

Pannon Egyetem
Műszaki Informatikai Kar
Villamosmérnöki és Információs Rendszerek Tanszék
Programtervező informatikus BSc szak

SZAKDOLGOZAT

Weboldal defacement detektáló eszköz fejlesztése

Pap Dávid

Témavezető: Dr. Szücs Veronika

Belső konzulens: Arányi Gábor

2023



PANNON EGYETEM

MŰSZAKI INFORMATIKAI KAR

Programtervező informatikus BSc szak

Veszprém, 2023. március 29.

SZAKDOLGOZAT TÉMAKIÍRÁS

Pap Dávid

Programtervező informatikus BSc szakos hallgató részére

Weboldal defacement detektáló eszköz fejlesztése

Témavezető: Dr. Szücs Veronika, egyetemi docens
Belső konzulens: Arányi Gábor, Ph.D hallgató

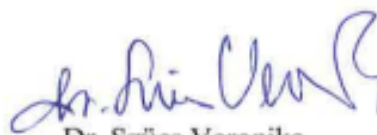
A feladat leírása:

A defacement az egyik legelterjedtebb támadási technika, amellyel gyakran elérhetetlenné teszik az eredeti weboldalakat, ezzel egyidejűleg támadó, adathalász webhelyre irányítják a felhasználókat, megsértve ezzel az oldal tulajdonosának és szolgáltatásait használó ügyfeleinek az üzleti érdekeit. A feladat a defacement támadásokat detektáló leggyakrabban használt szoftverek felkutatása, vizsgálata és összehasonlítása. A fejlesztendő program feladata, hogy mini-reverse proxyként a HTTP kommunikációt vizsgálja, továbbítsa a kiszolgálónak, és a kéréseket reguláris kifejezések segítségével szűrje. Az IP címek szűrésére használjon online blocklist-eket, emellett valósítsa meg a saját azonosítás alapján regisztrált potenciális támadó címek lokális tárolását a későbbi szűrésekhez. A feladat megvalósítása során vizsgálja meg a gépi tanulás technológia beépíthetőségének lehetőségét és korlátait a mini-proxy alkalmazásba.

Feladatkiírás:

- Vizsgálja meg a defacement típusú támadások legjellemzőbb technikáit,
- A közelmúlt felderített defacement támadásai alapján vizsgálja meg a működésükben megjelenő új vonásokat,
- Tervezzen meg egy lehetséges proxy kiszolgálót a forgalom előzetes szűrésére,
- Vizsgálja meg a gépi tanulás technológia alkalmazásának lehetőségeit, korlátait,
- Valósítsa meg és tesztelje az elkészített megoldást,
- Dokumentálja a teljes tervezés, megvalósítás és tesztelés folyamatot, valamint az eredményeket!


Dr. Süle Zoltán
egyetemi docens
szakfelelős


Dr. Szücs Veronika
egyetemi docens
témavezető

Hallgatói nyilatkozat

Alulírott Pap Dávid hallgató kijelentem, hogy a dolgozatot a Pannon Egyetem, Műszaki Informatikai Kar Villamosmérnöki és Információs Rendszerek Tanszékén készítettem a Programtervező informatikus BSc végzettség megszerzése érdekében.

Kijelentem, hogy a dolgozatban lévő érdemi rész saját munkám eredménye, az érdemi részen kívül csak a hivatkozott forrásokat (szakirodalom, eszközök stb.) használtam fel.

Tudomásul veszem, hogy a dolgozatban foglalt eredményeket a Pannon Egyetem, valamint a feladatot kiíró szervezeti egység saját céljaira szabadon felhasználhatja.

Dátum: Veszprém, 2023.05.02



Pap Dávid

Témavezetői nyilatkozat

Alulírott Dr. Szücs Veronika témavezető kijelentem, hogy a dolgozatot Pap Dávid a Pannon Egyetem Pannon Egyetem, Műszaki Informatikai Kar Villamosmérnöki és Információs Rendszerek Tanszékén készítette Programtervező informatikus BSc végzettség megszerzése érdekében.

Kijelentem, hogy a dolgozat védeésre bocsátását engedélyezem.

Dátum: Veszprém, 2023.05.09



Dr. Szücs Veronika

Köszönetnyilvánítás

Elsőként szeretném megköszönni a családomnak és barátaimnak, akik támogattak engem azáltal, hogy mellettem álltak, biztattak és néha hasznos tanáccsal láttak el. Továbbá köszönettel tartozom a témavezetőmnek, Szücs Veronikának.

Tartalmi összefoglaló

Szakedolgozatom témájának egy weboldal defacement támadást detektáló eszköz fejlesztését választottam, melynek feladata többek között az injekciós támadásoktól megvédeni a webszervert. Ez azért is fontos terület napjainkban, mert a felhasználói adatok védelme az egyik legfontosabb tényező a webes alkalmazások számára és egy WAF nagyban növelheti alkalmazásunk biztonságát. Mivel az injekciós támadások jelentik a legnagyobb veszélyt a webes alkalmazásokra, ezért ezek részletes áttekintése után hozzáálltam a feladat megvalósításához. A fejlesztett szoftver tartalmaz megoldást a főbb támadást vektorok kivédésére, mint például az SQL injection, az XSS vagy a Prototype Pollution.

A szoftver megoldásom egy WAF-proxy formájában készült el, ami a kéréseket továbbítja a kiszolgáló webszerver felé, majd továbbítja a válaszokat a kliens oldalra. Szakedolgozatomban részletesen bemutatom a WAF megoldások főbb koncepcióit és az alkalmazott technikákat. A védelem szabad megválasztása érdekében egy konfigurációs fájlban személyre lehet szabni a programot a kívánt beállításokkal. Továbbá a felhasználók számára a visszajelzést egy naplófájl és egy IP adatbázis segíti. A naplófájlban a program rögzíti az esetleges behatolási kísérleteket, illetve az IP adatbázisban menti a támadó IP címét, szükség esetén online ellenőrzi, hogy a cím nem szerepel-e feketelistákon.

A védelem tartalmaz egy megoldást, ami mintákat keres a kérésben, illetve egy gépi tanulást alkalmazó megközelítést. Elvégeztem mindkét megoldás részletes kiértékelését és tesztelését, továbbá írtam a gépi tanulás további lehetőségeiről a területen. A tervezés, megvalósítás és tesztelés folyamatát részletesen ismertettem.

Kulcsszavak: WAF, Injekciós támadások, Gépi tanulás, Biztonság

Abstract

I chose the development of a website defacement attack detection tool as the topic of my thesis, the task of which is to protect the web server from injection attacks. This is also an important area nowadays, because the protection of user data is one of the most important factors for web applications and a WAF can greatly increase the security of our application. Since injection attacks pose the greatest threat to web applications, I began implementing the task after a detailed review on them. The developed software includes a solution to prevent the most common attack vectors, such as SQL injection, XSS, and Prototype Pollution.

My software solution took the form of a WAF proxy, which forwards requests to the server web server and then forwards responses to the client. In my thesis, I present the main concepts of WAF solutions as well as the techniques used in detail. The program can be customized with the desired settings in a configuration file in order to freely choose the protection. A log file and an IP database also assist users with feedback. The program records any intrusion attempts in the log file, saves the attacker's IP address in the IP database, and if necessary, checks online whether the address is on blacklists.

Protection includes a solution that searches for patterns in the request as well as a machine learning approach. I thoroughly evaluated and tested both solutions, and I also wrote about the future possibilities of machine learning in the field. I've gone over the planning, implementation, and testing processes in great detail.

Keywords: WAF, Injection attacks, Machine learning, Security

Tartalomjegyzék

Jelölésjegyzék	10
1. Bevezetés.....	11
1.1. Kiterjedés	12
1.2. Korábbi precedensek	12
2. Injekciós támadások elemzése	13
2.1. XSS	13
2.2. SQLI	15
2.2.1. NoSQL injekció	16
2.2.2. ORM Injekció	16
2.3. Command Injection	16
2.3.1. LDAPi	17
2.4. Prototype Pollution	17
2.5. Egy injekciós támadás esettanulmánya	18
2.6. Konklúziók	19
2.6.1. Az injekciós támadások jövője	20
3. A WAF mint megoldás.....	22
3.1. In-Line és Out-of-Line megvalósítások	22
3.2. Védelem típusa	23
3.3. Egyéb feladatok	24
3.3.1. Integráció más biztonsági eszközökkel	24
3.3.2. Nemzetközi viszonylatban.....	25
3.3.3. Hazai viszonylatban.....	25
3.3.4. Limitációk.....	26
4. A szoftver bemutatása.....	26
4.1. Célkitűzések és követelmények	26
4.2. Tervezés	27
4.2.1. Proxy modul	28
4.2.2. Analízis modul.....	29
4.2.3. Egyéb tervezési feladatok	30

4.3.	Megvalósítás	31
4.3.1.	Proxy modul	32
4.3.2.	IP intelligence	37
4.3.3.	Szűrőimplementációk	37
4.4.	Konfiguráció, nyomon követés és adattárolás	39
4.5.	Feladatmegvalósítási nehézségek	40
4.6.	Az elkészített program SWOT elemzése	41
4.7.	Összegzés.....	42
5.	A gépi tanulás lehetőségei	42
5.1.	Adatok gyűjtése	42
5.1.1.	Overfitting és underfitting jelensége	43
5.2.	Feature engineering	43
5.2.1.	CountVectorizer.....	44
5.3.	Modellezés.....	44
5.3.1.	SGD	44
5.3.2.	XGB.....	44
5.3.3.	LinearSV.....	45
5.4.	Tesztelés és finomítás	45
5.4.1.	Kimenetek besorolása.....	46
5.4.2.	Accuracy	46
5.5.	Összegzés.....	47
6.	Tesztelés.....	48
6.1.	Tesztadatok előkészítése.....	48
6.2.	Aláírásalapú megközelítés	48
6.3.	Szabályalapú megközelítés	49
6.4.	Végtesztelés	49
7.	Továbbfejlesztési lehetőségek.....	50
7.1.	Adat menedzsment.....	50
7.2.	Skálázhatóság	50
8.	Zárójegyzetek.....	51
8.1.	Felhasználás.....	51

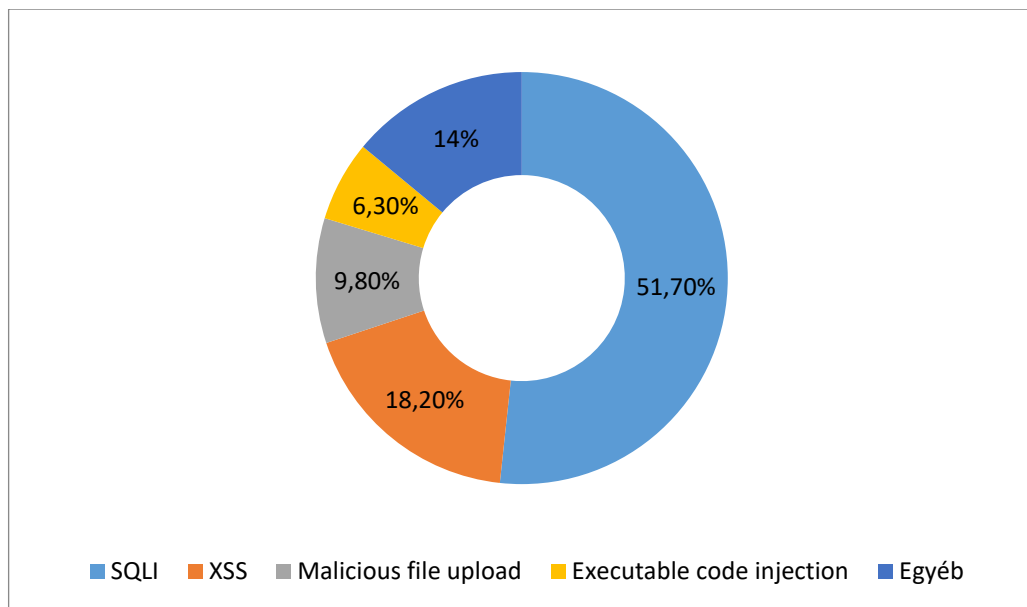
8.2. Összegzés.....	51
Irodalomjegyzék.....	52
Mellékletek	54
Ábrajegyzék.....	55
Táblázatjegyzék	56

Jelölésjegyzék

PWA	Progressive Web Application (Progresszív Web alkalmazás)
OWASP	Open Worldwide Application Security Project (Nyílt-világ alkalmazás kiberbiztonsági szervezet)
SQLI	SQL Injection
XSS	Cross Site Scripting
CDN	Content Delivery Network (Tartalomküldő hálózat)
WAF	Web Application Firewall (Web Alkalmazás Tűzfal)
WP	Wordpress
HTTP	Hypertext Transfer Protocol (Szövegátviteli protokoll)
OS	Operating System (Operációs rendszer)
RFC	Request For Comments (Kommentáláskérés)
DOM	Document Object Model (Dokumentum Objektum Modell)
ORM	Object Relational Mapping (Objektum Reláció Térképezés)
LDAP	Lightweight Directory Access Protocol (Pehelysúlyú Mappahozzáférési Protokoll)
JSON	JavaScript Object Notation (JavaScript Objektum Notáció)
API	Application Programming Interface (Alkalmazásprogramozási Felület)
MI	Mesterséges intelligencia
DDoS	Distributed Denial-of-Service (Eloszott túlterheléses támadás)
SSL/TLS	Secure Sockets Layer/Transport Layer Security (Titkosítási réteg)
MITM	Man In The Middle (Közbeékelődéses támadástípus)
AWS	Amazon Web Services
TCP	Transmission Control Protocol (Internetes átviteli protokoll)
DVWA	Damn Vulnerable Web Application (Sebezhető webes applikáció)
SGD	Stochastic Gradient Descent
XGB	eXtreme Gradien Boost
SVM	Support Vector Machine

1. Bevezetés

Az internet elterjedésével a Web technológiák váltak dominánssá, legyen szó böngészőben megjeleníthető applikációról vagy akár PWA-ról. Az aktív felhasználók száma is napról napra nő, 2022 júniusában ez a szám majdnem 5.5 milliárd volt [1]. Ma már minden vállalatnak rendelkezni kell egy Webes elérhetőséggel, ezért is fontos ezen applikációk védelme. Sajnos azonban a gyakorlatban ez nem mindig valósul meg kellőképp, egy OWASP kutatás [2] szerint a vizsgált oldalak 3%-án találtak valamifajta sebezhetőséget. A legelterjedtebb támadási formák közé tartozik az SQL Injekció, az XSS (Cross Site Scripting), kártékony fájlok feltöltése, valamint a végrehajtható kódrészletek beillesztése, ahogy a következő 1. ábra is mutatja [3]:



1. ábra Támadások gyakorisága [3]

A közelmúltban történt eKréta hack is rámutatott, milyen kódös terület a fejlesztőknek az XSS, vagy akár az SQL Injekció. Ugyanis habár a kódrészlet nyújt némi védelmet a legelterjedtebb kihasználhatóságok ellen, rengeteg más mód létezik injektálható kódok beillesztésére. 2020-ban minden vállalatra átlagosan 270 támadás jutott, ezek ellen például védelmet nyújtanak a CDN szolgáltatók (pl. CloudFlare) beágyazott WAF megoldásai. Ezeket azonban csak a nagyobb méretű Web applikációk tudják igénybe venni és nem is nyílt forráskódúak. Mégis a legnagyobb veszélyt a kezdő weboldal tulajdonosokra jelentik az injekciós támadások.

