SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5382-97879

IMPLEMENTÁCIA IDS PRE WEBOVÉ ÚTOKY

BAKALÁRSKA PRÁCA

2021 **Marek Mlynček**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5382-97879

IMPLEMENTÁCIA IDS PRE WEBOVÉ ÚTOKY
BAKALÁRSKA PRÁCA

Študijný program:	Aplikovaná informatika
Názov študijného odboru:	Informatika
Školiace pracovisko:	Ústav informatiky a matematiky
Vedúci záverečnej práce:	Ing. Peter Švec

Bratislava 2021 **Marek Mlynček**

Slovenská technická univerzita v Bratislave
Ústav informatiky a matematiky

Fakulta elektrotechniky a informatiky
Akademický rok: 2020/2021
Evidenčné číslo: FEI-5382-97879

ZADANIE BAKALÁRSKEJ PRÁCE

Študent:	**Marek Mlynček**
ID študenta:	97879
Študijný program:	aplikovaná informatika
Študijný odbor:	informatika
Vedúci práce:	Ing. Peter Švec
Miesto vypracovania:	Ústav informatiky a matematiky

Názov práce: **Implementácia IDS pre webové útoky**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Cieľom bakalárskej práce je zanalyzovať súčasný stav metód pre detekciu webových útokov a implementovať vlastné IDS (Intrusion Detection System) riešenie. Implementované riešenie bude zamerané na detekciu štandardných webových útokov ako sú napr. Cross-site scripting alebo SQL injection.

Úlohy:
1. Naštudujte aktuálne riešenia pre detekciu webových útokov.
2. Implementujte ukážkový prototyp IDS.
3. Otestujte prototyp na vybraných útokoch.
4. Porovnajte výsledky s existujúcimi riešeniami.

Zoznam odbornej literatúry:

1. Ishaque, M. – Hudec, L. Intrusion Detection System using Computational Intelligence Techniques. In *Proceedings of the 12th INDIACom; INDIACom-2018; IEEE Conference 2018 5th International Conference on "Computing for Sustainable Global Development", 14th – 16th March, 2018, Bharati Vidyapeeth's Institute of Computer Applications and Management (BVICAM), New Delhi (INDIA).* 1. vyd. S. l.: IEEE, 2018, s. 1432–1435. ISBN 978-93-80544-28-1.

Riešenie zadania práce od: 15. 02. 2021

Dátum odovzdania práce: 04. 06. 2021

Marek Mlynček
študent

Dr. rer. nat. Martin Drozda
vedúci pracoviska

Dr. rer. nat. Martin Drozda
garant študijného programu

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná informatika
Autor:	Marek Mlynček
Bakalárska práca:	Implementácia IDS pre webové útoky
Vedúci záverečnej práce:	Ing. Peter Švec
Miesto a rok predloženia práce:	Bratislava 2021

Účelom bakalárskej práce bolo na základe vykonanej analýzy navrhnúť *Intrusion Detection System* - IDS a vytvorenie správcovského panelu pre analýzu a spravovanie vytvoreného systému. V práci sa zaoberáme sledovaním webovej premávky a hodnotením jednotlivých HTTP správ z pohľadu bezpečnosti. Kontrola správ je zameraná na odhalenie škodlivého vstupu od používateľa (útočníka). Finálny systém sa skladá z dvoch častí a to z agenta na sledovanie a preposielanie HTTP správ a z panelu pre správu systému a zobrazenie výsledkov a štatistík. V prvej časti práce sa venujeme analýze útokov, ich dopadu na systémy a používateľov, na spôsoby a prístupy analýzy TCP tokov a spôsoby hodnotenia jednotlivých HTTP správ a ich obsahu. V ďalšej časti práce opisujeme samotnú implementáciu, podrobne opisujeme použité technológie pre vývoj IDS panelu a aj agenta. Vo finálnej časti môžeme vidieť výsledky našej práce a štatistiky úspešnosti proti simulovaným útokom.

Kľúčové slová: bezpečnosť, IDS, detekcia

ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Applied Informatics
Author:	Marek Mlynček
Bachelor's thesis:	Implementation of IDS for web attacks
Supervisor:	Ing. Peter Švec
Place and year of submission:	Bratislava 2021

The purpose of the Bachelor's thesis was based on the analysis to design an Intrusion Detection System and to create an Administration panel for further analyzing and system administration. In this work we deal with the web traffic monitoring and with the processing and scanning HTTP requests for the security reasons. The main purpose of the request scanning is to detect a possible malicious payload input by the user. The final system is composed of two main parts, these are the agent for the packet sniffing and sending requests to the IDS, and the admin IDS panel for system administration and traffic analysis. In the first section of our work we explain and analyze possible web application attacks, their consequences on the system and on other users, we analyze the different approaches of TCP stream sniffing and a different approaches for a HTTP request scoring. In the next section of the work we describe the implementation of our IDS, we talk about used technologies for agent and IDS development. The Final section we analyze the results of effectiveness of our IDS against some simulated attacks.

Keywords: security, IDS, detection

Poďakovanie

Chcel by som vyjadriť poďakovanie vedúcemu mojej práce Ing. Peter Švec za odbornú pomoc, jeho návrhy a postrehy a ochotu konzultovať akékoľvek problémy, s ktorými sme sa pri vypracovaní práce stretli.

Obsah

Zoznam obrázkov a tabuliek

Zoznam skratiek

API	Application programming interface
CGI	Common Gateway Interface
CSRF	Cross-Site Request Forgery
DBMS	Database Management Systems
DOM	The Document Object Model
DVWA	Damn Vulnerable Web Application
FTP	File Transfer Protocol
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IP	Internet Protocol
IPS	Intrusion Prevention System
ISO	International Organization for Standardization
LFI	Local File Inclusion
NSM	Network Security Monitoring
OS	Operačný Systém
OSI	Open Systems Interconnection
OWASP	The Open Web Application Security Project
REGEX	Regular Expressions
RFI	Remote File Inclusion
SIEM	Security Information and Event Management
SMB	Server Message Block
SQL	Structured Query Language
SSH	Secure shell
SSRF	Server-Side Request Forgery
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UML	Uniform Markup Language
URL	Uniform Resource Locator
XML	Extensible Markup Language
XSS	Cross-Site Scripting
XXE	External XML Entity

Úvod

Webové aplikácie sú aktuálne najviac používané druhy aplikácií a pomaly nahradzujú všetky *stand-alone* aplikácie. S nárastom ich používania sa zvyšujú aj požiadavky na aplikáciu ako aj jej celková rozmanitosť. S týmto javom je spojený aj nárast možných zraniteľností a "bezpečnostných dier"v aplikácii. Tieto zraniteľnosti predstavujú v reálnom svete skutočnú hrozbu. Útočníci sú schopní zo slabo zabezpečenej aplikácie odcudziť užívateľské dáta, prípadne interné dáta spoločnosti, manipulovať údaje, obmedzovať funkcionalitu aplikácie alebo spôsobiť jej nedostupnosť. Spomínané udalosti môžu mať veľký dopad na reputáciu spoločnosti alebo na iných používateľov a ich údaje. Podľa štatistík od organizácie *The Open Web Application Security Project* je až 77% webových aplikácií zraniteľných na *Cross-Site Scripting* a približne 35% obsahuje niektorú z *Injection* zraniteľností, ako je napríklad *SQL Injection.*[1]

Útoky na spomenuté zraniteľnosti majú svoju charakteristiku a vo väčšine prípadov ich vieme rozlíšiť od bežnej sieťovej premávky. To nám otvára možnosť automatizovanej detekcie útoku a analýze zachytenej hrozby. V tejto práci sa preto venujeme vyhodnocovaniu HTTP správ medzi klientom a serverom s cieľom odhaliť pokusy o útok na webovú aplikáciu.

V prvej kapitole sa venujeme práve tvaru a špecifickým prvkom, ktoré je možné pozorovať pri prevedení konkrétneho útoku. V druhej kapitole sme zhrnuli aké princípy a prístupy využívajú existujúce riešenia pre detekciu útokov na webové aplikácie a ktoré kroky v procese detekcie sú kľúčové pre dosiahnutie požadovaného výsledku. Podľa poznatkov a zistení uvedených v predchádzajúcich častiach práce, si v tretej kapitole vytvoríme návrh vlastného IDS riešenia. V štvrtej kapitole sa venujeme samotnej implementácii do simulovaného prostredia pre otestovanie funkcionality a účinnosti. Na koniec si v piatej kapitole zhrnieme dosiahnuté výsledky a pozorovania vlastného IDS systému.

1 Analýza útokov a ochrana

Webové aplikácie sa stali neodlúčiteľnou súčasťou našich životov. Neustále sa rozvíjajú a zlepšujú aby čoraz lepšie a efektívnejšie napĺňali naše požiadavky. Prudký vývoj však môže priniesť aj svoje riziká, medzi ktoré patria aj časté chyby a takzvané "diery v bezpečnosti". Dôkladne zabezpečiť svoju aplikáciu proti útokom nie je ani dnes jednoduchá úloha. Útoky sa vyvíjajú spolu s aplikáciami, automatizujú sa a je dôležité s nimi držať krok.

1.1 Typy útokov

Na rozdelenie a kategorizovanie útokov môžeme použiť viacero kritérií. Jedna z možností je rozdelenie podľa cieľa útok, čo môže byť napríklad server, administrátor alebo iný používateľ. Ďalší z možných spôsobov rozdelenia útokov je technológia, ktorú útočník využíva na zneužitie systému.

Pre jednoduchosť použijeme už definovaný rebríček zraniteľností od *OWASP Foundation[2]*. Organizácia OWASP pravidelne vyhodnocuje a analyzuje stav aplikácií v produkčnom prostredí z hľadiska bezpečnosti. Jedným z mnohých výsledkov tejto organizácie je aj rebríček *OWASP TOP10*, ktorý zobrazuje početnosť výskytu jednotlivých zraniteľností vo webových aplikáciách od najčastejšie sa vyskytujúcej až po zriedkavú.

OWASP Top 10 - 2017
A1: Injection
A2: Broken Authentication
A3: Sensitive Data Exposure
A4: XML External Entities
A5: Broken Access Controll
A6: Security Misconfiguration
A7: Cross-Site Scripting
A8: Insecure Deserialization
A9: Using Components with Known Vulnerabilities
A10: Insufficent Logging and Monitoring

Obr. 1: Tabuľka OWASP Top-10 zraniteľností webových aplikácií (2017)[2]

Každý zo znázornených útokov má svoje špecifické vlastnosti a prevedenie avšak za zraniteľnosť aplikácie považujeme aj nesprávnu konfiguráciu komponentov aplikácie, používanie neaktuálneho softvéru alebo aj veľmi triviálnu záležitosť ako je nedostatočne

komplexné heslo do systému. V našej práci sme sa rozhodli detailne venovať konkrétne dvom útokom a to *Cross-Site Scripting (XSS)* a *SQL Injection*, ktoré sú v OWASP Top10 tabuľke na pozíciách A1 a A7.

1.1.1 SQL Injection

Tento typ útoku je zameraný predovšetkým na databázu, jej správanie a logiku implementovanú vo webovej aplikácii[3]. Útočník sa snaží zistiť ako aplikácia interaguje s databázou a ako spracováva užívateľom vložený vstup. Cieľom tohto útoku je teda zneužiť logiku SQL výrazov, ktoré vznikajú na *back-ende* webovej aplikácie pomocou užívateľského vstupu, a následne komunikujú s databázou.

Pri nedostatočnom ošetrení používateľského vstupu môže mať takáto zraniteľnosť v aplikácii fatálne následky. Dopad na aplikáciu sa môže každým prípadom líšiť. Útočník môže mať neobmedzený prístup k databáze, tzn. všetky prihlasovacie mená a heslá ostatných registrovaných užívateľov, môže dokonca celú databázu zmazať. V niektorých prípadoch sa pomocou *SQL Injection* môže útočník v aplikácii identifikovať ako iný užívateľ alebo dokonca administrátor. Táto zraniteľnosť sa nazýva horizontálna / vertikálna eskalácia privilégií a je dosiahnuteľná pomocou *SQL Injection*.

