá nástroj *Nikto2* (CIRT.net, 2016). Kromě toho obsahuje i nástroj *URL Fazzer*, který může být použit pro vyhledaní skrytých souborů nebo adresářů umístěných na serveru pomocí *fuzzing* metody.

Třetí kategorie PT služeb používá nástroje soustředící se na testovaní infrastruktury. Pro ty používá nástroje jako TPC, UPD skeny nebo SSL Heathbread scanner, který zkoumá zranitelnost proti **SSL Heartbleed**. Trochu rozdílným nástrojem na skenování může být GHOST Wordpress sken, jehož funkčnost se soustředí na známa zranitelná místa wordpressu. Dalším nástrojem je **Bash ShellShock**, který kontroluje možnost spustit *bash* příkazy v URL, pokud server funguje pod operačním systémem Linux (Heartbleed Blog, 2016).

Služba funguje na kredity, jednotlivý anonymní uživatel získá každých 24 hodin zdarma pouze 40 kreditů, protože každý nástroj má určitou cenu v kreditech. Některé jsou zdarma, některé stojí 30 kreditů, aby je mohl uživatel bezplatně použít jen jednou za den. Pak tu jsou nástroje, které jsou cenově nad 40 kreditů, jsou k získání pouze po doplacení dalších kreditů. Samozřejmě se jedná se o ty nejlepší nástroje, které PT nabízí. Jde tedy o případ, kdy se celkově služba jeví na první pohled zdarma, ale veškeré důležité a užitečné nástroje jsou zpoplatněny. Výhodu oproti dalším službám na trhu má v tom, že není opravdu třeba nic instalovat, protože veškeré nástroje jsou k dispozici online, některé testy probíhají skrze uživatelův prohlížeč.

Nikto2

Jedná se o open source web server skener, který testuje webová rozhraní proti zranitelnosti, nebo zkoumá zastaralé verze serverů. Dále disponuje testováním přes 6.700 potencionálně nebezpečných program a souborů. Dále kontroluje konfiguraci serveru pro výskyt více jak jeden **index** soubor, HTTP server možnosti a snaží se identifikovat nainstalovaný software. Mimo jiné skenuje prvky a pluginy, které jsou v nedávné době nainstalované nebo aktualizované. Nikto2 je navržen tak, aby v co nejkratší době i za použití vyššího výkonu provedl veškeré testy.



Obrázek 2: Náhled webového portálu Pentesting-Tools.com.

Acunetix

Acunetix je mezinárodní společnost soustředící se předně na webovou bezpečnost. Nabízí širokou škálu nástrojů pro ochranu, pro kontrolu webu, kontroly celé sítě a pro slabá místa. Další z výhod jsou automatické nástroje pro report, kde si klient může nastavit, jak má být kontaktován. Acunetix sice poskytuje svůj skener pro detekci zranitelnosti zdarma, ale ten je velice omezen a slouží pouze pro demonstraci kvality nástrojů. Pokud uživatel chce kvalitnější skeny své aplikace, musí si připlatit. Mezi jeho klienty patří společnosti AVG, Sony nebo NASA, díky čemuž si nechává za své služby dobře zaplatit. Jeden kompletní test stojí 295 € (konzultace a deset skenerů stoji 9.995 € pro porovnání). Dále disponuje nástroji proti XSS, SQL injection, WordPress bezpečnosti, firewalls, SSL a zajištění sítě proti prolomení webové obrany (Acunetix Company, 2016).

Acunetix WVS je nástrojem automatického rozpoznání aplikačního firewallu a nástrojem pro správné nastavení maximální ochrany. Upravitelný skener využívá souborové filtry. Součástí tohoto nástroje jsou předem nastavené skenery, které se mohou podle vytíženosti serveru automaticky přerušit. Výpis hlavních testovací nástrojů, kterými disponuje:

- Cross-site Scripting
- SQL Injection

- DOM-based XSS
- CSRF útoky
- Directory Traversal
- prolomení CAPTCHA
- kontrola přihlašovacích formulářů.

V centru DeepScanu je plně automatizovaný webový prohlížeč, který zvládá pracovat se složitějšími technologiemi, jako jsou AJAX, SOAP/WSDL, SOAP/WCF, REST/WADL, XML, JSON, Google Web Toolkit (GWT) a CRUD operace, které by zvládl jakýkoliv prohlížeč. Tohle dovoluje Acunetix Vulnerability Scanner testovat webovou aplikaci přesně, jako by běžela uvnitř uživatelova prohlížeče, což dovoluje skeneru bezproblémově reagovat se složitějším ovládáním, které by uživatel udělal. Automatické skenery využívají integrovaný editor k exportování HTTP požadavků. Nechybí ani podpora analýzy webových aplikací vyvíjených v Ruby on Rails nebo Java Frameworks včetně Java Server Faces (JSF). Acunetix je k dispozici jako Software ke stažení nebo i jako online nástroj.

OWASP

Open Web Application Security Project (OWASP) je celosvětový *not-for-profit* (nekomerční) organizace zaměřená na zlepšovaní bezpečnosti softwaru. Jejich cílem je zviditelnění softwarové bezpečnosti, aby byli jednotlivci a organizace informováni o reálném celosvětovém nebezpečí. Vývoj první verze začal v USA. Dnes je tento open souce částečně i vyvíjen i v České republice, včetně každoročního setkání odborníků v Praze, které je zcela otevřené a zaměřené na bezpečnost a testovaní zranitelnosti (Owasp Organization, 2016).

Samurai Web Testing Framework

Samurai Web Testing Framework je virtuální prostředí podporované na virtuálních strojích, jako jsou VirtualBox nebo VMWare. Stáhnutelné obrazy jsou předem nakonfigurovány, aby fungovaly jako web pen-testing prostředí. K dispozici jsou pouze jako open source nástroje pro testování nebo jako útoky na webové stránky. Veškeré nástroje jsou podle vývojařů plně funkční a předem otestované. Samurai WTF je oproti ostatním svým způsobem unikát, přesto i on má problémy s komunikační podporou. Samurai byl vyvíjen jako nástroj pro aplikační testery, které hledají častá zrani- telná místa, jako jsou špatné konfigurace, XSS zranitelnost, SQLi anebo přípony souborů a další časté chyby v systémech. Dále je součástí prostředí několik nástrojů pro průzkum webových aplikací a serverů, souborů, adresářů nebo testovacích skriptů (MadIrish.net, 2016).

Kali Linux

Kali Linux, dálé zkratkou KL, je Linuxová distribuce podobná Debianu, která je cílená na pokročilé penestrační testování a kontroly bezpečnosti. KL obsahuje několik stovek nástrojů, které slouží k získání několik informacím, jako jsou penestrační testy, bezpečnostní výzkumy, forenzní nástroje a další. KL byl vytvořen, aby poskytl odborníkům na bezpečnost nástroj, kterým mohou bezpečnost kontrolovat. Některé z mnoha z nástrojů se soustředí na webovou bezpečnost, a proto je KL zmíněn. Nevýhodou těchto nástrojů může být nutná znalost unixových systémů a ovládatelnost příkazového řádku, protože většina nástrojů nemá grafické prostředí. KL je velice populárním nástrojem pro bezpečnost a má obrovskou podporu veřejnosti i firem soustředících se na bezpečnost. Jedná se o open source a jeho zdrojový kód se nachazí na stránkách společnosti Github (Kali Linux, 2016).

Obrázek 3: Použití nástroje sqlmap pro získání citlivých údájů z databáze (Picateshackz Company,2016).

2.6 Finální porovnání

Na trhu existuje mnoho nástrojů soustředících se na bezpečnost webových aplikací. Každá má rozdílný postup, jak docílit většího zabezpečení serveru proti webovým útokům nebo nalezení zranitelných míst. Každý framework, program nebo sada služeb mají své výhody, které je mohou upřednostnit před konkurencí. Oblast bezpečnosti je obrovská, a proto je možností přistoupit k řešení daného problému jinak. Takovéto řešení bude vytvořeno zdarma a na míru podle daných specifikací, což má své výhody. Pokud se podíváme, kolik dnes stojí bezpečností nástroje na zakázku, tak musíme očekávat cenu v tisících korunách. Na závěr kapitoly je třeba poznamenat, že žádný z nalezených produktů na ochranu nevyhovuje a nejlepší variantou je vytvoření zcela nového programu na ochranu serveru před útoky.

3 Návrh aplikace

Návrh celé aplikace je zaměřen na nasazení na určitý server MENDELU. Dále musí splnit požadavky Ústavu informačních technologií MENDELU. Návrh aplikace musí být jednoduchý. Pokud takový návrh bude jednoduchý, bude také zprostředkovatelný, z čehož plyne budoucí použitelnost aplikace.