Elterjedt a kezdő weboldal üzemeltetők körében a WordPress egyszerűsége és közösségi támogatása miatt. Nem nagy meglepetés, hogy a ma használatos weboldalak 43.2%-a WordPress-t használ. A veszélye pedig éppen az egyszerűségében rejlik, ugyanis mivel bárki közzé tehet pluginokat (kiegészítő tartalmakat) akaratlanul sebezhetőségeknek teheti ki a weboldat, ha nem figyel a beviteli mezők megfelelő megtisztítására. A WordPress oldalak esetén a legelterjedtebb sebezhetőség az XSS, sajnos itt a legnehezebb a megelőzés, mert nem tudni melyik plugin tartalmaz sebezhetőségeket. Elterjedtségéből adódóan pedig ez egyáltalán nem ritka jelenség, ugyanis átlagosan napi 10 WordPress sebezhetőség jelenik meg [4]. Ez nem csak plugin-okat érint, de a témákat is, ritkább esetben magát a WordPress alapszoftvert.

1.1. Kiterjedés

Habár a támadás módszerétől függ a kiterjedés, szélsőséges esetben akár a szervergépen kívül, a kapcsolódó hálózaton lévő kiszolgálókat elérve okozhat további károkat. Az injekciós támadásokat szintekre osztottam annak tekintetében, hogy mire károsak. A négy fő szint (környezetekkel a hatókört definiálva), amit a későbbiekben ki is fejtek részletesen:

- Adatbázis-támadások (MySQL, MSSQL)
- OS szintű támadások (Windows, OS X)
- Felhasználói szintű támadások (Chrome, Firefox)
- Futtató környezeti támadások (Node.JS, Apache)

Mint azt a lista is mutatja, a szervergép felett majdnem teljes kontrollt szerezhetünk az említett támadásokkal. A legnagyobb veszélyt az OS és adatbázis szintű támadások jelentik a szervergépen, illetve az adatbázisban tárolt szenzitív adatok miatt.

1.2. Korábbi precedensek

2015-ben az Impact Team nevű csoport tagjai az Ashley Maddison nevű oldalhoz szereztek hozzáférést. Az oldal érdekessége, hogy hűtlen házastársaknak kínált alkalmi társkeresési lehetőséget. A támadás során, amit feltehetően SQL injekció révén hajtottak végre az adatokhoz hozzáférve közzétették a felhasználók nevét és elérhetőségét.

A jelenleg is folyamatban lévő Orosz-Ukrán háborúban is teret kapott a defacement. A legprominensebb eset a háború kitörésének évfordulója alkalmából történt, amikor 32

orosz weboldalon a látogatókat egy videó az égő Kreml-ről fogadta [5]. Ebben az esetben azonban nem csak weboldalakat torzítottak el, hanem rádióadásokat is, ahol bombariadókat és téves információkat sugároztak.

2. Injekciós támadások elemzése

Az injekciós támadások nagyon gyakoriak és súlyos biztonsági kockázatot jelentenek. Egy 2020-as OWASP jelentés szerint például a webes alkalmazásokat ért támadások közel 45%-a injekciós támadás volt. A támadások közös pontja, hogy a HTTP kérések részeiben érkeznek, a fogadó webszervernek. A kérés mindhárom részében érkezhetnek kártékony karakterláncok, legyen szó a státusz sorról (status line), a fejlécekről (header) vagy az üzenet testéről (body). Általában azonban a fejlécek csak egy limitált száma szokott a kihasználhatóságok kiindulópontjaként szolgálni. Mivel a bejövő forgalmat vizsgáljuk a kérés (Request) típusú, illetve az általános mezők érdekesek számunkra. Ezek közül leggyakrabban az Authorization, Cookie és Form fejléceket használják a felhasználói oldalról való adatközléshez. Természetesen a protokoll felépítése miatt használhatnak a webapplikációk saját fejléceket, ezt általában „x,-szel kezdik egy RFC szabvány megkötése szerint [6]. Ez a kliensoldalról (amely forgalmat ellenőrizzük) általában nem jellemző. A következő szekciók célja, hogy egy rövid áttekintést adjon a nevezetesebb injekciós támadásokról és azok működéséről.

2.1. XSS

Az XSS vagy Cross Site Scripting egy ismert és elterjedt támadási módszer, aminek a lényege, mint általában az injekciós támadásoknak a tisztítatlan felhasználói bemeneten érkező kód futtatása. Hogy megismerjük miért lehetséges ez, a böngészők alapvető viselkedését kell átlátnunk. A böngészők egy úgynevezett renderelő motort használnak a weboldalak értelmezéséhez és megjelenítéséhez az előírt szabványoknak megfelelően. A renderelő motor értelmezi az HTML-t és CSS-t majd végrehajtja a JavaScript kódot. Ez a folyamat számos lépést tartalmaz, mint például a Document Object Model (DOM) fa létrehozását és a weboldal végső elrendezésének megjelenítését. Mivel az egész HTTP válaszüzenet egy karakterlánc formájában érkezik könnyű belátni, hogy a dinamikus, felhasználók által adott tartalmat is értelmezheti a böngészőnk futtatandó kódként.

Számtalan felhasználási módja létezik, többek között kitűnően alkalmas például a weboldal eltorzítására vagy a felhasználók sütijeinek (cookies) ellopására.

```
<script>alert(document.cookie)</script>
```

Ez elsősorban a gyanútlan felhasználókra veszélyes. Három kategóriára osztható a támadás, de ebből számunkra csak a tárolt (stored) és tükrözött (reflected) fontos. A harmadik fajta (DOM alapú XSS) azért nem érdekes, mert abban az esetben nem történik kommunikáció a webszerver felé, hanem lokálisan módosítják a felhasználók a már számukra megjelenített modellt (DOM). A tükrözött XSS gyakori példája mikor egy URL-re kattintva a felhasználó akaratlanul a kérést előre megszabott káros paraméterekkel adja le a webszerver felé, így mikor a böngésző megjeleníti a válaszként kapott oldalt a nem kívánatos paraméterekkel ezeket kódként értelmezve lefuttatja a felhasználói oldalon.

```
example.com/search?q=<script>alert('XSS')</script>
```

Ennél veszélyesebb a tárolt XSS, ami (ellentétben a tükrözötttel) a káros karakterláncot el is tárolja egy adatbázisban és a többi gyanútlan felhasználónak továbbítja, akiknek a böngészője le is futtatja azt. Általában HTML címkék beszúrásával valósítják meg a támadók. Hogy egy ismerős példát hozzak az eKréta *XSSPrevention()* függvénye véd az iframe, object, embed és script címkéktől. A fejlesztők temérdek más hiba mellett elfelejtették, hogy ez csak a töredéke a kihasználható jelöléseknek és többek között az elterjedt img címke ellen sem nyújt védelmet, aminek bármely attribútumát lehet használni ezáltal. Habár elsőre nem is gondolnánk, de a karakter kódolás változtatásával is ki lehet játszani a különböző szűrőket, szerencsére ma már ez egyre limitáltabb és kisebb szerepet játszik, mint ezelőtt, a böngészők fejlődésének köszönhetően.

```
<img src=#  
onerror=this.src=\"http://evilurl.org/?cookie=\"+document.cookie />
```

2.2. SQLI

Az SQL injekció veszélyessége abban rejlik, hogy a fertőzött felhasználói bemenetet SQL lekérdezés formájában a szervergépen lefuttatjuk. Amennyiben nincs rendesen megtisztítva a karakterlánc tartalmazhat karaktereket, amik a vártnál hamarabb lezárhatják a lekérdezést és saját kódjukat beillesztve futtatják tovább azt. Egy egyszerű példa:

```
'UNION SELECT username,password FROM users--
```

Ezáltal veszélynek kitéve az egész adatbázist. Ennek elkerülése végett ajánlott a bemenet megtisztítása, tárolt eljárások használata, ORM (Object Relational Mapping, ez a következő témakör), illetve még itt is lehetséges az adatbázis és a webszerver közé egy köztes WAF-ot implementálni, ami szűri a nem kívánatos kéréseket amennyiben a két alrendszer HTTP kommunikációt folytat. A támadások közül is ez a legelterjedtebb és a legnagyobb múlttal rendelkező. A weboldalak tesztelésére szerencsére léteznek szofisztikált eszközök, mint például az sqlmap, amik kérések sorozatával ellenőrzik a webhelyet. Szerencsére ezen próbálkozások detektálásában pont az segít, hogy még az adatbázisszoftver megállapításához is több kérés kell. A támadásnak két altípusa létezik (sok más csoportosítás mellett), a célzott (targeted) és a vak (blind). A lényegük, hogy a vak SQLI esetén a támadó nem kap direkt visszajelzést a kód lefutásáról, ezért a visszajelzési idő vagy igaz-hamis lekérdezések különbsége alapján folytatja tovább a kihasználhatóságok kiaknázását. Egy híres példa [7] amikor egy hacker egy pénzügyi szervezettől több mint 130 millió dollárt lopott, miután az adatbázisukat kedve szerint formálta egyszerű SQL lekérdezésekkel.

2.2.1. NoSQL injekció

Fontos megemlíteni, hogy a NoSQL rendszerekre is léteznek injekciós támadások. Ezek azonban jóval komplexebbek, a NoSQL rendszerek felépítéséből és az azzal való kommunikációból adódóan. Ebben szerepet játszik, hogy az SQL rendszerekkel ellentétben itt nincs egy sztenderd nyelv amire a lekérdezések épülnek. A lekérdezéseket formáját befolyásolják a(z):

- Adatbázis motor (MongoDB, Redis)
- Programozási nyelv (Python, Javascript)
- Keretrendszer (React, Node.js)

A közös jellemző a használt átviteli közeg, ami általában JSON, ritkább esetben XML. Egy példa, amiben az összes tárolt felhasználó nevét visszaadjuk:

```
db.users.find({'$ne: null'}));
```

a \$ne operátor kikeres minden nem null-lal egyenlő nevű felhasználót (not equal).

2.2.2. ORM Injekció

Az Object Relational Mapping többek között az SQL injekció egyik védelmi formájaként jött létre. Működését tekintve egy köztes közeget biztosít, ahol biztonságosan és szeparálva lehet lekérdezéseket összeállítani azáltal, hogy deklarált változókkal végzi a lekérdezéseket a szervertől a megszokott lekérdezés karakterláncoktól különbözően. Sajnos azonban a védelem nem mindig történik meg kellőképpen, így történt ez a népszerű Sequelize könyvtár esetében is [8]. Működését tekintve egy SQLI támadásnak felel meg.

2.3. Command Injection

A Command Injection mint azt a neve is sugallja, távoli kód futtatására biztosít lehetőséget. Ezeket a kódrészleteket, amiket a webservert részére küldenek a webalkalmazás a lokális rendszer shell-jében futtat. Ez a támadás a legveszélyesebb mert közvetlen a célrendszerhez szereznek hozzáférést a támadók és szabadon manipulálhatják azt.

Ha például egy keresés a webes API-n keresztül a rendszer shell-ben fut a következő kódrészlet visszaadja az ott található fájlok listáját.

```
' ; ls -la ; '
```

Legegyszerűbb megelőzése, hogy nem engedjük a felhasználóktól érkező bemenet közvetlen futtatását, vagy white list-ek alkalmazása.

2.3.1. LDAPI

Az LDAP (Lightweight Directory Access Protocol) injekció a rendszerek autentikációs lehetőségeinek kihasználhatósága nyomán jött létre. Mikor egy külső web applikáció a felhasználói adatok validálásához egy már létező például Windows szerver bejelentkezési szolgáltatását használja egy karakterlánc eljuttatásával. A következő példa egy olyan scenáriót mutat be, ahol minden tetszőleges fiókba szabadon beléphetünk.

```
admin)(password=*))(|(objectclass=*)
```

Működésében hasonlít az SQL injekcióra, de sokkal veszélyesebb mert itt nem csak az adatbázishoz fér hozzá a támadó, hanem az egész szervergépet tudja kezelni és ezáltal ott tetszőleges parancsokat lefuttatni. Szükséges esetben az egész helyi hálózati infrastruktúrát eléri a támadó. Ezek a támadások szerencsére annyira nem gyakoriak, mint az XSS vagy SQLI.

