KU LEUVEN

FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN

TECHNOLOGIECAMPUS GEEL

Ontwerp van een zelflerende Web Application Firewall

Vincent COX

Promotor: Gustaaf Vermeulen

Co-promotoren: Stijn Jans en Nico Cooman

Masterproef ingediend tot het behalen van de graad van master of science in de industriële wetenschappen: Elektronica-ICT

afstudeerrichting ICT

Academiejaar 2015-2016

VOORWOORD

In het masterjaar industrieel ingenieur is een masterproef een onderdeel van het programma, hierbij kijkt men of de student in staat is om een onderzoeksprobleem zelfstandig op te lossen. Ik heb gekozen voor een project rond security omdat dit zeer nauw aansluit bij mijn persoonlijke interesses en hobby's. Ik ben uiteindelijk uitgekomen bij 'The Security Factory', een bedrijf dat zich specialiseert in security en bestaat uit allemaal medewerkers met gelijkaardige interesses. Ik was hierdoor heel enthousiast om deel uit te maken van dit bedrijf en daar een project op te bouwen.

Uiteraard kruipt hier veel tijd in om zowel het onderzoek te doen als de thesis te schrijven. Hierbij is de hulp van een aantal mensen noodzakelijk en daarvoor zou ik graag enkele mensen willen bedanken.

Als eerste wil ik de heer Stijn Jans bedanken omdat hij de persoon is die mij in contact heeft gebracht met zijn bedrijf en voor verschillende praktische zaken heeft gezorgd. Ik zou ook graag de heer Nico Coomans willen bedanken omdat hij mijn directe begeleider en ook aanspreekpunt was als ik bepaalde vragen had. Zowel Nico als Stijn stonden mij steeds bij als er problemen waren en legden mij alles uit als ik iets niet begreep. Dat ging over zaken zowel binnen als buiten mijn masterproef.

Daarnaast wil ik graag de heer Gustaaf Vermeulen bedanken voor het regelen van verschillende zaken vanuit KU Leuven.

Ook wil ik graag mijn broer en ouders bedanken voor de financiële en morele steun tijdens mijn opleiding.

Ten slotte dank aan de jury voor de tijd en de moeite die nodig was om deze masterproef te lezen en te beoordelen.

SAMENVATTING

Het doel van deze masterpoef is het maken van een zelflerende webapplicatie firewall. De configuratie van de firewall gebeurt in een intuïtieve en eenvoudige web interface zodat het configureren van de firewall eenvoudig en snel is.

In het begin komt de nodige achtergrondinformatie aan bod zodat het doel en de noodzaak van deze masterproef duidelijk wordt. Daarna gaat het over de implementatie en het resultaat. Meer concreet bestaat de opbouw van deze masterproef uit drie grote delen.

In het eerste deel bespreken we de gevaren op het vlak van *cybersecurity* alsook de aanvallers en de gevolgen die hier aan vast hangen. Daarna worden de verschillende aanvalstypes overlopen die het meest worden gebruikt bij het aanvallen van webapplicaties. Deze informatie verduidelijkt het doel en de noodzaak van deze masterproef. Daarnaast geeft het eerste deel ook een duidelijk beeld waartegen deze firewall de applicaties moet beschermen.

Het tweede deel gaat vooral over de gedetailleerde projectdoelstellingen en de implementatie. Modsecurity is een module waar deze masterproef grotendeels mee is opgebouwd en daarom is het belangrijk om een aantal elementen en concepten van Modsecurity te bekijken.

Het laatste deel gaat over het testen van de applicatie, een essentieel onderdeel omdat zo de applicatie gefinetuned en getest kan worden op betrouwbaarheid.

De ontwikkelde zelflerende webapplicatie firewall is op een succesvolle manier ontwikkeld en biedt zeker potentieel voor verdere optimalisatie.

INHOUDSTAFEL

LIJST MET AFKORTINGEN

AWS	Amazon Web Services
CSS	Cascading Style Sheets
CVE	Common Vulnerabilities and Exposures
DVWA	Damn Vulnerable Web Application
GEO	Geografisch
HTML	HyperText Markup Language
ICT	Informatie- en communicatie Technologie
ID's	Identity Document
IIS	Internet Information Services
IP	Internet Protocol
OWASP	Open Web Application Security Project
PHP	Hypertext Preprocessor
RegEx	Regular Expression
SQL	Structured Query Language
UFW	Uncomplicated Firewall
ZAP	Zed Attack Proxy

LIJST MET FIGUREN

LIJST MET TABELLEN

INLEIDING

The Security Factory

The Security Factory is een bedrijf gelegen in Schelle (Provincie Antwerpen) dat diensten en oplossingen aanbiedt om de beveiliging van bedrijfsinfrastructuren en applicaties te verhogen.

De focus ligt vooral op begeleiding en training, maar ook op het testen van bestaande toepassingen, applicaties en de globale beveiliging van een bedrijf. Zo kunnen de zwakheden in kaart worden gebracht en kan er actief gezocht worden naar oplossingen.

Het team achter The Security Factory bestaat uit verschillende medewerkers met elk hun eigen specialisaties en expertises. Deze specialisaties worden gebundeld in het team om zo gedetailleerde oplossingen op maat te kunnen realiseren.

Figuur 1.1: *The Security Factory Logo.*

The Security Factory is een onderdeel van de Cronos Group. Deze groep participeert in meer dan 200 bedrijven en heeft in totaal 3500 werknemers.

Probleemstelling

Het internet heeft de afgelopen jaren een enorme groei gekend, maar helaas heeft de beveiliging ervan dit tempo niet kunnen volgen. Hierdoor komt steeds vaker in het nieuws dat bedrijven last hebben van ongeautoriseerde toegang door hackers. Tegenwoordig heeft elk bedrijf of KMO een webapplicatie of website waardoor klanten online bepaalde handelingen kunnen verrichten zoals bestellen, reserveren of inschrijven. Helaas zijn deze toepassingen niet altijd (goed) beveiligd waardoor hackers deze data kunnen blootstellen. Dit geeft deze bedrijven niet alleen een slecht imago, het is bovendien een gevaar voor de klanten omwille van *password-reuse* (eenzelfde wachtwoord voor meerdere websites). Daarnaast kunnen hackers deze persoonlijke data misbruiken. Een *Web Application Firewall* (WAF) laat enkel geverifieerde input door aan de webapplicatie zodat aanvallen worden afgeweerd. Helaas is het opzetten en onderhouden van een WAF tijdrovend en technisch veeleisend.

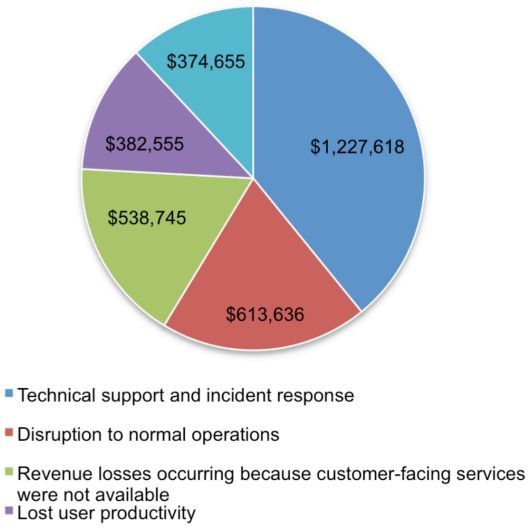
Deze masterproef gaat over de ontwikkeling van een zelflerende *Web Application Firewall* (WAF) die een minimale installatie heeft, zichzelf autonoom bijstuurt via continu monitoring en via een web paneel suggesties geeft om strengere *policies* toe te passen. Bovendien moet de WAF verschillende applicaties tegelijkertijd kunnen beveiligen. Bedrijven kunnen dan met deze zelflerende WAF toch hun webapplicatie beveiligen met beperkte kennis.

1 SECURITY VANDAAG DE DAG

Het gebeurt zeer frequent dat cyberaanvallen in het nieuws komen. De slachtoffers zijn zowel kleine als grote bedrijven maar ook overheidsinstanties en non-profit organisaties. Recent zijn zelfs een aantal kritieke sectoren geviseerd zoals ziekenhuizen [1], energiefaciliteiten [2], leger [3] en industrie [4]. Een aantal zaken komen nu meer gedetailleerd aan bod om een beter idee te krijgen van de impact van deze aanvallen en waarom tegenwoordig zoveel cybercriminaliteit bestaat.

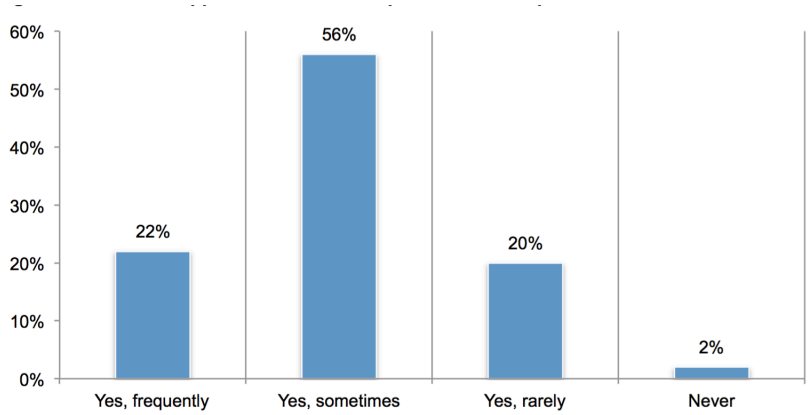
1.1 Schade door aanvallen

Een recent onderzoek [5] ondervroeg 594 medewerkers van grote organisaties. Aanvallen op webapplicaties hebben deze organisaties voor meer dan drie miljoen dollar aan schade gekost. De verschillende schadeposten staan weergegeven in de volgende figuur.



Figuur 1.1: *De kosten van een ernstige aanval op een web applicatie. [5]*

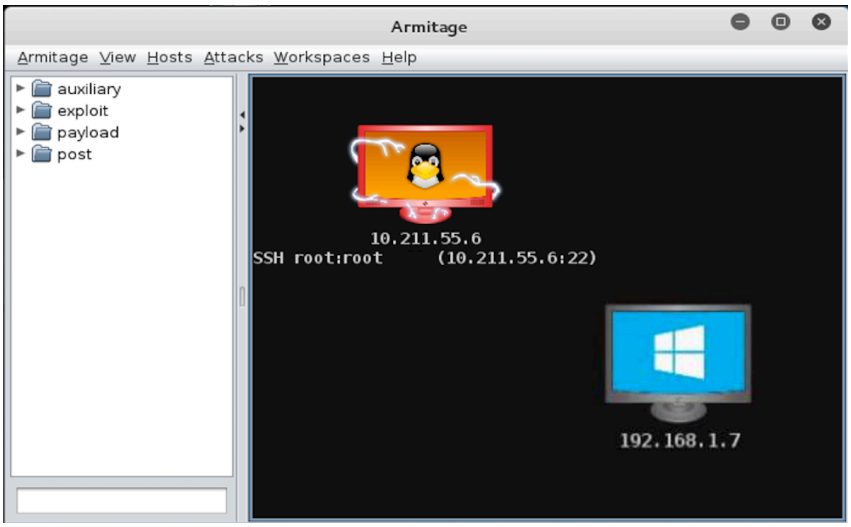
De grootste schadeposten zijn de technische interventies en ondersteuning. Daarnaast is het stilliggen van de normale werkzaamheden de tweede grootste schadepost. Ook de frequentie van aanvallen in de afgelopen twaalf maanden is zorgwekkend: amper twee procent van de bedrijven zijn in deze tijdsperiode niet aangevallen geweest.



Figuur 1.2: *Frequentie waaraan bedrijven of hun web applicatie is aangevallen in de laatste 12 maanden. [5]*

1.2 Hacking tools

Op het internet zijn een groot aantal *hacking tools* gratis te downloaden. Tools zoals Nmap maken het mogelijk om netwerken te verkennen door een volledige scan te draaien. Nog een andere veelgebruikte tool is SQLmap. Deze kan een automatische SQL-injectie uitvoeren op de meest gebruikte soorten databases. Armitage en Cobalt Strike (betaalde versie van Armitage) zijn ook bekende namen in de securitywereld, dit zijn grafische tools die het toelaten om met enkele clicks een computer of server te hacken [6].



Figuur 1.3: *Screenshot van Armitage. Linksboven een gehackte linux machine onder controle van de aanvaller en rechtsonder een windows 10 machine.*

Het besturingssysteem Kali is uitgerust met een groot aantal *hacking tools*, waaronder de *tools* die net al zijn vernoemd. Iemand met relatief weinig kennis heeft toch vrij krachtige software in handen. Daarnaast gebruiken hackers vaak Tor om aanvallen anoniem uit te voeren. Tor is een open netwerk voor anonieme communicatie gebaseerd op *onion routing* via *onion* servers. Het is duidelijk dat een goede beveiliging essentieel is omdat aanvallers een groot arsenaal aan tools binnen handbereik hebben en hun eigen identiteit kunnen beschermen tijdens deze aanvallen.

1.3 Kwetsbaarheden

Dagelijks komen er ook kwetsbaarheden uit, waarmee aanvallers applicaties en servers eenvoudig kunnen hacken. Enkele bekende sites zijn *0day.today* en *www.exploit-db.com*.

Op deze sites kunnen bezoekers per categorie zoeken naar gaten (*exploits* genoemd) in software. De meeste zijn ondertussen al gepatched, maar het nadeel is dat niet alle software tijdig wordt geüpdatet. Een goed voorbeeld is de "revslider" Wordpress *plugin*. Veel websites gebruiken deze *plugin* en na publicatie van deze *exploit* was er massaal misbruik om websites geautomatiseerd te hacken. Op een gegeven moment waren er meer dan honderdduizend [7] Wordpress websites gehackt. Bezoekers van deze sites kregen een groot arsenaal aan *exploits* voorgeschoteld (voornamelijk gebaseerd op Flash en Java) waardoor er duizenden computers geïnfecteerd werden.

1.4 Verouderde software

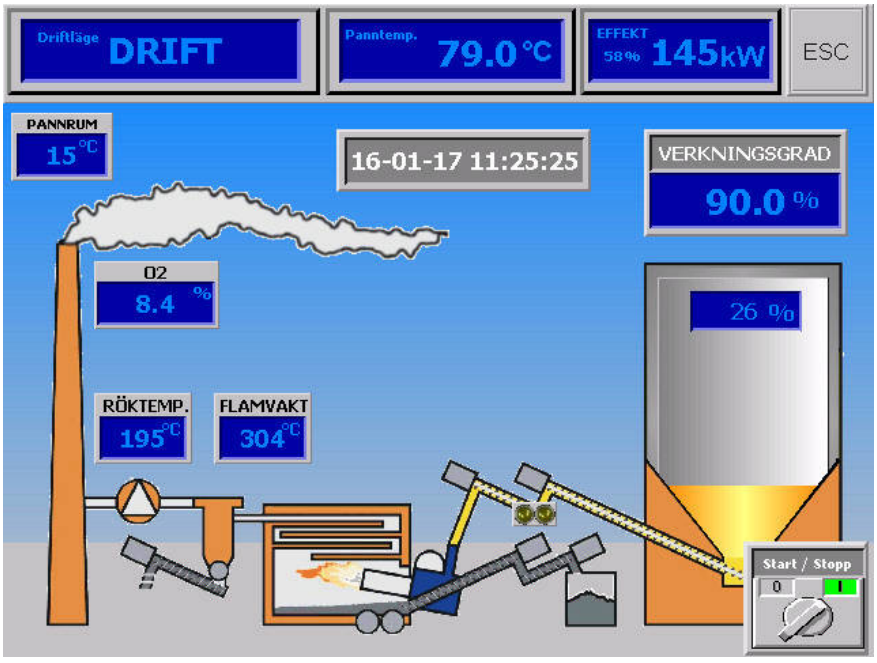
Software en websites updaten kan veel tijd in beslag nemen en het kost bovendien veel geld om nieuwe pakketten aan te kopen. Bedrijven stellen dit vaak uit waardoor deze oude software vaak enkele jaren draait zonder updates. Zoals in de vorige sectie beschreven staat, komen er geregeld mankementen in software boven water die aanvallers kunnen misbruiken.