Na to, aby sme vedeli overiť či je aplikácia zraniteľná voči *SQL Injection* potrebujeme vytvoriť vstup, ktorý by sa nejakým spôsobom pokúšal napadnúť SQL výraz, do ktorého bude tento vstup neskôr vložený. Takýto špeciálny vstup sa nazýva aj *payload* a v každej aplikácii sa môže líšiť, preto je potrebné na takýto test poznať SQL syntax rôznych DBMS systémov. Tento *payload* môže mať rôzne účely, ako je vyššie spomínané v tejto práci, ako ukážku si uvedieme dva príklady *payloadu* pre *SQL Injection*.

Majme aplikáciu, ktorá si od používateľa žiada ako vstup identifikátor používateľa a následne o ňom vypíše údaje. SQL výraz, ktorý by v tomto prípade *back-end* aplikácie spúšťal nad databázou môže byť nasledovný :

```
SELECT first_name, last_name FROM users WHERE user_id = '$id';
```

V prípade že vstup od používateľa nie je nijak kontrolovaný a obmedzený, útočník môže vytvoriť *payload*, ktorý bude nejakým spôsobom dopĺňať SQL výraz. Jeden z možných *payloadov* na vypísanie všetkých používateľov by mohol byť:

```
' OR 1 = 1 #
```

Po vložení tohto *payloadu* do SQL výrazu dostaneme finálny výraz:

```
SELECT first_name, last_name FROM users WHERE user_id = '' OR 1 = 1 #';
```

Ako vidíme tak tento *payload* napadol logiku výrazu a *SELECT* vráti všetky výsledky kde *user_id = ' '* alebo kde *1=1*, to znamená že bude vrátený každý záznam, ktorý sa nachádza v tabuľke *users*. Zaujímavejší *payload* by mohol byť taký, pomocou ktorého získame heslá všetkých používateľov, ktorý sú v aplikácii registrovaný. Takýto vstup by mohol vyzerať takto:

```
' UNION SELECT first_name, password FROM users #
```

Po vložení *payloadu* do aplikácie, vznikne finálny výraz:

```
SELECT first_name, last_name FROM users WHERE
user_id = '' UNION SELECT first_name, password FROM users #';
```

Pomocou tohto vstupu nám databáza vytvorí novú tabuľku, podľa parametrov zadaných v *UNION SELECT* a výsledok vráti spolu so základným naprogramovaným *SELECTom*. Takýmto spôsobom má útočník k dispozícii všetky záznamy uložené v databáze, vrátane hesiel.

Toto je ukážka princípu útoku na logiku SQL výrazov implementovaných v aplikácii. V praxi sa však často stretneme s maskovaním útoku, prípadne iným tvarom *payloadu* aby sa útočník vyhol prípadnej detekcii pokusu o útok alebo jeho úplnému zastaveniu. Je dôležité spomenúť najmä dva typy takéhoto maskovania a to zakódovanie *payloadu* do iného tvaru, napríklad do hexadecimálneho tvaru alebo vloženie komentárov na zmätenie detekcie napríklad: *UNI/*xxx*/ON SE/*xxx*/LECT*. V obidvoch prípadoch ide o valídny *payload*, ktorý bude spustený databázou ale je deformovaný za účelom pôsobiť ako nevinný vstup.

1.1.2 Cross-Site Scripting XSS

Veľmi populárny typ útoku s názvom *Cross-Site Scripting*, skrátene XSS, je útok, pri ktorom sa útočník pokúša spustiť kúsok špeciálne navrhnutého JavaScript kódu v prehliadači svojej obete[3]. *Payload* budeme v tomto prípade nazývať samotný skript, ktorý sa útočník pokúša spustiť v prehliadači inej osoby. Ide teda predovšetkým o útok na iných používateľov zraniteľnej aplikácie.
XSS môžeme rozdeliť do troch základných skupín:

- *Reflected XSS* - môžeme považovať za jednoduchší spôsob. Spravidla sa táto zraniteľnosť využíva vo funkcionalite vyhľadávania v rámci webovej aplikácie. Útok spočíva v tom, že útočník pošle svojej obeti URL, obsahujúce škodlivý *payload*. Obeť je po kliknutí na odkaz presmerovaná na zraniteľnú časť aplikácie, kde je ako vstup automaticky vložený škodlivý skript z URL. Takéto URL môže vyzerať napríklad takto:

http://vulnerable.host/?search=<script>MALICIOUS PAYLOAD</script>

- *Stored/Persistent XSS* - môže byť nebezpečnejšie z dôvodu, že nevyžaduje zasielanie URL ani inú interakciu s používateľmi. Ako už názov napovedá, ide o skript, ktorý je uložený na serveri, a ten nám ho pri každom načítaní sám vráti. Pre príklad možno uviesť príspevok vo fóre v zraniteľnej aplikácii. Útočník môže do svojho príspevku pridať HTML element obsahujúci škodlivý skript a aplikácia si takýto príspevok uloží do databázy. Každý ďalší používateľ, ktorý si od serveru vyžiada daný príspevok, ho dostane spoločne so skriptom útočníka, ten sa následne spustí v prehliadači. Nakoľko nemusí dôjsť k žiadnej interakcii medzi útočníkom a obeťou ako tomu bolo pri *Reflected XSS*, považujeme tento typ XSS za nebezpečnejší a v skutočnosti môže každá aplikácia, ktorú práve používame spustiť skripty útočníkov v našom prehliadači. Tvar aj obsah *paylaodu* môže byť vo všetkých prípadoch XSS rovnaký.

- *DOM Based XSS* - podobne ako pri *stored XSS* sa ani v tomto prípade nevyžaduje interakcia obete s útočníkom. Tento druh XSS využíva zraniteľnosť v logike aplikácie, ktorá zapisuje dáta do *Document Object Model (DOM)*. Útočník je v prípade zraniteľnej aplikácie schopný do dokumentu zapísať vlastný *payload*, a tým docieli jeho spustenie.

XSS *payload* je v porovnaní s *SQL Injection* omnoho zložitejší a nemá daný tvar, ktorý musí dodržiavať, keďže môže ísť o akýkoľvek skript. Útočník môže skript do aplikácie vložiť rôznym spôsobom, spravidla to býva niektorý z HTML elementov[1], kedy sa do aplikácie ako vstup vloží nový HTML element, obsahujúci nebezpečný skript, spustiteľný eventom podľa voľby útočníka. Rôznorodosť HTML elementov, ich atribútov a eventov, ktoré k nim patria, robí z detekcie tohto útoku pomerne zložitú úlohu.

1.1.3 Iné útoky

Existuje mnoho ďalších známych útokov na webové aplikácie, ktorým sa ale nebudeme v našej práci venovať až do takej hĺbky. Je však vhodné spomenúť a vysvetliť aspoň niektoré z nich nakoľko sa s nimi v tejto práci ešte stretneme.

- *External XML Entity* (XXE) - ide o druh zraniteľnosti, ktorá útočníkovi umožňuje komunikovať s aplikáciou pomocou XML vstupu. Výskyt takejto zraniteľnosti môže

[1] `https://portswigger.net/web-security/cross-site-scripting/cheat-sheet`

útočníkovi poskytnúť prístup k súborovému systému servera alebo iným externým zdrojom, s ktorými aplikácia interaguje.

- *Command Injection* - tento druh zraniteľnosti umožňuje útočníkovi spustiť ľubovoľný príkaz operačného systému na serveri, kde beží zraniteľná aplikácia. Zraniteľnosť za väčšinou vyskytuje v aplikáciách, ktoré poskytujú spúšťanie vybraných *shell* príkazov a nemajú dostatočnú ochranu, alebo v prípade, že aplikácia používa skript, ktorý následne spúšťa *shell* príkaz.

- *Cross-Site Request Forgery* (CSRF) - ide o útok, ktorý zneužíva skutočnosť, že používateľ je v aplikácii prihlásený a je vytvorená "dôvera"medzi používateľom a aplikáciou. Útočník potom dokáže v prípade zraniteľnej aplikácie vytvoriť požiadavku na server v mene prihláseného používateľa.

- *Server-Side Request Forgery* (SSRF) - pri tejto zraniteľnosti útočník dokáže vyvolať HTTP správu zo zraniteľného servera na akýkoľvek zvolený zdroj alebo inú aplikáciu.

- *Local / Remote File Inclusion* (LFI / RFI) - poskytuje útočníkovi možnosť dostať sa k súborom na serveri, ktoré nie sú verejne dostupné. Pri takto zraniteľnej aplikácii môže mať útočník prístup napríklad k súborom ako */etc/passwd* alebo k privátnym SSH kľúčom, ktoré môžu byť ďalej zneužité.

Všetky zo spomenutých útokov majú svoje špecifické vlastnosti a tvar payloadu. Na základe toho by sme vedeli jednotlivé útoky aj zachytiť a kategorizovať.

1.2 Ochrana pred útokmi

Je nesmierne veľa známych útokov a možností ako aplikáciu alebo jej logiku napadnúť a zneužiť. Ako zamedziť takýmto útokom? Bohužiaľ neexistuje žiadne univerzálne riešenie na všetky problémy a zraniteľnosti ale správnym prístupom k vývoju aplikácie a zabezpečeniu ďalšej jej funkcionality vieme útočníkovi prácu veľmi skomplikovať, prípadne niektorý typ útoku aj úplne znemožniť.

1.2.1 Bezpečné programovanie

Prvý krok v zabezpečovaní webovej aplikácie je správne a bezpečné programovanie jej funkcionality a spracovania dát od všetkých externých systémov aj používateľov. Väčšina zraniteľností, s ktorými sa stretneme sú spôsobené práve programátormi a ich nesprávnemu prístupu k zabezpečeniu aplikácie. Všetky podstatné informácie o bezpečnom programovaní sú zhrnuté v *OWASP Secure Coding Practice Quick Reference Guide*[2]

[2] `https://owasp.org/www-pdf-archive/OWASP_SCP_Quick_Reference_Guide_v2.pdf`

1.2.2 IDS / IPS

Ako doplnkovú vrstvu bezpečnosti pre aplikáciu môžeme využiť aj *Intrusion Detection System*. Na trhu už existuje široká škála takýchto systémov na detekciu rôznych útokov. IDS systémy majú rôzne zameranie, niektoré sa venujú detekciám a bezpečnosti siete, iné sa môžu venovať bezpečnosti aplikácie alebo servera a podobne. Záznamy, dáta a upozornenia generované systémom sú analyzované a vyhodnocované, z čoho je následne možné vytvárať reporty alebo bezpečnostné opatrenia v prípade prebiehajúceho útoku. IDS systémy ponúkajú správcovi výborný prehľad nad aplikáciou a premávkou, s ktorou sa aplikácia musí vysporiadať. Umožňujú včas detegovať a reagovať na možné nebezpečenstvo alebo útok a zamedziť tak možným škodám. Kombináciou bezpečného programovania a IDS systému získame veľmi dobre zabezpečenú aplikáciu a zároveň aj prehľad nad aktivitou používateľov a teda aj možných útočníkov.

1.2.3 Penetračné testy

Pravidelné penetračné testy dokážu odhaliť nedostatky, chyby v bezpečnosti a zraniteľnosti ešte pred tým ako to odhalí útočník so zlým zámerom. Samozrejme, ani tieto testy nie sú bezchybné a nedá sa povedať, že ak je aplikácia po teste bez nálezu zraniteľnosti, je úplne bezpečná. Každý tester má iné skúsenosti a znalosti, preto sa na to nemôžme úplne spoľahnúť, je to však výborný obraz o tom, v akom stave je aplikácia a jej funkcionalita z pohľadu bezpečnosti.