2.4. Prototype Pollution

A prototípus szennyezés (Prototype Pollution) egy viszonylag új és pont ezért sokak számára ismeretlen injekciós támadás, ami főleg JavaScriptben írt applikációkra veszélyes. Általában arra a szerveroldali kódra, ami adatkinyerést végez a kérésekből. Mert ha a JSON objektumot átalakítjuk objektummá annak a nem kívánatos részeihez is hozzáférhetnek a kérés küldői. Mivel a nyelvben az objektumok az öröklési elvek mezsgyéjén funkcionálnak a különböző objektumokból el lehet érni az őseiket a `__proto__` mező segítségével. Ezáltal az összes őst és leszármazottat kockázatnak kitéve.

```
{"__proto__": {"admin":true }}
```

Ez a támadás viszonylag új és a NodeJs térhódítása miatt egyre elterjedtebb, a jövőben pedig valószínűleg még elterjedtebb lesz. Szükséges esetekben mivel magához az objektumokhoz férnek hozzá, tetszésük szerint tudnak parancsokat futtatni a szervergépen. Különösen veszélyes ez a támadásforma mert a bemenet megtisztítása sem segít a problémán (amennyiben objektumként tároljuk érkezése pillanatában), ugyanis

már az információ megérkezése és változóba eltárolása pillanatában létrejött a fertőzött objektum, ennek következtében az összes leszármazott is megfertőződött a prototípus mező miatt. Mivel az objektum beérkezési fázisában nem igazán gondolnak a fejlesztők erre az eshetőségre itt meglehetősen jó ötlet egy WAF implementálása, ami folyamatosan ellenőrzi kéréseket a hasonló eshetőségek végett. Leginkább a kisebb webapplikációknál jelentkezik ez a jelenség és újabban egy Lodash nevű könyvtárnál, ami az összes 2020 előtti verziót érinti.

2.5. Egy injekciós támadás esettanulmánya

Hogy megértsük miként is működik egy támadás, egy újonnan felfedezett biztonsági rést mutatok be példaként. A kihasználhatóságot 2023 április 27-én fedezte fel Iyaad Luqman K, ami egy SQL injekciós támadási rés volt. Ez a kihasználhatóság a ChurchCRM 4.5.1-es verzióját és az összes ezelőtti verziót érintette, ami egy WordPress kiegészítő. Jól bizonyítja ez is, hogy a WP kiegészítők rengeteg sebezhetőségnek adnak teret, illetve az általam vizsgált kihasználhatóságok jelentős része WP plugin volt.

A kihasználhatóság az URL event paraméterében valósult meg, amiben a helytelen kezelés miatt bármilyen SQL scriptet lefuttatott a kiegészítő, amit Iyaad meg is tett. A

```
Action=List&Event=2+UNION+ALL+SELECT+1,NULL,CONCAT(%27Perseverance%27,usr_Username,%27:%27,usr_Password),NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL+from+user_usr--+&Type=Sunday%20School
```

paraméterek segítségével a WordPress szerver adatbázisa eldobja a usr_Username és az usr_Password táblákat, ezzel elveszítve az összes hitelesítési lehetőséget.

Sajnos az, hogy felfedezték a kihasználhatóságot még nem jelent megoldást a problémára, ugyanis a WP oldalak, amik ezt használják újabb verzióra frissítés előtt még ugyanúgy sebezhetők. Ez a kihasználhatóság csak frissítés után szűnik meg, és mivel ez nem történik meg minden esetben, továbbra is maradnak sebezhető oldalak az interneten emiatt.

2.6. Konklúziók

Az injekciós támadások jelentik a legnagyobb veszélyt a webes applikációkra ezért fontos ellenük védekezni. A védelem az ajánlottak szerint három módon történhet meg:

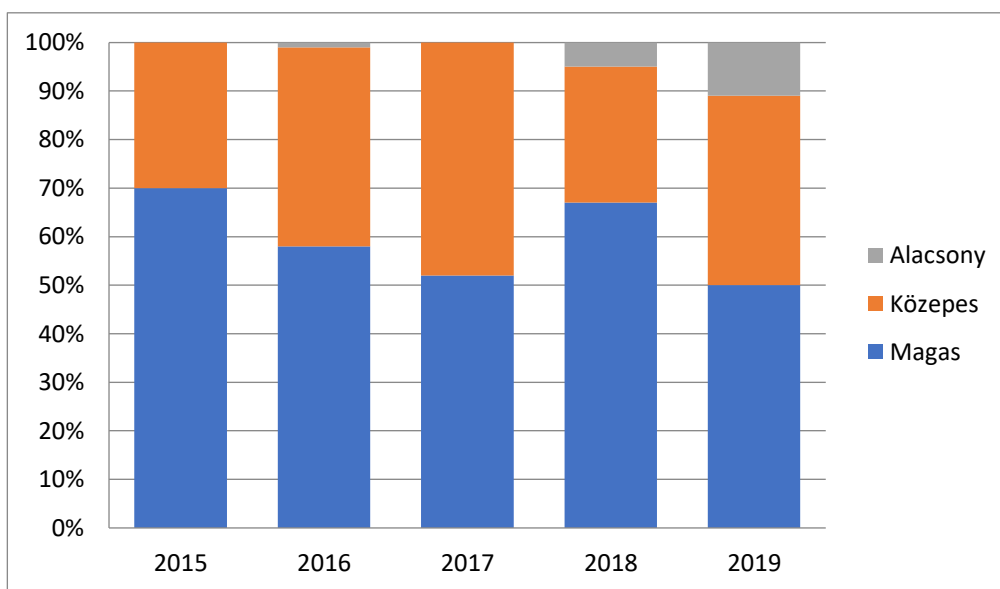
- Megfelelő bemeneti validációval „megengedő listákkal”
- Biztonságos API-val
- Speciális karakterek kicserélése

Megoldást jelentenek a problémára az előkészített metódusok (Prepared Statements), amik már előre megírt metódusok a különböző kihasználhatóságok megelőzése érdekében. Előfordulhat azonban, hogy ezek csak egy adott problémára vannak megírva és más rétegekben történő támadások ellen nem védenek.

Az injekciós támadások lehetnek általánosak, amelyek bármilyen sérülékeny rendszert céloznak, vagy célozottak, melyek egy bizonyos alkalmazást vagy rendszert célzottan támadnak. A célzott támadások általában nagyobb veszélyt jelentenek, mert a támadók gyakran rendelkeznek előzetes ismeretekkel a célalkalmazásról és az annak rendszeréről. Sok weboldal és alkalmazás szenvedett már el súlyos adatlopásokat és más káros következményeket az injekciós támadások miatt, és ezeknek a támadásoknak a megelőzése és a kezelése továbbra is fontos kihívást jelent a biztonsági szakemberek számára. Ezért választottam ezt a témát, illetve különösen a megelőzésre fókuszálva esett a választásom egy Web Application Firewall-ra, ami még a célba jutás előtt megvédi a szerveret az esetleges támadásoktól, azokat bizonyos kritériumok alapján szűrve. Rengeteg esetben már az is elegendő lépés, ha bizonyos IP címeket tiltanak le, amikről ismertek, hogy ezen támadások kiindulópontjaiként szolgálnak. A block listák általában TOR végpontokat (virtuális „alagutak” az internetes forgalom anonimmá tételére) vagy ismert spam címeket tartalmaznak.

2.6.1. Az injekciós támadások jövője

A támadások száma évenként változik, és nehéz pontos számokat adni. Az adatok eltérőek lehetnek a különböző forrásokból származó információk alapján. Azonban a keretrendszerek és újabb védelmi mechanizmusok elterjedése miatt visszaszorulóban van a számuk. Az OWASP (Open Web Application Security Project) által évente kiadott Top 10 biztonsági kockázatok listáján az injekciós támadások mindig jelen vannak, ami azt jelzi, hogy a probléma továbbra is jelentős kihívást jelent a kiberbiztonság számára. Néhány webapplikáció pedig továbbra is ragaszkodik a régebbi, olykor sebezhető technológiákhoz [9]. A támadások súlyosságát tekintve is csökkenő tendenciát mutatnak a statisztikák, ugyanis a ptsecurity.com felmérése [10] alapján jól látszik, hogy az alacsony és közép kategóriájú kockázatú támadások jelentkeznek többször azonos számú támadásokra vetítve, ahogy ez a 2. ábrán is látható.



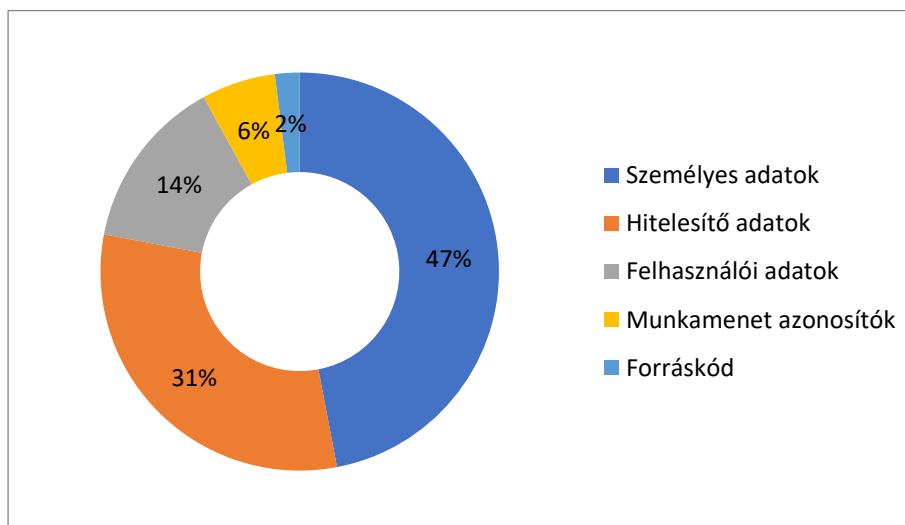
2. ábra Injekciós támadások súlyosság alapján

A támadások kiindulópontjául még mindig a kliens oldal szolgál az esetek 92%-ában és ez a tendencia nem valószínű, hogy csökken a jövőben.

Ennél aggasztóbb, hogy amennyiben az esetleges támadás megtörtént az esetek 16%-a, azaz minden 6. támadás során a támadók hozzáférhettek a kiszolgáló webszerverhez és annak operációs rendszeréhez.

Az ilyen támadások nem csak az érintett személyek személyes adatait veszélyeztetik, hanem a vállalkozások és szervezetek működését is jelentősen károsíthatják. Az adatlopások lehetősége különösen aggasztó a mai digitális korban, amikor az adatok és

az online kommunikáció egyre fontosabbá válik az üzleti világban és a mindennapi életünkben. A behatolások során ellopott fontos információk a következőképp alakultak:



3. ábra A behatolások során leggyakrabban megszerzett információk

3. A WAF mint megoldás

A WAF (Web Application Firewall) olyan biztonsági megoldás, amely védelmet nyújt a webalkalmazások elleni támadások ellen. Egy köztes réteget biztosít a webalkalmazás és a kliens között, amely lehetővé teszi a HTTP forgalom ellenőrzését és szűrését. Általában három típust különböztetünk meg. Mindhárom megoldásnak megvan a saját előnye és hátránya:

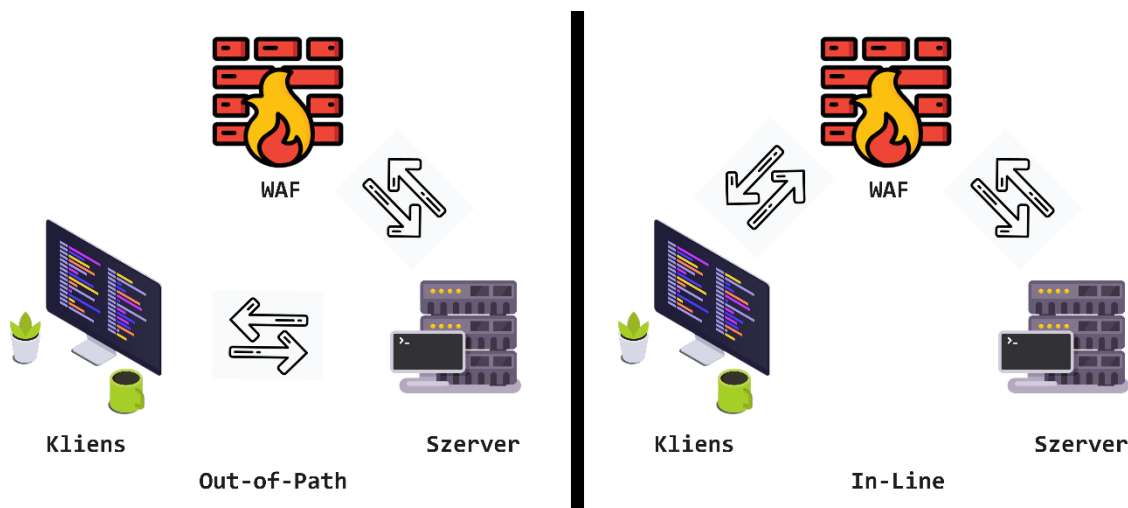
- **Felhő alapú:** A felhőalapú megvalósítás előnye a könnyű felállítás, illetve a fogyasztás alapján számolt árazás és a folyamatosan frissített védelem. Az utóbbi időben pont ezért ez vált a domináns, leginkább használt WAF megoldássá. Az egyetlen szereplők, akiknek ez a megoldás nem ajánlott a kormányzati szervek, adatbiztonsági okokból.
- **Szoftver alapú:** A szoftveralapú megoldás lényege, hogy virtuális eszközként vagy ügynökként fut, akár lokálisan (helyszíni), akár privát felhőben, akár nyilvános felhőben. Vannak kifejezetten mikroszolgáltatásként tervezett szoftver alapú WAF megoldások, mint például a Radware ami egy Kubernetes szolgáltatás. Valamint előnyei közé tartozik a magasfokú testreszabhatóság, ezzel a komplexitását növelve és felállítását, menedzselését nehezítve.
- **Hardver alapú:** A hardveralapú mindhárom közül a legköltségesebb, de egyben a leghatékonyabb megoldás. Mivel a helyi, lokális környezetben található dedikált hardveren, a szerver válaszsideje nagyban csökkenthető, ami nagy vállalatok esetében dollármilliárdokat jelenthet bevételben.

Összefoglalva, a felhőalapú, a szoftveralapú és a hardver alapú WAF mindegyike hatékony védelmet nyújt a webalkalmazások elleni kártékony támadásokkal szemben. Az üzemeltetőknek ki kell választaniuk a saját igényeiknek és a rendszerüknek legmegfelelőbb megoldást.