In een artikel van Bloomberg [8] stelt men dat nog een groot aantal bankautomaten op Windows XP draaien. Dit is redelijk problematisch omdat Microsoft de ondersteuning hiervan heeft stopgezet.

1.5 Zoekmachines

Op het internet zijn er ook zoekmachines die zijn gericht op alle toestellen die met het internet zijn verbonden. Dit gaat van webcams, gebouwen, bewakingscamera's, industriële SCADA-systemen, computers die VNC hebben openstaan, …

Een bekende zoekmachine is *shodan.io* en *censys.io*. Op deze sites kunnen bezoekers zoeken op kwetsbare softwareversies en slecht geconfigureerde webservers die hun data onbewust blootleggen. Dit zijn digitale goudmijnen voor hackers.



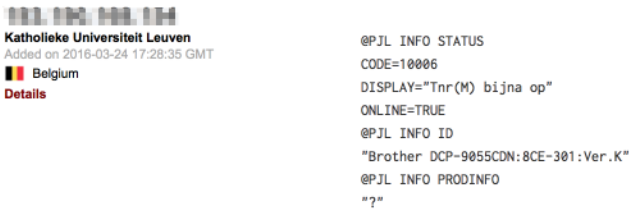
Figuur 1.4 *Industrieel bedieningspaneel gevonden via shodan.io [9].*

Het zoeken op de organisatie "Katholieke Universiteit Leuven" geeft zeer interessante voorbeelden. Deze zoekterm leverde bij *Shodan* 1688 resultaten op.

Protocol	Aantal
HTTP	686
HTTPS	621
SSH	121
FTP	13
DNS	12

Tabel 1.1: *Verschillende soorten services en hun aantallen.*

In deze data duiken ook een aantal printers op. De volgende figuur toont de informatie over een willekeurige printer.

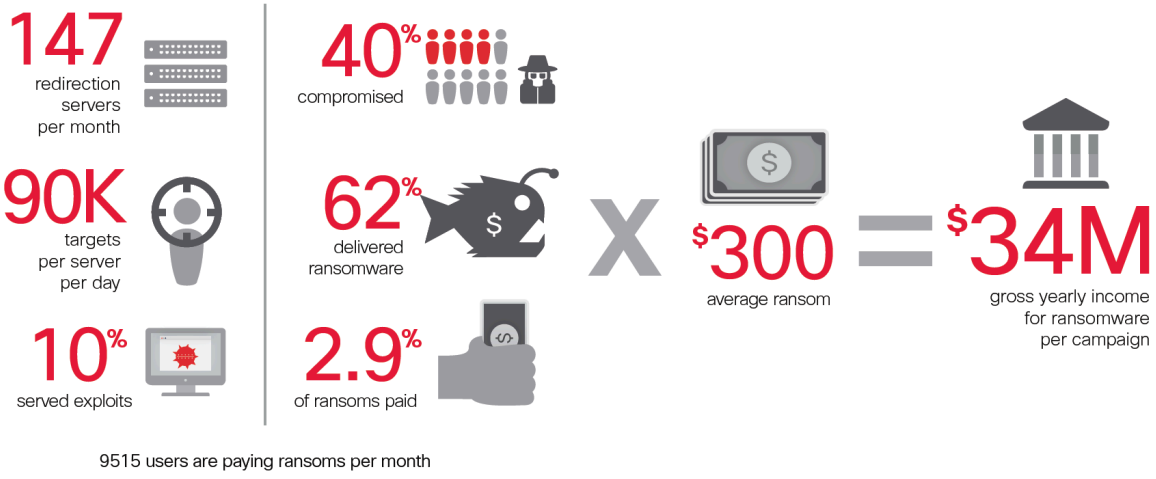


Figuur 1.5: *Een printer van de Katholieke Universiteit Leuven waarbij de inkt bijna op is.*

Daarnaast zijn er ook verouderde besturingssystemen zoals Windows XP en Linux 2.4 te vinden. Het is algemeen gekend dat Windows XP niet meer veilig is en aanvallers dit actief misbruiken. Het volledige rapport staat in bijlage 2.

1.6 Opbrengsten cybercriminaliteit

Een recent voorbeeld is de "Angler" exploit kit. Dit is een gesofisticeerde code die recente beveiligingsgaten in software gaat misbruiken om een computer te infecteren met virussen en *malware*. De meest misbruikte software zijn Java, Flash Player en Microsoft Silverlight. Het verspreiden van deze *exploit kit* gebeurt via geïnfecteerde advertentienetwerken, maar ook via gehackte websites. De grootste bron van inkomsten ontstaat door *cryptolockers* die aanvallers verspreiden via de *exploit kit*. Dit is een virus dat de bestanden op een computer versleuteld en de gebruikers onder druk zet om hun bestanden terug te krijgen voor een bepaald bedrag. Dit bedrag moet meestal in Bitcoin worden betaald zodat het traceren van de ontvanger onmogelijk wordt. De grootste verspreider van deze *exploit kit* verdiende op deze manier meer dan 30 miljoen dollar volgens het onderzoek van Cisco [10].



Figuur 1.6: *Opbrengsten Angler Exploit Kit. [10]*

Bovenstaande figuur toont hoe dit bedrag is berekend. Helemaal links wordt het meer technisch voorgesteld met het aantal geïnfecteerde servers en bezoekers. Midden van het schema wordt het al duidelijker waar het geld vandaan komt. Ongeveer 40% van de bezoekers die deze exploit kit voorgeschoteld kreeg, geraakten geïnfecteerd. Daarvan ontvingen 62% van de gebruikers een variant van *ransomware*. En uit deze groep betaalden 2,9% de afkoopsom om hun bestanden zogezegd terug te krijgen. Helaas zijn er nog altijd mensen die geen back-ups maken en als enige oplossing dan maar betalen.

1.7 Type aanvallers

Zoals in de vorige sectie is aangegeven, valt er daarmee veel geld te verdienen. Achter deze aanvallen zitten een groot aantal criminele organisaties en/of individuen. Soms zijn er ook zogenaamde "scriptkiddies" [11] die voor het amusement sites aanvallen met geautomatiseerde tools. Ze beschikken niet over uitgebreide kennis, maar door het gebruik van geautomatiseerde tools kunnen ze toch genoeg schade veroorzaken zodat deze groep ook niet te onderschatten is.

Als laatste zijn er ook nog de "overheidshackers". Deze groep zit op een veel hoger niveau dan de vorige groepen. Ze hebben het budget, middelen en de kennis om gesofisticeerde aanvallen uit te voeren. Een goed voorbeeld is het Stuxnet virus [12] dat Iraanse uraniumcentrifuges aanviel. Dit virus wist verschillende jaren onopgemerkt te werken en had de functionaliteit om gecontroleerd centrifuges te vernietigen.

1.8 Gevolgen

De gevolgen van een gehackte site zijn voelbaar bij zowel het bedrijf als bij de bezoekers van de site. Naast naamschade en het uitlekken van klantendata zijn er ook grote financiële verliezen. Bedrijven werken jaren aan hun reputatie en in één enkele hack kan het vertrouwen en de geloofwaardigheid van een merk instorten. Op het moment van een hack hangen slachtoffers uren aan de telefoon met *hosting providers*, ontwikkelaars en *security professionals* terwijl het bedrijf elke seconde naamschade ondergaat en inkomsten verliest. Als de gehackte site *malware* verspreid is de kans ook zeer groot dat Google *safebrowsing* deze site op de blacklist zet. Gebruikers krijgen dan de melding dat deze site *malware* verspreid wat veel mensen afschrikt.

> Safe Browsing shows people more than 5 million warnings per day for all sorts of malicious sites and unwanted software, and discovers more than 50,000 malware sites and more than 90,000 phishing sites every month. [13]

Als de hackers gebruikersdata kunnen bemachtigen is dit een nog een ander verhaal. Gegevens uit gehackte databases worden verkocht op de zwarte markt of worden gepost op bepaalde hackforums.

Prijzen [14] van deze data zijn uiteenlopend, maar vandaag kan men duizend gestolen emailaccounts kopen voor 0,50$ tot 10 dollar. Gevoeligere data zoals creditcard gegevens kosten $ $0.10 en $20 per stuk. De prijs hangt af van het land van herkomst, de hoeveelheid metadata en hoe recent de data is gestolen.

De website *haveibeenpwned.com* laat mensen zien of hun data is uitgelekt. Deze site is gemaakt door een bekende security ingenieur Troy Hunt die de wereld rondreist als consultant om de security van bedrijven te verbeteren.

Figuur 1.7: *De top tien gegevenslekken van haveibeenpwned.com [15].*

Dit is een interessant initiatief omdat veel mensen hetzelfde wachtwoord hergebruiken (ook wel *password reuse* genoemd) op websites, werkcomputers en andere toestellen. Als een hacker een wachtwoord heeft bemachtigd van een website kan bij *password reuse* ingelogd worden op bijvoorbeeld het mailaccount van het slachtoffer. Zo kan de aanvaller meer informatie verzamelen over andere sites en het wachtwoord resetten via het emailadres. Veel websites hebben namelijk een optie om het wachtwoord te resetten via het emailadres.

2 DOELSTELLINGEN EN VEREISTEN

Het voorgaande hoofdstuk beschrijft wat de gevaren zijn van gehackte websites en ook van gaten in software in het algemeen. In dit hoofdstuk komen de doelstellingen van de *Web Application Firewall* (WAF) aan bod die een website of webapplicatie zal proberen te beschermen. Het hoofddoel is dus een WAF te ontwerpen die aan de hand van bepaalde regels een website zal beschermen. Via een web interface zal de gebruiker gemakkelijk bepaalde regels kunnen toevoegen en verwijderen. Als laatste moet er ook een zelflerend gedeelte zijn zodat uit binnenkomend verkeer er bepaalde firewallregels kunnen voorgesteld worden.

2.1 Firewall

Om aanvallen en vreemde data tegen te houden, is het vanzelfsprekend een firewall te gebruiken. Omdat het ontwikkelen van een firewall tijdsintensief is, maakt deze masterproef gebruik van de *open source* firewall Modsecurity. Dit is een bestaande implementatie die volledig werkt, uitgebreid getest is en onder permanente ontwikkeling staat. De broncode staat op Github waar ook de ondersteuning voor ontwikkelaars te vinden is. De implementatie van deze masterproef is bijgevolg volledig uitgebouwd op Modsecurity.

Het grote nadeel van Modsecurity is dat er veel technische kennis voor vereist is. De bestandsstructuur, de firewall regels en de commando's moeten grondig gekend zijn om deze module juist te configureren. Daarom is bijkomend het maken van een eenvoudige web interface noodzakelijk zodat iemand met beperkte kennis van ICT toch een WAF kan instellen.

2.2 Web interface

De web interface moet zo eenvoudig mogelijk blijven om een zo groot mogelijk publiek aan te trekken. Het eenvoudig activeren en deactiveren van *firewall* regels moet met enkele muisklikken kunnen gebeuren. Daarnaast moeten er ook voldoende mogelijkheden aanwezig zijn zodat zelfs professionals hun eigen Modsecurity regels kunnen toevoegen. De web interface moet ook beschermd zijn met een login zodat buitenstaanders die geen toestemming hebben, geen gebruik kunnen maken van de web interface.

De veiligheid van deze web interface is een topprioriteit. Omdat de gebruiker hiermee de firewall aanstuurt, plakken hier ook bepaalde risico's aan vast. Daarom moeten er voldoende maatregelen genomen worden zodat er een veilige interactie gebeurt met de firewall en de gebruiker. Het hoofdstuk over de implementatie komt daar verschillende keren op terug.

2.3 Zelflerend gedeelte

De applicatie moet ook een zelflerend gedeelte hebben waardoor de gebruiker voorstellen krijgt om extra regels toe te voegen aan Modsecurity. Door bepaalde parameters in een applicatie te filteren, kan de veiligheid aanzienlijk verhoogd worden.

Een standaardformulier met een postcode zal enkel maar cijfers bevatten. Als de gebruiker dan instelt dat deze parameter enkel cijfers kan bevatten en geen symbolen of speciale karakters, kan injectie van commando's en data voorkomen worden.

Het is heel belangrijk dat de gebruiker deze regels zelf moet goedkeuren. Hierdoor is de gebruiker volledig verantwoordelijk voor de controle en kunnen verkeerde inschattingen

door het zelflerende gedeelte vermeden worden. De systemen die in productie staan, mogen immers geen onverwachte fouten bevatten door het gebruik van verkeerde regels.

2.4 Dataverzameling

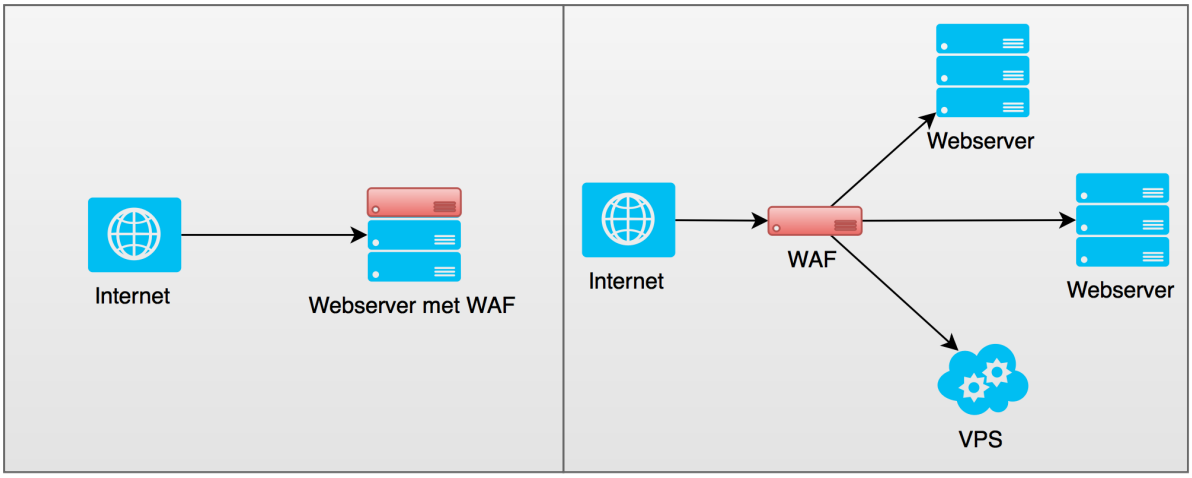
Natuurlijk moet de implementatie data verzamelen waarop het zelflerende gedeelte zijn berekeningen zal toepassen. Dit kan enkel als de firewall alle belangrijke informatie uit een binnenkomend verzoek haalt en opslaat. Dit moet efficiënt gebeuren en mag zeker de webserver niet te hard vertragen. Als de laadtijd van een website te lang duurt heeft dit een negatieve impact op de bezoekers.