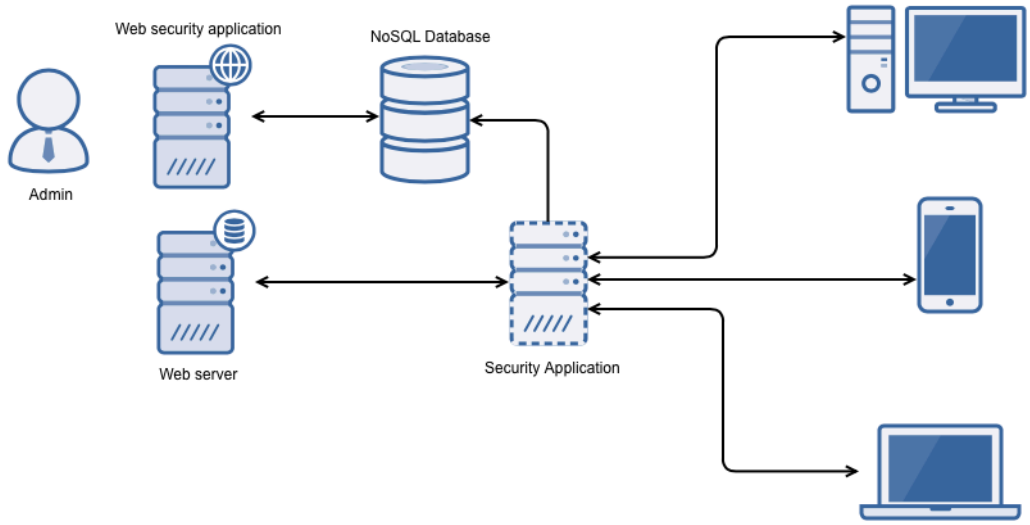
3.1 Analýza problému a návrh řešení

Ochrana se soustředí na webovou aplikaci, která se nachází na určitém serveru. Tato webová aplikace a stránky jsou k k dispozici pro všechny uživatele pod portem 80 a 443. Pokud bude webový útok cílen na daný server, musí projít těmito porty. Navržené řešení se soustředí na kontrolu právě těchto portů. Na jedné straně se nachází server obsahující citlivá data a na druhé straně uživatelé, mezi kterými se může skrývat útočník. Toto řešení má i výhodu, že nijak nezasahuje do dat uživatelů, která jsou již uložena na serveru. To jednak splňuje podmínku zadaní a dále nese i menší bezpečností riziko, aplikace není zneužitelná. Pokud bude nachystána síť pro webové útoky, je se třeba zaměřit na to, jak budou tyto útoky odchytávány. Je si třeba položit otázku, zda je třeba útok ihned zastavit. Pokud by toto platilo, mohlo by dojít k tomu, že aplikace bude bránit přístupu uživatelů, kteří nikterak neútočí. Návrhem je aplikace, která bude pouze kontrolovat vstupy, které bude předávat serveru. Pokud daný vstup rozpozná jako potencionální útok, pokusí se uložit o daném uživateli co nejvíce informací. Takové informace mohou například být IP adresa, typ rozpoznaného útoku atd. Tato data je třeba ukládat do databáze. Tato databáze musí být zabezpečena a umístěna na samém serveru pro menší riziko. Komunikace s takovouto databází by měla být šifrována, pokud se majitel programu rozhodne umístit databází jinak než lokálně. Je potřeba nasbíraná data zobrazit za pomoci webové aplikace, která bude tato data zobrazovat. Vhodné je použití různých grafů pro lepší vykreslení dat. Aplikace musí být úsporná a musí se soustředit se hlavně na propojení komunikace mezi uživatelem a serverem.

3.2 Komunikace mezi klientem a serverem

Aplikace bude poslouchat na zvolených portech. To znamená, že se k datům, která tudy budou proudit, musí dostat dříve než samotný server. To implikuje, že aplikace bude přijímat data od uživatelů, které si přečtou. Musí je přeposlat dál serveru. Server odpoví požadavku uživatele, ale tato data pošle aplikaci, s níž komunikuje místo uživatele. Ta data přepošle dále uživateli (více podrobností – schéma na obrázku číslo 4).

Největší důraz musí být kladen na to, aby byla aplikace neustále k dispozici. Pokud dojde k chybě na straně aplikace, není možné, aby mohl uživatel dále komunikovat se serverem. Dále by bylo vhodné veškeré údaje zapisovat a v případě



Obrázek 4: Schéma cele síťové komunikace.

problému je řešit. Nejdůležitějším krokem je, aby byla aplikace implementována pro zpracovaní více uživatelů naráz a aby uživatelé nebyli omezeni.

3.3 Rozpoznání jednotlivých útoků

Pokud budou veškerá vstupní data kontrolovaná, musí být i případný útok rozpoznatelný. Rozpoznatelný tak, aby se jednalo o co nejpřesnější výsledek a také, aby těchto testu bylo co nejvíce, což zaručí největší pravděpodobnost odhalení, zda se jedná o útok. Řešením tohoto problému je, že bude existovat adresář pravidel. Tento adresář bude obsahovat pravidla, pomocí kterých se budou vstupní data kontrolovat. Seznam pravidel bude otevřený majiteli serveru, který bude moci dále tato pravidla rozšiřovat, nebo upravovat ta stávající.

3.4 Ukládaní potecionálních útoků

Pro ukládaní všech potencionálních útoků je třeba databáze. Tato databáze nebude přímo součásti aplikace, proto bude na uživateli aplikace, aby si ji obstaral. Aplikace se k ní bude pouze připojovat a zapisovat do ní, nebo ji číst.

3.5 Zobrazení statistických dat

Uživateli bezpečnostní aplikace musí být zobrazeno co nejvíce potřebných dat, která budou dobře rozeznatelná a přehledně zobrazena. K tomu je zapotřebí několik grafů pro různé statistiky a analýzy podle jejich rozsahu. Aby toto zobrazení bylo ještě přehlednější, je zapotřebí interaktivních grafů. Díky interaktivnímu zobrazení si sám uživatel může zobrazené statistiky ihned upravit.Samotná webová aplikace zároveň musí být sama zabezpečena, aby zobrazila data pouze autorizované osobě.

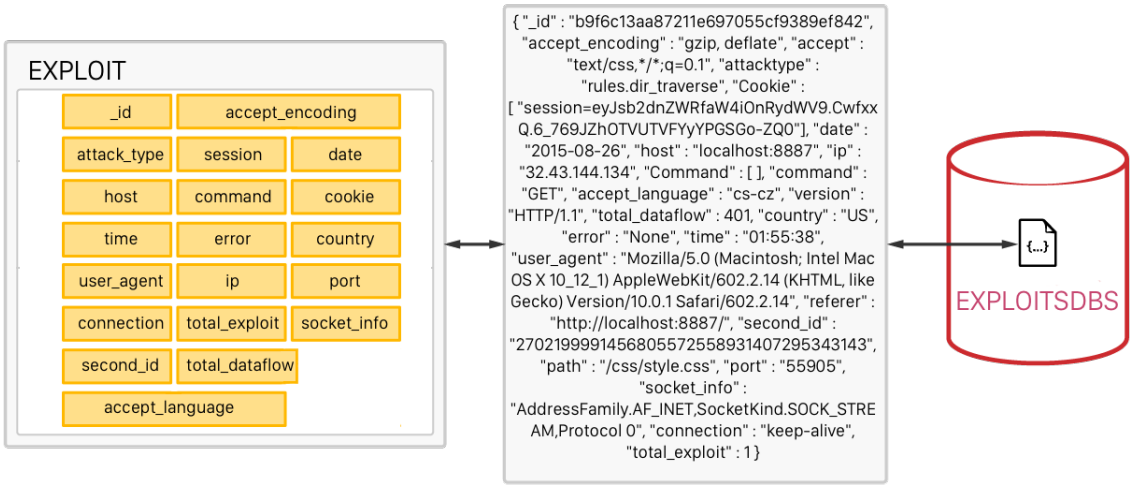
4 Tvorba aplikace

Navrženou aplikaci je třeba podle předem daného návrhu implementovat. Popis je zaměřen zejména na implementaci včetně použitých technologií jako například ukládaní dat. Důraz je kladen také na výhody, které zvolené technologie mají, zejména jejich význam a důvod jejich použití při tvorbě aplikace.