3.1. In-Line és Out-of-Line megvalósítások

Az In-Line megvalósítás azt jelenti, hogy a WAF közvetlenül a webalkalmazás előtt helyezkedik el, és az összes forgalmat átvizsgálja a webalkalmazás és a felhasználó között. Ez hatékonyabb védelmet biztosíthat a webalkalmazások számára, mivel a rosszindulatú forgalmat azonnal blokkolhatja, mielőtt az elérné az alkalmazást. Ezzel

szemben az Out-of-Line megvalósítás azt jelenti, hogy a WAF különálló komponensként működik, és a webalkalmazáshoz érkező forgalmat bizonyos kritériumok alapján a WAF-hoz irányítják át, ami átvizsgálja a forgalmat, majd a jóváhagyott forgalmat továbbítja a webalkalmazás felé. A koncepciókat a 4. ábrán szemléltetem. Ez a megvalósítás nagyobb rugalmasságot és skálázhatóságot tesz lehetővé, mivel külön szerveren vagy felhőalapú szolgáltatásként lehet futtatni a tűzfalszolgáltatást.



4. ábra In-Line és Out-of-Line szemléltetése

3.2. Védelem típusa

A védelem típusát nézve két kategória létezik, az aláírás-alapú és a szabály-alapú védekezés.

Az aláírás-alapú védelem lényege, hogy előre eltárolt minták alapján elemzi a kéréseket és amennyiben egyezéseket talál riasztást ad. Ezek a minták előre összeállított, már ismert támadások összetevőit tartalmazzák. Használata előnyös, mert könnyen frissíthető, ha új támadás típusokat fedeznek fel, viszont ebből adódóan a frissebb támadásvektoroktól csak frissítés után fog védeni. Azonban, ha kiterjedt és átfogó védelmet szeretnénk biztosítani sok mintát kell tárolnunk, ami tovább fokozza a válaszidőt. Általában ezért csak egy kis listát tartanak nyilván a releváns vektorokkal.

Ezzel ellentétben a szabályalapú megoldás egy viszonylag új védekezés, ami mesterséges intelligencia segítségével határoz meg úgynevezett szabályokat. A szabályok tekinthetők aláírások aláírásaként is, mert több hasonló aláírást képes kevesebb kóddal lefedni. Épp ezért a fő előnyei közé tartozik a gyorsaság, illetve a másik nagy előnye a könnyű fenntarthatóság, ugyanis csak nagyon minimális karbantartást igényel. Az MI alkalmazásából látszik továbbá, hogy kitűnően tudja kezelni a nulladik napi (ezelőtt

ismeretlen) támadásokat. A megközelítés hátulütője, hogy rengeteg false-positive, illetve true-negative érzékelést tapasztalhatunk használata során. Ez azt jelenti, hogy a rosszindulatú kérést néhány esetben nem detektálja és előfordulhat, hogy nem kártékony forgalmat tilt le.

3.3. Egyéb feladatok

A tartalomszűrés mellett sok egyéb extra szolgáltatással is rendelkeznek, amik tovább erősítik az alkalmazások védelmét. Ezek közé tartozik például a botnet-kezelés, a DDoS védelem vagy akár a webkaparás elleni védelem. Természetesen az egyes WAF megoldások különböző módon végezhetik ezeket a feladatokat, a botnet-kezelést általában IP feketelistákkal vagy Captcha-k alkalmazásával teszik. Egyre gyakoribb jelenség az SSL/TLS dekódolás is, ami lehetővé teszi a titkosított forgalom dekódolását és ellenőrzését. Természetesen ez egy In-Line megközelítés, amit reverse-proxy-k használnak. Az alap koncepciója, hogy a proxy felépít egy biztonságos TCP kapcsolatot a kliens és önmaga között, majd a szerver és önmaga között, aztán az üzeneteket dekódolja és újra titkosítja egy MITM (Man In The Middle) támadáshoz hasonlóan a kommunikáló felek között.

3.3.1. Integráció más biztonsági eszközökkel

Gyakran alkalmazhatók akár beépülő modulként az egyes WAF megoldások. Az egyik nyílt forráskódú megoldás, a ModSecurity is eredetileg egy Apache-modulként indult de ma megtalálható Nginx vagy MSIIS-modulként, vagy akár önálló kiszolgálóként is. A trendeket tekintve [11] az integráció más SIEM, illetve SOAR megoldásokba (kiberbiztonsági „eszköz-csomagok” a veszélyek felderítésére) folyamatosan növekszik. Egyre gyakoribb a CI/CD folyamatokba is integrálni a DevOps során (ami maga egy elterjedt és bevett technika), a biztonsági kockázatok folyamatos tesztelése érdekében. Az ilyen környezetek szervezetenként eltérő megvalósításokat használnak, ezért az egyes megoldásokat jelentősen személyre kell szabni az esetek nagyrésztében.

3.3.2. Nemzetközi viszonylatban

Az Egyesült Államokban használatuk nagyon elterjedt, mivel a szervezetek és vállalkozások nagy része kitüntetett figyelmet fordít a biztonságra. Az európai országokban is széles körben használnak WAF-okat, és az Európai Unió számos adatvédelmi és biztonsági szabályozással rendelkezik, amelyek előírnak kötelezettségeket a webes alkalmazások biztonságának.

Ma a legelterjedtebb WAF szolgáltatók közé tartozik például a Cloudflare, Imperva illetve az AWS. A Cloudflare és az AWS elterjedtségüket főleg a szolgáltatásaik minőségének és a mögöttük álló támogató cégeknek köszönhetik (FAANG¹). Ezek nagyrészt a felhőben működő megoldások tehát a közbeiktatásukhoz a hálózati forgalomnak kereszteznie kell, így a lokális felhasználók kéréseit is ki kell irányítani az internetre.

3.3.3. Hazai viszonylatban

Habár elsőre nem gondolnánk, de Magyarországon is jelen vannak WAF szolgáltatók, amelyek ha kisebb kapacitásban is, mint nemzetközi vetélytársaik de kiszolgálják a hazai ügyfeleiket. Az egyik legnagyobb WAF szolgáltatással is foglalkozó vállalat az Invitech, akinek ügyfelei közé tartozik a Budapest Bank vagy a hazai Erste Bank is. Alkalmazásuk azonban nem korlátozódik a nagyobb szervezetekre, hanem egyre több kisebb vállalkozás és weboldal is használja ezeket a megoldásokat a biztonságuk növelése érdekében. Az egyre növekvő számú internetes támadások és a webes biztonsági incidensek fokozott figyelmet igényelnek, így használatuk egyre fontosabbá válik a magyarországi webes környezetben is. Az elmúlt években rohamosan fejlődő 4iG nevű magyar vállalat is foglalkozik WAF és egyéb kiberbiztonsági megoldásokkal, elsősorban a nemzetközi szolgáltatók, mint például a Microsoft (Azure) vagy az Amazon (AWS) megoldásait implementálja. A 4iG-t azért is érdemes megemlíteni, mert a legfontosabb hazai banki és államigazgatási apparátusok védelméért felelősek.

¹ Facebook (jelenleg Meta Platforms), Amazon, Apple, Netflix és a Google

3.3.4. Limitációk

Bár hatékony védelmet nyújtanak a webes alkalmazások elleni támadások ellen, de vannak bizonyos korlátok, amelyeket érdemes figyelembe venni [12].

- **Álpozitívok:** Ez azt jelenti, hogy egy WAF nem minden esetben állapítja meg helyesen a rosszindulatú tevékenységeket, ezáltal sokszor indokolatlanul blokkolva bizonyos kéréseket.
- **Teljesítményromlás:** Többletköltséget jelenthet az alkalmazásnak, és hatással lehet a teljesítményére. Ez aggodalomra adhat okot a nagy forgalmú kritikus alkalmazásoknál vagy szigorú teljesítmény követelményeknél.
- **Nulladik napi támadások:** A gépi tanulással nem rendelkező WAF-okat előre fel kell készíteni a lehetséges támadástípusokra ezáltal, ha nincs megfelelően frissítve vagy egy régebbi szűrőt használ az újabb támadásokkal szemben védtelen marad.

Habár önálló megoldásként is jelen van sok helyen a WAF-ok csak egy részét képezik a többretegű védelmi mechanizmusoknak, a megfelelő és effektív működés érdekében folyamatos konfigurációt igényelnek.

4. A szoftver bemutatása

Egy saját fejlesztésű WAF, azaz Web Application Firewall elkészítése volt a feladatom, amit többek között az előzőekben ismertetett támadásvektorok kivédésére lehetne használni. Mindezt modern, könnyen követhető programozási paradigmák segítségével, a könnyebb megértés segítése érdekében a fontos algoritmusokat típusokkal és segítő kommentekkel kiegészítve a későbbi tovább fejlesztés érdekében. Céлом továbbá, hogy a megoldásom ne csak egy „Proof Of Concept” jelleggel valósuljon meg, hanem alkalmazható és effektív védelmet nyújtson akár magánszemélyek, vagy kisebb cégek számára.

4.1. Célkitűzések és követelmények

Egy könnyen kezelhető WAF megoldás, ami többek között képes a HTTP forgalom automatikus ellenőrzésére, amely a konfigurációban meghatározott webszerver címére

kerül továbbításra. Szükség esetén a TCP kapcsolatot bontja, ezzel a kártékony kérések célba jutását előzi meg. A védelem típusának sokszínűsége és ezek kezelhetősége érdekében egy jól összeállított konfigurációs fájlra is szüksége lesz az alkalmazásnak, hogy a végfelhasználók személyre tudják szabni a nekik megfelelő beállításokkal. Ezzel nem csak a védekezés módját lehet személyre szabni, hanem kivételek beállítására is lesz lehetőség. Sok esetben a válaszidő az egyik legfontosabb tényező a webes applikációknál, ezért bizonyos ellenőrzések kihagyásával ez jelentősen csökkenthető. Az imént említett védelem részletes beállítására is lesz lehetőség, amiben lehet váltani az aláírás és a szabályalapú védelem között. A WAF egy riasztó rendszert is alkalmaz, ami szükség esetén egy naplófájlban rögzíti az esetleges támadási kísérleteket és egyéb fontos információkat, a hozzájuk tartozó riasztási szinttel. A megoldás minimális mértékben módosíthatja a HTTP kéréseket, az esetleges integrációs problémák elkerülése érdekében. Az effektív védelem elérése érdekében IP feketelistákat használ a megoldás, melyek online és lokális adatok alapján szűrik a csatlakozni próbáló kiszolgálókat.

4.2. Tervezés

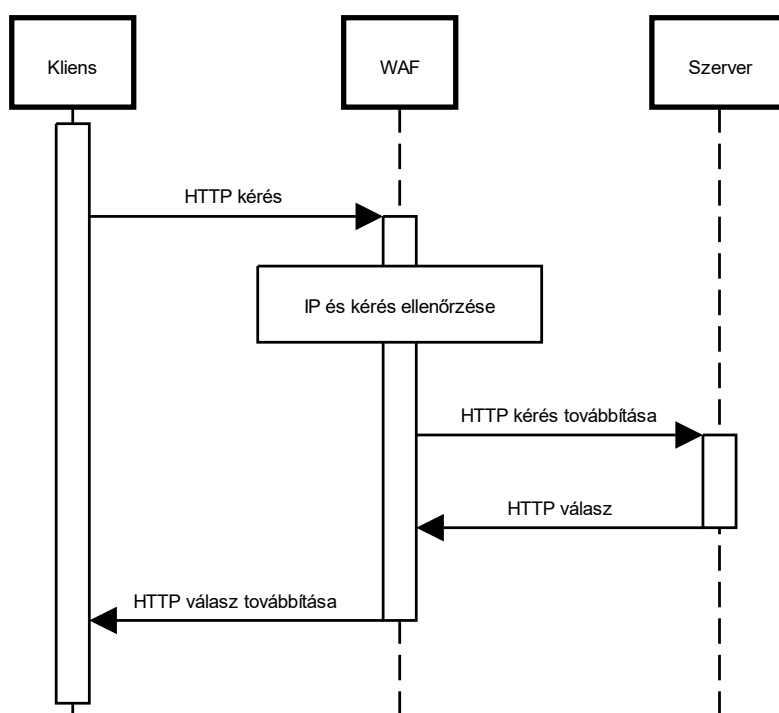
A tervezési fázisban megterveztem a rendszer előzetes architektúráját, amit majd a későbbiekben Python nyelven valósítok meg. Többek között a nyelv támogatottsága és gépi tanulást alkalmazó segédkönyvtárak miatt. Mivel közvetlen felhasználói interakció a program működése során nem szükséges, ezért a rendszer felhasználók általi elérhetőségét az előzetes konfigurációs fájlokra korlátoztam. További visszajelzési felületként szolgál a nyilvántartott IP adatbázis, ahol a rosszindulatú címek kerülnek tárolásra és a naplófájl, ahol minden fontosabb eseményről részletes leírást láthatunk. Ebből is látható, hogy a szoftver egy black-box rendszerként fog megvalósulni. A black-box vagy fekete doboz rendszerek olyan szoftverek, amik a felhasználói szempontból ismeretlen módon működnek. Ezen rendszerek fő előnyei az egyszerűség, a megbízhatóság és a biztonság. A felhasználóknak ugyanis a rendszer működtetéséhez csak minimális mennyiségű háttértudással kell rendelkezniük és a konfiguráción kívül a program más részét nem fogják látni. Emiatt javul a megbízhatóság is, mert a felhasználók nem tudnak akaratlanul kárt tenni a rendszerben, illetve a biztonság is jobb, ugyanis a rendszer részei nem kerülnek felfedésre.

4.2.1. Proxy modul

Az alkalmazás legfontosabb része az üzenetek továbbításáért felelős Proxy-réteg, ami egy reverse-HTTP-proxy. A proxy egy olyan szoftver (vagy hardver), ami a webservert és a kliens-applikáció közé ékelődve helyezkedik el, közvetíti a forgalmat közöttük, majd az ezekre a kérésekre küldött válaszokat is továbbítja. A proxyk bármilyen hálózati forgalom továbbítására alkalmasak, de esetünkben csak HTTP szabványú üzeneteket kell továbbítani.