2.5 Basisbescherming

Omdat de firewall in het begin nog geen data heeft om regels op te stellen, kan de gebruiker al wel een basisbescherming instellen tegen algemene en bekende aanvalsmethoden. Dit kan door de basisregels van Modsecurity te gebruiken. Deze regels beschermen tegen de zogenaamde "OWASP Top Ten" [16]. Dit is een lijst met de meest voorkomende beveiligingsrisico's bij webapplicaties. Hoofdstuk vier van deze masterproef gaat daar dieper op in.

2.6 Inzetmogelijkheden

De gebruiker moet de WAF op een bestaande webserver kunnen installeren, maar ook op een standalone WAF server die voor enkele webservers staat.



Figuur 2.1: *Inzetmodes van de WAF. Links staat een embedded opstelling en rechts een standalone installatie.*

De linkse figuur toont een implementatie bij kleine websites of webapplicaties. De rechtse figuur toont eerder een implementatie als de gebruiker een groot aantal sites wil beveiligen. Er is dan één centrale WAF die al het verkeer van de buitenwereld opvangt. De webservers die achter de WAF staan zijn van buitenaf niet bereikbaar zodat aanvallers hier niet aankunnen.

3 MODSECURITY

Modsecurity is een module die gebruikers bij een webserver kunnen plaatsen of als een standalone server kunnen gebruiken die dan voor enkele webservers staat. Momenteel zijn de webservers Nginx, Apache en IIS (microsoft) ondersteund [17].

Modsecurity is een *realtime* applicatie voor monitoren, loggen en access control. Omdat Modsecurity *open source* is, kunnen gebruikers de code volledig aanpassen waardoor dit zeer interessant is voor bedrijven en ontwikkelaars.

Enkele interessante mogelijkheden zijn [18]:

- **Real-time monitoring en access control**
  Omdat Modsecurity toegang heeft tot alle *web datastreams*, is het mogelijk om uitgebreide *logging* in te stellen.

- **Virtual patching**
  Hierdoor is het mogelijk om een applicatie die vatbaar is voor een gekende kwetsbaarheid, toch te *patchen* zonder de code van de software aan te passen. Modsecurity houdt dan de *requests* tegen die deze kwetsbaarheid misbruiken. Ontwikkelaars hebben hierdoor meer tijd om een *bug* uit de software te halen.

- **Beveiligen van web applicaties**
  Het is mogelijk om gevallen van *cross site scripting*, *cross site forgery*, SQL injectie en andere vaak gebruikte kwetsbaarheden tegen te houden met standaard firewallregels. Deze regels zijn opgesteld door OWASP, een instantie die zich bezighoudt met het verbeteren van de beveiliging bij webapplicaties.

- **Uitgebreide mogelijkheden**
  Modsecurity kan ook geavanceerde taken als XML *parsen* en *proxy requests* filteren. Het is ook mogelijk om Modsecurity als *reverse proxy* te gebruiken waardoor gebruikers een bestaand netwerk niet hoeven aan te passen.

Cloudflare, een bedrijf dat in deze sector zit, maakt ook gebruik van Modsecurity om hun klanten te beschermen tegen aanvallen. Ze gebruiken een aangepaste versie [19] van Modsecurity waarbij een aantal zaken in Lua zijn geschreven.

Het is wel zo dat Modsecurity standaard niets doet. Alleen door Modsecurity te vertellen wat niet mag, gaat deze module actie ondernemen.

3.1 Regelstructuur

Elke regel in Modsecurity is opgebouwd volgens hetzelfde formaat. De volgende code toont de structuur van de Modsecurity regels [17].

```
1.  SecRule VARIABLES OPERATOR [TRANSFORMATION_FUNCTIONS, ACTIONS]
```

Er zijn vier bouwblokken zichtbaar in bovenstaande formule, waarvan de laatste twee optioneel zijn.

Variabelen

Variabelen identificeren stukken van een HTTP-transactie waarmee een regel werkt. Modsecurity steekt alle informatie van elke transactie in variabelen zodat firewallregels deze informatie kunnen gebruiken. Deze variabelen bevatten ruwe bytes van data dus het kunnen ook speciale karakters en bytes zijn in plaats van platte tekst.

Operators

Operators specifiëren hoe Modsecurity een variabele moet analyseren. *Regular Expressions* (REGEX) zijn alomtegenwoordig.

Transformatie functies

Een regel kan één of meerdere transformaties bevatten. Deze transformaties dienen om bepaalde input te bewerken voordat een operator deze analyseert. Hackers coderen vaak hun kwaadaardige code om detectie te vermijden. Door bepaalde zaken te gaan decoderen kan Modsecurity toch de klare tekst gaan beoordelen.

Acties

Dit bepaalt wat er moet gebeuren als een regel een *match* (overeenkomst) maakt. Enkele voorbeelden zijn het rapporteren naar een logfile, blokkeren van een *request* of het dreigingsniveau verhogen.

Met bovenstaande bouwblokjes kunnen gebruikers van Modsecurity heel verfijnde en effectieve regels maken. Het toelichten van alle mogelijke variabelen, operators en transformatie valt buiten de scope van deze masterproef. Het boek "Modsecurity Handbook" van Ivan Ristic is zeer geschikt voor diegenen die hier interesse voor hebben.

3.2 Mappenstructuur

Modsecurity maakt ook gebruik van een vaste mappenstructuur. Net zoals bij Apache zijn er mappen voor configuratie, modules en standaardregels. Afhankelijk van het *operating system* kan dit anders zijn, maar de gebruiker kan best de standaard mappen gebruiken voor configuratie. Als de gebruiker toch afwijkt van deze standaardstructuur, kunnen er bij updates bepaalde zaken overschreden worden en dit kan ook leiden tot fouten. Buitenstaanders die achteraf de configuratie willen aanpassen kunnen ook moeilijkheden ondervinden om bepaalde zaken terug te vinden.

Een overzicht van de mappenstructuur bij Ubuntu wordt weergegeven in de volgende tabel.

Locatie	**Beschrijving**
/var/log/apache2/	Logboekbestanden worden hier opgeslagen
/etc/modsecurity/	Configuratiebestanden van Modsecurity
/usr/share/modsecurity-crs/	Configuratiebestanden van Modsecurity
/usr/share/modsecurity-crs/base_rules/	Standaard regels die beschermen tegen veelvoorkomende aanvallen

***Tabel 3.1**: Mappenstructuur van Modsecurity op een Ubuntu systeem [17].*

3.3 Apache configuratie

De configuratie van Modsecurity vereist enkele wijzigingen in de Apache configuratie. In de volgende code wordt een standaard configuratie voor een website gegeven.

```
1.  <VirtualHost *:90>
2.      SecAuditLog /var/log/apache2/modsec_audit_hacked_site.log
3.      SecDebugLog /var/log/apache2/modsec_debug.log
4.      ServerAdmin admin@example.com
5.      ServerName hack-site.com
6.      DocumentRoot /var/www/hacksite
7.      ErrorLog ${APACHE_LOG_DIR}/error.log
8.      CustomLog ${APACHE_LOG_DIR}/access.log combined
9.  </VirtualHost>
```

Om Modsecurity te activeren moeten er enkele regels bijkomen:

```
1.  <VirtualHost *:90>
2.      SecRuleEngine On
3.      SecRequestBodyAccess On
4.      IncludeOptional virtual-hosts/hack_site/enabled-rules/*.conf
5.      IncludeOptional virtual-hosts/hack_site/custom_suggestions/*.conf
6.      SecAuditEngine RelevantOnly
7.      SecAuditLog /var/log/apache2/modsec_audit_hacked_site.log
8.      SecDebugLog /var/log/apache2/modsec_debug.log
9.      SecDebugLogLevel 3
10.     ServerAdmin admin@example.com
11.     ServerName hack_site
12.     DocumentRoot /var/www/hacksite
13.     ErrorLog ${APACHE_LOG_DIR}/error.log
14.     CustomLog ${APACHE_LOG_DIR}/access.log combined
15. </VirtualHost>
```

In de volgende tabel staan de betekenissen van deze parameters.

Parameter	**Beschrijving**
SecRuleEngine	Deze parameter geeft aan in welke modus Modsecurity moet werken. Dit moet dus op aan (“On”) staan om de firewall te activeren. Maar het kan ook in “DetectionOnly” staan waardoor de firewall alles logt, maar niets blokkeert.
SecRequestBodyAcces	Deze parameter geeft aan of Modsecurity in de body van een request moet kijken. Dit heeft een zekere prestatie impact, maar is een vereiste om *POST-requests* te kunnen filteren.
IncludeOptional	Dit gaat de nieuwe firewall regels van de implementatie toevoegen aan Modsecurity.
SecAuditLog en SecDebugLog	Deze parameters geven de locatie aan waar Modsecurity meldingen en fouten moet opslaan.
SecDebugLogLevel	Deze parameter stelt het logniveau in. Niveau drie is aanbevolen omdat dit de voornaamste gebeurtenissen opslaat. Een hoger nummer betekent meer bijkomende info en debuginformatie.

***Tabel 3.2**: Parameters in het configuratiebestand van Apache [20].*

3.4 Updates en ondersteuning

Modsecurity is een *open source* project onder actieve ontwikkeling. Op geregelde momenten komen er updates en nieuwe versies beschikbaar. De ontwikkeling gebeurt deels door betaalde werknemers van TrustWave, maar vooral door ontwikkelaars die gratis aan dit project werken.

Omdat Modsecurity een opensource project is, bestaat er *community support* op Github[1]. Dit wil zeggen dat er grotendeels vrijwilligers zullen proberen te helpen bij problemen.

Er bestaat ook de mogelijkheid om commerciële ondersteuning te krijgen via Trustwave. Dit bedrijf geeft ondersteuning bij algemene problemen, maar ook bij updates en onderhoud.

Daarnaast zijn er ook commerciële firewall regels beschikbaar [21]. Deze regels worden dagelijks geüpdatet en hebben meer functionaliteit dan de basisregels die standaard in Modsecurity zitten.

[1] Github is een veelgebruikte website om aan softwareontwikkeling te doen. Github biedt zowel gratis als betaalde accounts voor opensourceprojecten. Het is de meest populaire hostingsite gebasseerd op Git.

4 AANVALSTYPES

Om een idee te krijgen over de verschillende aanvalstypes waarmee de firewall te maken krijgt, komen in dit hoofdstuk een aantal belangrijke concepten aan bod.

4.1 OWASP-organisatie

OWASP (Open Web Application Security Project) is een non-profit organisatie die zich specialiseert in het onderzoeken van de beveiliging bij webapplicaties. Iedereen is vrij om deel te nemen en al het materiaal is gratis beschikbaar. OWASP beveelt geen producten of diensten aan omdat de organisatie een neutrale positie wil innemen [22].

4.2 OWASP Top Ten

OWASP is bekend voor hun "OWASP Top Ten" lijst omdat die in de securitywereld als een *checklist* wordt aanzien. De onderstaande tabel toont de laatste versie van de OWASP-top die is opgemaakt in 2013.

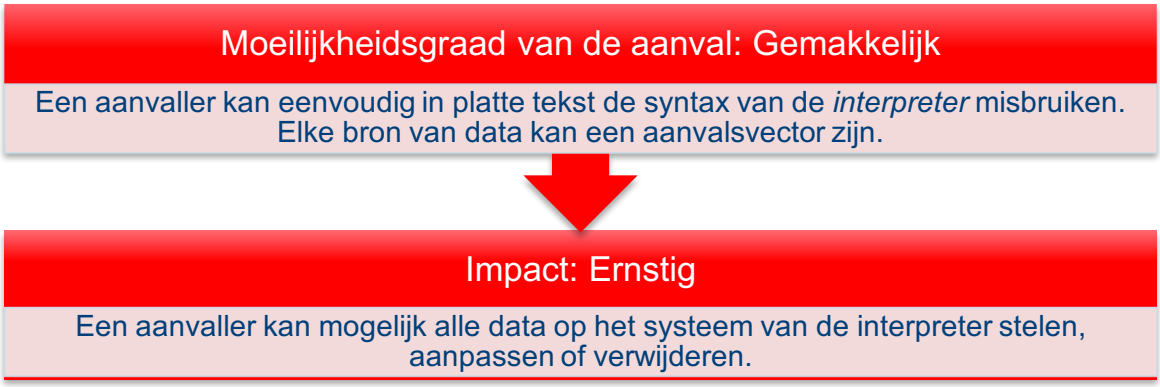
1	Injection
2	Broken Authentication and Session Management
3	Cross-Site Scripting (XSS)
4	Insecure Direct Object References
5	Security Misconfiguration
6	Sensitive Data Exposure
7	Missing Function Level Access Control
8	Cross-Site Request Forgery (CSRF)
9	Using Components with Known Vulnerabilities
10	Unvalidated Redirects and Forwards

Tabel 4.1: *De "OWASP top ten" lijst die is opgemaakt in 2013 [16].*

Deze tien categorieën worden nu verder besproken. De moeilijkheidsgraad, impact en een voorbeeld van een aanvalsscenario schetsen een duidelijk beeld tegen wat de firewall moet beschermen. Merk op dat de moeilijkheidsgraad van de aanval een combinatie is van de nodige kennis, tools en het al dan niet vaak voorkomen van dit beveiligingslek op het internet.

4.2.1 Injectie

Injectie risico's bij SQL en LDAP[2] treden op als een aanvaller onbekende data naar een *interpreter*[3] stuurt in de vorm van een commando of query. De data kan namelijk zodanig gekozen zijn dat het systeem kwaadaardige commando's uitvoert of dat het data zonder autorisatie vrijgeeft.



Voorbeeld van een aanvalsscenario

Een kwetsbare applicatie gebruikt onbekende data in de volgende SQL-opdracht:

```
1.  String query = "SELECT * FROM gebruikers WHERE gebruikerID='" +
    request.getParameter("id") + "'";
```

De aanvaller kan dan de "id" parameter aanpassen naar onderstaand voorbeeld.

```
1.  ' or '1'='1
```

Dit kan in praktijk gebeuren door volgende URL te gebruiken:

```
1.  http://example.com/app/accountView?id=' or '1'='1
```

Dit geeft dan alle rijen van de tabel "accounts" terug, wat normaal niet zou mogen.

---

[2] *Lightweight Directory Access Protocol* (LDAP) is een netwerkprotocol dat beschrijft hoe gegevens uit *directoryservices* benaderd moeten worden over bijvoorbeeld TCP/IP.

[3] Een *interpreter* is een computerprogramma dat de broncode van computerprogramma's vertaalt in een voor de processor begrijpelijke vorm en meteen uitvoert, in tegenstelling tot een *compiler*, die programma's opslaat in een dergelijke vorm.

4.2.2 Broken Authentication and Session Management

Applicatiefuncties die instaan voor de authenticatie en sessie management zijn vaak verkeerd geïmplementeerd. Hierdoor kunnen aanvallers sleutels en *session tokens* aanpassen en zich voordoen als iemand anders.

Voorbeeld van een aanvalsscenario

Een vliegtuigmaatschappij met een slecht beveiligde website zet sessie ID's in de URL:

```
1.  http://example.com/sale/saleitems;jsessionid=2P0OC2JSNDLPSKHCJUN2JV?dest=Hawaii
```

Een geauthentiseerde gebruiker van deze site wil deze link doorsturen naar enkele vrienden via email. Wanneer zijn vrienden op deze link klikken kunnen ze zijn sessie gebruiken en aankopen doen via zijn creditcard.

4.2.3 Cross-Site Scripting (XSS)

XSS-risico's treden op wanneer een applicatie onbekende data doorstuurt naar de webbrowser van een bezoeker zonder validatie van de gegevens. Hierdoor kunnen aanvallers bepaalde scripts uitvoeren in de browser van de bezoekers.