Ďalšou možnosťou sú automatizované testy zraniteľností pre webové aplikácie. Tie dokážu odhaliť veľa známych zraniteľností za pomerne krátky čas a sú vhodné na pravidelné a naplánované kontroly aplikácie. Medzi známe produkty v tejto oblasti patria napríklad :

- App Scaner
- Burp Suite
- Zed Attack Proxy
- Nessus

Tieto skenery taktiež vytvoria dobrý obraz o stave a bezpečnosti aplikácie ale neslúžia ako úplná náhrada skúseného penetračného testera, ktorý dokáže odhaliť aj komplikovanejšie problémy, ktoré je veľmi zložité automatizovať.

2 Súčasné IDS riešenia

V tejto kapitole si priblížime známe IDS systémy, ich funkcionalitu, výhody a nevýhody a pozrieme sa na ich prístup k detekcii možnej hrozby, prípadne iných anomálií. Bližšie si vysvetlíme, čo je dôležité monitorovať a akými parametrami sieťovej premávky je vhodné sa zaoberať ak chceme zaručiť čo najefektívnejšie odhaľovanie útokov na aplikáciu.

2.1 Intrusion Detection/Prevention System

Intrusion Detection System[3] je zariadenie alebo *softvér*, ktorý slúži na monitorovanie siete alebo iných systémov, so zámerom detegovať škodlivú aktivitu alebo porušenie bezpečnostných zásad a pravidiel[4]. Každá podozrivá aktivita alebo útok je následne nahlásená buď správcovi alebo centrálne zaznamenaná pomocou *Security Information and Event Management (SIEM)*[4] systému. SIEM systém môže kombinovať výstup z viacerých zdrojov a využíva techniky na filtrovanie bezpečnostných incidentov pre rozlíšenie či ide o skutočný útok alebo len falošný poplach.

V čom sa líši od *Intrusion Prevention System (IPS)* ? IPS systém je schopný autonómne vykonať predom nastavené kroky v prípade detekcie narušenia bezpečnosti. Systém na vzniknuté riziko reaguje, takže hlavý rozdiel je, že IPS dokáže v niektorých prípadoch zamedziť ďalšiemu útoku alebo zneužitiu aplikácie, kde IDS systém nás o prebiehajúcom útoku len informuje.

Na trhu už samozrejme existuje viacero funkčných a osvedčených IDS / IPS riešení. Každé z nich má svoje výhody a nevýhody, niektoré sú zamerané na sieťovú premávku a sledujú aj protokoly nižších vrstiev OSI-ISO modelu ako napríklad TCP, IP, UDP, ICMP a podobne. Iné sú zas zamerané na ochranu webovej aplikácie a pracujú viac s protokolmi na aplikačnej vrstve sieťového modelu. Neexistuje teda nič ako univerzálne najlepšie riešenie a je na správcovi systému, ktorou cestou sa vydá, a ktorý z dostupných produktov mu vyhovuje najviac. Vhodným prístupom pri výbere konkrétneho IDS systému je porovnanie existujúcich populárnych riešení a na základe ich charakteristiky a skúseností iných používateľov vybrať vhodné riešenie pre vlastnú potrebu.[5]

Ako príklad sme zvolili dva veľmi populárne IDS riešenia a to **Snort** a **Suricata**.

[3] `https://www.checkpoint.com/cyber-hub/network-security/`

[4] `https://www.varonis.com/blog/what-is-siem/`

[5] https://www.softwaretestinghelp.com/intrusion-detection-systems/

2.1.1 Snort

Ide o *Open Source Intrusion Prevention System* IPS. Používa systém pravidiel, ktoré pomáhajú definovať škodlivú aktivitu na sieti. Pomocou týchto pravidiel deteguje podozrivé správy a na základe ďalšej analýzy vytvorí upozornenie pre používateľa.
Funkcie:

- Monitorovanie a ukladanie sieťových správ
- Sledovanie hrozby
- Blokovanie na základe známych signatúr
- *Real-time* update pre bezpečnostné signatúry
- Detailné reporty
- Detekcia rôznych parametrov ako použitý OS, SMB sondy, CGI útoky, *buffer overflow* útoky a *stealth scan* otvorených portov.

Nevýhody:

- *Upgrade* je často nebezpečný
- Nestabilný s Cisco chybami

2.1.2 Suricata

Je *Open Source* rýchly a robustný systém pre detekciu hrozby na sieti. Suricata jadro je schopné *real-time* detekcie škodlivých dát(IDS), prevencia útokov (IPS), bezpečnostné monitorovanie sieťovej premávky (NSM) a *offline* spracovávanie pcap súborov.
Funkcie:

- Zbiera dáta na aplikačnej vrstve
- Schopnosť monitorovať protokoly nižších vrstiev ako napríklad TCP, IP, UDP, ICMP
- *Real-time* hľadanie sieťových aplikácií ako SMB, HTTP, and FTP
- Integrácia aj so *softvérom* tretích strán ako Anaval, Squil, BASE a Snorby
- Vstavaný skriptovací modul
- Detekcia na princípe známych bezpečnostných signatúr ale aj na princípe anomálií

Nevýhody:

- Komplikovaný proces inštalácie
- Menšia komunita ako pri Snort IPS

Pri výbere vhodného IDS vstupuje do hry viacero faktorov. Veľkosť kapitálu pre zriadenie riešenia, mohutnosť monitorovanej siete, použité technológie a podobne.

2.2 Monitorovanie sieťovej premávky

Dôležitý aspekt pri detekcii prebiehajúceho útoku, prípadne pokusu o napadnutie aplikácie je monitorovanie a analýza sieťovej premávky a jej parametrov. Ak sa chceme venovať bezpečnosti webovej aplikácie, je vhodné sledovať práve dáta posielané HTTP protokolom.

Každý vstup od používateľa, či už ide o útok alebo nie, musí prejsť HTTP protokolom zo zariadenia používateľa až na server, kde je nasadená webová aplikácia. Vstup je v tvare jednoduchého reťazca znakov. Nad týmito reťazcami je možné robiť kontrolu či ide o nedovolený vstup alebo regulárnu premávku. Je viac prístupov, ktoré môžeme zvoliť. Jeden z nich je kontrola povolených znakov, tzv. *whitelisting* a ak reťazec obsahuje nedovolené znaky, upozorniť na túto skutočnosť. Ďalší z prístupov môže byť *blacklisting*. Problém s týmto prístupom je ten, že musíme rozlíšiť, do ktorého vstupného poľa daný vstup smeruje, pretože do niektorých vstupných polí v aplikácii môžu ísť aj znaky, ktoré považujeme za možné nebezpečenstvo ako napríklad <>'; a podobne. Keďže takáto kontrola vyžaduje prácu s reťazcami, hľadanie zhody a porovnávanie reťazcov, je vhodné pre tento prípad použiť REGEX.

3 Návrh IDS

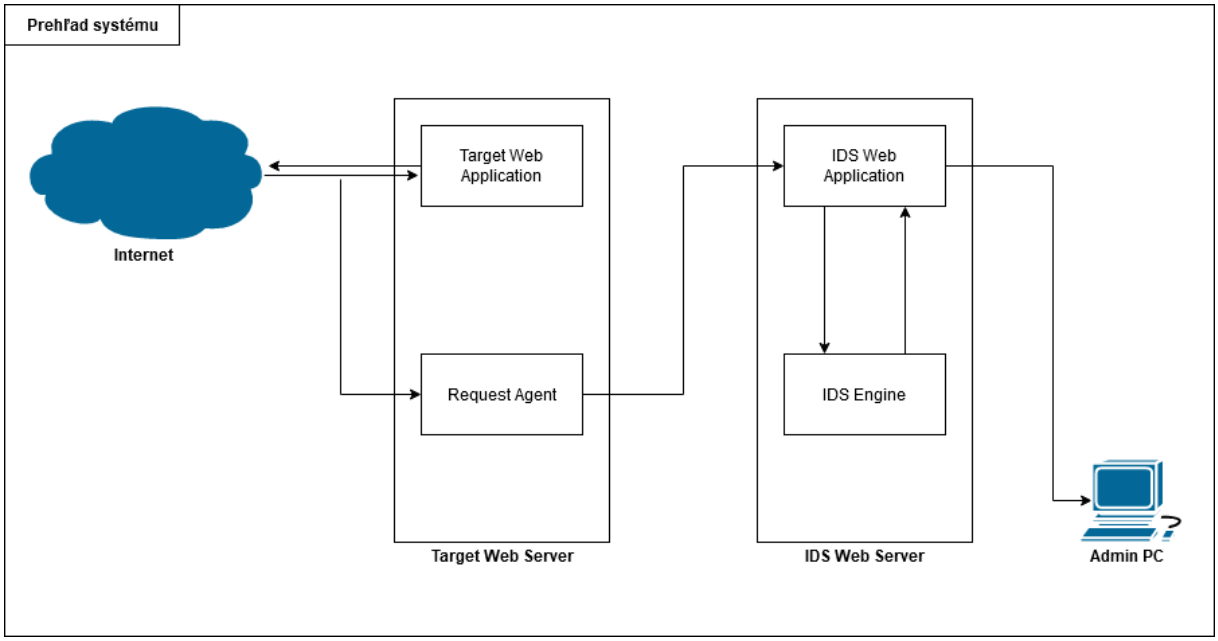
Na základe analýzy danej problematiky v predchádzajúcich častiach tejto práce, vieme určiť kroky potrebné pre návrh, vývoj a implementáciu vlastného IDS riešenia. Vieme akú funkcionalitu má navrhovaný systém obsahovať a čo od neho môže používateľ očakávať. S týmito informáciami si môžeme presne definovať cieľ bakalárskej práce.

3.1 Cieľ vlastného riešenia

Dôležitá súčasť tejto bakalárskej práce je osvojiť si metódy a techniky monitorovania webovej premávky a vyhodnocovania hrozieb. Porovnať a analyzovať existujúce IDS a IPS riešenia. Osvojiť si techniky a charakteristiky známych webových útokov, so zameraním na *SQL Injection* a *Cross-Site Scripting*. Na základe získaných poznatkov navrhnúť a implementovať vlastné riešenie na zachytenie spomenutých webových útokov. Simulovať útoky na testovacej aplikácii a vyhodnotiť štatistiky úspešnosti detekcie týchto útokov vlastným IDS riešením.

3.2 Architektúra systému

Systém, ktorý navrhneme pre splnenie vyššie uvedených cieľov sa bude skladať z dvoch kľúčových častí a to z agenta na monitorovanie TCP prúdov dát a z IDS správcovského panelu, ktorý slúži ako grafické rozhranie pre správcu systému na ďalšie nastavovanie a sledovanie premávky.



Obr. 2: Prehľad implementácie komponentov IDS riešenia

3.2.1 C++ Agent

V tejto časti práce sa venujeme zachytávaniu sieťovej komunikácie medzi serverom a klientom. Z danej komunikácie potrebujeme vyťažiť niekoľko kľúčových informácií pre podrobnejšiu analýzu, zozbierať ich do vhodnej štruktúry a odoslať na ďalšie spracovanie. Agent by mal zachytiť predovšetkým HTTP správy vytvorené klientom a jeho zdrojovú IP adresu.

3.2.2 Jadro IDS a používateľské rozhranie

Na zobrazenie správ prijatých od agenta použijeme webovú aplikáciu. Tá bude používateľovi poskytovať informácie o stave premávky, výsledky analýzy HTTP správ, štatistiky a ďalšie iné nastavenia IDS riešenia. Na *back-ende* tejto aplikácie bude bežať samotné jadro pre hodnotenie HTTP správ pomocou REGEX pravidiel. Pravidlá sú tiež súčasťou nastavení v IDS paneli.

3.3 Zachytávanie sieťovej premávky

Hlavnou úlohou agenta je zachytenie alebo sledovanie sieťovej premávky medzi klientom a serverom. To najdôležitejšie čo z tejto komunikácie potrebujeme získať pre detekciu známych webových útokov je práve správa posielaná pomocou HTTP protokolu na aplikačnej vrstve OSI-ISO sieťového modelu. Vhodným parametrom pre zachytenie je aj zdrojová IP adresa, teda adresa klienta. Takto zachytené dáta sa potom pošlú na spracovanie do IDS systému.