4.1 Ukládaní dat

Dnes se pracuje s velkým objemem dat. Ať už se jedná o obchodní data nebo data uživatelů, díky moderním technologiím (cloud, mobily, sociální média a big data) přibývají. Dat, se kterými firmy operují a která musí zpracovat, je velmi mnoho. Z toho důvodu bylo třeba pracovat s daty ve velkém množství. Ukládat je a dále s nimi pracovat v co nejkratším čase bez jakéhokoliv problému. Relační databáze, dnes nejpoužívanější databázový systém, mají své výhody zvláště v přehlednosti. Pokud se ale soustředíme na samotná Big Data, pak je vhodné použít NoSQL. Ty fungují na jednoduchém principu, kde každá hodnota má vlastní klíč. Kromě velkého množství dat lze aplikovat NoSQL databázi na vstupní hodnoty, která se může často měnit podle počtu vstupních parametrů. Tady lze aplikovat dynamická schémata pro změnu podle požadavku, se kterými nemá NoSQL na rozdíl od relačních databází problém. Ty jsou fixované a předem definované a pro ukládání dalších dat je třeba lidského zásahu pro změnu schématu. To může zpomalit proces vývoje, nejen z důvodu manuální operace, ale i proto, že může mít vliv na další aplikace nebo služby. V porovnání NoSQL databáze plně podporují agilní vývoj, díky jeho nepotřebě definovaní schématu, jak jsou data modelována. Datový model je definován aplikačním modelem v podobě objektu. Data jsou čtena a zapisována ve formátu JSON, který je de facto standardem produkci dat pro web, mobily nebo IoT aplikace. Další výhodou je eliminace přetížení ORM frameworků a zjednodušení aplikačního vývoje možností zápisu a čtení objektů bez potřeby je rozdělovat (MongoDB Inc., 2016). Jeden objekt může číst nebo zapisovat jeden document, jako v je to v aplikaci (viz obrázek č. 5).

V porovnání s relační technologií, distribuční NoSQL databáze dělí data na několik oddílu a distribuuje je do několik instancí, které nemají sdílené zdroje. V přidání mohou být data replikována do jedné nebo více instancí pro velkou dostup-nost. Zatímco relační databáze jako Oracle vyžadují zvláštní software pro replikaci, NoSQl databázi ji nevyžadují. Tato technologie využitelná v tomto případě pro ukládaní dat pro útoky ze dvoud důvodů. HTTP požadavky se mohou měnit dynamicky a počet pravidel, který by je měl kontrolovat, bude přibíhat, z toho důvodu se hodí dynamická obměna dat. Dále si je třeba uvědomit, že v budoucnu bude dat přibívat, lze předpokládat nárůst útoků. Pokud by měly být aplikace používány i další roky, je třeba počítat s velkou sumou dat, i proto se zde hodí aplikovat NoSQL databázi (Couchbase, 2016). Segment NoSQL databází v současnosti významně roste a prospívá především v oblasti big data a real-time webu.



Obrázek 5: Příklad zapisu jednoho záznamu, tento obrázek také symbolizuje model databáze.

MongoDB

MongoDB se řadí mezi NoSQL databáze a patří zároveň mezi ty nejpopulárnější. Díky její popularitě je k nalezení vedlejších knihoven pro podporu dalších aplikaci jako Django nebo právě použitý framework Flask. MongoDB místo tradičních relačních databází využívajících tabulky používá dokumenty podobné formátu JSON a dynamické databázové schéma, které umožňuje vytváření a integraci dat pro aplikace. Tento postup může cílit k rychlé a jednoduché práci s daty, zvláště pak při velkkém množství dat (MongoDB Inc., 2009).

Navržený projekt NoSQL databáze firmou MongoDB Inc. je od roku 2009 open-source (MongoDB Inc., 2016). Díky své popularitě je toto backendové řešení součástí mnoha velkých projektů, jako jsou webové aplikace a stránky. Příkladem jsou známé firmy a společnosti Craiglist, eBay, Foursquare, SourceForge, Viacom a New York Times. MongoDB je aktuálně nejpopulárnější NoSQL databázový systém. Je k dispozici zdarma pod GNU Affero General Public License a jazykové ovladače jsou dostupné pod Apache Licence (MongoDB Inc., 2016).

4.2 Programovací nástroje použité k tvorbě

Python3

Python 3 je nová verze jazyka Python. Aktuální verze tohoto jazyka je 3.5.3, tato verze byla použita pro implementaci aplikace. Důležitým poznatkem je jeho nekompatibilita s Pythonem 2.x. Při pokusu aplikovat syntaxi verze Pythonu 2.x nebude možné úspěšně spustit na interpretru Pythonu 3. Verze 3.x je více méně stejná, ale v detailech, obzvláště built-in objektech jako jsou slovníky nebo datový typ string (podpora utf-8), je rozdílná. Změnilo se mnoho funkcí, z nichž některé již nadále nejsou podporovány a z verze 3.x byly natrvalo odstraněny. Dále byl vylepšen Glo-

bral Interpreter Lock, takže při použití *multiprocessing* knihoven je rychlejší než verze 2.x. Také standardní knihovna byla přeorganizována v několika klíčových místech. Snad jedinou nevýhodou verze 3.x vůči 2.x je menší podpora knihoven, kterou verze 2.x má. Těchto knihoven není tolik, jejich počet se neustále snižuje. I přes tyto výhody, které má Python 3, je zvláštní, jak je verze 2.x stále více využívána, a to i oproti modernější verzi 3.x. Python 3 je odkaz tohoto interpretovaného jazyka, který bude v budoucnu nadále používán, dokud nebude nahrazen zcela jiným jazykem, což se nemusí stát v nejbližším desetiletí (Python Software Foundation, 2008).

Javascript, HTML a CSS

Za zmínku stojí, že i tyto jazyky, jako je **Javascript**, Turingovsky kompletní jazyk **CSS** a tágovací jazyk **HTML**, byly použity při vývoji webové aplikace. Stejně tak jako byl použit standard HTML5 a CSS3, byla použita i CSS knihovna *bootstrap*.

4.3 Webová aplikace

Flask

Jedná se o framework, kterému se někdy přezdívá mikroframework, nevyžaduje speciální nástroje ani knihovny. Neobsahuje žádnou databázovou aplikační vrstvu nebo žádnou další komponentu, kterou poskytují předem nainstalované knihovny třetích stran, jak je to běžné u frameworků. Podporována jsou rozšíření, která se mohou přidat k aplikačním funkcím, jako kdyby byly implementovány v samotném jádru Flasku. Rozšíření existují pro objektově relační mapery, validace forem, manipulace s nahráváním, mnoho veřejných ověřovacích technologií a několik běžných nástrojů. Tato rozšíření jsou aktualizována daleko častěji něž jádro samotného frameworku, proto mohou být přívětivější pro vývoj aplikací. Aktuální stabilní verze je 0.11 vydaná v květnu 2016, autoři ho stále považují za nehotový. I přes nižší známost Flasku existuje mnoho webových aplikací, které jsou tvořeny právě Flaskem. Následuje jednoduchá ukázka použití knihovny Flask v Pythonu 3 (Flask Organization, 2016).

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def home():
    if not session.get('logged_in'):
        return render_template('login.html')
    else:
        return render_template('index.html')

def login():
    if request.form['password'] == 'password'
     and request.form['username'] == 'admin':
        session['logged_in'] = True
    else:
        flash('Wrong password!')
    return home()

@app.route("/logout")
def logout():
    session['logged_in'] = False
    return home()