Voorbeeld van een aanvalsscenario

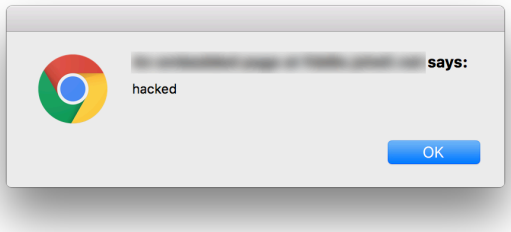
Code webpagina:

```
1.  <?php echo $_GET['id']; ?>
```

Een aanvaller stuurt een link naar de gebruiker die weinig argwaan heeft omdat de URL van een bekende en vertrouwde site is.

```
1.  https://example.com/index.php?id=<script>alert('hacked');</script>
```

Als de gebruiker op deze link klikt geeft de browser de volgende melding weer:

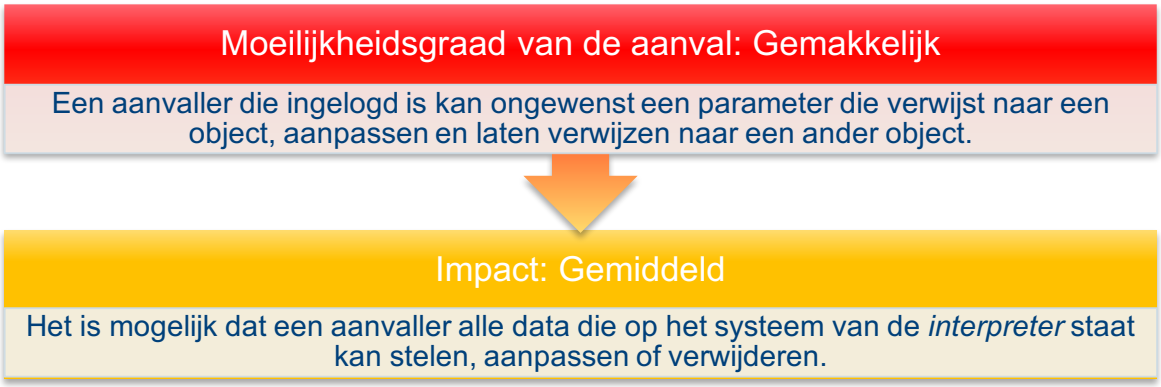


Figuur 4.1: *Kwaadaardige code in de vorm van een pop-up bericht als voorbeeld.*

In dit voorbeeld ziet het slachtoffer enkel een onschuldig pop-up bericht, maar de aanvaller kan bijvoorbeeld ook scripts laten uitvoeren die gegevens kunnen stelen.

4.2.4 Insecure Direct Object References

Dit aanvalstype treedt op als een ontwikkelaar een object zoals een bestand, map of databasesleutel blootstelt. Zonder te kijken of een gebruiker toestemming heeft om andere waardes op te vragen kan een aanvaller dit misbruiken om data ongeautoriseerd te bekijken en aan te passen.



Voorbeeld van een aanvalsscenario

Een kwetsbare webapplicatie toont de gebruikersinformatie van de huidige gebruiker (Vincent) op:

```
1.  https://example.com/accountinfo.php?account=vincent
```

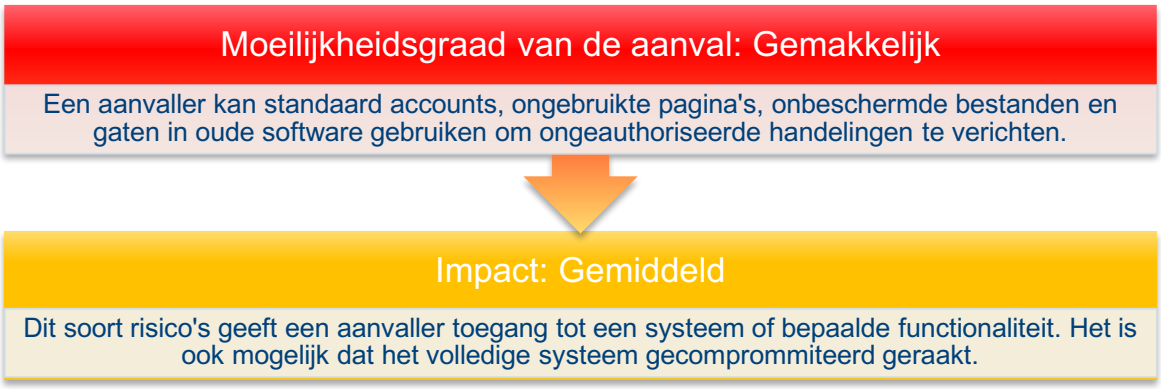
De aanvaller kan de link gebruiken met een andere naam als account.

```
1.  https://example.com/accountinfo.php?account=bert
```

Door deze aanpassing is de informatie van gebruiker Bert zichtbaar. De applicatie valideert niet of de huidige gebruiker toestemming heeft om de informatie van gebruiker Bert te bekijken.

4.2.5 Security Misconfiguration

Goede security vereist een veilige configuratie van alle componenten uit het geheel. Dit houdt in dat zaken als de applicatie zelf, de gebruikte *frameworks*, de applicatieserver, webserver, databaseserver en het platform allemaal veilig moeten zijn. Als er één zwakke schakel is, kan het volledige systeem kwetsbaar worden.
Veiligheidsmaatregelen moeten niet alleen worden geïmplementeerd, maar ook onderhouden. Als de software niet geüpdatet is, kan dit uitgroeien tot een groot beveiligingsrisico. Daarnaast zijn de standaardinstellingen van bijvoorbeeld webservers heel onveilig [16].



Voorbeeld van een aanvalsscenario

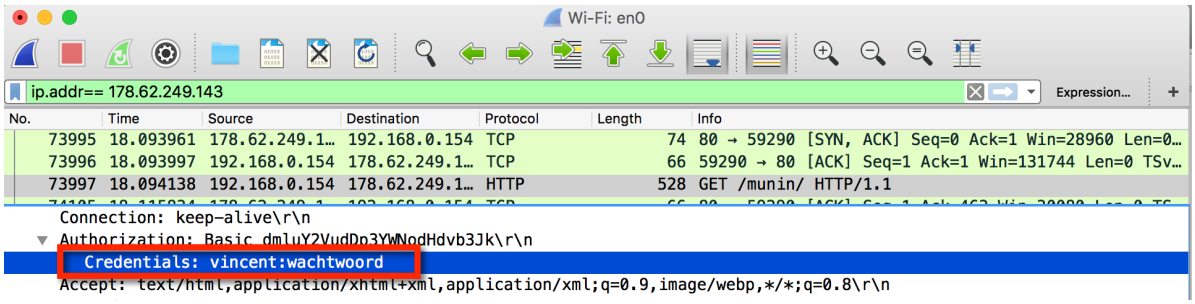
Het onderhoudspaneel is automatisch geïnstalleerd via een pakket. Het standaardaccount "admin" met wachtwoord "admin" is niet uitgeschakeld. Een aanvaller kan via Google deze standaardlogingegevens vinden en zo toegang krijgen tot het onderhoudspaneel.

4.2.6 Sensitive Data Exposure

Veel webapplicaties bieden niet voldoende bescherming aan gevoelige data zoals creditcards, wachtwoorden en identiteitskaarten. Aanvallers kunnen deze zaken stelen en aanpassen om zo creditcardfraude te plegen en zich voor te doen als iemand anders. Gevoelige data moet altijd extra beschermd zijn met recente encryptie, zowel bij de opslag als bij het versturen naar de browser van een gebruiker.

Voorbeeld van een aanvalsscenario

Een loginformulier staat op een website die niet communiceert over HTTPS. Aanvallers kunnen het netwerkverkeer afluisteren en zo wachtwoorden in platte tekst aflezen.



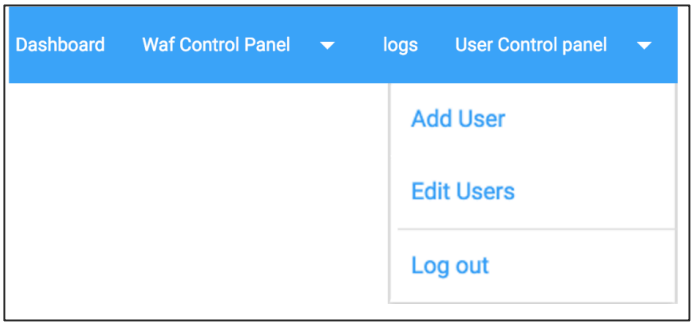
Figuur 4.2: *Screenshot van Wireshark, een bekende tool voor het inspecteren van netwerkverkeer.*

4.2.7 Missing Function Level Access Control

De meeste webapplicaties verbergen bepaalde beheerdersfunctionaliteiten bij gewone gebruikers. Bijvoorbeeld: een lid van een vereniging mag geen administratieve taken verrichten zoals andere leden verwijderen of toevoegen. Als een aanvaller weet welke pagina's deze functionaliteit bevat en deze pagina niet nakijkt of een gebruiker hiertoe toegang heeft, kunnen er gevaarlijke gevolgen zijn.

Voorbeeld van een aanvalsscenario

In het menu van de implementatie staat bij gebruikers die beheerder zijn een link om bepaalde gebruikers te bewerken.



Figuur 4.3: *Screenshot van het menu in de implementatie.*

De knop "Add User" verwijst bijvoorbeeld naar "functions/add_user.php". Deze pagina mag normaal gezien enkel maar toegankelijk zijn voor beheerders. Als een gewone gebruiker toch deze pagina kan gebruiken, is dit een risico. Er moet altijd gecontroleerd worden of een handeling is toegelaten. Het is niet voldoende om enkel ingelogde gebruikers toegang te geven tot bepaalde zaken. De applicatie moet ook nakijken of het type gebruiker hier toestemming voor heeft.

4.2.8 Cross-Site Request Forgery (CSRF)

Een CSRF-aanval gaat een ingelogde gebruiker zonder toestemming bepaalde handelingen laten verrichten. Dit gebeurt bijvoorbeeld door een link te sturen naar het slachtoffer die bij het openen een nieuwe administratorgebruiker aanmaakt. Het administratoraccount kan dan misbruikt worden om de werking van de applicatie te verstoren en data te stelen.

Voorbeeld van een aanvalsscenario

Een gebruiker is ingelogd op een webapplicatie en kijkt zijn e-mails na. Een aanvaller stuurt een link naar een slachtoffer:

```
1.  https://example.com/create_user.php?username=vincent&password=123
```

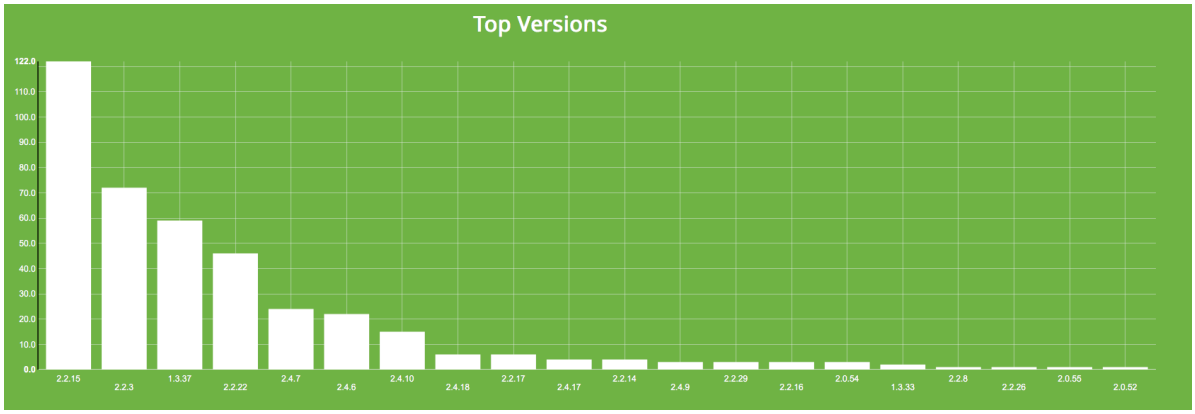
Als het slachtoffer daarop klikt, wordt een gebruiker aangemaakt met username "vincent" en wachtwoord "123". Dit account kan dan misbruikt worden.

4.2.9 Using Components with Known Vulnerabilities

Als een webapplicatie gebruik maakt van componenten zoals *libraries*, *frameworks* en andere softwaremodules is de kans groot dat hier kwetsbaarheden in gevonden worden na verloop van tijd. Kwetsbaarheden in deze componenten kunnen dan de volledige applicatie of server ondermijnen.

Voorbeeld van een aanvalsscenario

Een aanvaller kijkt welk type webserver de organisatie KU Leuven gebruikt.



Figuur 4.4: *Via shodan.io kan er gezocht worden per organisatie. Deze grafiek toont hoeveel elke Apache versie wordt gebruikt door de organisatie KU Leuven.*

De grafiek geeft aan dat de meest gebruikte versie Apache 2.2.15 is. Online kan deze versie eenvoudig opgezocht worden en de aanvaller kan dan zien dat er 24 verschillende **mogelijke** beveiligingsrisico's zijn.

#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score
1	CVE-2011-3192	399	1	DoS	2011-08-29	2013-11-15	7.8
The byterange filter in the Apache HTTP Server 1.3.x, 2.0.x through 2.0.64, and 2.2.x through 2.2.19 allows remo expresses multiple overlapping ranges, as exploited in the wild in August 2011, a different vulnerability than CVE-2							
2	CVE-2013-2249				2013-07-23	2013-08-30	7.5
mod_session_dbd.c in the mod_session_dbd module in the Apache HTTP Server before 2.4.5 proceeds with save o which has unspecified impact and remote attack vectors.							
3	CVE-2012-0883	264		+Priv	2012-04-18	2013-09-17	6.9
envvars (aka envvars-std) in the Apache HTTP Server before 2.4.2 places a zero-length directory name in the LD_ working directory during execution of apachectl.							
4	CVE-2013-1862	310		Exec Code	2013-06-10	2014-03-05	5.1
mod_rewrite.c in the mod_rewrite module in the Apache HTTP Server 2.2.x before 2.2.25 writes data to a log file v commands via an HTTP request containing an escape sequence for a terminal emulator.							

Figuur 4.5*: Top 4 mogelijke kwetsbaarheden in Apache 2.2.15. [23]*

De aanvaller kan zo de werking van de servers verstoren of inbreken in de servers.

4.2.10 Unvalidated Redirects and Forwards

Webapplicaties sturen gebruikers vaak door naar andere pagina's of websites en gebruiken onbekende data om te kijken wat de bestemming moet zijn. Zonder validatie kunnen aanvallers slachtoffers doorsturen naar *phishing* of *malware* websites.