Je viac možností, ako pristupovať k sledovaniu HTTP správ. V našom prípade ide o tvorbu *Intrusion Detection System* riešenia a preto je postačujúce, ak premávku len čítame. Nie je potrebné na správy nijak reagovať ani ich zastavovať prípadne blokovať, ako by to bolo v prípade *Intrusion Prevention System*. Postačuje ak zozbierame z komunikácie dáta, s ktorými chceme ďalej pracovať, a v definovanej štruktúre ich odovzdáme nášmu IDS systému. Veľmi vhodný prístup pre tvorbu *softvéru* na sledovanie premávky je aj použitie proxy servera. Použitím takéhoto servera by celá komunikácie medzi klientom a serverom prechádzala navyše aj cez proxy server. Táto možnosť má viacero výhod a jednou z nich je aj možnosť správy označené ako hrozbu úplne zablokovať a tým zabrániť možnému útoku na aplikáciu.

3.4 Hodnotenie HTTP správ

Kľúčovou úlohou IDS systému je odhalenie podozrivej alebo nebezpečnej komunikácie klienta so serverom. Pre splnenie tejto požiadavky využijeme práve REGEX pravidlá, ktoré budú kontrolovať každú správu, prijatú od agenta. Každá správa môže byť pred odo-

slaním útočníkom ľubovoľne modifikovaná, preto musíme kontrolovať všetky parametre, ktoré by mohli obsahovať škodlivý obsah. Je potrebné sledovať všetky štandardné vstupy od používateľa, či už ide o parametre v URL alebo v tele HTTP správy ak ide o POST metódu.

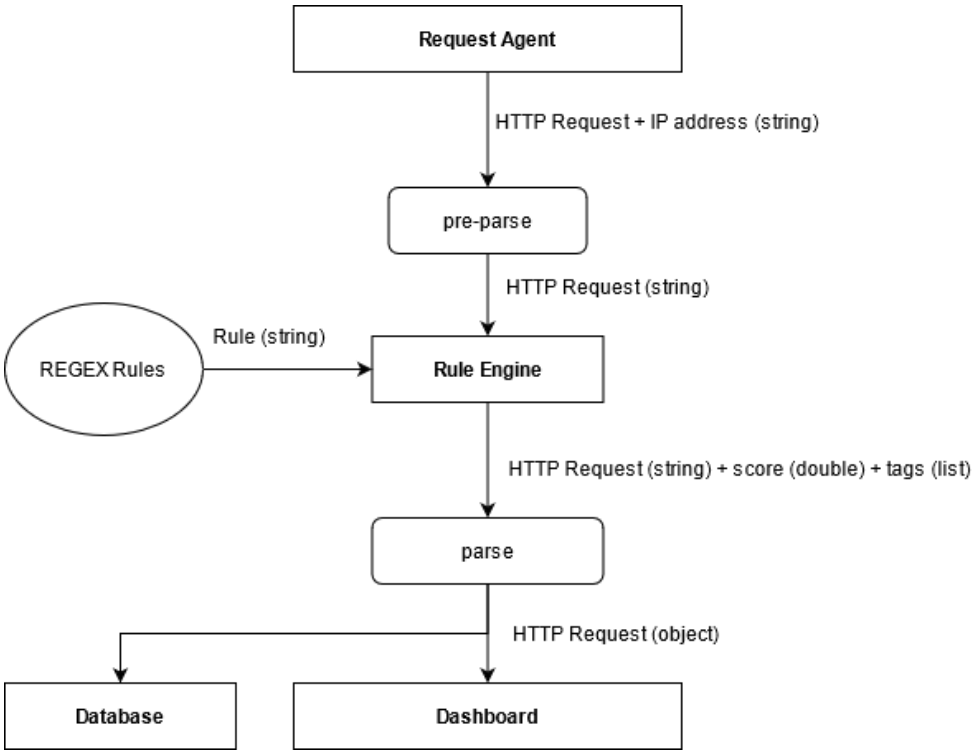
Na kontrolu správy je potrebné vytvoriť REGEX pravidlá. Dôkladnosť kontroly a prísnosť pravidiel závisí od správcu systému. Ak by sme chceli zachytiť každý útok typu *SQL Injection* mohli by sme jednoducho vytvoriť pravidlo pre detekciu jednej úvodzovky, keďže ide o základnú časť pri pokusoch o tento útok. Ten istý prístup môžeme využiť aj pri zachytávaní *Cross-Site Scripting*. Správca môže vytvoriť pravidlo, ktoré označí ako útok každú správu, ktorá obsahuje znaky '<' a '>'. Tým by sme detegovali každý pokus o vloženie HTML prvku a tým aj všetky pokusy o XSS útok. Takýto prístup ale vedie k obrovskému množstvu falošných detekcií, čo môže niekedy spôsobiť veľa nepríjemností. Je bežné, že používateľ tieto znaky používa, aj keď sa na aplikáciu nesnaží útočiť, preto musíme vytvoriť komplexnejšie pravidlá, ktoré sa viac približujú syntaxi útoku.[5]
Jednoduchý príklad pravidla pre detekciu *SQL Injection order by* príkazu môže byť:

```
['")]*\s*order(\s)+by(\s)+[0-9]+
```

Vysvetlenie:

- **['")]*** : pravidlo by malo očakávať, že kontrolovaný reťazec sa bude začínať jedným zo znakov uvedených v medzi hranatými zátvorkami. Veľká väčšina útokov na databázu začína práve týmito znakmi a to je aj dôvodom ich použitia vo veľa pravidlách.
- **(\s)*order(\s)+by(\s)+** : táto časť pravidla kontroluje či sa v reťazci nachádza SQL príkaz *order by*, ktorý pri *SQL Injecton* útoku slúži na odhalenie počtu selektovaných stĺpcov výrazom na *back-ende* webovej aplikácie.
- **[0-9]+** : na koniec aby príkaz pracoval správne, musí útočník pridať číslo, ktoré reprezentuje ním odhadovaný počet stĺpcov.

Spojením väčšieho množstva obdobných pravidiel vieme vytvoriť istú vrstvu ochrany pre aplikáciu a zlepšiť tak detekciu útokov. Po tom, ako je HTTP správa zachytená pravidlom a označená ako podozrivá, je jej pridané skóre, ktoré predstavuje mieru nebezpečenstva pre monitorovaný systém. Spodná hranica tohto skóre je 0 a to predstavuje premávku, ktorú systém vyhodnotil ako bezpečnú. Horná hranica nie je určená a platí, čim vyššie skóre, tým väčšie podozrenie. Správa, ktorá je pravidlom zachytená, navyše dostane označenie (tag), podľa toho, ktoré pravidlo ju zachytilo. Používateľské rozhranie navrhnuté k systému umožňuje takto označené správy jednoducho filtrovať a analyzovať.

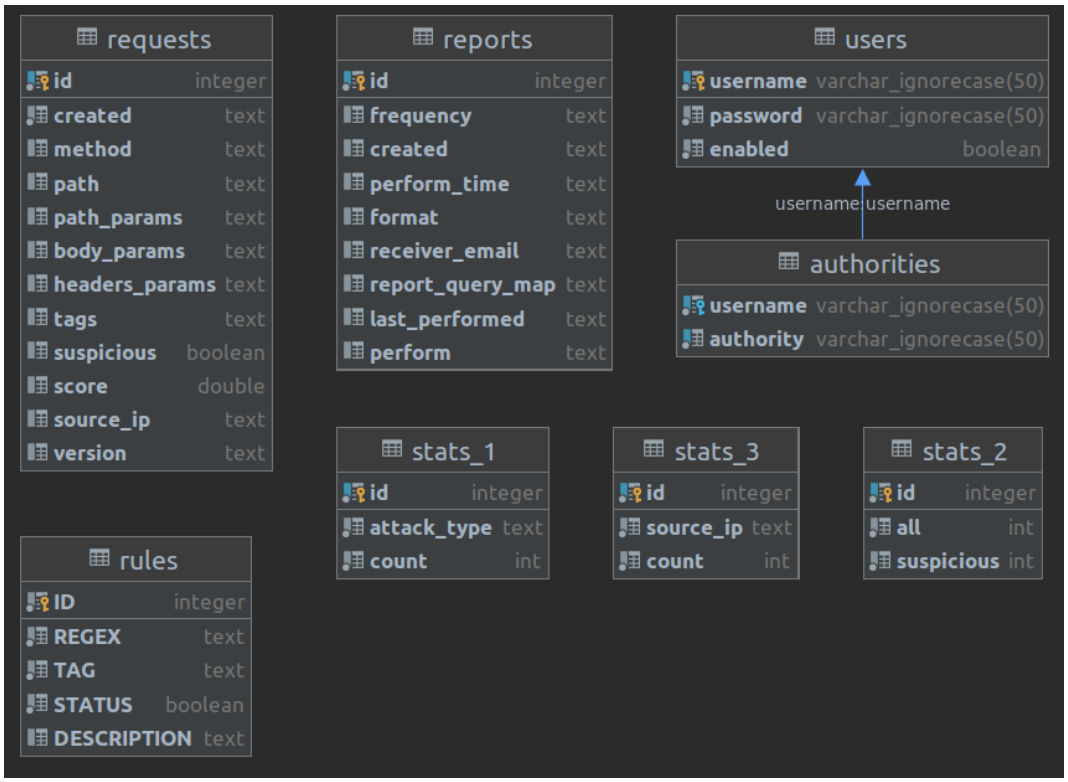


Obr. 3: *Data-flow* diagram spracovania HTTP správy

Na obrázku číslo 3 môžeme vidieť, že z komponentu *Request Agent* príde HTTP správa spolu so zdrojovou IP adresou vo forme jednoduchého reťazca (*string*). Od správy sa následne oddelí IP adresa a nasleduje samotné hodnotenie. Správa je hodnotená ako jeden celý reťazec, to znamená ako jeden riadok textu. Týmto vieme zabezpečiť, že pravidlo bude aplikované na všetky parametre správy, nakoľko útočník má nad odosielanou správou úplnú kontrolu, nebezpečný *payload* sa môže nachádzať v hociktorej časti správy. Po aplikovaní všetkých pravidiel nasleduje spracovanie správy ako aj výsledkov jej hodnotenia. Všetky informácie spojíme do jedného objektu a uložíme do databázy, prípadne zobrazíme v IDS paneli.

3.5 Návrh databázy

Naša databáza bude obsahovať jednoduché tabuľky a bude typu SQLite. Bude slúžiť pre ukladanie pravidiel navrhnutých a pridaných správcom pomocou používateľského webového rozhrania systému. Ďalšou položkou budú samotné HTTP správy, ktoré budú v databáze uložené už po ohodnotení všetkými pravidlami a majú priradené svoje skóre ako aj tagy. Odtiaľto budú správy neskôr zobrazované v niektorých častiach IDS panelu. V databáze uložíme taktiež cykly pre odosielanie reportov a štatistiky potrebné pre hlavný *Dashboard* IDS panelu a to v tabuľke *reports*.



Obr. 4: Návrh tabuliek pre databázu

Tabuľky *users* a *authorities* sa starajú o autentifikáciu používateľa pri prihlasovaní do IDS panelu. Obsahujú používateľské meno a *hash* hesla. V tabuľke *authorities* je uložená rola, ktorú má konkrétny používateľ pridelenú. V našom prípade sa pracuje iba s rolou *ROLE_ADMIN*. Posledné tri tabuľky *stats_1-3* obsahujú údaje pre *Dashboard*, ktoré sa následne zobrazia do grafov.