if __name__ == "__main__":
    app.secret_key = os.urandom(12)
    app.run(host='localhost',port=5000,debug=True)
```

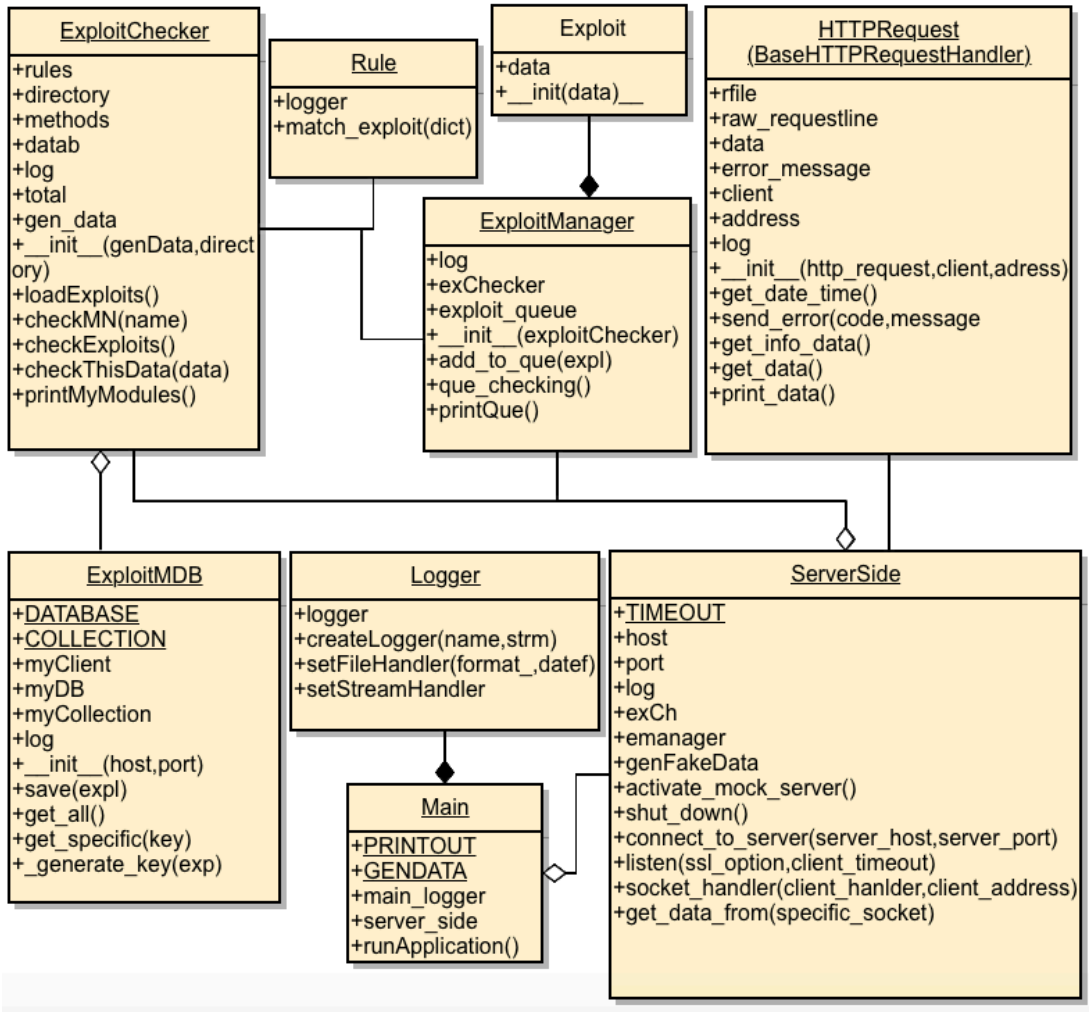
Flask je vytvořen v Pythonu, jedná se o opensource. Kromě jeho podpory Pythonu 3, nemá problém spolupracovat s databzí MongoDB a přizpůsobit se dynamické struktuře databáze. Tento mikroframework je neustále ve vývoji a autoři slibuji další funkce v budoucnu, zároveň chtějí zachovat jeho jednoduchost.

D3js

D3js je využita hlavně při vývoj menších aplikací pro zobrazení dat v podobě interaktivních grafů různé podoby. D3 umožňuje shluknout libovolná data, poté na ně aplikovat datově řízené transformace k zápisu do dokumentu. **D3** lze například použít ke generování HTML tabulky pro pole čísel. Stejná data lze využít ke tvorbě interaktivních SVG pruhů s přechody a specifickou interakcí. Efektivně manipuluje daty a dokumenty na základě údajů ke správné prezentaci dat výhodnou flexibilitou. Vystihuje všechny funkce webových standardů, jako je HTML, SVG a CSS. S nízkou

náročností na výkon je D3 je velmi rychlý a podporuje velké soubory dat nebo dynamické chování pro interakci s animací.

4.4 Implementace

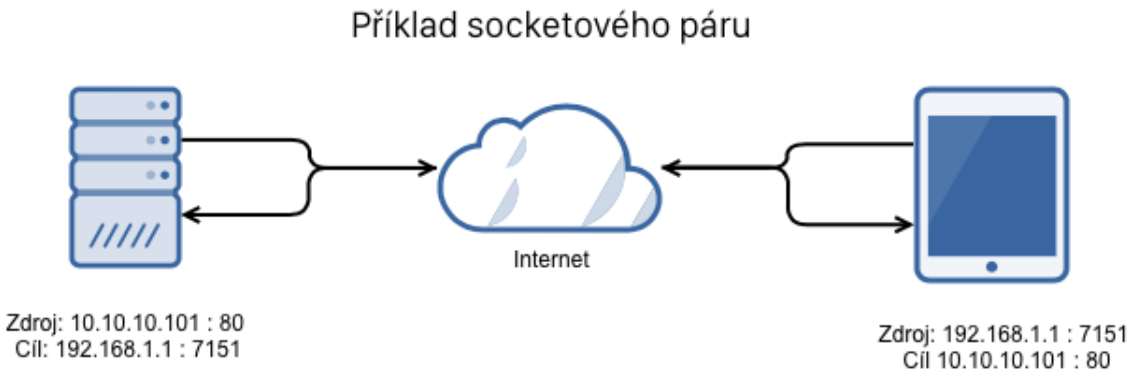


Obrázek 6: Uml Class Diagram celé aplikace.

Propojení serveru

Důraz je kladen na propojení klienta se serverem, které je důležité pro aplikaci. Zároveň probíhá čtení jednotlivých HTTP požadavků, z nichž mohou být některé uloženy. Aby aplikace fungovala, musí poslouchat na portu 80 a 443, na který jsou očekávány příchozí požadavky. V momentě, kdy se klient připojí, vytvoří se socket (navázaní spojení přes určitou IP adresu a port). Po úspěšném propojení klienta

s aplikací je třeba propojit je se serverem. Toto propojení není přímé, jelikož požadavky, které jsou přijaty, jsou přeposlané serveru a jeho odpověď je nejdříve poslaná aplikaci a následně přeposlána klientovi. Je tedy třeba vytvořit ihned další socket pro propojení aplikace a serveru. Celá tato operace musí zabrat maximálně sekundu, aby uživatel nepocítil znatelné zpomalení. Toto propojení funguje následovně: Při inicializaci se vytvoří instance *Server_side*, která bere vstupní parametry host adresu, port, na kterém má přijímat požadavky, a exploit manažera (viz podkapitola Kontrola vstupních dat). V momentě zavolání funkce *activate_mock_server* se vytvoří socket pro klienta s parametry *SOL_SOCEKT* a *SO_REUSEADDR*. Ty zajistí, aby mohl být socket použit pro naslouchání požadavku a aby probíhal dokud nebude požádán o ukončení, nebo násilně ukončen. Bez ukončení výjimkou tento socket začne naslouchat na portu a na adrese, která byla předána v konstruktoru.



Obrázek 7: Příklad zapisu jednoho záznamu.

Dále se pokusí program spustit *mock_server* (falešný server) na záložním portu v případě, že zadaný port byl už zabrán, nebo se ho nepodařilo zabrat (nedostatek práv např.). Pokud vše dosavadně proběhlo bez problému, zavolá se funkce *listen*. Ta po zavolání vytvoří proces pro kontroly zásobníku, který je použit pro zkontrolování jednotlivých požadavků (viz podkapitola Kontrola exploitů). Tento proces je spuštěn na pozadí systému jako deamon. Funkce **listen**, která je zavolána, spustí smyčku *while*, jejíž obsahem je tvorba proměnné *client* s nastavením časového limitu, pokud je třeba SSL. Funkce *socket_handler* vytvoří proces běžící jako daemon, ale běží na jiném vlákně. Pokud se něco pokazí, vyšle klientovi tento proces chybný kód 500, aby ho upozornil, že chyba není na jeho straně spojení. Díky více vláknům je možné zpracovat několik uživatelů zároveň. Pro zjednodušení se tento proces pokusí nejdříve připojit přes socket server a přijmout požadavek od uživatele skrze funkci *get_data_from*. Předtím než je ihned pošle serveru, je tento požadavek parsován (viz podkapitola Parsování HTTP požadavku) a vložen do zásobníku pro kontrolu (kontrola exploitů). Původní požadavek je poslán serveru a odpověď získaná stejnou funkcí *get_data_from*, ale od serveru a poslána klientovi. Po celou tuto operaci musí

být otevřené sockety pro klienta i pro servery, které se navzájem nijak neovlivňují. Funkce, která použije pro získání požadavků, pracuje s bytovými daty a používá buffer, do kterého načítá po velikosti 1024 bajtů, dokud je možno čist vstupní data, jinak je celá suma vrácena skrze funkci. Druhá zmíněná smyčka je ukončena, první stále čeká na nové požadavky klientů, dokud není celá aplikace řádně ukončena. To probíhá zavoláním funkce *shut_down*, která ukončuje frontu, uzavírá hlavní socket pro poslouchání na zvolených portech a ukončuje veškerá neukončená vlákna a ukončuje celou aplikaci.

Parsování HTTP požadavku

Pro zpracování HTTP požadavku je ho třeba parsovat pro získání potřených dat. Jelikož jsou data v byte kódu (byte string formát pro Python), je třeba převést string pro snazší zpracování a pro získání dat ho parsovat. V praxi se pomocí knihovny **BytesIO** čte byte stringu a dale ho knihovna **BaseHTTPRequestHandler** parsuje. Na převedená data se zavolá funkce *get_data*, která převádí vstupní data na datový typ hašovací tabulka, v Pyhtonu se jedná o *dictionary* (slovník). K získaným datům se přídá aktuální čas a datum, informace o socketu nebo IP adresa, port, které jsou tvořeny během parsování a které jsou přidány také do struktury. Navíc podle IP adresy je rozpoznána země původu (knihovna **geolite2**) a uložena ve formátu ISO dané země. Například pokud se jedná IP adresu registrovanou v České republice, bude výsledný string ve formátu CZ. Z dat jsou odstraněny přebytečné mezery pro lepší přehlednost dat.

Ukládaní dat

Ukládaní dat probíhá skrze MongoDB databázi, která je ovládaná knihovno **pymongo**, která je podporována i Pythonem 3. Samotný model si ukládá veřejné proměnné DATABASE a COLLECTION. Jedná se o Python, proto není třeba nastavovat proměnné public ani static. Ty slouží k přístupu určité databáze skrze string *database* (například název atabase – exploitsdbs). V konstruktoru se vytvoří inicializace Mongo klienta s host adresou a portem (nechá výchozí 27017). Vytvoří se proměnné klient a kolekce, které mají za cíl propojit databází s kolekcí. Pro uložení dat persistentně je třeba zavolat funkci *save*. Po zavolání je nejdříve zkontrolován vstupní parametr, zda-li je typu slovník. Pokud ano, vytvoří se primární klíč, což je hexadecimální haš generován pomocí knihovny uuid4. Kromě vytvoření primárního klíče je vytvořen i sekundární klíč, který slouží čistě jako záložní klíč, pokud by nastal problém identity dat skrze primární klíč. Kromě obou klíčů je vytvořen parametr *total_exploit* s hodnotou jedna. Ten je ukládán, aby zvětšil celkový počet uložených potencionálních útoků uložených v databázi. Každý prvek v kolekci má právě hodnotu jedna. Voláním skrze funkci *Mongoclient.insert* se data vloží do zvolené databáze a kolekce. Pokud vše proběhlo bez problému, funkce save vrací True, jinak False. Modul obsahuje další funkce, které nabízí jiné operace s databází, například pro čtení dat z databáze.