Voorbeeld van een aanvalsscenario

De aanvaller stuurt het slachtoffer een link die doorverwijst naar een nagemaakte site.

```
1.  http://www.bank.com/redirect.php?url=namaak-bank.com
```

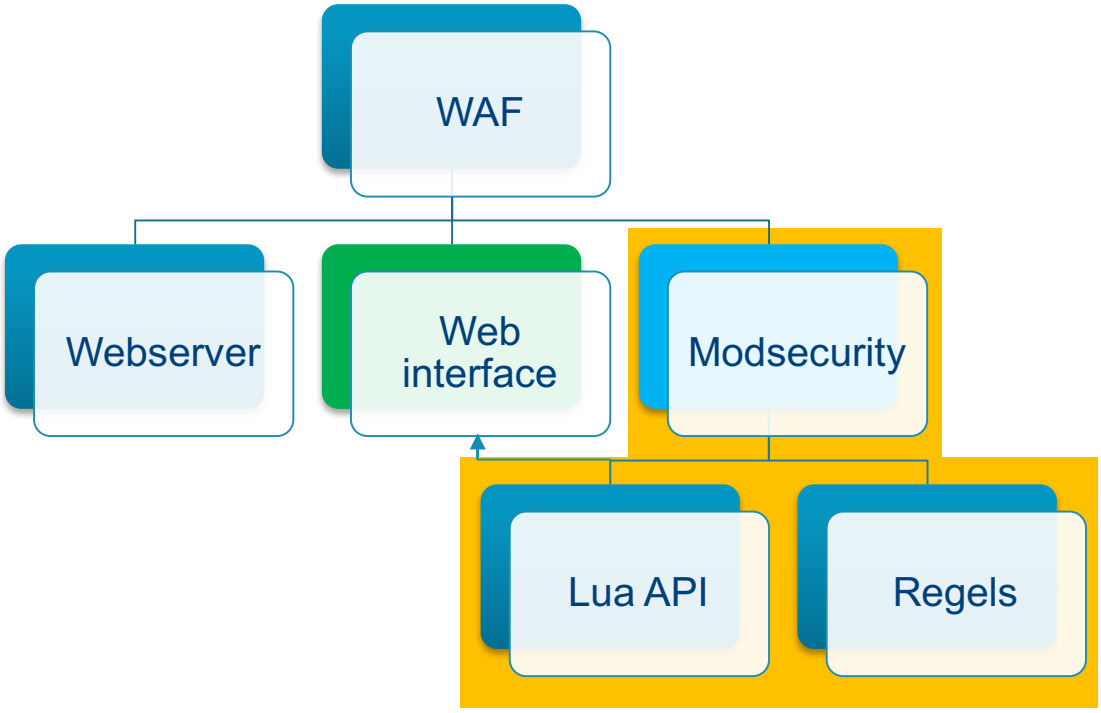
Omdat de link waarop het slachtoffer klikt het domein bevat van "bank.com" denkt de gebruiker dat hij op de echte site zit terwijl hij zijn bankgegevens invult op de namaakwebsite.

5 IMPLEMENTATIE

Dit hoofdstuk bespreekt de uitvoering van de masterproef. Omwille van de complexiteit is dit opgedeeld in aparte delen. Een aantal zaken zijn ingewikkelder geïmplementeerd dan normaal nodig, dit is bewust gedaan om de veiligheid van het geheel te waarborgen. Omdat deze masterproef vooral focust op het security aspect is het natuurlijk een noodzaak om veiligheid voorop te stellen.

5.1 Structuur van de WAF

De voornaamste onderdelen van de implementatie zijn de webserver, web interface en het zelflerende gedeelte (oranje). Dit zelflerende gedeelte is dan nog eens opgesplitst in de Modsecurity module die de configuratie regels en de Lua API bevat.



Figuur 5.1: *Overzicht van de verschillende componenten waaruit de WAF is opgebouwd.*

De webserver is het gedeelte dat de websites en applicaties van de gebruiker omvat. Dit kan ook enkel als *proxy*[4] dienen als de gebruiker de firewall in een *standalone* opstelling gebruikt. De web interface laat de gebruiker toe om de firewall in te stellen en te configureren. Het is met andere woorden het bedieningspaneel van dit project.

De blokken in het oranje veld werken achter de schermen en hiermee zal de gebruiker niet in direct contact staan. De web interface zorgt er namelijk voor dat achter de schermen bepaalde mechanismen van deze blokken in actie schieten. Deze acties zullen de Modsecurity module configureren en de verzamelde data terug in de web interface tonen.

[4] Een *proxy* is een soort van tussenpersoon die het verkeer doorstuurt naar een andere locatie. Het genereert zelf geen data, maar zorgt voor de routering naar de eindbestemming.

Omdat onze web interface bepaalde handelingen moet verrichten als *root user*[5] wordt er beroep gedaan op een veilige methode om deze handelingen toe te laten. Een webserver draait zelf nooit als *root user* om veiligheidsredenen. Dit probleem is opgelost door een concept uit te werken op basis van een *daemon*. Een *daemon* is een proces dat op de achtergrond draait en bij bepaalde gebeurtenissen in actie treedt om bepaalde taken te verrichten. Dit concept vormt de brug tussen de veilige omgeving en de *root* omgeving waar verkeerde handelingen extreme gevolgen hebben.

5.2 Operating System

De implementatie gebruikt het Ubuntu besturingssysteem. Dit is een Linux distributie gebaseerd op Debian. Het voordeel van Debian is dat vrijwilligers deze distributie onderhouden in plaats van een commerciële organisatie. Dit betekent dat nieuwe updates enkel vrijkomen als het publiek dit klaar acht. Bij commerciële organisaties forceren de aandeelhouders de datum van de updates ten koste van de kwaliteit, met alle gevolgen van dien. Updates zijn hierdoor zeer stabiel, maar het nadeel is wel dat de updates niet de allerlaatste versies van bepaalde software bevatten. Voor een webserver die met het internet is verbonden, is dit niet echt een probleem. Het tegendeel is zelfs waar: bugs door nieuwe software zijn in productie helemaal niet gewenst.

5.3 Web interface

De web interface maakt het mogelijk voor de gebruiker om eenvoudig de Modsecurity regels aan te passen en andere handelingen te verrichten. Zoals eerder vermeld is het een topprioriteit om een zo eenvoudig mogelijke interface te maken. Dit trekt een groot publiek aan omdat het toegankelijker is voor beginners en snel in gebruik is voor ervaren gebruikers.

Voor de webpagina's worden de industriestandaards HTML5, CSS en PHP gebruikt. Om de gebruiksvriendelijkheid te optimaliseren wordt JQuery gebruikt. Deze javascript-taal laat de interface er vloeiend uit zien met animaties, speciale opmaak en het dynamisch laden van data.

De taal van de web interface is bewust in het Engels gemaakt. De doelgroep wordt hiermee veel groter en Engels is in de ICT-wereld meestal de voertaal.

5.3.1 Thema

De web interface gebruikt het bekende CSS thema "Materialize CSS" [24]. Het voordeel van dit framework is dat ontwikkelaars het enkel maar moeten linken aan de pagina's van een applicatie. Door bepaalde klasse namen te gebruiken genereert het framework automatisch een mooie opmaak. Deze klasse namen zijn uitvoerig gedocumenteerd op de website. Zo kunnen ontwikkelaars focussen op de functionaliteit in plaats van veel tijd te verliezen met de opmaak in orde te krijgen.

Nog een voordeel van dit thema is dat het automatisch schaalt naar mobiele platformen. Ontwikkelaars noemen dit "responsive design". Toestellen zoals tablets en

[5] De *root user* is het gebruikersaccount dat alle machtigingen heeft van een machine. Meestal wordt gewerkt met een user account, dat is een account met veel minder machtigingen. Meestal volstaat dit laatste, maar om bepaalde zaken te installeren en te configureren zijn nu eenmaal *root* rechten nodig.

smartphones kunnen de pagina's op hun eigen formaat tonen, zonder in te leveren op leesbaarheid van tekst of bruikbaarheid van de gebruikersinterface.

Figuur 5.2: *Het responsive design zorgt dat de interface op alle formaten optimaal wordt getoond.*

5.3.2 Gebruikersbeheer

In de webapplicatie is het mogelijk om meerdere gebruikers aan te maken. Zo kunnen meerdere personen inloggen op de applicatie en bepaalde zaken configureren. Het gebruikersbeheer is geprogrammeerd met de laatste nieuwe aanbevelingen op het vlak van beveiliging.

De applicatie versleutelt wachtwoorden met de beste functies die de PHP-versie van het systeem toelaat. Er zit een compatibiliteitsbibliotheek in zodat ook oudere PHP-versies van een relatief goede beveiliging kunnen genieten.

Daarnaast is de applicatie ook beschermd tegen *brute-force* aanvallen. Als een aanvaller verschillende keren een verkeerde login gebruikt, treedt er een vertraging op. Als dit aantal blijft toenemen, blokkeert de applicatie het IP-adres dat de foutieve login's heeft veroorzaakt.

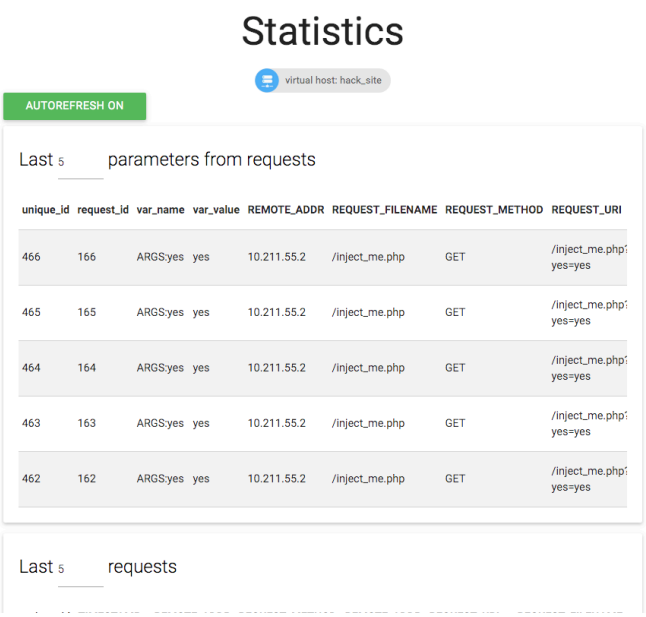
5.3.3 Firewall modus

De firewall kan in drie configuraties worden ingesteld:

- Uit: geen logging en ook geen bescherming van de firewall
- *monitor mode*: geen bescherming van de firewall, maar de applicatie verzamelt wel informatie voor het zelflerende gedeelte. Dit is met andere woorden de leermodus. Tijdens deze modus kan de gebruiker ook statistieken bekijken van de aanvallen die gedetecteerd zijn met de ingestelde firewall regels. Merk op dat de firewall alle *requests* door laat, dus ook de aanvallen.
- Aan: dit beschermt de firewall én verzamelt informatie voor het zelflerende gedeelte.

5.3.4 Realtime overzicht

In de webapplicatie is er een realtime overzicht van de binnenkomende parameters en *requests*. De gebruiker kan allerhande info terugvinden zoals het IP-adres vanwaar een *request* gestuurd is, de methode, de opgevraagde URL, de parameters enzovoort. Dit kan handig zijn om bepaalde zaken te testen.

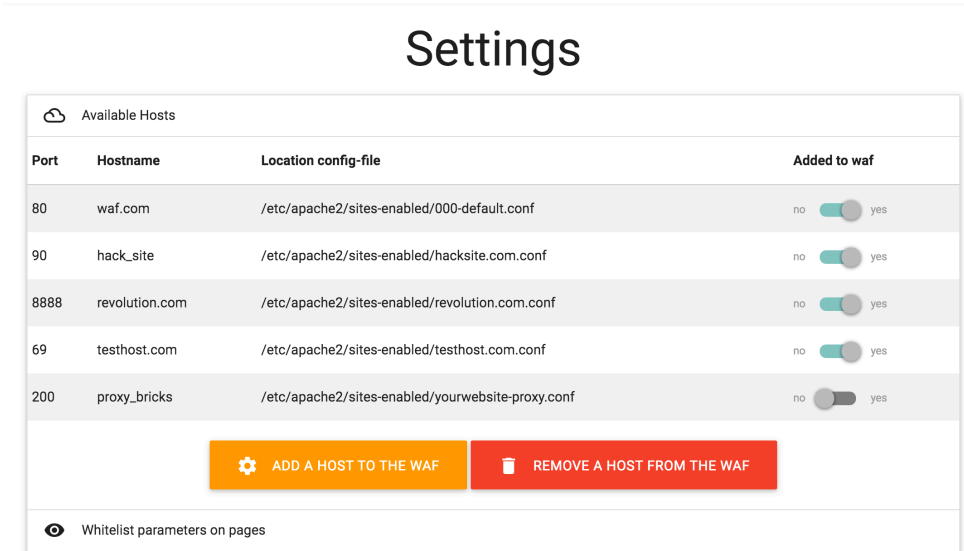


***Figuur 5.3**: Realtime weergave van de binnenkomende requests.*

De gebruiker kan ook kiezen om meer *requests* te tonen door het veldje aan te passen.

5.3.5 Instellingen

Op de pagina voor de instellingen kunnen nieuwe websites en applicaties toegevoegd worden. Dit kan automatisch gebeuren, maar ook manueel. Het automatische proces is natuurlijk het snelste en zal in de meeste gevallen ook volstaan. Als de gebruiker werkt met zeer ingewikkelde configuratiebestanden is de manuele optie beter. De stappen worden volledig uitgelegd door een zogenaamde installatiewizard.



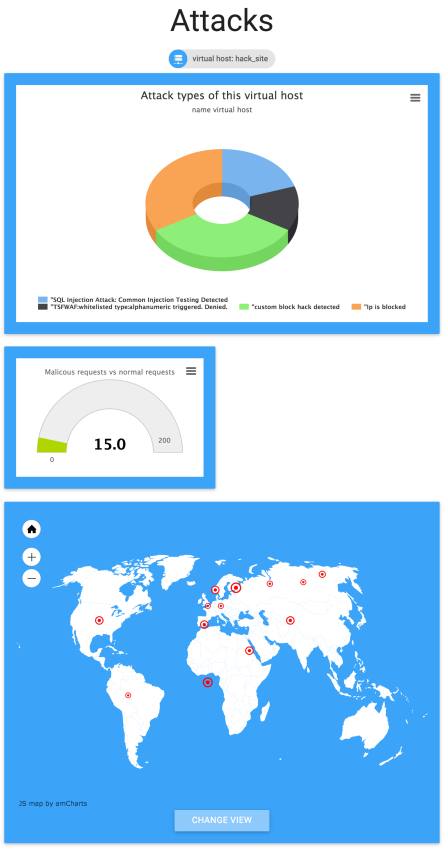
***Figuur 5.4:** Pagina van de instellingen.*

Bij de instellingen is er ook de mogelijkheid om bepaalde parameters te blacklisten op bepaalde pagina's. Dit is vooral wenselijk bij login formulieren. De data die onze scripts verzamelen worden niet in een formaat opgeslagen dat wachtwoorden beschermt. De gebruiker kan hier een parameter op een pagina blacklisten, bijvoorbeeld "wachtwoord" op de pagina "login.php".

Als laatste kan de gebruiker ook de verschillende firewall modi instellen voor elke website. Het is aangeraden om bij kritieke websites of applicaties eerst de monitor modus te gebruiken.

5.3.6 Aanvalsoverzicht

In de implementatie kan de gebruiker ook een rapport terugvinden over de aanvallen. Zaken als het type van de meest voorkomende aanvallen of locaties van de aanvallers zijn grafisch weergegeven. Onderstaande figuur toont een screenshot van dit overzicht.



Figuur 5.5: *Een deel van het aanvalsoverzicht in de web interface.*

Dit geeft de gebruiker inzicht over welk type verkeer er op de websites binnenkomt.

5.3.7 Opstellen van firewall regels

In de applicatie is er ook de mogelijkheid om eenvoudige firewallregels op te stellen. Een IP of een pagina kan worden geblokkeerd, maar kan ook gewhitelist worden. Een IP of pagina dat op deze *whitelist*[6] staat, wordt volledig vrijgesteld van alle Modsecurity regels.

[6] Een whitelist is een lijst met toegelaten acties, identiteiten of andere kenmerken.

De combinatie van een IP op een pagina blokkeren of *whitelisten* gaat ook. Hierdoor kan men pagina's die in productie zijn of die kwetsbaar zijn, afschermen voor buitenstaanders.