3.6 Spracovanie výsledkov

Pre informovanie správcu o stave aplikácie a potenciálnych hrozbách sme zvolili práve tri riešenia. Prvým z nich sú grafy, ktoré tvoria hlavný *Dashboard*. Budú zachytávať vyhodnotenie najčastejšie aplikovaných útokov proti chránenej aplikácii, nebezpečné IP adresy a podobne. Toto riešenie slúži na celkový prehľad o aktuálnych udalostiach a zhrnutie za vybraný časový úsek.

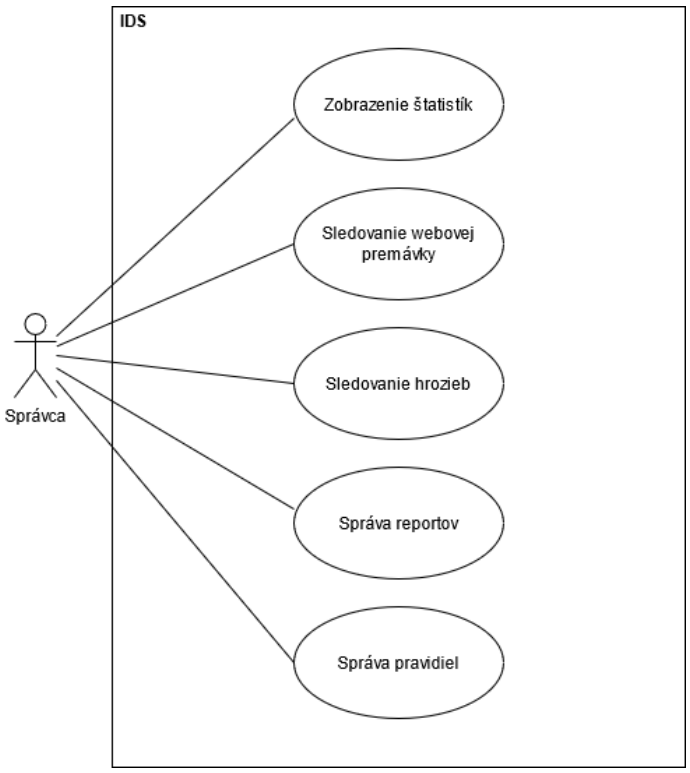
Druhou formou spracovania je prideľovanie tagov spomenutých v sekcii 3.4. Tieto tagy reprezentujú zachytenú hrozbu pomocou definovaného pravidla a každá HTTP požiadavka, so zámerom uškodiť, by mala byť takýmto tagom označená. Takto označené záznamy sa zobrazujú v IDS paneli a je možné v nich prehľadávať a ďalej analyzovať zvolený vektor útoku.

Ďalším riešením je reportovací systém, ktorý má za úlohu v jednom súbore zhrnúť vybrané časové obdobie a vytvoriť report o hrozbách a udalostiach vhodných na ďalšiu analýzu. Nebezpečné HTTP správy zachytené systémom sú neskôr zahrnuté do pravidelných reportov, ktoré systém v cykloch vyhotovuje v požadovanej forme a odosiela na uvedenú e-mailovú adresu.

Všetky tri spomenuté riešenia majú za úlohu čo najefektívnejšie upozorniť správcu o výskyte novej bezpečnostnej udalosti. Každé z nich má svoje výhody a nevýhody ale v kombinácii tvoria systém, v ktorom by mal mať správca prehľad a byť informovaný o každej udalosti pre ďalšiu analýzu prípadne prepracovanie aplikácie alebo jej logiky, aby chyba nebola viac zneužiteľná.

3.7 Návrh používateľského rozhrania

Aplikáciu vieme rozdeliť do základných častí, podľa funkcie, ktorú má napĺňať. Grafické rozhranie by malo byť intuitívne a ľahko zapamätateľné pre jednoduché a rýchle používanie. Diagram nižšie zachytáva prípady použitia, ktoré sú v tejto sekcii neskôr podrobnejšie popísané. Znázorňuje aké akcie môže správca vykonávať pre dosiahnutie svojho cieľa.



Obr. 5: UML Diagram prípadov použitia IDS systému

- P01: Zobrazenie štatistík - Správca sa po kliknutí na záložku *Dahsboard* presunie na hlavnú obrazovku, na ktorej sú zobrazené grafy a štatistiky vyhodnotené z IDS systému
- P02: Sledovanie webovej premávky - Správca je po kliknutí na záložku *Live Traffic* schopný pozorovať a filtrovať aktuálne prebiehajúcu premávku na monitorovanej webovej aplikácii
- P03: Sledovanie hrozieb - Správca je po kliknutí na záložku *Threats* schopný pozorovať a filtrovať všetky zachytené hrozby na monitorovanej webovej aplikácii
- P04: Správa reportov - Správca je po prejdení do sekcie *Report Management* schopný pridávať/odstraňovať parametre reportov, alebo vytvárať nové.
- P05: Správa pravidiel - Po kliknutí na sekciu *Rule Engine* správca môže pridávať alebo odstraňovať REGEX pravidlá.

4 Implementácia IDS

Po dokončení návrhovej časti projektu máme zozbierané dostatočné množstvo informácií a poznatkov o požadovanej funkcionalite, architektúre a celkovom dizajne výsledného *softvéru*. V tejto sekcii sa budeme venovať samotnému vývoju a implementácii *softvéru*, ako aj technológiám použitým na jeho zhotovenie.

4.1 Výber aplikácie na demonštrovanie útokov

Keďže cieľom práce je zachytiť útoky na webovú aplikáciu, budeme potrebovať aplikáciu, ktorá nám poslúži ako cieľ pre útočníka. Pre tento účel sme vybrali už hotovú webovú aplikáciu *Damn Vulnerable Web Application*[6]. DVWA je zraniteľná webová aplikácia, napísaná programovacím jazykom PHP a ako databázu využíva MySQL. Jej cieľom je pomôcť profesionálom zaoberajúcim sa informačnou bezpečnosťou vyskúšať si svoje schopnosti a nástroje v útokoch na webovú aplikáciu tak, aby v skutočnosti nikomu neublížili. Pomáha vývojárom lepšie pochopiť procesy správneho zabezpečenia webových aplikácií.

Zámerom DVWA je precvičiť si útoky na niektoré z najznámejších webových zraniteľností, s rôznymi úrovňami zabezpečenia.

4.2 Použité technológie v IDS

Pri práci sme použili viacero dostupných technológií a pre lepšie pochopenie si ich v tejto sekcii vysvetlíme, pretože sa neskôr v tejto práci budú ešte často spomínať. V nasledujúcich kapitolách dokumentu opíšeme aj použité knižnice a všetky súvisiace časti potrebné pre splnenie požadovanej funkcionality z predchádzajúcej kapitoly.

4.2.1 C++

Na prácu s *paketmi* a sieťovou komunikáciou a teda na tvorbu komponentu agenta, využijeme programovací jazyk C++. Pomocou vhodných knižníc vieme monitorovať TCP prúdy dát a odosielať ich ďalej na spracovanie do IDS webového panelu.

C++ je viac paradigmový programovací jazyk vyššej úrovne na všeobecné použitie, ktorý umožňuje pracovať aj s prostriedkami nízkej úrovne. Má statickú typovú kontrolu, podporuje procedurálne programovanie, dátovú abstrakciu, objektovo orientované programovanie, ale aj generické programovanie. Od 90-tych rokov 20. storočia patrí k najpopulárnejším programovacím jazykom, používa ho až vyše 95% *enginov* počítačových hier.[7]

4.2.2 C++ knižnice

- **libtins :** Pre účely monitorovania sieťovej premávky použijeme knižnicu libtins. Ide o *high-level*, multiplatformovú knižnicu, ktorá slúži na sledovanie a prácu so sieťovými paketmi. Jej hlavným účelom je priniesť vývojárovi možnosť jednoducho a efektívne vytvárať nástroje, ktoré majú za úlohu vytvárať/posielať/prijímať alebo manipulovať sieťové *pakety*.[8]

- **curl :** Na odosielanie zachytených dát použijeme knižnicu curl. Ide o *open source softvér*, používaný ako príkaz v príkazovom riadku alebo v skriptoch kde slúži najmä na prenos dát. Curl sa navyše používa aj v autách, televíznych setoch, smerovačoch, tlačiarňach, audio príslušenstve, mobilných zariadeniach a tak ďalej. Slúži ako jadro prenosu dát pre mnoho webových a iných *softvérových* aplikácií. Najaktuálnejšia verzia je 7.75.0, vydaná tretieho februára 2021.[9]

4.2.3 Java

Už v analytickej časti sme sa rozhodli, že IDS panel bude vyvinutý ako webová aplikácia. Rozhodli sme sa použiť známy programovací jazyk Java, s ktorým už máme nejaké skúsenosti. Pri vývoji v Jave použijeme *Springboot framework*, ktorý bude bližšie popísaný v ďalšej podkapitole. Java a *Spring Boot* poskytuje všetky potrebné prvky pre tvorbu nášho systému.

Java je objektovo orientovaný programovací jazyk. Je vyvíjaný spoločnosťou Oracle. Jeho syntax vychádza z jazykov C a C++. Zdrojové programy sa nekompilujú do strojového kódu, ale do medzistupňa, tzv. *byte-code*, ktorý nie je závislý od konkrétnej platformy. Java patrí k najrozšírenejším a najpoužívanejším programovacím jazykom. Tiež úplne dominuje na platforme Android.

4.2.4 Spring Boot

Spring Boot je projekt postavený na *Spring Frameworku*. Ponúka jednoduchšiu a rýchlejšiu cestu pre založenie, nastavovanie, a celkový chod jednoduchých a webovo orientovaných aplikácií. V skratke je *Spring Boot* kombinácia *Spring Frameworku* a *Embedded* serverov ako napríklad Tomcat, Jetty. *Spring Boot* nevyžaduje XML konfiguráciu a poskytuje množstvo rozširovacích modulov čím výrazne znižuje čas potrebný na vývoj a testovanie aplikácie.[10]

4.2.5 Thymeleaf

Thymeleaf je moderný *engine* na generovanie šablón pre webové aj *stand-alone* prostredia.

Jeho hlavným cieľom je priniesť elegantné a prirodzené šablóny pre vývoj aplikácií - HTML, ktoré bude správne zobrazené v prehliadači a taktiež funkčné ako statický prototyp, čo prináša možnosť lepšej spolupráce v tímoch vývojárov.

S modulmi pre *Spring Framework*, možnou integráciu vlastných nástrojov, možnosťami pridávať inú vlastnú funkcionalitu, je Thymeleaf ideálny pre vývoj HTML5 JVM webových aplikácií a mnoho ďalších.[11]

4.2.6 SQLite

SQLite je *open source* relačná databáza. Pôvodne vydaná v roku 2000, bola navrhnutá aby pre aplikácie poskytovala pohodlný spôsob manažovania dát bez zbytočných nákladov, ktoré prichádzajú s dedikovanými relačnými DBMS. SQLite je známe vďaka svojim hlavným prednostiam ako napríklad jednoduchá prenositeľnosť, jednoduché používanie, kompaktnosť, efektivita a spoľahlivosť. [12]

4.2.7 Knižnica na tvorbu grafov - Chart.js

Ide o jednu z veľmi populárnych a ľahko použiteľných knižníc pre tvorbu grafov. Chart.js poskytuje pripravené modely vizualizácie pre rôzne typy dát s minimom programovania. Takto vytvorené grafy sa automaticky upravujú podľa ostatných elementov a sú responzívne. Knižnica ponúka rôzne druhy grafov ako napríklad histogram, koláčový graf a podobne. [13]

4.2.8 Knižnica na tvorbu Excel dokumentov - GemBox

Ide o komponent, ktorý umožňuje čítať, vytvárať editovať a konvertovať dokumenty pomocou jedného jednoduchého API. Je jednoduchý na použitie a vyžaduje len Javu (od verzie 8), takže sa dá nasadiť rýchlo bez použitia iných licencií. Niektoré z jeho hlavných funkcií sú napríklad : práca so súbormi .xlsx, .xls, .csv, .html a ďalšie, Ochrana a šifrovanie súborov, podporuje aj prácu s obrázkami v dokumentoch a veľa ďalších.[14]

4.3 Funkcionalita IDS

V tejto kapitole a jej častiach vysvetlíme ako sme so spomenutými technológiami pracovali, kde sme ich využili a ako sme pomocou nich naplnili stanovené ciele a návrhy v časti 3 tejto práce.