A reverse proxy vagy fordított proxy, ellentétben egy általános proxyval a szervergép közvetlen közelében helyezkedik el. Azonban általában ezek nem csak az üzenetek továbbítását végzik, hanem sok más egyéb hasznos funkcióval is rendelkeznek. Például tehermentesítik a szervergépet azért, hogy a gyakoribb kérésekre érkező válaszokat eltárolják és ezekre beérkezés esetén rögtön továbbítják a választ. Ezen kívül elosztott rendszerek esetén a kéréseket elosztják a többi szervergép között, akik a webalkalmazást kiszolgálják. Esetünkben azonban ezek a funkciók nem kerülnek implementálásra, mert ezek inkább rendszer-menedzsmenti kérdések, mintsem kiber-biztonságiak.

A proxy tervezett működését az alábbi szekvencia diagrammon szemléltetem.



5. ábra A proxy tervezett általános működése

4.2.1.1. Eseményalapú és szálalapú rendszerek

A szálalapú rendszerek alapelve, hogy több szálát futtat egyidejűleg a rendszer. WAF-ok esetén ez azt jelenti, hogy a szálak egy-egy HTTP folyamat kezelnek ezzel a válaszidőket jelentősen javítva. Ezzel ellentétben az eseményalapú megközelítés a beérkező HTTP kéréseket figyelve és azokra reagálva hajtja végre a szükséges ellenőrzéseket. Az eseményalapú megközelítés általában hatékonyabb, mint a szálalapú megközelítés, mert kisebb a várakozási idő és kevesebb a szükséges memória. A szoftveremet a két megoldás hibrid változataként terveztem, főleg teljesítmény optimalizálási okokból.

4.2.2. Analízis modul

Az analízis feladatokat a tervezés során két csoportra bontottam, az aláírásalapú és a szabályalapú megközelítés szerint, amit a felhasználók maguk állíthatnak szükségleteik szerint.

4.2.2.1. Aláírásalapú védelem

Az aláírásalapú védelem megtervezésének első lépése volt, hogy megvizsgáltam a támadás típusok különböző karakterisztikáit és ismert lehetséges alternatíváit. Első körben az elterjedt támadásokra fókuszáltam, majd további kevésbé elterjedt típusokra kerestem példákat. Miután a lehetséges típusokkal és jellemzőikkel megismerkedtem elkezdtem próba reguláris kifejezéseket (regex) írni rájuk, amik detektálják őket. A regex-ek megállapításában nem találtam segítséget, mivel ezeket általában a kiberbiztonsági cégek ipari titokként őrzik, ezzel növelve hatékonyságukat. A későbbi könnyű karbantarthatóság érdekében az aláírásokat a konfigurációs fájlokban tervezem tárolni, hogy onnan könnyen módosíthatóak legyenek. Ezzel naprakész biztonságot garantálva az alkalmazásoknak.

4.2.2.2. Szabályalapú védelem

A szabályalapú védelem megtervezése a legutolsó részét képezte a feladat megvalósításnak, mert nem volt előzetes tapasztalatom a gépi tanulás területén. Ezért egy kisebb kitérő után, amely során megismerkedtem a főbb koncepciókkal és néhány módszerrel, rátaláltam a Bag-of-Words modellekre. Ez a megközelítés lehetővé teszi, hogy viszonylag egyszerűen keressünk mintákat a szövegben a bizonyos részletek

előfordulását alapul véve. A megvalósítás részben tárgyalom a további alkalmazott megoldásokat, mert főleg a tesztelések segítségével tudtam csak javítani az eredményeket.

4.2.3. Egyéb tervezési feladatok

Az előzetes előkészületi folyamatokhoz a kimeneti adatok meghatározása is hozzátartozott. Az adatbázis és a naplófájlok formátumához tartalmi kritériumokat határoztam meg, mint például a támadó IP címének, a támadás időpontjának, a használt támadás típus, illetve a kizárás okának feljegyzése. Mivel nem csak a támadások kerülnek feljegyzésre a naplófájlban, egy újabb listában ezeket is feljegyeztem miről milyen értesítést szeretnék adni a felhasználóknak.

4.2.3.1. Fejlesztéshez használt eszközök

A feladat megvalósításához a JetBrains PyCharm nevű IDE-t (Integrated development environment) választottam a virtuális, projekthez kapcsolt python környezet miatt, valamint a tervezett adatbázissal való könnyebb interakció miatt. További pozitívum volt, amit csak a fejlesztés későbbi szakaszaiban vettem észre a SciView, ami adat vizualizálásra volt nagyon hasznos, főleg a gépi tanulás implementálásánál.

A HTTP kérések küldéséhez Postman-t használtam, ami API (Application Programming Interface) fejlesztésre, tesztelésre lett tervezve. Először is, mert egyszerű és felhasználóbarát. Továbbá a tesztelés folyamatát teljes egészében lehet automatizálni, ezzel rengeteg időt spórolva.

A tesztelendő alkalmazásnak a DVWA-t (Damn Vulnerable Web Application) választottam, mert ez egy olyan webes applikáció, ami szándékosan biztonsági résekkel lett ellátva. Főleg oktatási célokra lett kifejlesztve Ryan Dewhurst által. Majd a DVWA

applikáció futtatására egy Docker konténert használtam. Ezt a 6. ábrán részletesen szemléltetem egy tevékenységdiagram keretében.



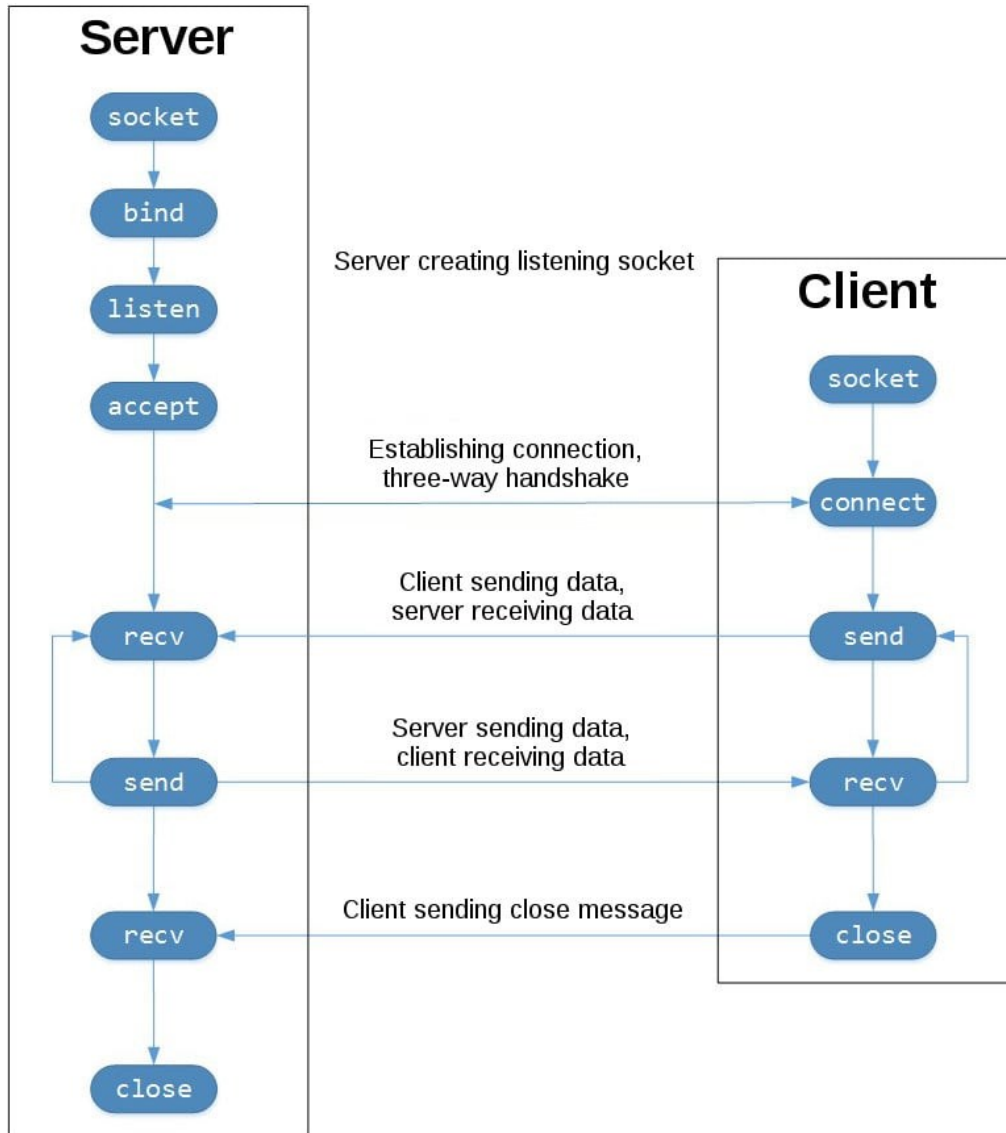
6. ábra Rendszer terve és a használt technológiák (tevékenységdiagram)

4.3. Megvalósítás

Az alábbi szekcióban bemutatom a programom működését, kód példákkal és a főbb koncepciók ismertetésével. A feladat megvalósítása során a legnagyobb kihívást a socket kommunikáció megértése jelentette. Szerencsére a megvalósított megoldásban az összes többi funkció implementálása mellett erre is találtam optimális megoldást.

4.3.1. Proxy modul

A TCP socketeket összekötő kommunikációt a Pythonban beépített socket modullal oldottam meg, handler osztályok segítségével. A 7. ábrán szemléltetem egy socket csatlakozását.



7. ábra Socket kommunikáció lépései [17]

Egy handler a klienskommunikációért, míg egy a szerverkommunikációért felelős, de mindkét osztály alapjául vagy ősének az InstanceHandler osztály szolgál.

4.3.1.1. ServerInstanceHandler

```
def connect_server_socket(self):
    self.close_socket()
    try:
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socket.connect((self.address, self.port))
        self.socket.settimeout(2)
    except socket.error:
        sys.exit("Target not listening, terminating process...")
```

A leszármazott *ServerInstanceHandler* csatlakoztatására szolgáló függvényt láthatjuk, ami egy továbbító állapotú portot hoz létre. Esetünkben a „socket.SOCK_STREAM” jelenti, hogy egy TCP portot hozunk létre a „socket.AF_INET” pedig IPv4-es címre utal. Implementáltam egy kilépési mechanizmust is, a későbbi felhasználók munkájának megkönnyítése érdekében, ha a specifikált szerver felé valamilyen oknál fogva nem létesíthetünk csatlakozást.

Amennyiben a port átkerült a kívánt állapotba és kérés érkezik a proxy-ra a *forward()* függvény továbbítja azt a kérés minimális módosításával, amit a [Továbbítandó kérések előkezelése](#) részben részletesen leírok.

```
def forward_message(self, message: str, start_port) -> bytes:
    self.connect_server_socket()
    if not start_port == 80:
        message.replace(str(start_port), str(self.port))
    else:
        host = (message.partition("Host: ")[2]).partition("\r\n")[0]
        message = message.replace(host, f"{host}:{self.port}")
    self.socket.sendall(message.encode())
    return self.receive()
```

A *receive()* függvény a fogadás módját befolyásolja. A konfigurációban állítható maximális üzenet méret alapján ennyi byte adatot fogad, majd kiolvassa a fejlécben megkapott „Content-Length” alapján, hogy még hány byte adatot fog a szerver küldeni. Ezt követően, fogadja a megadott számú byte-ot. Ha a szerver nem használ Content-Length fejléct addig fogadja a bytefolyamot, amíg az nem áll meg 2 másodpercnél több időre. Ez főleg akkor fontos, ha nem megfelelő a küldött csomag, és nem jelzik a végét. A HTTP protokoll, így vagy „Transfer-Encoding”-al jelzi az üzenet tényleges hosszát.

A chunk megoldás főleg nagyobb méretű képek vagy valós idejű streamelés során használatos. Erre azonban a proxy nincs felkészítve, Content-Length fejléceket alkalmazó szerverekkel javasolt a használata, ugyanis ez a legelterjedtebb mód.

```
def receive(self) -> bytes:
    response = self.socket.recv(self.message_size)
    content_length = ((response.partition(b"Content-
Length:")[2]).partition(b"\r\n")[0]).decode()
    if len(content_length) > 0:
        content_length = int(content_length)
        while True:
            if content_length == len(response.partition(b"\r\n\r\n")[2]):
                break
            response += self.socket.recv(self.message_size)
        return response
    else:
        while True:
            try:
                data_part = self.socket.recv(self.message_size)
                response += data_part
```

A szervertől így kapott adatokat továbbítja a kliens részére. Itt már ellenőrzéseket nem végzünk a kapott válaszon.

4.3.1.2. ClientInstanceHandler

A folyamat másik részéhez használatos *ClientInstanceHandler* osztály a klienshez való csatlakozásért felelős. Itt egy hallgató állapotú port létrehozásával figyeli a beérkező kéréseket, illetve hatékonysági megfontolásokból itt történik az IP elemzése amire a későbbiekben ki is térek.

```
def connect_client_socket(self):
    self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    self.socket.bind((self.address, self.port))
    self.socket.listen(1)
    self.client_conn, self.address = self.socket.accept()
    if not is_IP_safe(self.address):
        self.client_conn.close()
        raise InvalidIPException()
    self.client_conn.settimeout(2)
```

A *forward_comm()* függvénnyel pedig csak tovább delegáljuk a kérés tartalmát elemzésre a többi modul számára, majd a választ visszaküldjük a kliens számára. Az „socket.SO_REUSEADDR” segítségével hozzá tudjuk rendelni a porthoz még akkor is, ha azt egy másik folyamat használja.

4.3.1.3. Észlelési módszertan

Felmerült bennem ötletként, hogy a WAF-ok általános detektálása a támadóknak egy igen könnyű feladat. Ugyanis a különböző WAF megoldások valamilyen formában, egy lenyomatot vagy fingerprint-et hagynak a kérésben. Ez megoldásonként változó, de általános és elterjedt praktika. Ezek lehetnek többek között:

- Saját cookie-k bevezetése (Yunso WAF)
- További fejlécek használata (AWS)
 - A fejlécekben véletlenszerű karakterláncok, a támadók összezavarására
- A szerver fejléc módosítása (WTS WAF)
- A válasz body részében (DotDefender)
- Hiba esetén más státuszkóddal válaszolva (WebKnight)

A könnyű detektálhatóság elkerülése érdekében a proxy csak a „Host” fejléct módosítja, amit a következőkben tárgyalok.