5.3.8 Veiligheidsmaatregelen

Vanaf dag één is er met security in het achterhoofd geprogrammeerd aan de WAF. Het heeft weinig zin om een WAF te maken die zelf een beveiligingsrisico vormt. Daarom zijn een aantal methodes gebruikt om de veiligheid te garanderen.

5.3.8.1 *Root* machtigingen

Het is altijd wenselijk om een webserver met zijn webapplicaties niet als *root* te laten opereren. Hierdoor blijft de schade beperkt wanneer één van de web applicaties is gehackt. Als de webapplicatie toch als *root* draait, dan heeft de aanvaller mogelijk ook controle over de andere webapplicaties en services die op deze machine draaien. In het hoofdstuk "*root daemon*" komt een mogelijkheid aan bod om met de web interface toch bepaalde *root*handelingen uit te voeren.

5.3.8.2 PDO

De code van de web interface maakt gebruik van PHP Data Objects (PDO). Het voordeel hiervan is dat dit beschermt tegen SQL-injectie. MySQL functies daarentegen vereisen *escaping* en andere methodes om de input vrij te maken van SQL-injectie. Het is zelfs dan nog niet waterdicht en daarenboven zijn deze commando's verouderd (werken niet meer in PHP7 en later) [25].

5.3.8.3 Gebruikers-input validatie

De input die de gebruiker in zijn browser kan aanpassen en eindigt bij bepaalde *root*-acties wordt op drie plaatsen gevalideerd: bij de *client* zelf, op de webserver en op *root*-niveau. Ontwikkelaars mogen namelijk nooit vertrouwen op de input van de gebruiker omdat bepaalde gevormde input dan ongewenste effecten heeft, zeker als dit op *root*-niveau wordt uitgevoerd. Zelfs de data die door de web interface van de WAF wordt gestuurd naar *root*-niveau wordt gevalideerd. Dit moet verzekeren dat in het geval dat de web interface van de WAF gecompromitteerd is, er toch geen kwaadaardige commando's op *root*-niveau uitgevoerd kunnen worden.

5.3.8.4 Weergave van data

Omdat de web interface de aanvalsparameters toont bij de statistieken, is dit een direct gevaar voor XSS-aanvallen. Dit is opgelost door overal de "*htmlspecialchars()"* in PHP te gebruiken bij het tonen van data.

```
1.  $output = '<script>alert("xss")</script>';
2.  echo $output; // XSS
3.  echo htmlspecialchars($output);  // Geen XSS
```

5.4 Configuratie webserver

Deze masterproef maakt gebruik van een Apache webserver. Dit type server is momenteel het meest gebruikt. Nochtans zou de WAF met minimale aanpassingen ook kunnen werken op NGINX, dit is een webserver die in populariteit sterk aan het toenemen is en veel sneller is als Apache.

5.4.1 Headers

De webserver moet ook een aantal *headers* meegeven om het risico op bepaalde aanvallen aan de gebruikerskant te verkleinen. Dit is gerealiseerd door de volgende code te gebruiken op elke pagina in de PHP-code.

```
1.  header( 'X-Frame-Options: SAMEORIGIN' );
2.  header("X-XSS-Protection: 1; mode=block");
3.  header("X-Content-Type-Options: nosniff");
4.  header_remove('x-powered-by');
5.  header_remove('Server');
6.  ini_set('ServerTokens', 'ProductOnly');
7.  ini_set('ServerSignature', 'Off');
8.  ini_set( 'session.cookie_httponly', 1 );
```

Regel één tot en met drie beschermen tegen XSS-aanvallen in de browser van de gebruiker. Regel vier tot en met zeven verwijderen de versienummers van Apache en PHP zodat aanvallers niet direct weten welke (al dan niet kwetsbare) versie de webserver gebruikt. De laatste regel beschermt de cookies (en dus ook de waardevolle sessie-ID[7]) tegen scripts die deze info willen stelen.

5.4.2 PHP-configuratie

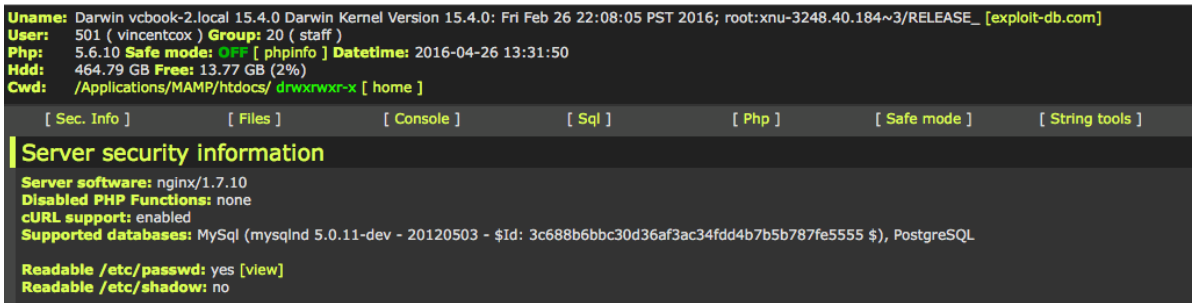
Ontwikkelaars kunnen de PHP-instellingen aanpassen om de beveiliging van de volledige server te verhogen. Deze instellingen staan in het php.ini bestand in de configuratie map van de server.

Door functies uit te schakelen die niet zijn gebruikt in de applicaties neemt de veiligheid enorm toe.

```
1.  disable_functions =
    posix_mkfifo,posix_getlogin,posix_ttyname,getenv,get_current_user,proc_get_status,
    get_cfg_var,disk_free_space,disk_total_space,diskfreespace,getcwd,getlastmo,getmyg
    id,getmyinode,getmypid,getmyuid,proc_open,shell_exec,popen,system,apache_child_ter
    minate, apache_setenv, define_syslog_variables, escapeshellarg, escapeshellcmd,
    eval, exec, fp, fput, ftp_connect, ftp_exec, ftp_get, ftp_login, ftp_nb_fput,
    ftp_put, ftp_raw, ftp_rawlist, highlight_file, ini_alter, ini_get_all,
    ini_restore, inject_code, mysql_pconnect, openlog, passthru, php_uname,
    phpAds_remoteInfo, phpAds_XmlRpc, phpAds_xmlrpcDecode, phpAds_xmlrpcEncode, popen,
    posix_getpwuid, posix_kill, posix_mkfifo, posix_setpgid, posix_setsid,
    posix_setuid, posix_setuid, posix_uname, proc_close, proc_get_status, proc_nice,
    proc_open, proc_terminate, shell_exec, syslog, system, xmlrpc_entity_decode
2.  open_basedir = /Applications/MAMP/htdocs
```

[7] Na het succesvol inloggen op een website krijgt een gebruiker vaak een sessie-ID dat opgeslagen wordt in een cookie. Dit is eigenlijk een tijdelijke sleutel die bewijst dat de gebruiker is ingelogd en wat zijn identiteit is.

De volgende twee figuren tonen aan waarom bovenstaande extra regels een wereld van verschil maken. De twee figuren tonen screenshots van een *web shell*[8] die op een website staat.



Figuur 5.6: *Screenshot van een web shell op een onbeschermde host.*

Figuur 5.7: *Screenshot na het toevoegen van de code aan het php.ini bestand.*

De eerste figuur toont duidelijk aan dat er veel informatie via PHP te vinden is. Er zijn ook volledige rechten om bepaalde bestanden uit te lezen, zelfs buiten de map van de website. Dit is helemaal niet gewenst. In de tweede figuur vallen al deze informatie en rechten weg waardoor de omgeving veel veiliger is, zelfs als er een aanval is gelukt.

5.4.3 MySQL-configuratie

Door de databasegebruikers enkel toegang te geven tot bepaalde tabellen kan de veiligheid enorm toenemen. Moest een hacker toegang krijgen tot een gebruikersaccount, dan is de schade beperkt tot alleen de databases waartoe dit account toegang heeft. Als ontwikkelaars de standaardinstellingen gebruiken, heeft dit account toegang tot alle databases wat niet wenselijk is.

Een bijkomende effectieve maatregel is het toelaten van minimale acties door een gebruikersaccount. Handelingen zoals "DROP table" verwijderen een tabel en zijn in de verkeerde handen destructief. Dit kan tot gegevensverlies leiden en is voor de werking van veel implementaties zelfs niet nodig. In deze masterproef krijgt elk databaseaccount de minimale rechten om zo een veilig geheel te krijgen.

---

[8] Een aanvaller gebruikt vaak een *web shell* op een gecompromiteerde server. Hiermee kan de aanvaller gemakkelijk bestanden downloaden, commando's uitvoeren en bestanden aanpassen.

5.5 Root daemon

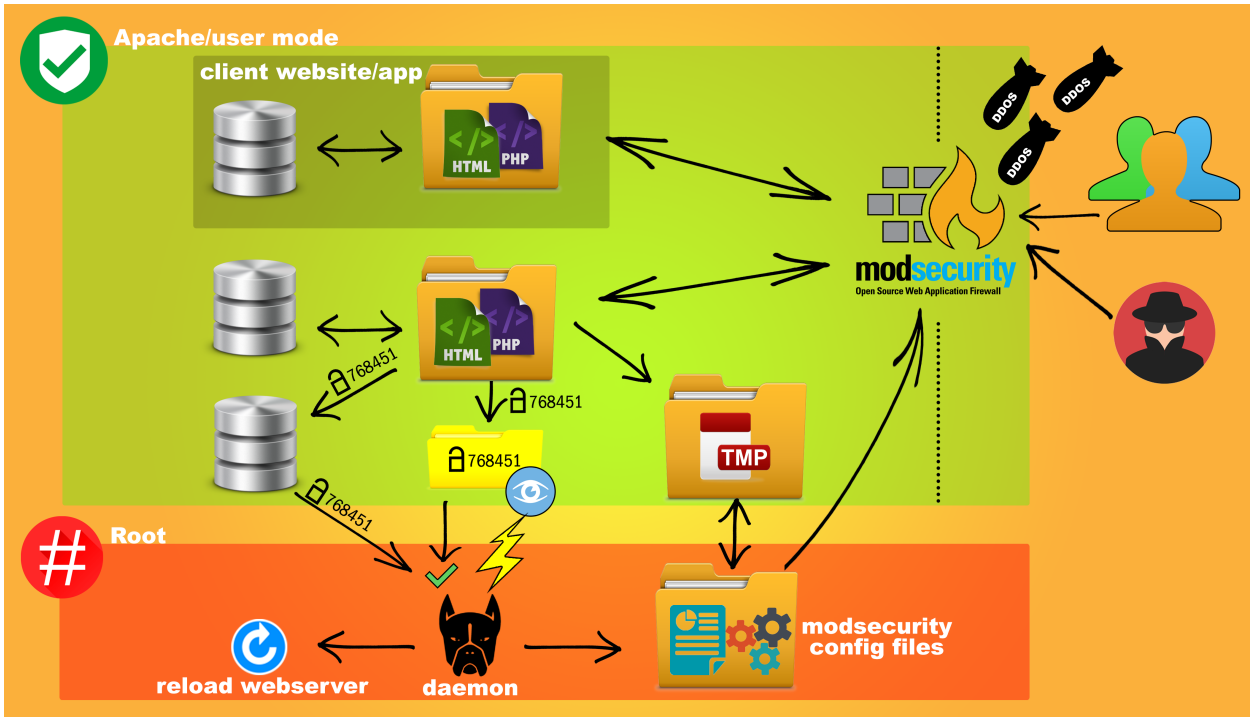
Bij het ontwerpen van de WAF stelden er zich twee fundamentele problemen:

- Permissie: bepaalde handelingen moeten door de webapplicatie als *root* uitgevoerd worden, maar de webapplicatie draait in *usermode*.
- De webapplicatie heeft geen toegang tot de configuratie files van Modsecurity. Deze staan buiten de Apache *root* map.

Deze problemen zijn opgelost door middel van een *daemon*. Een daemon is een proces dat op de achtergrond draait en bij bepaalde gebeurtenissen in werking treedt.

5.5.1 Werking daemon

De volgende figuur toont de werking en structuur van de daemon.



Figuur 5.8: *Werking van de daemon.*

De figuur bestaat uit twee grote vlakken: het *root*gedeelte in het rood en *user mode* in het groen. Een webserver gaat nooit als *root* draaien omwille van veiligheidsredenen. Een lek in een applicatie zou dan *root* toegang geven tot een aanvaller wat niet gewenst is. Nochtans zou de webapplicatie bepaalde *root*handelingen moeten uitvoeren. Dit is opgelost met een *daemon* die bepaalde *watchfolders* (geel) in de gaten houdt.

Stel dat de webapplicatie een *root* handeling wil uitvoeren, dan gaat deze een *token* genereren en dit in zowel de database als de *watchfolder* steken. Omdat er een nieuwe file in de *watchfolder* komt, gaat de daemon een signaal krijgen en het *token* van in de database en *watchfolder* vergelijken. Komen deze *tokens* overeen dan gaat de daemon een bepaalde *root*handeling uitvoeren. Omdat de *daemon* als *root* draait, heeft deze toestemming hiervoor.

Er zijn verschillende *root* handelingen gedefinieerd:

- Het herladen van de webserver. Hierdoor worden de regels en instellingen in de configuratiebestanden actief.
- Het kopiëren van de Modsecurity regels naar een tijdelijke map waardoor onze webapplicatie deze regels kan aanpassen.
- Het terug kopiëren van de tijdelijke map naar de Modsecurity configuratie map.
- Het activeren/deactiveren van Modsecurity regels.
- Toevoegen van een *virtual host* aan de firewall.

5.5.2 Daemon configuratiebestand

Alle variabelen zoals de databasenamen, tabelnamen en bestandlocaties zijn opgeslagen in een apart "daemon.cfg" bestand. Als de gebruiker een andere databasenaam wil gebruiken kan hij of zij dit eenvoudig aanpassen op één plaats. De gebruiker moet dan niet overal in het script waarden gaan aanpassen.

5.5.3 Veiligheidsoverwegingen bij de daemon

Omdat de daemon als *root* draait, moeten er een aantal zaken met extra zorg behandeld worden om de veiligheid van de server te garanderen. Een fout in de daemon kan de volledige server compromitteren.

5.5.3.1 Input validatie

De *daemon* moet altijd input van buitenaf valideren. De beste manier is het gebruik van een *case*-functie zodat de *daemon* enkel de toegelaten operaties uitvoert. De volgende code laat zo een *case*-functie zien in Bash.

```
1.  case "$input_buitenwereld" in
2.  "1")
3.      do_this()
4.      ;;
5.  "2" | "3")
6.      do_what_you_are_supposed_to_do()
7.      ;;
8.  *)
9.      do_nothing()
10.     ;;
11. esac
```

Input die toch rechtstreeks in een commando terecht komt, kan men best valideren met behulp van een "regular expression" of in het kort "regex" genoemd. Dit kan door het definiëren van een zoekpatroon en te gaan kijken of de input aan dit patroon voldoet. Enkele voorbeelden zijn geldige e-mails, creditcardnotatie en telefoonnummers.

5.5.3.2 Bash quotes

Ontwikkelaars moeten altijd gebruik maken van aanhalingstekens bij variabelen in de Bash taal. Als dit niet gebeurt, kan dit een zeer ernstig beveiligingsrisico zijn waarbij aanvallers willekeurige code kunnen uitvoeren van buitenaf. De volgende code geeft een voorbeeld van dit beveiligingsrisico.