4.3.1 C++ Agent

Tento komponent nášho systému nie je priamo spojený s webovou aplikáciou. Ide o agenta implementovaného na serveri, ktorý máme v pláne monitorovať, ako je znázornené na obrázku číslo 2. Tento komponent je naprogramovaný v jazyku C++ s použitím dvoch kľúčových knižníc a to libtins a curl, ktoré sme si predstavili v sekcii 4.2.2.

Monitorovanie sieťovej premávky na serveri, kde je tento agent implementovaný je realizované práve pomocou knižnice libtins. Veľká časť agenta bola inšpirovaná práve verejne dostupným príkladom, ktorý vývojári knižnice libtins ponúkajú na svojej stránke ako možný príklad použitia[6]. S použitým kódom vieme sledovať TCP tok dát medzi serverom a klientom. Z tohto toku vyberieme práve dve časti a to zdrojovú IP adresu HTTP požiadavky klienta, a samotnú požiadavku v tvare jedno-riadkového reťazca. Tieto dva parametre z toku medzi klientom a serverom predstavujú dôležitú súčasť v hodnotení rizikovosti komunikácie. V používateľskej HTTP požiadavke budeme neskôr hľadať potenciálne hrozby a pre čiastočnú identifikáciu a rozlíšenie zariadenia, z ktorého útočník požiadavku zaslal, prevezmeme z toku aj IP adresu.

Je dôležité povedať, že agent TCP tok nijak neovplyvňuje ani nemení, jediná akcia nad týmto tokom je čítanie dát. Takto prečítané dáta uložíme do jedného reťazca v tvare i=<ip_adresa>&r=<http_správa>. V systéme môže správa vyzerať nasledovne:

```
i=127.0.0.1&r=GET /dvwa/images/login_logo.png HTTP/1.1
              Host: localhost
              User-Agent: Mozilla/5.0
              (X11; Ubuntu; Linux x86_64; rv:86.0)
              Gecko/20100101 Firefox/86.0
              Accept: image/webp,*/*
              Accept-Language: en-US,en;q=0.5
              Accept-Encoding: gzip, deflate
              Connection: keep-alive
              Referer: http://localhost/login.php
              Cookie: PHPSESSID=1ggpab1sscec0g5lf8madq6plk;
              security=impossible
              If-Modified-Since: Sun, 22 Nov 2020 21:38:28 GMT
              If-None-Match: "2380-5b4b8e5f02e2e"
              Cache-Control: max-age=0
```

Takto vytvorenú štruktúru dát teraz potrebujeme poslať do webovej aplikácie IDS panelu a to komponentu, ktorý tieto dáta očakáva. Ide o *controler* na ceste /api/process, ktorý je hlavným mostom v komunikácii medzi agentom a IDS systémom. Všetky zachytené správy sa posielajú na spracovanie práve tomuto komponentu, kde sú spracované a pripravené pre ďalšiu analýzu. Agent na odosielanie spomenutých dát využíva knižnicu

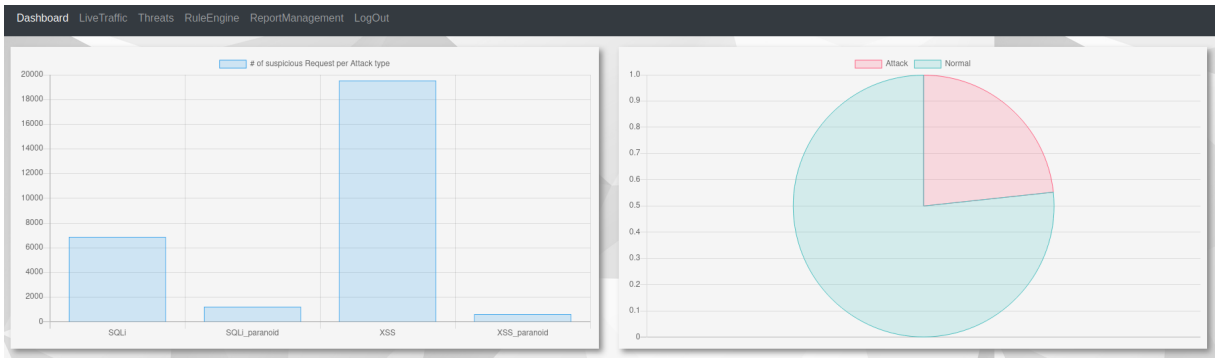
[6]http://libtins.github.io/examples/http-requests/

curl. Pomocou funkcií, ktoré knižnica obsahuje vieme vybrať dáta, ktoré chceme odoslať. Knižnica vytvorí POST požiadavku a do jej tela vloží našu zhotovenú štruktúru, tá je následne odoslaná na zvolenú URL adresu.

Prvým krokom po prijatí správy v IDS paneli je rozdelenie na dve časti, IP adresu a samotnú HTTP správu. Správa sa následne hodnotí všetkými pravidlami v databáze, ktoré majú status *On*. Po získaní skóre sa správa rozdelí na menšie časti a uloží do databázy. O týchto procesoch bude viac informácií v ďalších častiach tejto kapitoly.

4.3.2 Dashboard

Hneď po prihlásení do IDS panelu sa nám zobrazí *dashboard*, ktorý obsahuje niekoľko grafov. Jednotlivé grafy zobrazujú dáta zozbierané systémom a uložené v tabuľkách *stats_1-3* spomenutých v sekcii 3.5 tejto práce.



Obr. 6: Záložka Dashboard v IDS paneli.

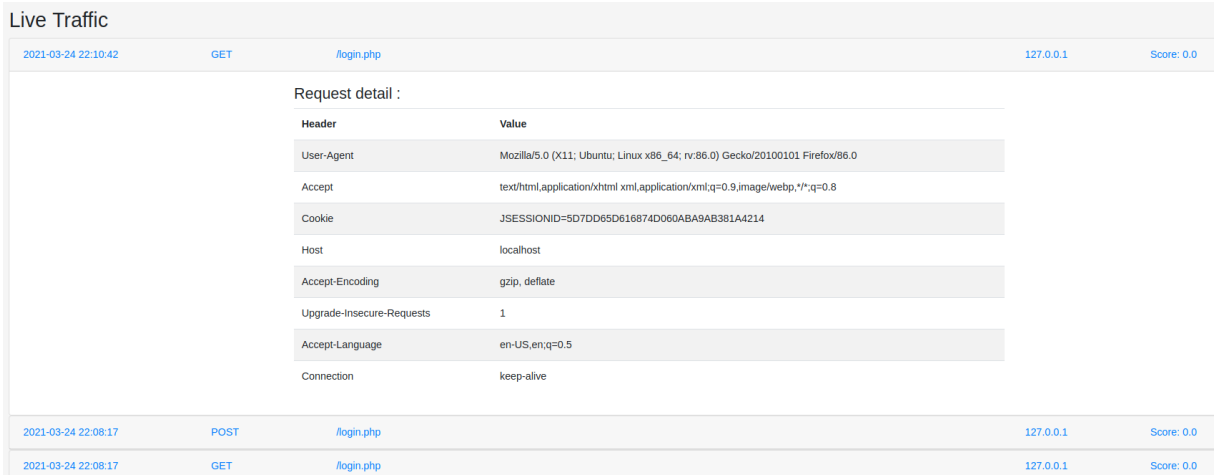
Dashboard má za úlohu pre správcu vytvoriť štatistický obraz o situácii a premávke, ktorú sme zachytili na monitorovanom serveri. Tri pripravené grafy zobrazujú najčastejšie sa vyskytujúce útoky, pomer útokov oproti bežnej premávke a najviac rizikové IP adresy.

Na tvorbu grafov bola použitá JavaScript knižnica Chart.js, ktorá umožňuje veľmi jednoduché grafické spracovanie dát, príklad vidíme na obrázku číslo 6.

4.3.3 Live Traffic & Threats

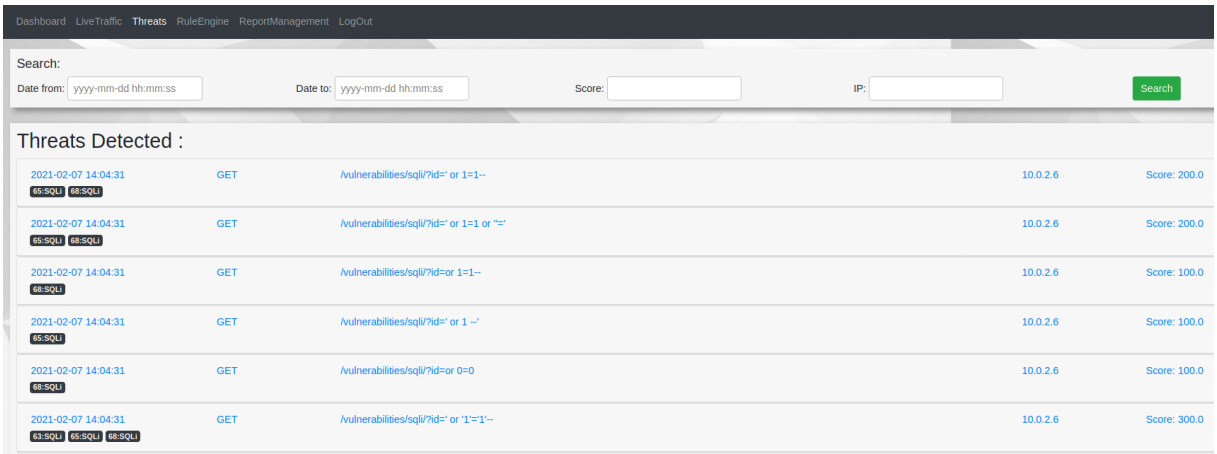
Táto sekcia panelu ponúka správcovi systému možnosti vyhľadávania a analýzy všetkých HTTP správ prijatých IDS systémom. Na detailnejšie prehliadanie správ je k dispozícii filter, v ktorom má správca možnosť vybrať len tie správy, na ktoré sa chce pozrieť detailnejšie. Takto vybrané správy sú potom zobrazené do spodného panelu v takzvaných kartách s možnosťou detailnejšieho zobrazenia konkrétnej karty. Tá obsahuje kompletné informácie o HTTP správe ako je použitá metóda, cieľová cesta, hlavičky a telo správy, ak sa jedná o POST metódu.

Panel *Threats* je konštruovaný rovnakým spôsobom ako panel *LiveTraffic*, avšak s



Obr. 7: Detail karty v paneli *LiveTraffic*

tým rozdielom, že *Threats* pracuje len so správami, ktoré majú skóre väčšie ako 0 a sú teda považované za možnú hrozbu alebo útok.