Logováni

Každá aplikace potřebuje záznam výstupu programu, aby v případě chyby mohl technik reagovat snadno an specifický problém. Instance loggeru je vytvořena v main.py je všem dalším modulům globálně předávána. Existuje několik druhů zápisu, které se dělí podle toho jak důležitý výstup je. Dělí se na INFO, DEBUG, ERROR a nebo WARNING. Například ERROR bude použit u výpisu kdy se stala závazná chyba, která způsobila pád aplikace. Toto rozdělení pomáhá k přehlednějšímu výpisu. Modul **Logger** obsahuje dvě funkce. Funkce *SetFileHandler* vrací nastavenou instanci souborového handleru s cestou pro soubor (i s názvem), a setStreamHandler s instací *stream handler*, která se stará o výpis veškerého vstupu na standardní výstup.

Každá aplikace potřebuje záznam výstupu programu, aby v případě chyby mohl technik reagovat snadno na specifický problém. Instance loggeru je vytvořena v main.py, je globálně předávána všem dalším modulům. Existuje několik druhů zápisu, které se dělí podle toho, jak důležitý výstup je. Dělí se na INFO, DEBUG, ERROR anebo WARNING. Například ERROR bude použit u výpisu, kdy se stala závazná chyba, která způsobila pád aplikace. Toto rozdělení pomáhá k přehlednějšímu výpisu. Modul **Logger** obsahuje dvě funkce. Funkce *SetFileHandler* vrací nastavenou instanci souborového handleru s cestou pro soubor (i s názvem) a *stream handler* s instací stream handler, která se stará o výpis veškerého vstupu na standardní výstup.

Kontrola pravidel

Volaní jednotlivých pravidel uložených v adresáři **rules** se načítají jednotlivá pravidla, ta jsou aplikována na vstupní data. Prvně je vytvořena instance modulu pro spojení s databází. Načtení pravidel v zadáním adresáři se uskuteční pomocí funkce *loadExploits*. Nejdříve za pomocí regexu profiltruje vstupní soubory v adresáři a povolí jenom soubory s koncovkou **py**. Dále si do nově vytvořené hašovací tabulky ukládá informace o jednotlivém pravidlu jako název, cesta k souboru a modul. Pokud načtení pravidel proběhne v pořádku, jsou pravidla zkontrolována. Funkce *CheckExploits* kontroluje každé načtené pravidlo, zda obsahuje zvolenou metodu a jestli tato metoda vrací True nebo False. Tímto je kontrolováno, jestli funkce alespoň dokáže data hodnotit boolenou hodnotou, zda se jedna o útok. Pokud u kontrolovaného pravidla není splněna jakákoliv podmínka, dané pravidlo není přidáno mezi kontrolovaná pravidla v běhu aplikace.

Kontrola vstupních dat

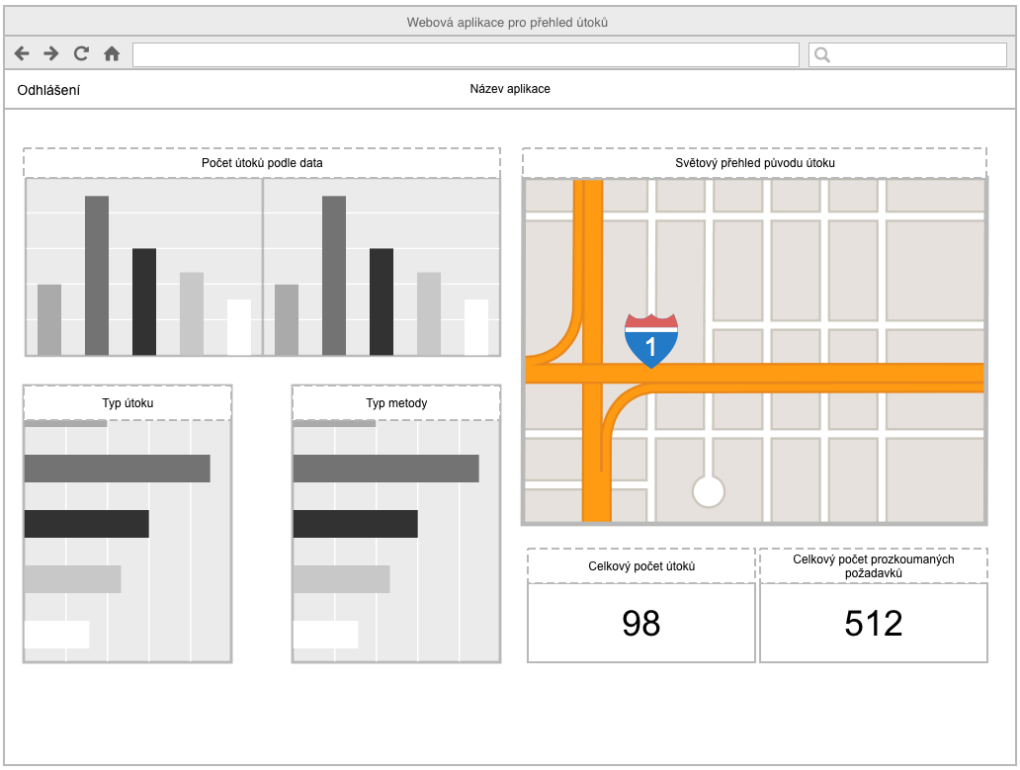
Kontrola vstupních dat začíná v momentě, kdy se v modulu propojení serveru spustí proces jako daemon. Při vzniku se vytvoří prázdný zásobník zásobník pomocí knihovny **multiprocessing**. Jedná se o speciální zásobník, který má funkci ukládat do své datové struktury data tak, aby k nim mohlo mít v určitém momentě

přístup jenom jedno vlákno. Tím se zajistí, aby další vlákna nepřepsala hodnoty, se kterými hlavní vlákno pro kontrolu pracuje. Data, která jsou obsažena v zásobníku, jsou postupně kontrolována. Zásobník funguje na principu – mám data, zkontroluji je, pokud nemám data, pak počkám. Čas, který proces počká, je 0.0130 mili-sekund, poté se znovu zeptá na stav zásobníku. Pokud nebude prázdný, začne proces kontroly nanovo. Tento opakovaný proces značně ušetří celkový výkon serveru.

Webová aplikace

Při tvorbě aplikce byly použity cizí knihovny nebo části kódu, pokud tak bylo učiněno, byl v části kódu dopsán zdroj a licence.

Webová aplikace je tvořena skrze knihovnu **flask**. Jedná se o jednoduchou aplikaci obsahující stránku pro přihlášení zabezpečenou skrze heslo a stránku s obsahem dat v podobě grafů. Hlavní struktura se skládá ze souboru **app.py**. Ten obsahuje veškeré nastavení, co se týče backendu stránky. Jedná se přihlášení, odhlášení, získaní dat z databáze a převedení na formát JSON. Aplikace se dělí strukturně na static a templates adresáře. Static obsahuje veškeré potřebné knihovny pro běh aplikace (CSS,JS). Templates obsahuje klasické html soubory pro zobrazení informací (index.html).



Obrázek 8: Drátěný model návrhu webové aplikace.

Usecase Diagram

Usecase Diagram pro přihlášní a použití aplikace.
Aktoři

- Administrátor

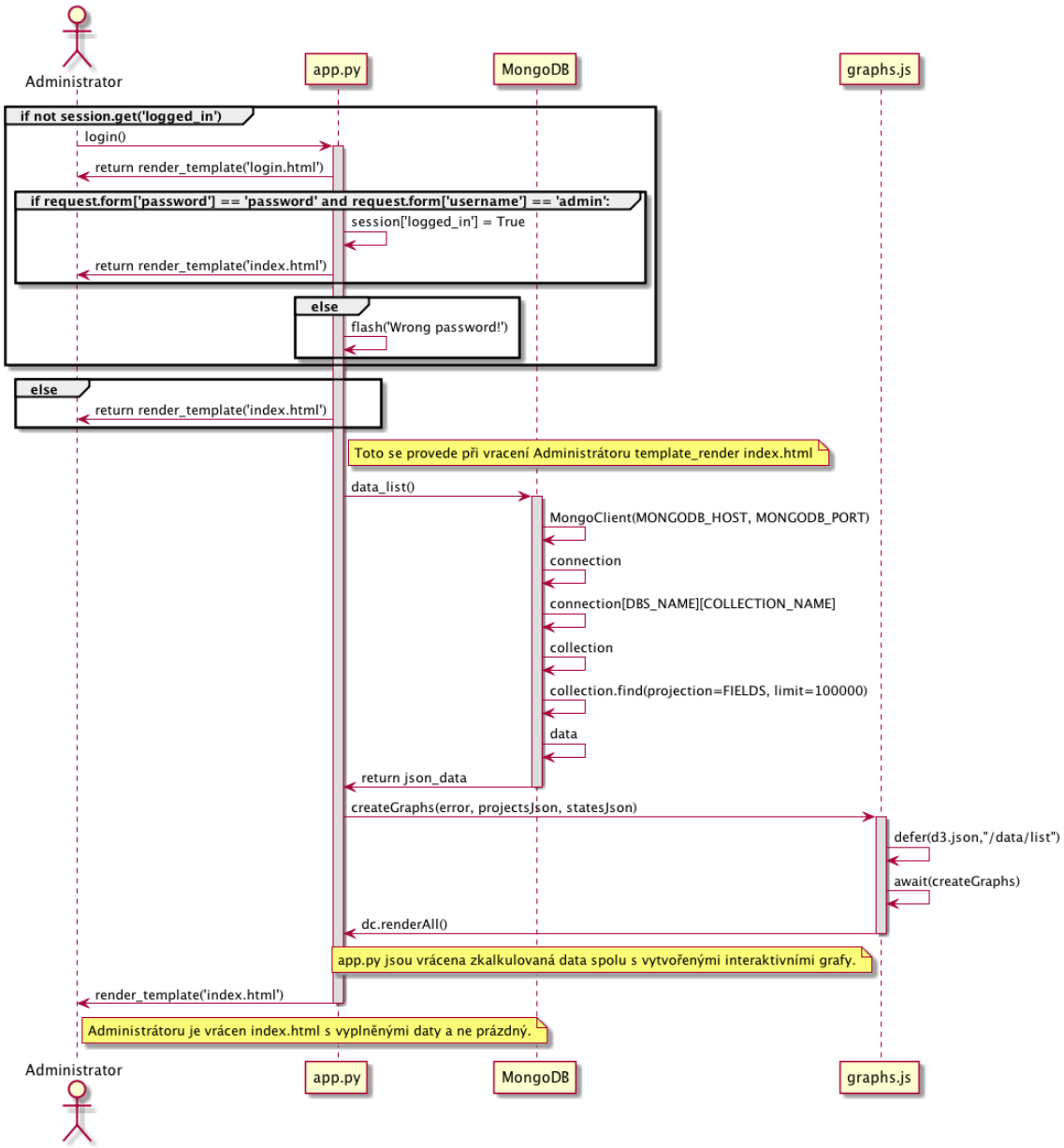
Předpoklad

- Administrátor má heslo a uživatelské jméno pro přihlášení

Scénař

1. Administrátor nastaví url webové aplikace
2. Pokud má Administrátor prázdnou přihlašovací relaci (session), pak systém zobrazí přihlašovací okno.
3. Administrátor vyplní přihlašovací údaje jako je přihlašovací jmeno,heslo a klikne na odeslat.
4. Systém zkontroluje zadané vstupní udaje. Pokud souhlasí s těmi, které systém má, pak Administrátoru se nastaví session na hodnotu True a je přeposlá na hlavní stránku aplikace.
5. (Volitelné) Administrátor zvolí mapu a pomocí skorolovaní myši si přiblíží mapu Evropy.
6. Přes celé okno mapy je zobrazena Evropa.
7. Administrátor přesune myš na Anglii.
8. Systém zobrazí malé okénko přes mapu, které zobrazí údaje o počtu útoků na server pochazejících z Anglie.
9. (Volitelné) Administrátor klikne na graf zobrazjící analýzu dat a zvolí časový úsek přetažením myši z května na duben.
10. Systém upraví vešekerá hodnoty podle zvoleného časového úseku kromě mapy.
11. Administrátor klikne na označenou plochu a celou přetáhné mimo graf.
12. Systém resetuje vešekeré data na původní hodnoty.
13. (Volitelné) Administrátor zvolí metodu GET na grafu metod.
14. Systém upraví veškeré grafy a data a grafy podle zvolené metody kromě mapy.
15. Administrátor klikne na stejný sloupec.
16. Systém resetuje data a grafy na půdvodní hodnotu.

Sekvenční Diagram



Obrázek 9: Sekveční Diagram pro pro přihlášení a spuštění aplikace.

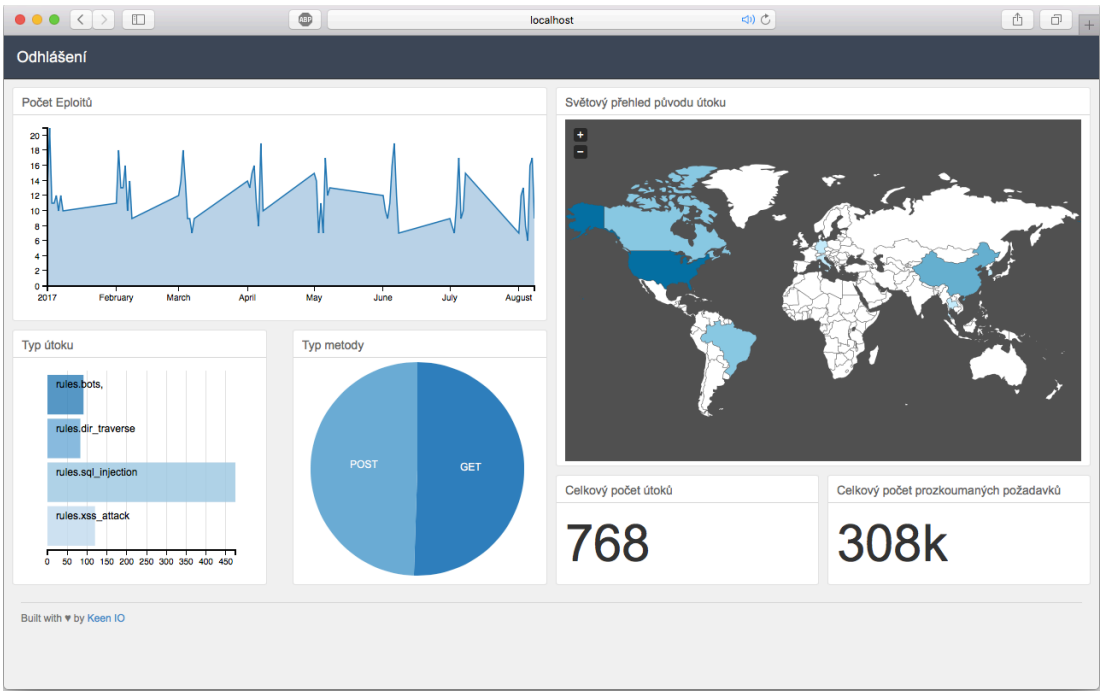
Zobrazení dat

Data, jak již bylo zmíněno, jsou zobrazena pomocí grafů pro zpřehlednění. Kromě klasických sloupcových grafů jsou data zobrazena pomocí vektorové mapy nebo číselných údajů. Na veškeré použité javascriptové knihovny se vztahuje licence GNU 3. Pro použití sloupcových grafů, jejichž výskyt je nejčastější, byla použita knihovna

D3js. Grafy zobrazují data, jako je datum útoku, typ nebo použitá metoda. Pro vykreslení mapy je použita knihovna **vectorMap**. Před tím než jsou grafy vykresleny, musí být načteny z prostředí aplikace. Stačí načíst předem připravená JSON data, která se musí přečíst, některá data se zkalkulují a následně jsou vykreslena. Po každém načtení stránky se proces opakuje, o aktualizaci JSON dat se stará aplikace. Grafy obsahují i možnost interaktivity s myší, uživatel může upravit data podle postupného klikání.