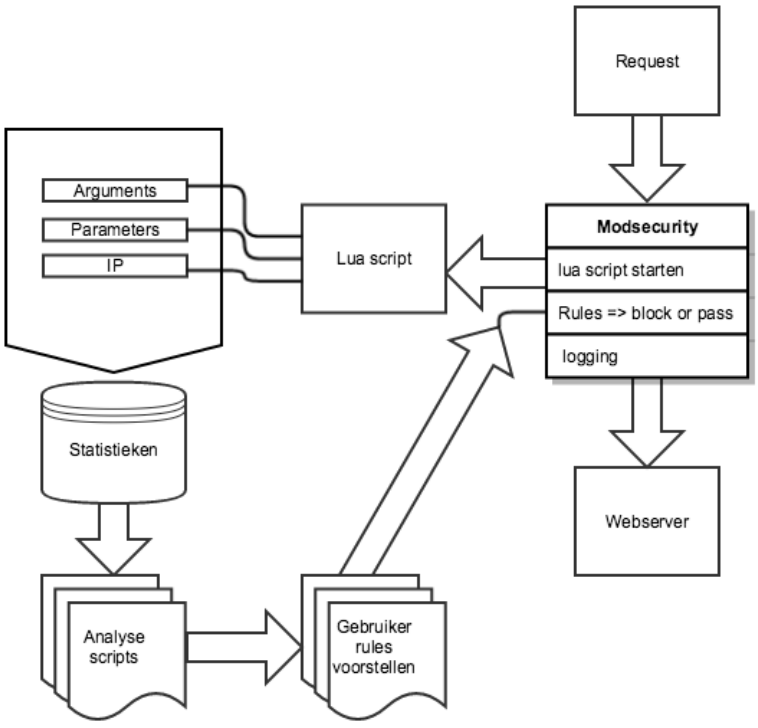
```
1.  root@ubuntu:/home/vincentcox# echo /home/*
2.  /home/vincentcox
3.  root@ubuntu:/home/vincentcox# echo "/home/*"
4.  /home/*
```

De eerste twee regels tonen de input en output aan zonder de aanhalingstekens. De gebruikersnaam van het systeem komt in de output te staan. Dit is niet gewenst omdat dit interessante informatie geeft aan hackers. Regel drie en vier tonen een veilige manier aan waar de output correct is weergegeven.

In de *daemon* is dus gebruik gemaakt van aanhalingstekens om gevaarlijke input tegen te houden.

5.6 Zelflerende gedeelte

De volgende figuur toont het verloop van een *request* door Modsecurity naar de webserver.



Figuur 5.9: *Schema van het verwerken van een request bij de Modsecurity firewall.*

Bovenstaande figuur toont rechtsboven een *request* dat binnenkomt bij Modsecurity. Een *request* (of in het Nederlands een verzoek genoemd) is een aanvraag om een bepaalde pagina te zien, of om data in te dienen en het resultaat ervan te bekijken. Modsecurity gaat aan de hand van de firewall regels bekijken of dit *request* mag aankomen bij de webserver. Als dit verzoek niet voldoet aan de regels blokkeert

Modsecurity. Het verzoek eindigt met andere woorden in de virtuele vuilbak. Als tweede gaat de implementatie via Modsecurity elke parameter van een *request* opslaan in een database. Dit is praktisch gerealiseerd door gebruik te maken van de Lua API omdat dit eenvoudig per *request* bepaalde argumenten en parameters kan uit lezen. De implementatie gebruikt hierna de data om statistieken aan te maken en het voorstellen van nieuwe regels.

5.6.1 Opslag van data

De opslag van de data gebeurt via een MySQL-server. Deze server is momenteel een van de meest populaire databaseservers. Het voordeel dat deze databaseserver zo populair is, maakt dat ontwikkelaars in PHP (web interface), Bash (daemon), python (analysescripts) en Lua (Modsecurity data verzameling) eenvoudig kunnen verbinden met de MySQL database. De API is ofwel standaard ingebouwd of kan eenvoudig via een *library* worden toegevoegd.

De snelheid van een MySQL database is zelfs snel genoeg om het geheel vlot te laten functioneren bij grote hoeveelheden data. Bovendien maken de MySQL *query's* het eenvoudig om snel en efficiënt de juiste data te kunnen opvragen.

5.6.2 Lua logging module

Deze module zorgt voor de opslag van alle parameters bij een *request* in de MySQL database. Omdat er ook parameters zullen zijn die kwaadaardige code bevatten (zoals SQL-injectie) is er enige zorg vereist om deze data correct op te slaan. In PHP zijn er altijd "prepared statements" gebruikt. Dit is een methode om parameters veilig in de database op te slaan.

Omdat er geen "prepared statements" ondersteuning bestaat voor MySQL in de Lua-taal, is het zeer moeilijk om nog SQL-injectie tegen te gaan. Er bestaat wel een zogenaamde "strip slashes" functie, maar deze gaf in bepaalde gevallen nog altijd foutmeldingen wat niet wenselijk is. De applicatie moet natuurlijk ook parameters die SQL-injectie bevatten, loggen.

Uiteindelijk is dit opgelost door variabelen die injecteerbaar zijn te coderen met base64.

```
1.  Gewone tekst:                                    SQL injectie ')–
2.  Base64 encoding toegepast:                   U1FMIGluamVjdGllICcpLS0=
```

Base64 zet een string met spaties en speciale tekens altijd om in een alfanumerieke tekst (met soms een "=" op het einde). Hierdoor wordt een SQL-injectie omgezet naar een veilige vorm die de applicatie in de database kan opslaan.

Zelfs als de aanvaller de kennis heeft dat de implementatie gebruik maat van base64, is het niet mogelijk om hierop te anticiperen. Base64 decoding op klare tekst is niet mogelijk. Ook al forceert de aanvaller dit, dan bekomt hij of zij enkel speciale unicode karakters. Een webserver verstaat deze tekens toch niet en verwerkt het request bijgevolg niet.

De databasegebruiker die het Lua script gebruikt, heeft enkel toegang tot de *requests* database. Ook al gebeurt er SQL-injectie, dan zijn de andere databases niet in gevaar. De machtigingen zijn ook minimaal voor deze gebruiker waardoor destructieve acties als "DROP TABLE" niet zijn toegelaten.

5.6.3 Analyse scripts

Voor de analyse scripts is de voorkeur naar Python gegaan. Dit is een veelzijdige taal met dynamische *typing*, veel *packages* en *extensies* waardoor er veel resultaat komt uit relatief weinig code.

De hoofdtaak van het python script is om de verschillende soorten parameters van een *request* te gaan classificeren in een aantal types:

- Alfanumeriek
- Getallen (*Digits*)
- E-mailadressen
- Symbolen
- …

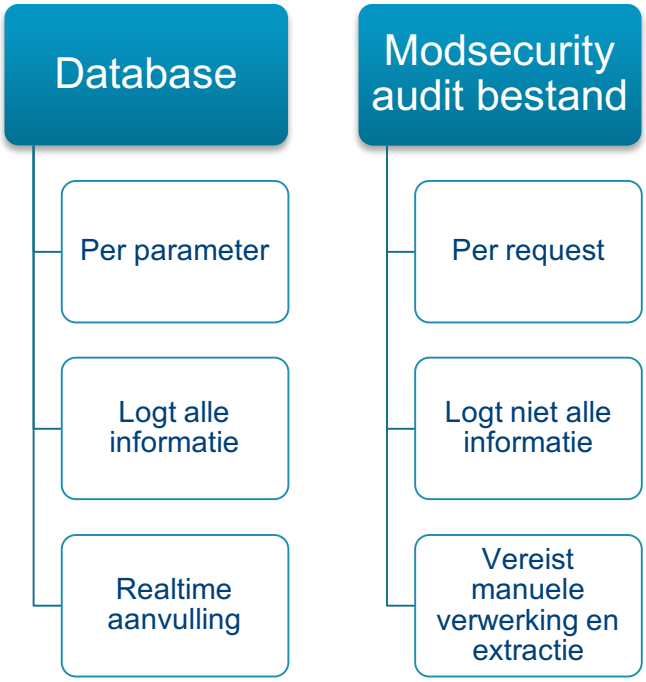
Het script gaat ook tellen hoe vaak een bepaald type bij een parameter op een pagina hoort. Het type dat het meeste voorkomt stuurt het script door naar de web interface. De volgende code toont een stukje van de code dat het script doorstuurt naar de web interface.

```
1.  [ ['id', 'alphanumeric', '/index.php', 40], ['id', 'symbols', '/index.php', 1]]
```

In dit geval detecteert het script veertig keer het type als alfanumeriek bij parameter "id", en één keer als type symbolen. Aan de hand van deze informatie gaat de web interface dan aan de gebruiker voorstellen om een Modsecurity regel te genereren die enkel deze parameter als alfanumeriek doorlaat. De kans is namelijk groot dat het type symbool afkomstig is van een kwaadaardige input. Natuurlijk moet de gebruiker deze regel goedkeuren om problemen met de webapplicaties van de gebruiker te vermijden.

5.6.4 Weergave van suggesties

Er zijn in totaal twee bronnen van data die de applicatie gebruikt voor zijn bewerkingen. De eerste en voornaamste is de database die alle gegevens van een *request* bevat. Als tweede is er het logboek van Modsecurity dat elke overtreding gaat bijhouden. Afhankelijk van het ingestelde log-niveau kan deze ook alle normale *requests* bijhouden.



Figuur 5.10: *Overzicht van de beschikbare informatie.*

Om de data eenvoudig en visueel voor te stellen is er veel gebruik gemaakt van bepaalde JQuery *libraries*. Dit zijn een soort van "plugins" die bepaalde taken kunnen uitvoeren op de webpagina's.

Zo zijn er twee methodes gerealiseerd om de gebruiker te laten omgaan met de voorgestelde data: een "TagCloud" en "Tree Filtering".

5.6.4.1 TagCloud

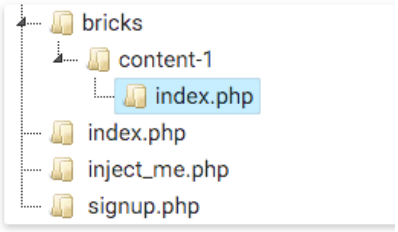
Dit concept is vooral interessant als de gebruiker de beste aanbevelingen eerst wil aanpakken. De grootte van een woord toont aan dat dit type zeer vaak is gevonden voor een bepaalde parameter.

Figuur 5.11: *Screenshot van de zogenaamde "TagCloud".*

Als de gebruiker dit woord heeft aangeklikt, komt dit in een lijst te staan waar het type nog aanpasbaar is indien het programma dit verkeerd geclassificeerd heeft. Daarna kan de gebruiker op "generate rules" klikken waardoor de applicatie Modsecurity regels aanmaakt. De gebruiker kan deze regels nog eventueel aanpassen en toevoegen aan de andere firewall regels.

5.6.4.2 Tree filtering

Bij *Tree filtering* zijn alle parameters zichtbaar en kan de gebruiker op basis van een pagina filteren. Dit is interessant voor grote websites waarbij de grote hoeveelheden data soms overweldigend kunnen zijn.



Figuur 5.12: *Screenshot van de zogenaamde "TreeView".*

De gebruiker kan door op een pagina of map te klikken in de *Tree-view* de parameters filteren. Ook hier kan de gebruiker regels laten genereren.

5.6.4.3 Eigen classificatietypes

De gebruiker kan zelf ook nieuwe types definiëren in het classificatieproces. In de web interface kan de gebruiker een nieuwe *regex* toevoegen via enkele muisklikken.

5.6.5 Blacklisting van gevoelige parameters

Bepaalde webapplicaties maken gebruik van een login formulier. Dit type formulieren versturen meestal ook wachtwoorden naar de webserver. Het is niet wenselijk om deze gegevens in de database van deze implementatie op te slaan. Deze data wordt achteraf verwerkt en moet dus in platte tekst beschikbaar zijn. Wachtwoorden daarentegen moeten gehasht en gesalt zijn zodat een hacker nooit over wachtwoorden in platte tekst beschikt bij een lek in een database.

In de web interface bestaat de mogelijkheid om parameters van bepaalde pagina's te *blacklisten* zodat de applicatie deze overslaat bij het loggen.

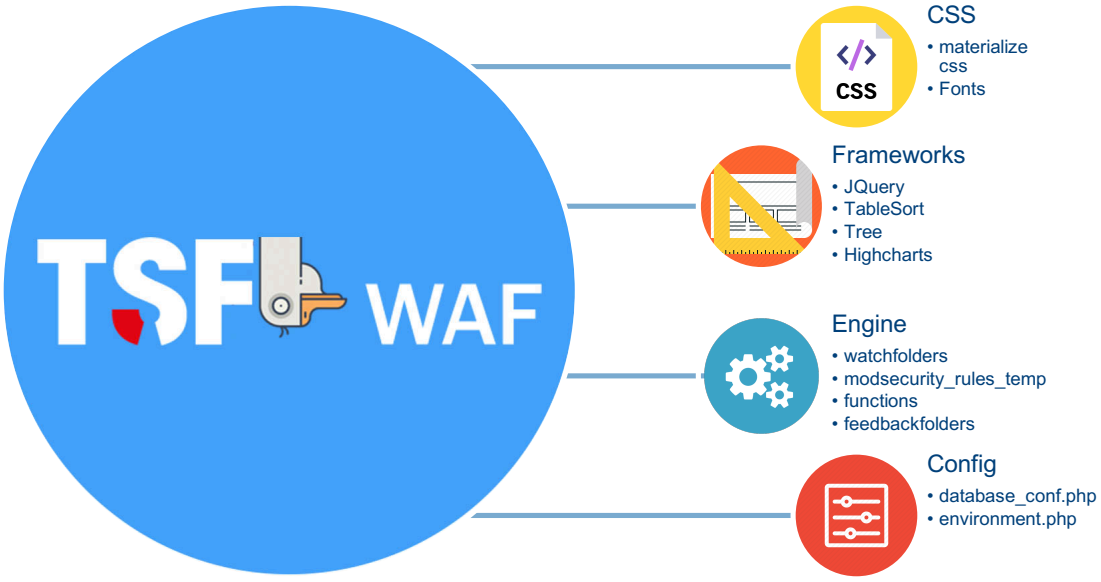
5.6.6 Statistieken van aanvallen

De firewall logt alle *requests* die zijn tegengehouden. De applicatie verzamelt alle relevante data in dit logboek en toont dit aan de gebruiker in de vorm van grafieken en tabellen.

Zaken zoals het IP-adres, tijdstip, pagina en parameter zijn hiervoor het belangrijkste. Een interessante functie van de applicatie is het opzoeken van het IP-adres in een GEO-locatie database. De locatie komt dan op een wereldmap te staan waardoor de gebruiker een mooi overzicht krijgt van de aanvallen.

5.7 Bestandsindeling

De applicatie is gestructureerd opgebouwd zodat toekomstige aanpassingen snel kunnen plaatsvinden. Het aanpassen van bijvoorbeeld het menu gebeurt op één plaats in plaats van op elke pagina. De configuratie van de PHP-database gebeurt ook op één plaats zodat de gebruiker niet alle functies opnieuw moet aanpassen. De volgende figuur toont een kort overzicht van de bestandsindeling.



Figuur 5.13: *Overzicht van de bestandsindeling.*

De map CSS bevat alle CSS-code en lettertypes voor de opmaak. De "Frameworks" map bevat de JQuery *plugin* en andere *plugins* die bepaalde functies op zich nemen zoals het ophalen van data, dynamische opmaak en visuele animaties. De "Engine" map bevat de verzameling van *watchfolders*, functies en *feedbackfolders*. Zoals de naam al doet vermoeden is dit de motor van de WAF. Dan is er nog een map met alle configuratie instellingen voor de database aan te sturen.