Obr. 8: Záložka *Threats* v IDS paneli

4.3.4 Rule Engine

Táto sekcia poskytuje rozhranie pre správu pravidiel nášho IDS riešenia. Ako prvú vidíme možnosť pre pridávanie pravidiel. Na to, aby sme pravidlo vytvorili potrebujeme zadať jeho REGEX reprezentáciu ako bolo vysvetlené v sekcii 3.4 tejto práce. Pravidlo taktiež vyžaduje svoj tag. Ten má za úlohu veľmi stručne vysvetľovať, aký útok má pravidlo zachytiť. V našom prípade sme použili tagy:

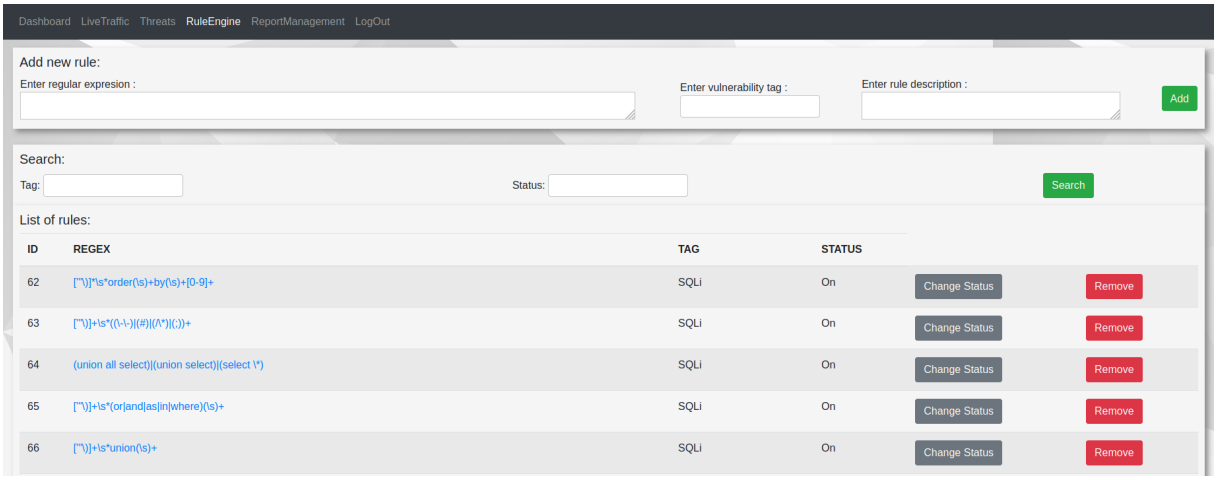
- SQLi - *SQL Injection*
- SQLi_paranoid - *SQL Injection* s pravdepodobnosťou výskytu väčšieho množstva *false-positive* detekcií

- XSS - *Cross-Site Scripting*
- XSS_paranoid - *Cross-Site Scripting* s pravdepodobnosťou výskytu väčšieho množstva *false-positive* detekcií

Vždy, keď IDS systém vyhodnotí skúmanú správu ako podozrivú, pridelí jej tag podľa toho, ktoré pravidlo správu takto vyhodnotilo. Tieto tagy sú neskôr zobrazené ako súčasť kariet v paneli *LiveTraffic* a *Threats* ako vidíme na obrázku číslo 8.

Ako možnosť pri pridávaní pravidla je aj jeho popis, pre jeho rýchlejšie pochopenie v prípade že ho chceme zmazať alebo inak upraviť.

Ďalšou položkou v opisovanom paneli *RuleEngine* je zoznam už vytvorených pravidiel, uložených v databáze. Zobrazuje sa tu celé pravidlo , jeho tag a taktiež jeho status, ktorý hovorí či je pravidlo aktívne alebo neaktívne. To znamená, že pokiaľ pravidlo vytvára príliš veľké množstvo *false-positive* detekcií, máme možnosť ho deaktivovať, úplne zmazať alebo inak upraviť.



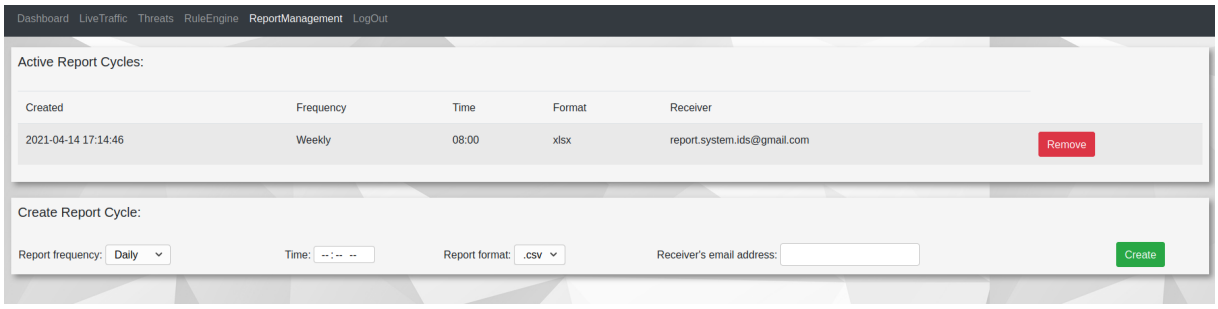
Obr. 9: Záložka *RuleEngine* v IDS paneli.

4.3.5 Report Management

Systém ponúka aj možnosť vytvárania pravidelných reportov, ktoré môžu slúžiť pre ďalšiu analýzu prípadne ako dáta pre algoritmy strojového učenia. Záložka *ReportManagement* slúži práve na správu a vytváranie cyklov pre tvorbu reportov.

Na tvorbu reportov bol použitý komponent GemBox. Ten nám poskytuje rozhranie pre spracovávanie súborov priamo v kóde a tak isto aj možnosť vybrať formát zhotoveného súboru.

Máme možnosť si vybrať či chceme report vytvárať denne, týždenne alebo mesačne, v akom čase a taktiež v akom formáte. Report sa vytvára vždy za obdobie od času založenia



Obr. 10: Záložka *ReportManagement* v IDS paneli.

až po zvolenú možnosť. Posledná možnosť v tvorbe cyklu je e-mailová adresa príjemcu. Na túto adresu bude report doručený.

	A	B	C	D	E	F	G	H	I	J
1	Created	Method	Path	Version	Path params	Headers	Body params	Tags	Source IPv4	Score
2	2021-01-17 0:09:16	POST	/login.php	HTTP/1.1	{}	{Origin= http://localh	{user_token=73e88f3	[TEST]	127.0.0.1	1

Obr. 11: Report vytvorený IDS systémom

Report obsahuje všetky informácie, ktoré sú dostupné aj v záložke Threats. Každá správa vyhodnotená ako hrozba a spadá do vybraného časového úseku, bude zaradená do pravidelného reportu. Príklad prijatého reportu môžeme vidieť na obrázku číslo 11.

5 Výsledky a testovanie

Po úspešnom dokončení implementačnej časti práce, si v tejto kapitole ukážeme priebeh a výsledky testov nášho riešenia. Na prevedenie testov sme použili dva nástroje, ktoré sú v základnej výbave operačného systému Kali Linux a to *sqlmap* a *wfuzz*. V nasledujúcich podkapitolách si bližšie opíšeme zvolený postup testovania ako aj použité nástroje a *wordlisty*.

5.1 SQL Injection

Testovanie detekcie útoku na túto zraniteľnosť sme previedli pomocou OS Kali Linux. Ide o distribúciu Linuxu, zameranú na penetračné testovanie a obsahuje veľké množstvo nástrojov, ktoré môžeme využiť aj pre účel otestovania nášho IDS systému.

Použité nástroje :

- *sqlmap* - je nástroj pre automatizovanú detekciu zraniteľností, hlavne typu *SQL Injection*.
- *wfuzz* - nástroj vytvorený pre *bruteforce* útoky na webové aplikácie.

Kombináciou nástroja *wfuzz* a správne zvoleného zoznamu *payloadov*, vieme simulovať útočníka, ktorý do aplikácie zadáva škodlivé vstupy a tým otestovať v akej miere je náš IDS systém schopný takýto útok detegovať.

Tabuľka 1: Výsledky testov (SQLi) vlastné

Názov nástroja / wordlistu	Počet HTTP správ	Počet zachytených hrozieb	Úspešnosť (%)
seclists/Generic Blind	42	39	92,86
seclists/Generic SQLi	246	151	61,38
seclists/Quick SQLi	77	60	77,92
wfuzz/SQL	125	51	40,80
wfuzz/sql_inj	42	29	69,05
sqlmap / default	146	128	87,67
Spolu	678	458	67,55

V tabuľke 1 môžeme vidieť percentuálnu úspešnosť zachytenia útoku pri jednotlivých

wordlistoch použitých *payloadov*. V prvom stĺpci vidíme názov a zdroj *wordlistu*, v prípade *slqmap* nie je *wordlist* uvedený, nakoľko tento nástroj pracuje aj bez zvoleného *wordlistu*. V tom prípade sme použili nástroj bez zmeny nastavení. V druhom stĺpci môžeme vidieť počet všetkých HTTP správ zaslaných nástrojom *wfuzz* do monitorovanej zraniteľnej aplikácie. Tretí stĺpec znázorňuje počet správ, ktoré naše riešenie označilo ako hrozbu a teda rozoznalo, že ide o útok na aplikáciu. V poslednom stĺpci je zobrazená úspešnosť odhalenia hrozby v percentách.

V niektorých prípadoch pozorujeme úspešnosť nižšiu ako 70% a pri použití *wordlistu* SQL.txt z nástroja *wfuzz* úspešnosť klesla na 40,80%. Dôvodom nízkej úspešnosti je, že niektoré z *wordlistov* obsahujú aj vstupy, ktoré sú príliš podobné bežnému vstupu. Naše pravidlá sú nastavené tak, aby generovali minimálny počet *false-positive* detekcií. Preto reťazce ako napríklad "!", ")", "//", "delete", "having", a podobne, obsiahnuté vo *wordliste* s najnižšou úspešnosťou nepovažujeme za kritické a pre aplikáciu nepredstavujú priame riziko. Máme možnosť do IDS systému doplniť pravidlá tak, aby sme úspešnosť aj v tomto prípade zvýšili, avšak počet *false-positive* detekcií by sa rapídne zvýšil a tým skomplikoval analýzu možného útoku.

Zoznam použitých *wordlistov seclists*[7]:

- seclists/Fuzzing/SQLi/Generic-BlindSQLi.fuzzdb.txt
- seclists/Fuzzing/SQLi/Generic-SQLi.txt
- seclists/Fuzzing/SQLi/quick-SQLi.txt

Zoznam použitých *wordlistov wfuzz*:

- wfuzz/Injections/SQL.txt
- wfuzz/vulns/sql_inj.txt

5.2 XSS Cross Site Scripting

Postup pri testovaní detekcie útoku na túto zraniteľnosť bol rovnaký ako v podkapitole 5.1. Tento krát sme použili len jeden nástroj a to *wfuzz*, ktorý je taktiež opísaný v predchádzajúcej časti. Pri detekcii XSS útoku sme sa zamerali na syntax HTML elementov. Keďže niektoré z použitých *wordlistov* obsahujú len *payload* typu HTML elementov, dokázali sme v niektorom prípade dosiahnuť úspešnosť aj 100%.

[7]https://github.com/danielmiessler/SecLists

Tabuľka 2: Výsledky testov (XSS) vlastné

Názov nástroja / wordlistu	Počet HTTP správ	Počet zachytených hrozieb	Úspešnosť (%)
seclists/BruteLogic	113	83	73,45
seclists/PortSwigger	2537	2537	100,00
seclists/Jhaddix	94	78	82,98
seclists/RSNAKE	68	59	86,76
seclists/Vectors Mario	327	156	47,71
wfuzz/XSS	32	32	100,00
Spolu	3171	2945	92,87

Vidíme že pri tomto teste je celková úspešnosť vyššia ako 90% a pri *wordlistoch PortSwigger* a XSS sa nám podarilo každú škodlivú HTTP správu označiť ako hrozbu. Dôvodom je, že pravidlá na detekciu XSS útoku sme navrhli práve podľa takto zostavených *payloadov*. Opačným prípadom je *wordlist* s názvom *Vectors Mario*, kde vidíme úspešnosť nižšiu ako 50%. Tento *wordlist* obsahuje veľmi atypické *payloady*, pričom mnohé z nich nepredstavujú priame ohrozenie bezpečnosti aplikácie.
Príklad *payloadov* z *wordlistu Vectors Mario*:

```
-->{}
</a>//["'‘-->]]>]</div>
XXX
*['<!--']{}
x='<%'
[B]
```

Tak isto ako pri niektorých prípadoch v predchádzajúcej podkapitole, sme sa aj v tomto prípade rozhodli pre prístup s minimálnym počtom *false-positive* detekcií. Uvedený príklad *payloadov* je teda dobrovoľne prepustený bez označenia ako hrozba. Pri analýze premávky v IDS paneli, je možné však doplniť pravidlo aj pre takéto typy *payloadu*.