4.3.1.4. Továbbítandó kérések előkezelése

A proxy működésének elengedhetetlen része a kérés minimális szintű módosítása. Erre azért van szükség mert egy opcionális fejléc „elárulja” a webszervernek a kérés beérkezési helyét. Amennyiben erre a webszerver fel van készítve egy 301-es hibakódú „Moved permanently” üzenettel válaszol, aminek a „Host” fejléce tartalmazza a tényleges desztinációt. Ezt felhasználva a böngésző a proxy-t megkerülve a webszervert közvetlenül próbálja elérni az előző válaszban kapott címen és porton. A tesztelés során ez a WordPress oldalaknál okozott főleg problémát. Ezt a funkcionalitást a WordPress biztosította, mert az ezt kiszolgáló Apache szerver nem ellenőrzi ezen fejlécek jelenlétét.

4.3.1.5. Adatok közlése HTTP kérésben

A felhasználói adat közlése a http protokoll szabályok szerint 3 formában történhet. A HTTP protokoll alapvetően szimpla sortörésekkel választja el egymástól a különböző fejléceket. A kérés első sorában a kérési sor vagy request line található, amiben a kérés módja (GET, POST, etc.) és a HTTP verzió között található a felkeresett cím. Ami már csak az elérési út, paraméter és az anchor (XPath jelölő²) részeket tartalmazza. Az anchor az oldalon való eligazodást segíti, dokumentum-béli hiperhivatkozásként. A paraméter,

² Az XPath az XML dokumentumban bizonyos címkék keresésére szolgál

pontosabban keresési paraméter vagy query parameter szolgál leggyakrabban kihasználhatóságok kiindulópontjaként. Az anchor vagy „horgony” kevésbé, de az XPath injekció révén előfordulhat ez is. Ez a fajta támadás leggyakrabban GET kérések esetén használatos. A megértést segítő 8. ábrán bemutatok egy HTTP kérést.

```
POST /home?data=sealion HTTP/1.1
Content-Type: application/json
User-Agent: PostmanRuntime/7.30.0
Accept: */*
Postman-Token: be5759c7-8749-4730
Host: localhost:8080
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 17
Cookie: Cookie_1=value

{"data":"Kansas"}
```

8. ábra Egy HTTP kérés számunkra fontos adatmezői

Egy nagyon gyakran használatos fejléc, ami felhasználói információközlésre használatos a Cookie, amiket a kommunikáció során a szerver és a kliens, minden üzenettel továbbít egymás felé.

Előfordulhat azonban, hogy a rosszindulatú felhasználó módosítja ezeket, és amennyiben a fogadó applikáció helytelenül van megírva, ellenőrzés nélkül csak beillesztve lefuttatja a felhasználó által módosított cookie tartalmát. Azonban ez elég szélsőséges eset, itt inkább a cookie poisoning jellemző, ahol más felhasználóktól lopott cookie-k segítségével az ő munkamenetüket (session) hamisítva az adott felhasználót személyesítik meg. A cookie poisoning ellen egy alapvető WAF nem nyújt határos védelmet.

A harmadik lehetőség, ha a kérés body részében található az adat. Ez a fejléc rész után dupla sortörés után következik a kérésben. A formátumot egy „Content-Type” nevű reprezentációs fejléc adja meg és a fogadó fél ezt használva állapítja meg, hogy kell az adatot értelmezni. Amennyiben ez hiányzik (vagy hibás) a böngészők egy bizonyos MIME sniffing technikát alkalmaznak, amivel lényegében kitalálják az érkező adat formátumát. Ez nem csak klienseknél, de szerveroldalon is előfordul, amennyiben pedig a kérés tartalmaz egy „X-Content-Type-Options: nosniff” fejlécet könnyen letiltható a viselkedés. Ez a leggyakoribb, mert általában a POST kérés body részében közlünk adatot. Természetesen a kérés más részeiben is lehetséges az adatok átadása a

szerverapplikáció felé, de mivel ezek csak nagyon ritkán és komplexebb alkalmazások esetében jellemzők, ezzel az esettel nem kívánok foglalkozni szakdolgozatomban. Megvalósításomban a config fájlban szükség szerint állítható a védelem kiterjedése az imént felsorolt három HTTP mezőre.

4.3.2. IP intelligence

Az IP intelligence egy olyan megoldás, ami a hálózati forgalom (IP címek) ellenőrzését és elemzését foglalja magában. A megvalósított programom egy lokális feketelistát tart nyilván, ahova a rosszindulatú karakterláncokat küldő felhasználók IP címei tárolja a tervezési fázisban meghatározott módon. Továbbá a webszerveret felkereső IP címeket lekéri online található DNSB-től (Domain Name System Blocklist). Amennyiben valamelyik tartalmazza, a kérés valószínűleg egy támadótól jöhet ezért ezt terminálni kell. A lokális adatbázis gyorsítótárazási megoldásként is működik, ugyanis az IP címek DNSB-ből való lekérdezése egy időigényes folyamat, mert több forrás áll rendelkezésre. A megvalósítás SQLAlchemy-t használ, ami egy ORM (Object-relational mapping) csomag a könnyű adatbáziskezelés megvalósításáért.

4.3.3. Szűrőimplementációk

Mindkét implementált megoldás először a bytestream-ként érkező kérést dictionary-vé konvertálva ellenőrzi a konfigurációban beállított szempontok szerint.

4.3.3.1. Aláírásalapú szűrő

A *manual_approach()* függvény, ami a manuális szűrőt tartalmazza a konfiguráció alapján összehasonlítja a specifikált reguláris kifejezéseket a kérés szükséges részeivel. Támadásvektor észlelése esetén pedig a naplófájlban megfelelő riasztással tájékoztatja a rendszer üzemeltetőit.

4.3.3.2. Szabályalapú szűrő

A szabályalapú szűrő megvalósításához a tesztelt módszerek közül a leghatékonyabbat választottam, ami az XGBC (eXtreme Gradient Boosting Classifier) volt. A teszt adatokat a HttpParamsDataset [13] Kaggle repozitóriumból töltöttem le a modell tanításához, amit 90%-ban tanításra és 10%-ban teszteléshez használtam. Ehhez a `train_test_split` metódust

használtam, ami az adatokból véletlenszerűen választotta az elemeket, ezzel elkerülve az esetleges hibás mintavételezést. A `train_test_split` metódus egy `X` és egy `Y` halmazt képez, a megadott `feature` (jellemzők) és `target` (célváltozók) alapján, amiket feloszt tanító és tesztelő halmazokra. Hogy a kiválasztás jobb eredményt adjon a metódusban található „`stratify`” paraméter használatával megadható, hogy melyik paramétert szeretnénk egyenlő arányban látni az adatokban.

Ha felosztottuk az adatainkat a `scikitlearn` `CountVectorizer` osztályát használva megvizsgáljuk az adathalmazunk minden tagját. Ez a megközelítés, pontosabban az `analyzer` paraméter értékét „`char`”-ra állítva a karakterek előfordulását veszi figyelembe és elsősorban ez alapján hoz döntéseket. A `min_df` paraméterrel megadtam, hogy a 20-nál kisebb előfordulású karaktereket hagyja figyelmen kívül, mert az adathalmazom tartalmazott más nyelvekből származó karaktereket is kis mennyiségben.

A karakterek előfordulását a `vocabulary_` adattag segítségével is lekérhetjük, ez esetben 68 volt. Az `XGBoost`-tal pedig a karakterek „`feature importance`” azaz jellemzőinek fontosságát kérhetjük le, ami az adathalmazomon a következőképp alakult:

)	=	{	,	#	'	"	-	:	.
0.7306	0.1116	0.0367	0.0217	0.0152	0.0130	0.0117	0.0112	0.0085	0.0056

1. táblázat Az előfeldolgozás eredménye

Az adatok előkészítése és transzformációja után egy pipeline készül az előfeldolgozás (`count_vectorizer`) és a választott modell-ből (`XGBoost`) ami egy egyszerű `fit()` művelettel az adatokat betöltve készen is áll az adatok kiértékelésére a `predict()` függvénnyel. Amennyiben támadást észlel, azt a naplófájlban riasztásként a felhasználóknak jelzi.

4.3.3.3. Adatok tisztítása és formázása

Szerencsére a talált `HttpParamsDataset` könyvtár adatai csak minimális előkészítést igényeltek, mert bár az adatok csoportokba voltak rendezve az anomália besorolást korrigálnom kellett. Néhány esetben nem volt megfelelő a CSV (comma-separated values) formátum, ezt is korrigáltam a `pandas` nevű könyvtárral.

4.4. Konfiguráció, nyomon követés és adattárolás

A konfigurációhoz Toml (Tom's Obvious Minimal Language) nyelvet használtam, mert pythonban könnyen integrálható a configparser modul révén. A legfontosabb beállítások módosítására nyújt lehetőségeket, illetve a fentiekben említett védelem konfigurálására. Új támadástípusok elleni védelem automatikus hozzáadására is nyújt lehetőséget.

```
[base]
accept_from = "127.0.0.1"
local_IP = "127.0.0.1"
in_port = 8080
out_port = 80
message_size = 4096
[analysis]
block_malicious_IP = true
[analysis.request]
type = ["GET", "POST", "UPDATE", "DELETE"]
part = ["status_line", "body"]
[ruleset]
use_manual = true
manual_rules = [
    {name = "XSS", regex = '(<)+(\s)*(script|body|img|image|iframe|
    {name = "SQLi", regex = '([\x27]([\x27]|^[^[\x27]])*)*.*(ALTER
    {name = "Prototype_pollution", regex = '{(\S|\s)*_proto_'}]}
use_ML = false
```

9. ábra A konfiguráció tartalma

Az adatbázis a már említett IP feketelistát tartalmazza és SQLi adatbázist használ SQL Alchemy összekötéssel. Ez főleg absztrakciós szempontból tűnt jó választásnak, másrészt mert egy ORM használatával nem szükséges komplexebb folyamatok esetén az egész folyamatot menedzselni. Teljesítmény szempontjából is jól optimalizált, így nem annyira költségesek az adatkezelési folyamatok. A konfigurációs fájlt a 9. az adatbázis szerkezetét a 10. míg a naplófájl a 11. ábra hivatott bemutatni.

	blacklistID	ip_address	reason	source	detected_at
1		1 89.23.211.54	spam	local_exemption	2023-03-29 16:56:14
2		2 89.23.11.43	spam	local_exemption	2023-03-29 16:56:14

10. ábra Az adatbázis végleges megvalósítása

A naplófájl megvalósításához a logger modult használtam. Az üzenetek csoportosítása érdekében a riasztási időt és szinteket is csatoltam a különböző jelentések leírásához, valamint támadás esetén a történt offenzíva leírását.


```
[17-Apr-23 17:37:50] CRITICAL - Trace of Prototype_pollution detected in body
[17-Apr-23 17:39:48] INFO - Connection established from local machine
[17-Apr-23 17:39:48] CRITICAL - Trace of XSS detected in body
[17-Apr-23 18:13:21] INFO - Connection established from local machine
[17-Apr-23 18:13:21] CRITICAL - Attack vector detected by AI in Cookie
[17-Apr-23 18:56:36] INFO - Connection established from local machine
[17-Apr-23 18:56:36] CRITICAL - Attack vector detected by AI in status line
```

11. ábra A naplófájl tartalma

Ezek az elemek együttesen segítik a WAF hatékony működését és megkönnyítik a hibaelhárítás folyamatát.

A megvalósítási folyamat követhetőségéért egy verziókövetőt (git) is használtam, hogy nyomon tudjam követni az esetleges változtatásokat és esetleges hibák esetén az online GitHub szerverekről a kód elvesztése esetén pótolni lehet azt.

4.5. Feladatmegvalósítási nehézségek

A feladat megvalósítása során a legnagyobb nehézséget az jelentette, hogy a megkezdésekor csak nagyon csekély ismeretekkel rendelkeztam a WAF megoldások tekintetében. A legtöbb probléma azonban a proxy megvalósítása közben adódott, ugyanis csak nagyon kis számban léteznek pythonban megvalósított proxy megoldások, ezért saját megoldást kellett készítenem a problémára socketek segítségével.

A másik problémás terület a gépi tanulás implementálása volt, amivel szintén a szakdolgozat írása során ismerkedtem meg. Itt a különböző modellek kiválasztása, illetve megfelelő pontosság elérése okozta a legnagyobb feladatot.

Ha nehézségnek nem is nevezném, de a téma feldolgozás első felében javascript-ben terveztem a feladat megoldását és ezt a Tervezés II. tárgy során újra átgondolva döntöttem a Python és a gépi tanulást használó megközelítés mellett. További nehezítő tényező volt, hogy a kódot gyakran újraírtam, hogy követhető és átlátható maradjon. Az absztrakció és a modulokra bontásnak köszönhetően egy követhető szerkezet jött létre a fent részletezett három modulból.

4.6. Az elkészített program SWOT elemzése

A SWOT (Strengths, Weaknesses, Opportunities, Threats) elemzés egy olyan eszköz, amivel általában szervezetek tényezőit vizsgálják. Azonban ez esetünkben is alkalmazható a fejlesztett program leírására és elemzésére.