```
$(function(){
  //console.log(mapData)
  $('#world-map-gdp').vectorMap({
    map: 'world_mill',
    series: {
      regions: [{
        values: mapData,
        scale: ['#C8EEFF', '#0071A4'],
        normalizeFunction: 'polynomial'
      }]
    },
    onRegionTipShow: function(e, el, code){
      el.html(el.html()+' (Počet útoku - '+mapData[code]+')');
    }
  });
});
```

Zobrazený kód v Javascriptu ukazuje ukázku použití knihovny vectorMap. Data jsou načtena skrze proměnou gdpData. Po načtení dat se vykresle mapí, jejíž jednotlivé země jsou obarveny podle počtu útoků. Čím více útoků pochází z dané země, tím bude více do tmavě modra. Podporována je interakce s myší pro přiblížení mapy, případně se po najetí ukazatele na určitou zemi ukáže hodnota s obsahem počtu útoků.



Obrázek 10: Ukázka běhu aplikace s grafy.

5 Testování aplikace

Testování je důležitou částí každého vývoje aplikace. Vzhledem k velikosti aplikace jsou i zde obsaženy testy. Podoba testu může být různá, ale jejich princip je stejný. Přehození zvoleného vstupu a očekávaného výstupu, to platí pro testovaní jazyka Python. Kromě úspěšného výstupu se testuje selhání. Dostatečné testovaní může zabránit, aby program obsahoval kritikou chybu, která by vedla k častým pádům aplikace, které tudíž by následně byla nepoužitelná v plném provozu.

5.1 Druhy testů pro python

Jazyk Python má své druhy testů, ale ty nejpoužívanější jsou následující:
Jedná se o testovací knihovny **doctest**,**tox**,**pytest** anebo **unitest**. Každá z těchto knihoven se liší způsobem testovaní. Všechny testují se zvolený vstupem a následně kontrolují očekávaný výstup. Například knihovna doctest vloží testovací kód přímo do funkce jako komentář. Tento komentář slouží k testovaní kódu, který se zavolá se vstupním parametrem na dalším řádek očekávaný výstup (viz kód).