Zoznam použitých *wordlistov seclists*:

- seclists/Fuzzing/XSS/XSS-BruteLogic.txt
- seclists/Fuzzing/XSS/XSS-Cheat-Sheet-PortSqigger.txt

- seclists/Fuzzing/XSS/XSS-Jhaddix.txt
- seclists/Fuzzing/XSS/XSS-RSNAKE.txt
- seclists/Fuzzing/XSS/XSS-Vectors-Mario.txt

Zoznam použitých *wordlistov wfuzz*:

- wfuzz/Injections/XSS.txt

5.3 Porovnanie a štatistika

Na porovnanie dosiahnutých výsledkov s niektorým z existujúcich IDS riešení sme zvolili systém Suricata. Ide o pomerne populárny a ľahko dostupný systém navrhnutý na prevenciu alebo detekciu webových ale aj iných útokov. Suricata na detekciu hrozieb využíva podobný systém pravidiel ako náš IDS systém. Priebeh testov ako aj použité nástroje a *wordlisty* sú totožné s nástrojmi v predošlej kapitole tejto práce.

Tabuľka 3: Výsledky testov (SQLi) Suricata

Názov nástroja / wordlistu	Počet HTTP správ	Počet zachytených hrozieb	Úspešnosť (%)
seclists/Generic Blind	42	10	23,80
seclists/Generic SQLi	246	53	21,54
seclists/Quick SQLi	77	2	2,60
wfuzz/SQL	125	6	4,80
wfuzz/sql_inj	42	0	0,00
sqlmap / default	146	0	0,00
Spolu	678	71	10,47

Z výsledkov úspešnosti systému Suricata, v tabuľkách 3 a 4 môžeme pozorovať, že počet zachytených hrozieb je podstatne nižší ako pri našom riešení. Jedným z dôvodov je skutočnosť, že naše riešenie bolo pripravené presne na tieto dva typy útokov a Suricata pokrýva obrovskú škálu iných útokov, zraniteľností a anomálií, ktoré sa môžu v sieťovej premávke vyskytnúť.

Pri testovaní systému Suricata sme pravidlá nijak neupravovali ani nepridávali nové, použité boli len tie zo základného balíka pravidiel, ktoré sú automaticky dostupné hneď

Tabuľka 4: Výsledky testov (XSS) Suricata

Názov nástroja / wordlistu	Počet HTTP správ	Počet zachytených hrozieb	Úspešnosť (%)
seclists/BruteLogic	113	7	6,19
seclists/PortSwigger	2537	186	7,23
seclists/Jhaddix	94	55	58,51
seclists/RSNAKE	68	9	13,24
seclists/Vectors Mario	327	24	7,34
wfuzz/XSS	32	14	43,75
Spolu	3171	295	9,30

po inštalácii a nasadení systému.

Tabuľka 5: Porovnanie úspešnosti testovaných systémov

Zraniteľnosť	Úspešnosť vlastné (%)	Úspešnosť Suricata (%)
XSS	92,87	9,30
SQLi	67,55	10,47

Pravidlá na zachytenie hrozby boli pre náš vlastný IDS systém navrhované a testované aj pomocou testov a *wordlistov* spomenutých v tejto sekcii. Náš systém bol vďaka tomu na tieto typy *payloadov* pripravený čo môžeme pozorovať aj v tabuľke 5, kde dosiahol podstatne lepšie výsledky úspešnosti zachytenia hrozby ako Suricata IDS. Pokiaľ by sme naše pravidlá preniesli do systému Suricata, tak isto by sme dokázali zvýšiť jeho účinnosť v absolvovaných testoch.

Záver

V úvodných častiach práce sme získali prehľad o problematike webovej bezpečnosti, najčastejších útokoch a ich forme a uviedli sme si prečo je dôležité sa tejto téme venovať. Hlavným cieľom bolo účinne detegovať možnú hrozbu alebo prebiehajúci útok na aplikáciu a to práve skúmaním a analýzou sieťovej premávky so zameraním na správy aplikačnej vrstvy.

Vykonaním prieskumu súčasného technologického stavu v tejto oblasti sme získali prehľad o rôznych prístupoch k analýze HTTP správ ako aj o možnostiach implementácie vlastného riešenia. V práci sme sa zamerali na dve konkrétne zraniteľnosti a to *Cross-Site Scripting* a *SQL Injection*. Zhodnotili sme možnosti ich automatizovaného odhalenia a vytvorili sme systém pravidiel, ktorý má za úlohu škodlivé správy detegovať. Následne sme navrhli prehľadné a jednoduché používateľské rozhranie pre správcu, ktoré poskytuje pohodlný prístup k funkcionalite a manažmentu IDS systému. Pre zjednodušenie implementácie je systém navrhnutý vo forme webovej aplikácie čo zvyšuje dostupnosť a pohodlnejší prístup. Systém taktiež obsahuje modul na preposielanie sledovaných HTTP správ do správcovského panelu na spracovanie a analýzu. Všetky dôležité udalosti a bezpečnostné incidenty má správca prehľadne zobrazené v panely či už v podobe grafov, reportov alebo tabuliek.

Pre uvedenie riešenia do produkčného prostredia by bolo potrebné vylepšenie niektorých aspektov aplikácie, hlavne z hľadiska bezpečnosti, keďže vo veľa prípadoch môže dochádzať k spracovávaniu citlivých osobných údajov. Naše riešenie pokrýva dve zraniteľnosti, v praxi sa však stretneme s podstatne väčším spektrom možných zraniteľností. Zaujímavým prínosom by bolo využitie umelej inteligencie so zameraním na detekciu anomálií, ktoré by dokázalo zachytiť aj útoky, ktoré nie sú také populárne a bežne sa na takéto útoky nedokážeme pripraviť.

Zoznam použitej literatúry

1. TECHNOLOGIES, positive. *Web application vulnerabilities: statistics for 2018*. [B. r.]. Dostupné tiež z: https://www.ptsecurity.com/ww-en/analytics/web-application-vulnerabilities-statistics-2019/. [Online, cit. 2021-05-10].
2. OWASP FOUNDATION, Inc. *Top 10 Web Application Security Risks*. [B. r.]. Dostupné tiež z: https://owasp.org/www-project-top-ten/. Online; cit. 14.1.2021.
3. SHEMA, Mike. *Hacking web apps: detecting and preventing web application security problems*. Newnes, 2012.
4. SUNDARAM, Aurobindo. An introduction to intrusion detection. *Crossroads*. 1996, roč. 2, č. 4, s. 3–7.
5. K. K. MOOKHEY, Nilesh Burghate (zost.). *Endpoint Protection: Detection of SQL Injection and Cross-site Scripting Attacks*. https://www.broadcom.com/. Dostupné tiež z: https://community.broadcom.com/symantecenterprise/communities/community-home/librarydocuments/viewdocument?DocumentKey=001f5e09-88b4-4a9a-b310-4c20578eecf9&CommunityKey=1ecf5f55-9545-44d6-b0f4-4e4a7f5f5e68&tab=librarydocuments.
6. DEWHURST, Ryan. *Damn Vulnerable Web Application (DVWA)*. [B. r.]. Dostupné tiež z: https://github.com/digininja/DVWA. Online; cit. 12.2.2021.
7. WIKIPEDIA. *C++*. [B. r.]. Dostupné tiež z: https://sk.wikipedia.org/wiki/C%2B%2B. Online; cit. 12.2.2021.
8. COSTIN, Adrian a ZINCA, Daniel. Extending the libtins library with SIP and RTP classes. In: *2020 International Symposium on Electronics and Telecommunications (ISETC)*. 2020, s. 1–4.
9. STENBERG, Daniel. *Everything curl*. 2018.
10. JAVATPOINT. *What is Spring Boot*. [B. r.]. Dostupné tiež z: https://www.javatpoint.com/spring-boot-tutorial. Online; cit. 12.2.2021.
11. THYMELEAF. *Thymeleaf*. [B. r.]. Dostupné tiež z: https://www.thymeleaf.org/. Online; cit. 12.2.2021.
12. OWENS, Mike a ALLEN, Grant. *The definitive guide to SQLite*. Springer, 2010.
13. DA ROCHA, Helder. *Learn Chart. js: Create interactive visualizations for the Web with Chart. js 2*. Packt Publishing Ltd, 2019.

14. GEMBOX. *GemBox.Spreadsheet for Java: Fast read and write for XLSX, XLS, ODS, CSV and HTML — all from one easy interface.* [B. r.]. Dostupné tiež z: `https://www.gemboxsoftware.com/spreadsheet-java`. Online; cit. 13.2.2021.

Prílohy

A Štruktúra elektronického nosiča

/src

· Zdrojové kódy IDS systému *.java* a *.html*

/request_agent

· Zdrojový kód agenta *.cpp* a konfiguračný súbor *.conf*

/regex_pravidla.txt

· Zoznam navrhnutých REGEX pravidiel

/thesis.pdf

· Písomná časť bakalárskej práce

B Zoznam navrhnutých REGEX pravidiel

```
62 - "['""\)]*\s*order(\s)+by(\s)+[0-9]+"          - SQLi
63 - "['""\)]+\s*((\-\-)|(#)|(/\*)|(;))+"          - SQLi
64 - (union all select)|(union select)|(select \*) - SQLi
65 - "['""\)]+\s*(or|and|as|in|where)(\s)+"        - SQLi
66 - "['""\)]+\s*union(\s)+"                       - SQLi
68 - "(or|and)(\s)+('|"")*[a-z]*[0-9]*('|"")*(\s)*(=|!=|>|<|>=|<=)
     (\s)*('|"")*[a-z]*[0-9]*('|"")*"              - SQLi
72 - sleep\s*\(|exec\s*\(|substr\s*\(              - SQLi_paranoid
74 - <[a-z]+\s+[a-z]+                              - XSS
76 - <script>                                      - XSS_paranoid
77 - <h[1-9]+\s+[a-z]+                             - XSS
```

C Používateľská príručka

Príručka pre implementáciu IDS do vlastného prostredia.

Implementácia IDS riešenia:
Vo vybranom vývojovom prostredí pre Java aplikácie (IntelliJ IDEA - otestované) vytvoríme *Maven* projekt a vložíme /src súbor so zdrojovými kódmi. Doplníme všetky požadované *Maven* závislosti a skompilujeme program. Prostredie IntelliJ IDEA po spustení programu automaticky vytvorí inštanciu Tomcat servera, kde je naša aplikácia už nasadená a štandardne je prístupná na adrese http://localhost:8080/.

Implementácia databázy:
Vytvoríme databázu so štruktúrou uvedenou v kapitole 3.5 tejto práce (SQLite - otestované). Štruktúra tabuliek ako aj názvy tabuliek a stĺpcov sa musia presne zhodovať s tabuľkami na obrázku číslo 4 kapitola 3.5.

Implementácia REGEX pravidiel:
Zo súboru regex_pravidla.txt vložíme všetky pravidlá do databázy, pomocou nasadeného IDS panelu alebo pomocou vývojového prostredia cez SQL príkaz.

Implementácia Agenta:
V priečinku /request_agent potrebujeme skompilovať program request_agent.cpp (g++ - otestované). Na kompiláciu programu potrebujeme kompilátoru nalinkovať 2 C++ knižnice, ktoré program používa a to libtins a curl. Po úspešnej kompilácii môžeme v súbore request_agent.conf nastaviť niektoré z parametrov ako napríklad sledovaný *interface* a iné. Vytvorený spustiteľný program po kompilácii spustíme na zariadení, kde sa nachádza aplikácia, ktorú chceme monitorovať a určíme *interface*, ktorý sa bude monitorovať.