- **Erősségek (Strengths):** Megvédi a mögöttes webalkalmazást a nem kívánt támadásvektoroktól, illetve növeli a felhasználók bizalmát a webalkalmazással szemben. A legtöbb megoldásba könnyű integrációt biztosít, mert csak a HTTP folyamatot analizálja. Hatékonyan detektálja a támadás típusokat, amelyekre felkészítjük, mint például az SQLi vagy az XSS.
- **Gyengeségek (Weaknesses):** Sajnos előfordulhat, hogy a védelem nem nyújt megfelelő biztonságot és néhány kérés átjut rajta. Mivel nincs 100%-os védelem, ami lefed minden rendszert és támadást ez előfordulhat. Továbbá előfordulhat az is, hogy blokkolunk olyan kéréseket, amik nem voltak rosszindulatúak. Ezek habár rontják a webalkalmazás felhasználóinak élményét elkerülhetetlenek.
A megoldásom leginkább kisléptékű felhasználásra lett tervezve, mert a nagyobb vállalatok elosztott rendszereihez nem biztosít megfelelő integrálási lehetőségeket.
- **Lehetőségek (Opportunities):** Egyre elterjedtebb a WAF megoldások használata így az irántuk való kereslet is növekedik. A webalkalmazások biztonsága mind az általam kidolgozott, mind a piacon elérhető megoldásokkal jelentősen növelhető. Az előzetes piacfelmérés során arra jutottam, hogy egyszerre használnak aláírás és szabályalapú identifikálást a különböző WAF megoldásokban, ezért az én megoldásom is ezt a megközelítést alkalmazza.
- **Veszélyek (Threats):** A védekezés szempontjából legnagyobb veszélyt a rosszindulatú felhasználók megkülönböztetése jelenti a mindennapi forgalomtól. Ezen kívül azonban számos nehézségnek kell megfelelni, mint például a növekvő verseny a piacon vagy akár a naprakészség kérdése. Ha a WAF megoldásunkat nem fejlesztjük az hamar elavulttá és sebezhetővé válhat az új támadásvektorokkal szemben.

4.7. Összegzés

Az igényfelmérés során megvizsgált lehetséges megoldások mérlegelésével kiválasztottam a szükséges és elvárt funkciókat, majd az ezekhez szükséges technológiák kiválasztásával folytatva a feladatot a célkitűzések szerint elkészítettem. Minden kitűzött funkció teljes megvalósításra került. A telepítés és konfigurálás meglehetősen egyszerű, nem igényel nagyobb szakmai tudást, a konfigurációs fájlnak köszönhetően. A szoftver Windows, Linux és MacOS rendszereken is tesztelésre került. Leginkább Windows rendszeren került sor a tesztelésre, más platformokon néhány esetben hibára futottam, de igyekeztem javítani azokat. A további, működést javító lehetőségeket a [Továbbfejlesztési lehetőségek](#) részben taglalom, illetve a szoftver tesztelési metodikáját a [Tesztelés](#) részben részletesen leírom.

5. A gépi tanulás lehetőségei

Ez a rész a gépi tanulás integrálásával foglalkozik a WAF-ok mindennapi használatában. Hatékonyságát tekintve sokkal hatásosabb és pontosabb lehet, mint az aláírásalapú vetélytársai. Ezen kívül képesek gyorsan és hatékonyan azonosítani a fenyegetéseket, és alkalmazkodni a változó támadási módszerekhez, ami a kiberbiztonság szempontjából kulcsfontosságú.

Az adatbányászat fontosságát is érdemes kiemelni ezen a területen. Ezáltal a WAF-ok képesek azonosítani olyan korábban kevésbé észlelt támadás módszereket, amik a korábbi aláírásalapú szűrőkön könnyen átcúsztak.

A következő részekben bemutatom a ML (Machine learning) vagy gépi tanulás általános lépéseit az implementálás felé.

5.1. Adatok gyűjtése

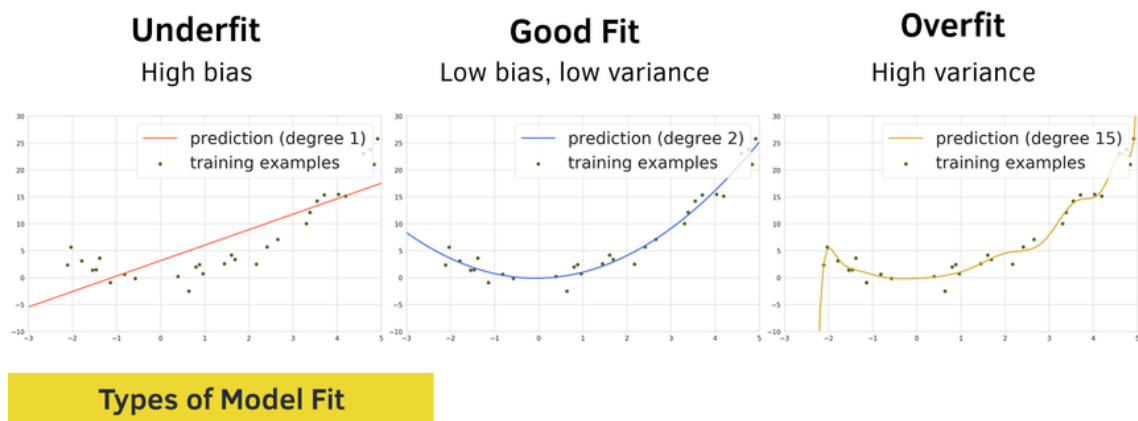
Közös vonás az összes gépi tanulást alkalmazó módszerben, hogy az első lépés az adatok gyűjtése. Ez a legfontosabb, hogy a későbbiekben készített modell megfelelő mennyiségű adattal rendelkezzen. Az adatokat érdemes a már korábban használt WAF alkalmazásokból beszerezni mert így biztos helyes adatokat használunk az elemzésre.

5.1.1. Overfitting és underfitting jelensége

Az overfitting vagy túltanultatás jelensége akkor jelentkezik, ha a modell túl jól illeszkedik a megadott adathalmazra. Ez ahhoz vezet, hogy a lényegi információ helyett az adathalmaz egyedi abnormitásait is megtanulja.

Ezzel ellentétben az underfitting, vagy alultanulás akkor fordul elő ha az adathalmaz nem illeszkedik megfelelően a modellre ezáltal alacsony pontosságot eredményezve. Ez a jelenséget gyakran az okozza, hogy a modell túl egyszerű ahhoz, hogy a probléma összetettségét megragadja. Ezt hivatott szemléltetni a 12. ábra.

Ez azért nagyon fontos, mert a modellünk pontosságát és általánosító képességét befolyásolja. Elkerülésük érdekében fontos a megfelelő minőségű és mennyiségű adat, vagy a megfelelő komplexitású modell kiválasztása.



12. ábra A jelenségek szemléltetése [14]

5.2. Feature engineering

A folyamat célja, hogy a nyers adatokból átalakítással és kiválasztásokkal olyan adatokat készítsen elő, amik jellemzőkkel lettek ellátva. Még hozzá olyan jellemzőkkel, amik jelentőséggel bírnak a célváltozó előrejelzésének szempontjából. A folyamat során az adatokat (itt általában HTTP kérések) számszerűvé alakítani sok szempont alapján lehet, de mindegyikkel jellemvonásokat vagy angolul feature-öket jelölünk ki, amelyek jól reprezentálják az adatokat. Az egész folyamat legfontosabb része a feature engineering, mert ha nem megfelelő jellemvonások kerülnek kiválasztásra, illetve akár ha helytelen súlyokkal, az egész predikció folyamata helytelen eredményeket hozhat.

5.2.1. CountVectorizer

Egy lehetséges megközelítés a feature engineering lépés megoldásához, ha egy lépéssel például a *CountVectorizer()* függvény segítségével a kérésünkben található adattereket felosztjuk és a felosztott tulajdonságokhoz, vagy feature-ökhöz rendelünk súlyokat. Ezeket a hozzárendelt értékeket pedig a modell a következő lépésben hasznosítani tudja a helyes döntéshozásra.

5.3. Modellezés

Sok esetben a modell kiválasztásán múlik az eredményünk helyessége. A tesztelésem során is a három tesztelt megoldás eltérő eredményeket adott megegyező adathalmazon. A modellválasztást az adathalmazhoz kell igazítani, mert nagy mértékben függ a jellemzőitől és a felhasználási céltól. A használt modellek az olvasott szakirodalom alapján kerültek kiválasztásra, mérlegelve azok előnyeit, hátrányait és alkalmazhatóságait.

5.3.1. SGD

A Stochastic Gradient Descent egy olyan felügyelt gépi tanulási módszer, ami segítségével az adathalmazunkat tudjuk osztályozni, használható regresszióra (regression) és klasszifikációra (classification) is. A regresszió folytonos értékek becslési problémáira nyújt megoldást, míg a klasszifikáció objektumokat kategorizál ismert csoportokra. Esetünkben a WAF a kéréseket jó és rosszindulatú csoportokba klasszifikálja, azaz besorolja.

5.3.2. XGB

Az eXtreme Gradien Boost [15] egy felügyelt gépi tanulási módszer, ami döntési fákon alapul. Mint a neve is jelzi, gradient-boosting megközelítést alkalmaz, ami egy úgynevezett ensemble tanulási módszer. Az ensemble francia szó jelentése együttes, ami itt arra utal, hogy ez a tanulási módszer több gyenge tanulót kombinálva alkotja meg a sokkal effektívebb osztályozó vagy regressziós modellt.

5.3.3. LinearSV

A SVM (Support Vector Machine) szintén egy felügyelt gépi tanulási módszer, ami egy olyan hipersíkot, azaz egy kisebb dimenziójú szeparálót keres a térben reprezentált adatok között, ami maximális távra van a legközelebbi adatpontoktól, ezzel nagyon effektív szeparációt érve el akár a klaszterezési, vagy regressziós feladatok során.

5.4. Tesztelés és finomítás

Az adathalmazt általában két részre osztják, amiből a nagyobb rész a modell tanítására a kisebbik halmaz pedig a tesztelésre használatos. Itt fontos kiemelni, hogy nem elégséges az adatokat véletlenszerűen kiválasztani, mert előfordulhatnak nem kívánatos forgatókönyvek. Az első például esetünkben, ha az adathalmaz sok jóindulatú és kevés rosszindulatú kérést tartalmaz (ami realisztikus, mert az összes kérés kis százaléka ilyen) a kiválasztás során megeshet, hogy nem kerül rosszindulatú kérés a teszhalmazba, ezzel helytelen működést eredményezve. Ennek elkerülése érdekében ajánlott a beépített szétválasztás használata, ahol a scikitlearn esetében kiválaszthatjuk, hogy melyik attribútum szerint szeretnénk egyenlően elosztani az adathalmazt.

5.4.1. Kimenetek besorolása

Fontos megjegyezni, hogy habár az ML megközelítést alkalmazó megoldások törekednek a magas pontosság elérésére, így is előfordulhatnak olyan esetek amikor nem megfelelően állapítja meg a támadásvektorokat. A detektálás és ennek helyességét tekintve 4 állapotot különböztetünk meg:

- **True-positive:** akkor lép fel, ha a WAF helyesen detektálja a támadást és blokkolja is azt.
- **True-negative:** a szoftver helyesen megállapítja, hogy a kérés nem tartalmaz támadásvektorokat.
- **False-positive:** amikor a szűrő hibásan észlel egy ténylegesen nem támadó aktivitást és a tevékenységet rosszindulatúnak értékeli. Ez téves riasztáshoz vezethet, amikor a rendszer blokkolja a valódi felhasználók jogos tevékenységeit.
- **False-negative:** a besorolások közül ez a legrosszabb egy WAF esetén, mert itt helytelenül azonosítva a kérést nem titulálja rosszindulatúnak a valóban káros forgalmat, így a támadók könnyen átjutnak a védelmen.

A védelem érdekében a legfontosabb szempont a false-negative és a false-positive detektálások elkerülése, mert míg az előbbi veszélynek teszi ki a webalkalmazást, az utóbbi jelentősen rontja a felhasználói élményt.

5.4.2. Accuracy

A pontosság vagy accuracy egy olyan metrika, amivel a modell előrejelzéseinek helyességi rátáját tudjuk megadni. Ezt a helyes predikciók és az összes predikció hányadosaként kapjuk [16], vagy:

$$Pontosság = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

Ahol TP a True-positive, TN a True-negative, FP a false-negative és FN a False-negative. Azonban ez nem mindig a legjobb metrika, mert bár magas pontosságot kaphatunk bizonyos esetekben (90%-95%) ez sok False értéknél is lehet magas. Például, ha egy WAF esetében nézzük és a helyes megállapítások száma 900, amiből 10 a True-positive és 890 a True-negative, illetve az összes detekció száma 1000, amiből 10 a False-positive és 90 a False-negative. Ha ezt az előbbi képletbe behelyettesítjük 90%-os pontosságot

kapunk, ami jónak mondható, de a 100 rosszindulatú kérésből a WAF csak 10 kérés esetében állapította meg a támadásvektort, így 90 kérés gond nélkül továbbjutott a webalkalmazás felé. A pontosságot ebben az esetben a True-negative-ok magas aránya javítja, ezt a jelenséget „Imbalanced Classification”-nek nevezzük. Ezért van szükség más metrikák használatára, mint például a recall és precision.

5.4.2.1. Precision

A precision, vagy precízió megadja esetünkben, hogy a pozitív észlelések hány százaléka volt valóban helyes. Egy modell, ami nem detektál hamis pozitívokat 100%-os precízióval rendelkezik. Más szóval azt állapítja meg, hogy mennyi az eredmények között a valós pozitívok aránya, azaz, hogy mennyire helyesek az eredmények. Ennek megállapítása a következő:

$$\text{Precízió} = \frac{TP}{TP + FP} \quad (2)$$

5.4.2.2. Recall

A recall, vagy szenzitivitás azt mutatja meg, hogy a tényleges pozitív észlelések hány százaléka lett helyesen azonosítva. Ebből következik, hogy egy modell, ami nem ad hamis-negatívokat 100%-os recall-al rendelkezik. Ennek kiszámítása:

$$\text{Szenzitivitás} = \frac{TP}{TP + FN} \quad (3)$$

5.5. Összegzés

Összefoglalva, a gépi tanulás hatalmas lehetőségeket nyújt a területen, jelentős mértékben javíthatja a rendszerek hatékonyságát és biztonságát. A hagyományos aláírás alapú megoldásokkal szemben a zero-day (nulladik napi) támadásokkal szemben is nyújt védelmet és ez a gépi tanulást alkalmazó módszerek jelentős előnye. A felsorolt értékmérők (pontosság, precízió és szenzitivitás) a tesztelés során a modell kiértékelésében lesz segítségünkre, ezek függvényében tudunk dönteni az eredményünk helyességéről.