```
def get_data_from(self,specific_socket):
        """Get data from client or server.
        Return sum of byte string or empty
        byte string.
        >>> get_data_from('')
        b''
        """
        totall_data = b''
        while True:
            buffer_data = b''
            try:
                buffer_data = specific_socket.recv(1024)
                #self.save_data.append(bytes.decode(buffer_data))
                if not buffer_data:
                    break
            except socket.timeout:
                break
            except Exception as e:
                self.log.error("""Failed to recive data
                from specific socket: %s""" % e)
            totall_data = totall_data + buffer_data
        return totall_data

if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

Další knihovny, jsou jako unitest2, pro testovaní používají speciálně navr žený samotný testovací soubor pro testovaní specifických částí programu. Ty obsahují speciální metody pro testovaní zvolených modulů a jsou volány zvlášť.

Zmínit je třeba i **Tox**, protože se liší od ostatních testovací knihoven tím, že pro testovaní kódu aplikuje různé verze Pythonu, které jsou zvoleny. Například aplikaci chceme otestovat verzemi 3.5.2 a 3.5.3, tak je zadáme jako vstupní parametry testu. Pokud je testovaný kód kompatibilní s těmito verzemi, zjistí se to podle výstupu testu.

5.2 Průběh testování

Každý modul aplikace byl testován samostatně podle zvolených testů. Testovaní zjistilo mnoho chyb, které aplikace obsahovala během vývoje. Tudíž lze testovaní považovat za úspěšné. Aplikaci je třeba ovšem neustále testovat, a to i při dalším vývoji. Sice bylo navrženo, aby byl každý nový modul automaticky zkontrolován, ovšem ani to nemusí být zárukou bezchybného kódu.

6 Nasazení aplikace

Pro plné využití každé aplikace je třeba návodu instalace. Pomocí manuálu je možné aplikaci nainstalovat, tak i později spustit. Z čehož plyne využití aplikace, která je nepoužitelná, pokud nevím, jak ji nainstalovat. Různé programy a aplikace mohou obsahovat instalační skript. V tomto případě by bylo tento skript složité aplikovat na server a z důvodu nebezpečí potřeby práva superuživatele byla zvolena cesta postupného návodu instalace. Celý proces je vysvětlen do detailu se všemi potřebnými kroky pro instalaci na daný server.

6.1 Potřebné prvky

Prvním potřebným a nejdůležitějším prvkem je Python interpreter, přesněji nejnovější verze Python 3.52. Pro tuto verzi Pythonu je aplikace jak navržena, tak i testována. Z tohoto jazyka a interpretérů pramení další kroky pro úspěšnou instalaci.

Zde se jedná o knihovny pro programovací jazyk Python *tabulate*,*geolite2*, *maxmindb*, *pymongo* a *flask*. Pokud nebudou tyto knihovny instalovány, dojde k pádu aplikaci. Knihovna Flask jako jediná z knihoven pro Python slouží k běhu webové aplikace nikoliv backendové části, která běží jako daemon na serveru pro sběr informací. Dalším krokem je instalace databáze, kterou aplikace vyžaduje pro zápis nasbíraných dat.

6.2 Instalace

Pro jednoduchou instalaci lze využít manažér balíčků PIP pro jazyk Python. Obsahuje veškeré zmíněné knihovny. Po instalaci specifikovaných knihoven následuje instalace databáze. Pokud server neobsahuje databázi MongoDB, tak instalace probíhá ve třech krocích. Nejdříve se instaluje samotná databáze MongoDB, a to skrze baličkovacího mažera linuxové distribuce (např. dpkg). Dále vytvoření zprávy "root" adresáře /data/db, ale i s nastavením veřejného práva, aby měl program k tomuto adresáři přístup. Posledním krokem je první spuštění databáze s příkazem `mongod` a výstupem programu na standardní výstup.

Umístění aplikace záleží na administrátoru serveru, stačí, aby byla aplikace umístěna na serveru a soubor main.py měl dostatečná práva na spuštění.

6.3 Přidaní dalších pravidel

Jak již bylo zmíněno, aplikace byla navržena tak, aby ji bylo možné snadno rozšířit. Cíleno je na seznam pravidel, který je možné neomezeně upravovat, přidávat další pravidla. Stačí se jenom držet pravidel, která jsou vysvětlena v souboru *example.py*.

```
import logging
logger = logging.getLogger('ruleLogger')
# When writing error use logger to write error etc logger.error,
# info, warning
# Main function match_exploit
def match_exploit(data):
    if isinstance(data,dict):
        for key,val in data.items(): pass
    # Take data from other class, this data will be in
    # specific format:
    # defaultdict(<class 'list'>,
    # {'Accept': ['*/*'],
    #  'Accept-Encoding': ['gzip,-deflate'],
    #  'Accept-Language': ['cs-cz'],
    #  'Client_address': ['IP-Address:127.0.0.1,port:59196'],
    #  'Command': ['GET'],
    #  'Connection': ['keep-alive'],
    #  'Cookie': ['SQLiteManager_currentLangue=2'],
    #  'Error': [None],
    #  'Host': ['localhost:8887'],
    #  'If-Modified-Since': ['Thu,-01-Sep-2016-14:59:13-GMT'],
    #  'If-None-Match': ['W/"29c816-1-53b7374f45240"'],
    #  'Path': ['/MAMP/feed/needsUpdate.txt'],
    #  'Referer': ['http://localhost:8887/MAMP/?language=Eng.'],
    #  'Socket_info':['AddrFamily.AF_INET,Socket.SOCK_STREAM,
    #   Protocol-Protocol-0'],
    #  'Time': ['2016-09-02_21:14:40'],
    #  'User-Agent': ['Mozilla/5.0 (Macintosh;Intel-OSX10_11_6),
    #   AppleWebKit/601.7.7-(KHTML,Gecko)-Version/9.1.2-Safari'],
    #  'Version': ['HTTP/1.1'],
    #  'X-Requested-With': ['XMLHttpRequest']})

    return False
    # This function should return True if it finds specific
    # exploit else False Important!! If function doesn't return
    # True or False, it will be considered not working

# If you want to test your module use pythonic way to do it:
if __name__ == "__main__":
    pass
    # All your tests
```

7 Závěr

Na závěr je třeba zhodnotit celou práci a zároveň objasnit, jaký celkový přínos práce a aplikace přináší. Práce se soustředí na zkoumání současných hrozeb pro webové technologie. Aplikace se zase soustředí na to, jak tyto útoky rozpoznat a pomocí webové aplikace vytvořit analýzu těchto hrozeb. Přináší metodu filtrovaní vstupních dat, kterých může být v podobě HTTP požadavků mnoho. Každý tento požadavek může a nemusí obsahovat škodlivá data za cílem škodit. Veškerá tato vstupní data nejsou žádným způsobem upravena ani ovlivněna, ale pouze posílaná dále serveru. Ty jsou pak zobrazena v podobě webové aplikace, kterou má pod kontrolou administrátor serveru. Tento jeden způsob z mnoha se snaží zlepšit bezpečnost a tím i kvalitu služeb, které web nebo webová aplikace může přinášet a chránit data uživatelů. K ochraně dat a bezpečnosti jsou využity technologie, které se na trhu nevyskytují tak často a přináší další způsob, jak bojovat proti nebezpečí webu. Aplikace přes relativně dlouhý vývoj není zcela hotová, a to z důvodu, že lze předpokládat a konstatovat, že stále existují další rozšíření, kterými lze aplikaci obohatit.

Z hlediska závěrečné rekapitulace funkčnosti aplikace se celá aplikace se dělí na backendouvou a frontendovou část. Backendová běží na serveru jako proces daemon, který má za cíl sběr specifických dat, podezřelých z obsahu infikovaného kódů. Frontendová část tato data zobrazuje v přehledné podobě interaktivních grafů, které jsou aktualizovány podle obsahu a počtu dat. Spouštění je uskutečněno skrze soubor **main.py**, který volá další potřebné moduly, a to textbfserverside. Ten se stará o spojení dat mezi uživatelem a serverem. Jedná jako zpostřekovatel dat a zároveň data od uživatele vstupuje kontroluje. Tato data v podobě HTTP požadavku nejdříve parsuje, pro možnosti s nimi pracovat, a následně je kontroluje. Kontrola je za pomocí speciálního adresáře pravidel, která jsou navržena navržena pravě pro kontrolu webových útoků. Pokud jedno a více pravidel označí pravě kontrolovaný požadavek jako potencionální útok, je uložen do databáze. Tyto potencionální útoky jsou zobrazeny pomocí webové aplikace, která z databáze útoky čte a zobrazuje. Pro zobrazení využívá interaktivní grafy a mapu k přehlednějšímu zobrazení analýzy dat. Tato data jsou k dispozici pro administrátora a zabezpečeny pro přístup pověřeným osobám.

Jak již bylo zmíněno, aplikaci je možné dále rozšířit, a to například přidáním dalších pravidel. Tento seznam není omezen a aplikace je navržena tak, aby výpočet kontroly nezatěžoval komunikaci serveru s uživateli. Pravidel může být mnoho a mohou se lišit i náročností. Díky menší omezenosti času mohou být přidána pravidla s komplexnějším přístupem kontroly jako neuronové sítě. Dále se může aplikace soustředit na zjištění potencionálních chyb v lokálních PHP souborech. Dále použití několika knihoven pro Python, jako je ExeFilter, což je open sourceový framework pro filtraci souborů ve formátu e-mailů, webových stránek nebo souborů. Dále snažit se detekovat stejné obsahy, které může případně promazávat. **PyClamAV** je knihovna pro přidání detekci viru. Rozšířitelné jsou i interaktivní grafy, jejichž počet není omezen a na internetu jich je k dipozici celá řada.

8 Seznam zdrojů

ACUNETIX COM. *Web App Security – Check your Site for Web Application Vulnerabilities*. [online]. [cit. 2016-06-16]. Dostupné z: http://www.acunetix.com/websitesecurity/web-app-security/.

ACUNETIX COM. *Ordering Acunetix*. [online]. [cit. 2016-06-14]. Dostupné z: http://www.acunetix.com/ordering/.

ACUNETIX COM. *SQL Injection (SQLi)*. [online]. [cit. 2016-08-24]. Dostupné z: http://www.acunetix.com/websitesecurity/sql-injection/.

ACUNETIX COM. *Blind SQL Injection: What is it?*. [online]. [cit. 2016-06-24]. Dostupné z: http://www.acunetix.com/websitesecurity/blind-sql-injection/.

ACUNETIX COM. *Cross-site Scripting (XSS) Attack*. [online]. [cit. 2016-05-30]. Dostupné z: http://www.acunetix.com/websitesecurity/cross-site-scripting/.

ACUNETIX COM. *Blind XSS: The Ticking Time Bomb of XSS Attacks*. [online]. [cit. 2016-08-13]. Dostupné z: http://www.acunetix.com/blog/articles/blind-xss/.

ACUNETIX COM. *AJAX security: Are AJAX Applications Vulnerable to Hack Attacks?*. [online]. [cit. 2016-10-20]. Dostupné z: http://www.acunetix.com/websitesecurity/ajax/.

CIRT.NET *Nikto2*. [online]. [cit. 2016-08-14]. Dostupné z: https://www.cirt.net/Nikto2.

COUCHBASE *Why NoSQL*. [online]. [cit. 2016-11-30]. Dostupné z: http://www.couchbase.com/nosql-resources/why-nosql.

DEDE ,D. *MySQL.com compromised*. [online]. [cit. 2016-08-27]. Dostupné z: https://blog.sucuri.net/2011/03/mysql-com-compromised.html.

FLASK ORG. *User's Guide*. [online]. [cit. 2016-12-01]. Dostupné z: http://flask.pocoo.org/docs/0.12/.

HEARTBLEED BLOG *The Heartbleed Bug*. [online]. [cit. 2016-08-15]. Dostupné z: http://heartbleed.com.

INFORMATION COMMISSIONER'S OFFICE COMPANY *TalkTalk gets record £400,000 fine for failing to prevent October 2015 attack*. [online]. [cit. 2016-07-30]. Dostupné z: https://ico.org.uk/about-the-ico/news-and-events/news-and-blogs/2016/10/talktalk-gets-record-400-000-fine-for-failing-to-prevent-october-2015-attack/.

KALI LINUX *What is Kali Linux*. [online]. [cit. 2016-07-18]. Dostupné z: http://docs.kali.org/introduction/what-is-kali-linux.

Lemon, S. *Mass SQL Injection Attack Targets Chinese Web Sites*. [online]. [cit. 2016-010-23]. Dostupné z: http://www.pcworld.com/article/146048/article.html.

Long, J. *Google Hacking*. Nové Sady: Zone Press, 2005. 472 s. ISBN 80-86815-31-5..

MadIrish.net *Samurai Web Testing Framework*. [online]. [cit. 2016-08-14]. Dostupné z: http://www.madirish.net/188.

Michael Zino *ASCII Encoded/Binary String Automated SQL Injection Attack*.[online]. [cit. 2016-11-23]. Dostupné z: http://www.bloombit.com/Articles/2008/05/ASCII-Encoded-Binary-String-Automated-SQL-Injection.aspx.

MongoDB Inc. *The AGPL*. [online]. [cit. 2016-08-14]. Dostupné z: http://blog.mongodb.org/post/103832439/the-agpl.

MongoDB Inc. *MongoDB Manual Contents*. [online]. [cit. 2016-08-18]. Dostupné z: https://docs.mongodb.com/manual/contents/.

MongoDB Inc. *What is NoSQL?*. [online]. [cit. 2016-01-12]. Dostupné z: https://www.mongodb.com/nosql-explained.

MongoDB Inc. *Our Story*. [online]. [cit. 2016-10-19]. Dostupné z: https://www.mongodb.com/company.

Ormandy, T. *Analysis and Exploitation of an ESET Vulnerability*. [online]. [cit. 2016-010-15]. Dostupné z: https://googleprojectzero.blogspot.cz/2015/06/analysis-and-exploitation-of-eset.html.

Owasp Org. *SQL Injection*. [online]. [cit. 2016-07-17]. Dostupné z: https://www.owasp.org/index.php/SQL_Injection.

Picateshackz Com. *Sqlmap attack*. [online]. [cit. 2016-09-22]. Dostupné z: http://picateshackz.com/wp-content/uploads/2015/01/8.png.

Poulsen, K. *Guesswork Plagues Web Hole Reporting*. [online]. [cit. 2016-01-19]. Dostupné z: http://www.securityfocus.com/news/346.

Python Software Foundation *Python 3.0 Release*. [online]. [cit. 2016-05-22]. Dostupné z: https://www.python.org/download/releases/3.0/.

RedTeam Secure Consulting *Web Application Penetration Testing*. [online]. [cit. 2016-07-23]. Dostupné z: http://www.redteamsecure.com/application-penetration-testing/.

Robert Auger *Path Traversal*. [online]. [cit. 2016-07-13]. Dostupné z: http://projects.webappsec.org/w/page/13246952/Path%20Traversal.

Tichý, J. *Cross-site scripting*. [online]. [cit. 2016-03-16]. Dostupné z: http://www.phpguru.cz/clanky/cross-site-scripting.

Twitter Cor. *RedHack Tweet*. [online]. [cit. 2016-08-13]. Dostupné z: http://twitter.com/RedHack_EN/statuses/350461821456613376.

Yap, J. *450,000 user passwords leaked in Yahoo breach*. [online]. [cit. 2016-01-29]. Dostupné z: http://www.zdnet.com/450000-user-passwords-leaked-in-yahoo-breach-7000000772/.