6. Tesztelés

A tesztelési fázis során először ellenőriztem a program megfelelő működését a lehetséges beállítások tekintetében, majd hozzákezdtam a szűrők hatásfokának teszteléséhez. A teszteléshez alkalmazott adatokat a következő fejezetben taglalom. Ezek segítségével ellenőriztem a szabály és aláírásalapú megoldásaimat az ismertetett értékelő metrikák segítségével. A követelményeknél említett hatásfokot, aminek meg kell felelnie a fejlesztett programnak, 90%-ban állapítottam meg. Ennek a tesztelt adathalmazon minden szűrő megfelelt.

6.1. Tesztadatok előkészítése

A teszteléshez használt adathalmazt több forrásból állítottam össze a python pandas modul segítségével. Többek közt a GitHubon bárki számára elérhető HttpParamsDataset és PayloadBox könyvtárakat használtam, amiket elláttam a támadás típusát jelző címkével. Habár ez a címke jelenleg csak adatáttekinthetőségi szempontokból van jelen, a program továbbfejlesztésével a modell meg tudná állapítani a támadás típusát is, nem csupán a jelenlétét.

6.2. Aláírásalapú megközelítés

Az aláírás alapú tesztelés során főként a megírt reguláris kifejezéseket teszteltem, amik XSS, SQLi és Prototype pollution támadásokra tesztelésre is kerültek. Itt teljes mértékben tudtam alkalmazni a gépi tanulást alkalmazó szűrő tanításához használt adathalmazt. A használt adatok tekintetében mindenhol elértem a megkívánt hatékonyságot, ami a legrosszabb esetben is 98% feletti érték volt.

Az XSS támadások kivédésére írt regex a tesztelt 7144 támadásból 6996-ot sikeresen kivédett, ami 97,92%-os pontosságot jelent. Az eredmények és a ki nem védett bementek alapos elemzése után észrevettem, hogy ez a hatásfok alábecslés, ugyanis több szemantikai hibát is tartalmazott az adathalmaz.

Az SQL injekció szűrő a tesztelt 10850 SQLi támadásból 10676-ra állapította meg helyesen a támadás típust. Ez 98,39%-os pontosságot jelent, ami szintén megfelel az elvártaknak.

A prototype pollution ellenőrzéséhez sajnos csak kevés tesztadatot találtam. Ez azért lehetséges, mert egy kevésbé ismert és használt támadás típusról lévén szó nem annyira elterjedt a használata. A 43 tesztadtból 43-at sikeresen észlelt így maradéktalanul, 100%-ban megfelelt.

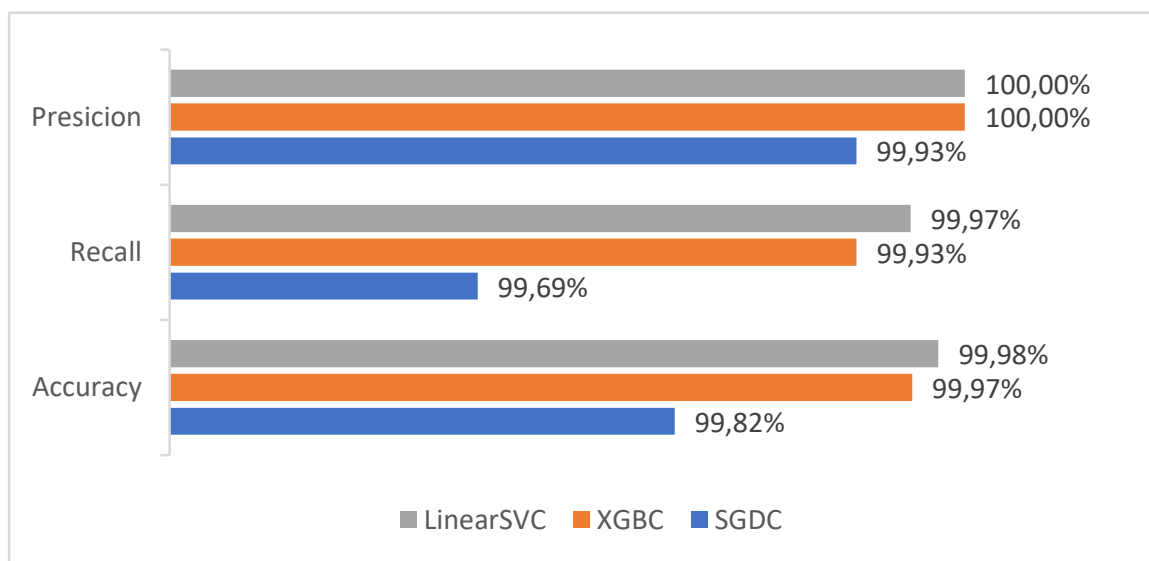
	XSS	SQLi	Prototype pollution
Összes (db)	7144	10850	43
Észlelt (db)	6996	10676	43
Pontosság (%)	98%	98%	100%

2. táblázat A tesztelés eredménye

6.3. Szabályalapú megközelítés

A modellek sikerességét az adathalmazon a scikitlearn által beépített függvényekkel könnyen ellenőrizni tudtam, így itt egyéb tesztesetek írására a statisztikák előállítására nem volt szükségem.

Az alábbi táblázat a modelleket értékeli az 5.4.2 részben ismertetett szempontok szerint:



13. ábra A tesztelés eredménye

6.4. Végtesztelés

A WAF megoldás kézi tesztelését a DVWA (Damn Vulnerable Web Application) applikáción végeztem el a böngésző alkalmazásomból. Ez egy olyan webalkalmazás, amely sebezhetőségeket tartalmaz, amelyeket a felhasználók különböző tesztekkel kipróbálhatnak. Ezeken a teszteken magas hatékonysággal megfelelt a WAF

megoldásom, és a támadások jelentős részét képes kiszűrni ezzel megvédve az alkalmazást.

7. Továbbfejlesztési lehetőségek

A fejlesztett szoftver jelenleg In-Line WAF megoldásként használható, de a válaszdő növekedés minimalizálása érdekében egy Out-of-Path megoldás implementálása jelentősen javítana rajta. Azonban ez a megoldás jóval komplexebb és a kiszolgáló szerverszolgáltatást is hozzá kellene hangolni, hogy egyes üzeneteket továbbítson a WAF-nak elemzés céljából. Az analízisműveletek pontosságán is lehet javítani, akár az aláírás, akár a szabályalapú védelmet. Az aláírásalapú detekciót további támadások lenyomatát tartalmazó regex mintákkal, a szabályalapú szűrőt pedig más megközelítésű, pontosabb ML algoritmusokkal lehetne javítani. A továbbfejlesztést nagyban elősegítené a felhasználható adathalmaz növelése, ezzel pontosabbá lehetne tenni az aláírásalapú megközelítést, még pontosabb minták használatával, illetve a szabályalapú szűrőt is jobban lehetne tanítani.

7.1. Adat menedzsment

További fejlesztési lehetőségként a WAF megoldásba egy modult lehetne beépíteni, ami rögzíti a kéréseket, majd ezek alapján fejleszti tovább a védelmi mechanizmusait. Ezekkel az extra funkciókkal tovább bővítve a programunkat szükségünk lenne egy skálázhatóságot támogató megoldásra, amivel a szervergép terhelését megfelelően el tudjuk osztani.

7.2. Skálázhatóság

A skálázhatóság érdekében a szoftver elosztását támogató megoldások bevezetése lenne javasolt, amivel akár több szerveren elosztva tudná vizsgálni a webszerverek forgalmát, ezzel a válaszdőt és a terhelést csökkentve.

8. Zárójegyzetek

8.1. Felhasználás

A szoftver lokálisan futtatható, rendeltetésszerűen működik és teljes mértékben tesztelésre került, ahol minden téren megfelelt. A futtatáshoz csupán egy Python környezetre van szükségünk, a használt modulok előzetes telepítésével, amit a *requirement.txt* fájlban találunk. A futtatás előtt a konfigurációban a kívánt beállítások és ellenőrzési metódusok megadásával biztosítjuk a kívánt védelem típusát. Majd a *main.py* fájl futtatásával elindítjuk a WAF megoldást, ami legenerálja az adatbázist és a naplózófájlokat. Ha mindez megtörtént a beállított IP címet és portot böngészőből felkeresve a WAF-proxy kezeli a kommunikációnkat a webszerver irányába a kérések előzetes szűrésével.

8.2. Összegzés

A szakdolgozat elkészítése egy rendkívül izgalmas feladatnak bizonyult, mert bár a feladat megkezdésekor nem rendelkezttem a szükséges ismeretek jelentős részével, a kidolgozás folyamán újabb és újabb koncepciókkal ismerkedtem meg és tanultam meg ezek használatát. A HTTP és socket kommunikációval való megismerkedés tartott a legtovább, de elengedhetetlen volt a megvalósításhoz. Ezen kívül a gépi tanulást alkalmazó műveletek bizonyultak még nehéznek, de egyben érdekesnek, amik teljesen ismeretlenek voltak eddig számomra. Ez egy jó lehetőség volt az előbb említettek megismerésére és megtanulására.

Remélem egy érdekes és lényegre törő leírást sikerült létrehoznom a főbb koncepciók ismertetésével a szakdolgozatomban.

Irodalomjegyzék

- [1] Miniwatts Marketing Group, „internet world stats,” Miniwatts Marketing Group, 21 01 2023. [Online]. Available: <https://internetworldstats.com/stats.htm>. [Hozzáférés dátuma: 20 11 2022].
- [2] OWASP Top 10 team, „OWASP,” OWASP, 2021. [Online]. Available: https://owasp.org/Top10/A03_2021-Injection/. [Hozzáférés dátuma: 24 11 2022].
- [3] L. S. Vailshery, „Statista,” statista, 19 01 2023. [Online]. Available: <https://www.statista.com/statistics/806081/worldwide-application-vulnerability-taxonomy/>. [Hozzáférés dátuma: 01 12 2022].
- [4] Cydave, „wordfence.com,” Wordfence, 2023. [Online]. Available: <https://www.wordfence.com/threat-intel/vulnerabilities/>. [Hozzáférés dátuma: 20 12 2023].
- [5] D. Ahmed, „hackread.com,” Hackread, 25 02 2023. [Online]. Available: <https://www.hackread.com/hackers-deface-russia-websites-ukraine/>. [Hozzáférés dátuma: 20 03 2023].
- [6] K. Moore, „Message Header Extensions for Non-ASCII Text,” Network Working Group, University of Tennessee, 1996.
- [7] Kim Zetter, „Wired,” Wired, 26 03 2010. [Online]. Available: <https://www.wired.com/2010/03/heartland-sentencing/>. [Hozzáférés dátuma: 10 10 2022].
- [8] Liran Tal, „snyk.io,” Snyk, 11 09 2019. [Online]. Available: <https://snyk.io/blog/sequelize-orm-npm-library-found-vulnerable-to-sql-injection-attacks/>. [Hozzáférés dátuma: 03 12 2022].
- [9] w3techs, „Historical yearly trends in the usage statistics of server-side programming languages for websites,” w3techs.com, 2023.
- [10] ptsecurity, „ptsecurity.com,” Ptsecurity, 13 02 2020. [Online]. Available: <https://www.ptsecurity.com/ww-en/analytics/web-vulnerabilities-2020/>. [Hozzáférés dátuma: 20 12 2022].

- [11] A. A., „abdalslam.com,” Abdalslam, 03 03 2023. [Online]. Available: <https://abdalslam.com/web-application-firewalls-waf-statistics>. [Hozzáférés dátuma: 20 03 2023].
- [12] sourcedefense, „sourcedefense.com,” Sourcedefense, [Online]. Available: <https://sourcedefense.com/glossary/limitations-of-waf/>. [Hozzáférés dátuma: 30 03 2023].
- [13] Morzeux, „kaggle.com,” Kaggle, 04 09 2020. [Online]. Available: <https://www.kaggle.com/datasets/evg3n1j/httpparamsdataset>. [Hozzáférés dátuma: 14 04 2023].
- [14] Curiously, „curiously.com,” Curiously, 17 10 2019. [Online]. Available: <https://curiously.com/posts/hackers-guide-to-fixing-underfitting-and-overfitting-models/>. [Hozzáférés dátuma: 20 04 2023].
- [15] S. S. Dhaliwal, „Effective Intrusion Detection System Using XGBoost,” p. 24, 21 06 2018.
- [16] Google, „developers.google.com,” Google, 18 07 2022. [Online]. Available: <https://developers.google.com/machine-learning/crash-course/classification/accuracy>. [Hozzáférés dátuma: 20 04 2023].
- [17] N. Jennings, „realpython.com,” Realpython, [Online]. Available: <https://realpython.com/python-sockets/>. [Hozzáférés dátuma: 10 03 2023].

Mellékletek

Mappaszerkezet

```
+Szoftver
|   .gitignore
|   config.toml
|   main.py
|   requirement.txt
+---Analysis
|   __init__.py
|   helperFunctions.py
|   IPChecker.py
+---Data
|   __init__.py
|   loader.py
|   payload_full.csv
+---Proxy
|   __init__.py
|   instanceHandler.py
+---testing
|   testing.py
|   payload_full.csv
+Szakirodalom
|   Effective Intrusion Detection System Using XGBoost.pdf
|   Message Header Extensions for Non-ASCII Text.txt
```

Ábrajegyzék

1. ábra Támadások gyakorisága [3]	11
2. ábra Injekciós támadások súlyosság alapján.....	20
3. ábra A behatolások során leggyakrabban megszerzett információk.....	21
4. ábra In-Line és Out-of-Line szemléltetése.....	23
5. ábra A proxy tervezett általános működése	28
6. ábra Rendszer terve és a használt technológiák (tevékenységdiagram)	31
7. ábra Socket kommunikáció lépései [17]	32
8. ábra Egy HTTP kérés számunkra fontos adatmezői	36
9. ábra A konfiguráció tartalma	39
10. ábra Az adatbázis végleges megvalósítása	39
11. ábra A naplófájl tartalma	40
12. ábra A jelenségek szemléltetése [14].....	43
13. ábra A tesztelés eredménye.....	49

Táblázatjegyzék

1. táblázat Az előfeldolgozás eredménye	38
2. táblázat A tesztelés eredménye	49