6 TESTEN VAN DE WAF

Een belangrijk deel van deze masterproef is het testen van de applicatie. De focus ligt hier vooral op het security aspect: een WAF interface die het geheel onveiliger maakt is natuurlijk het tegenovergestelde van het doel. Daarom is het belangrijk dat er verschillende pentesten op de webapplicatie zijn uitgevoerd waardoor zwakke schakels gelokaliseerd en opgelost kunnen worden.

6.1 Pentesting tools

Online zijn een groot aantal pentesttools beschikbaar. Een groot deel zijn betalend, maar er zijn ook een aantal gratis te gebruiken tools. Bij deze masterproef zijn vooral "Burp Suite" en "OWASP ZAP" gebruikt bij het testen.

6.1.1 Burp Suite

Dit is een uitgebreide tool die vooral bedoeld is om manueel te gaan testen. Met veel precisie kunnen onderzoekers de meest verborgen beveiligingsrisico's vinden. Veel gebruikers noemen het dan ook vaak "the swiss army knife for security testing" [26]. Het nadeel is wel dat bepaalde functies heel uitgebreid zijn en dit kan verwarrend zijn voor beginnende gebruikers.

6.1.2 OWASP ZAP

OWASP Zed Attack Proxy (ZAP) is een gemakkelijk te gebruiken programma dat het mogelijk maakt om beveiligingsrisico's te vinden in webapplicaties en websites. Het is meer een "click and point" omgeving waardoor dit niet altijd geschikt is voor gevoelige plaatsen in webapplicaties of systemen in productie. Voor beginners is dit een interessante tool omdat dit relatief weinig kennis vereist en de interface zichzelf uitwijst.

6.2 Testapplicaties

Om wat trafiek te genereren en zo data te kunnen verzamelen bij de WAF zijn er enkele websites en applicaties opgezet. In totaal zijn er twee verschillende scenario's die data moeten verzamelen: normale sites met het grootste deel gewone bezoekers en een aantal kwetsbare applicaties die worden aangevallen.

6.2.1 Normale applicaties

De normale opstelling maakt gebruik van een Wordpress installatie met standaard demo inhoud en plug-ins. Wordpress vertegenwoordigt 26,3% [27] van alle websites op het internet, omgerekend is dit 59,2% marktaandeel in *content management systems*. Dit is bijgevolg een realistische testomgeving.

Daarnaast zijn er ook verschillende eenvoudige formulieren die het mogelijk maken om bepaalde parameters beter te kunnen testen.

Statische websites ontvangen meestal geen parameters en het heeft bijgevolg geen zin om statische websites te testen.

6.2.2 Kwetsbare applicaties

Het simuleren van beveiligingsrisico's gaat met behulp van applicaties die bewust slecht zijn opgebouwd.

Een bekend voorbeeld is "Damn Vulnerable Web Application" (DVWA). Dit is een PHP/MySQL webapplicatie bedoelt voor testen uit te voeren door security onderzoekers en web ontwikkelaars. Het hoofddoel is ontwikkelaars de gevaren van slecht opgebouwde applicaties te laten begrijpen en aan te leren hoe ze veilige platformen kunnen bouwen.

Een ander voorbeeld is "Bricks" wat ook een PHP/MySQL applicatie is. Dit is ook een applicatie die ontwikkeld is door OWASP.

Het is wel belangrijk dat deze applicaties niet publiek bereikbaar zijn. Deze applicaties zijn redelijk bekend in de securitywereld en dit kan opgemerkt worden door andere personen of *crawlers*[9] die dan hiervan misbruik maken en de server kunnen overnemen.

6.3 Testomgeving

Onlinediensten zoals DigitalOcean en Amazon Web Services (AWS) maken het goedkoop en gemakkelijk om snel een eigen server op te zetten. Voor tien dollar per maand krijgen gebruikers al een server met 1gb ram en 30Gb opslag [28]. Via SSH kan al het configuratiewerk gebeuren en DNS-diensten zitten er ook gratis bij.

Voor deze masterproef is de functie "snapshots"[10] zeer interessant. Een beheerder kan dan een volledig geconfigureerde server als een *diskimage* opslaan. Daarna kan hij of zij een server met juist dezelfde instellingen starten uit deze *diskimage*. Dit bespaart veel tijd omdat het letterlijk enkele muisklikken kost om een snapshot te maken en een nieuwe server daaruit op te starten.

Na een destructieve test waarbij een automatische tool alle mogelijke parameters overspoelt, kan de instantie verwijderd worden na diagnose. De database kan dan behoorlijk aangetast zijn en het zou te veel tijd kosten om de schade terug te rollen. Vandaar dat het interessant is om met één enkele klik de instantie te verwijderen en een nieuwe instantie aan te maken om nieuwe testen uit te voeren.

Bovendien is het ook mogelijk om meerdere instanties intern te verbinden in hetzelfde datacentrum.

6.3.1 Firewall instellingen

Zoals eerder vermeld is het belangrijk dat een aantal zaken niet publiek bereikbaar zijn. Gelukkig is het vrij eenvoudig om gerichte firewall regels in te stellen zodat enkel eigen verkeer toegelaten is. Met de tool "UFW" (Uncomplicated Firewall) is dit, zoals de naam al doet vermoeden, eenvoudig om in te stellen en zeer effectief. Deze tool

---

[9] *Crawlers* zijn scripts die de inhoud van websites scannen om bepaalde informatie te verzamelen. Een bekend voorbeeld is de *crawler* van Google, maar er zijn ook *crawlers* die e-mailadressen verzamelen voor kwaadaardige doeleinden.

[10] *Snapshots* zijn momentopnames van een opslagmedium. In virtualisatieprogramma's en cloudtoepassingen is dit een vaak gebruikte functie.

beschermde de servers van publiek verkeer. Hierdoor was het mogelijk om een *cloud*-server toch privé op te stellen.

6.4 Resultaten

Er zijn in totaal drie belangrijke testen die meer inzicht geven in het eindoordeel. De impact van de firewall op de prestaties van de webserver bepaalt of gebruikers dit project effectief in productie kunnen gebruiken. Bij een te grote impact zal de gebruikerservaring in het gedrang komen en zullen bedrijven dit nooit gebruiken. De tweede test zal inzicht geven over de bescherming die de basisregels van Modsecurity kunnen leveren. De laatste test zal bepalen of de veiligheid van de implementatie in orde is. Een kwetsbare firewall spreekt zichzelf tegen.

6.4.1 Prestatie impact

De eerste test onderzoekt de prestatie impact van de firewall. Dit is getest door de vertraging bij de laadtijd van een pagina te meten als de firewall is ingeschakeld. Als de firewall is ingeschakeld dan staat Modsecurity aan en is er ook *logging* van de *requests* via de applicatie.

Deze testopstelling gebruikt twee servers die via een intern netwerk zijn verbonden in hetzelfde datacentrum. Het voordeel van dit internetnetwerk is dat er geen trage *hops* zijn die de test beïnvloeden. De serverconfiguratie wordt in volgende tabel weergegeven.

	Testserver
CPU	2.40 GHz – 1 core
RAM	512 MB
SWAP	4 Gb

Tabel 6.1: *Specificaties Testserver.*

De gebruikte tools voor het uitvoeren van de testen zijn “Siege” en “Apache Benchmark”. Dit zijn veelgebruikte stresstools om servers tot het uiterste te drijven waardoor ontwikkelaars inzicht krijgen of hun applicatie grote druk kan weerstaan. Via deze tools krijgt men een vergelijking hoeveel milliseconden de firewall vertraagt als de firewall is ingeschakeld.

Er zijn drie testscenario’s:

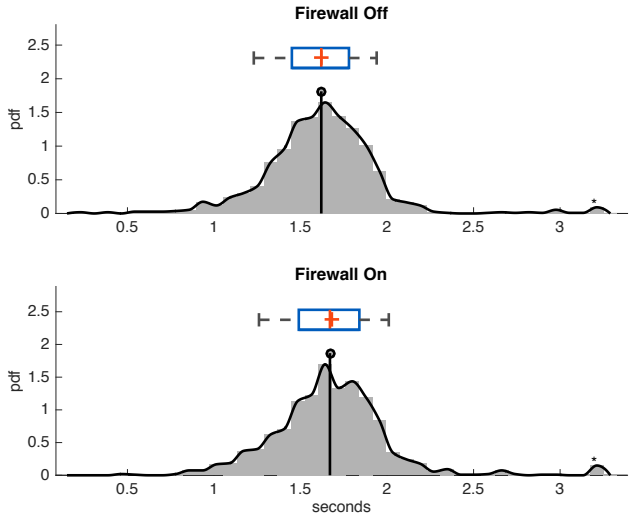
- De startpagina van een normale Wordpress installatie.
- Zoeken van een woord op een Wordpress website. Hierbij is het zoekwoord een parameter die wordt meegegeven.
- Een lege pagina waar verschillende parameters worden meegegeven.

6.4.1.1 Testscenario startpagina Wordpress

De tool gaat de startpagina van een doorsnee Wordpress website opvragen. Het gebruikte commando met zijn parameters:

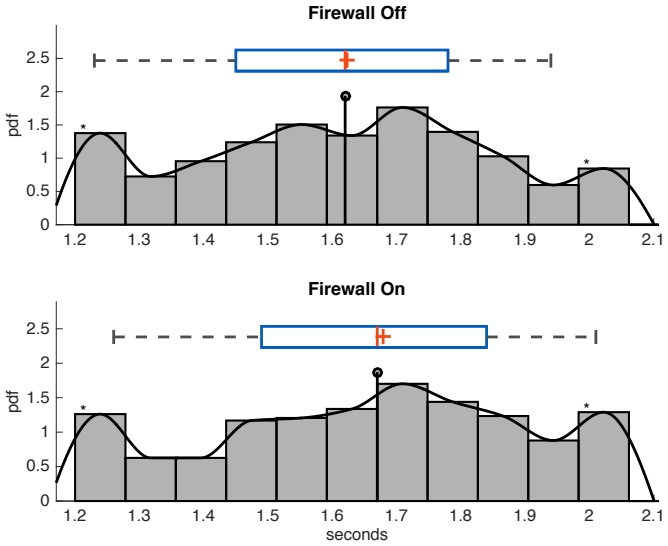
```
1.  /usr/bin/siege -d1 -c10 -t5m -v -l "http://10.133.4.192:333/"
```

Deze test duurt vijf minuten en zal tien gebruikers op hetzelfde moment simuleren. Er is ook een random wachttijd van maximaal één seconde. De volgende figuur toont de verdeling van de laadtijden op de startpagina.



***Figuur 6.1:**. Testscenario startpagina Wordpress. Bovenaan staat de firewall uit, onderaan staat de firewall aan.*

Op het eerste zicht lijkt er een minimaal verschil te zijn. De volgende figuur zoomt even in op het centrum.



***Figuur 6.2:** Ingezoomd op het centrum van de vorige figuur.*

De volgende tabel toont de verschillen tussen respectievelijk de mediaan en het gemiddelde van de laadtijden bij de uitgeschakelde en ingeschakelde firewall.

Mediaan verschil	0.0500 s
Gemiddelde verschil	0.0577 s

Tabel 6.2: *Resultaten van het testscenario bij de Wordpress startpagina.*

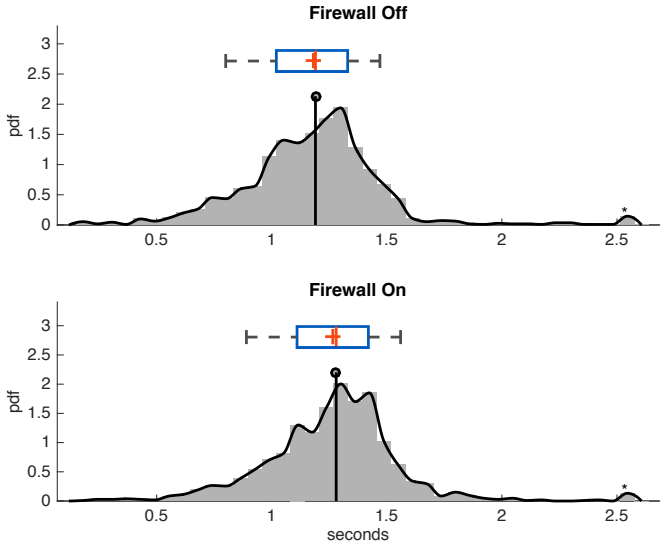
Een gemiddelde van 50 milliseconden extra is zeker aanvaardbaar. Bij een netwerkfirewall is dit niet acceptabel, maar bij een webapplicatie firewall wel omdat de laadtijd van een website in de ordegrootte zit van één tot twee seconden.

6.4.1.2 Testscenario zoekfunctie met een GET-request

Deze test is gelijkaardig aan de vorige test, alleen is er een extra GET-parameter aanwezig. Dit gebeurt via het volgende commando te gebruiken:

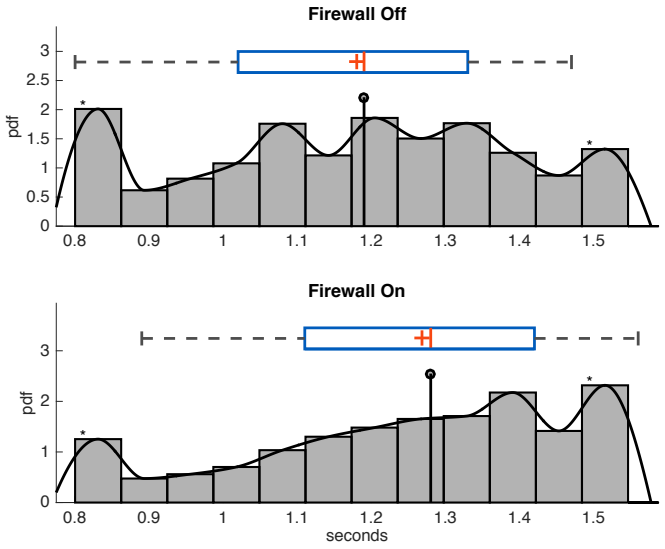
```
1.  /usr/bin/siege -d1 -c10 -t5m -v -l
    "http://10.133.4.192:333/search?keyword=testString"
```

Hier duurt de test ook weer vijf minuten met tien gesimuleerde, gelijktijdige gebruikers. De volgende figuur toont de verdeling van de laadtijden.



Figuur 6.3: *Testscenario zoekfunctie met een GET-request. Bovenaan staat de firewall uit, onderaan staat de firewall aan.*

Hier is het verschil iets groter als in de vorige test. De uitvergrootte figuur geeft dit duidelijker aan.



Figuur 6.4: *Ingezoomd op het centrum van de vorige figuur.*

De volgende tabel toont opnieuw de verschillen tussen respectievelijk de mediaan en het gemiddelde van de laadtijden bij de uitgeschakelde en ingeschakelde firewall.

Mediaan verschil	0.0900 s
Gemiddelde verschil	0.0872 s

Tabel 6.3: *Resultaten van het testscenario bij een GET-request.*

Met een mediaan van 90ms en een gemiddelde van 87ms is de impact groter dan in het voorgaande scenario. Omdat de applicatie meer parameters opslaat is dit een logisch gevolg. Deze resultaten beginnen al richting 100 milliseconden te eindigen wat tegen de grens van het acceptabele aanleunt.

6.4.1.3 Testscenario blanco pagina met parameters

Deze test maakt gebruik van de "Apache Benchmarking" tool. Omdat deze tool anders functioneert, is het zeker interessant om hiermee ook een vergelijking op te stellen. Dit brengt extra variatie in onze testmethoden.

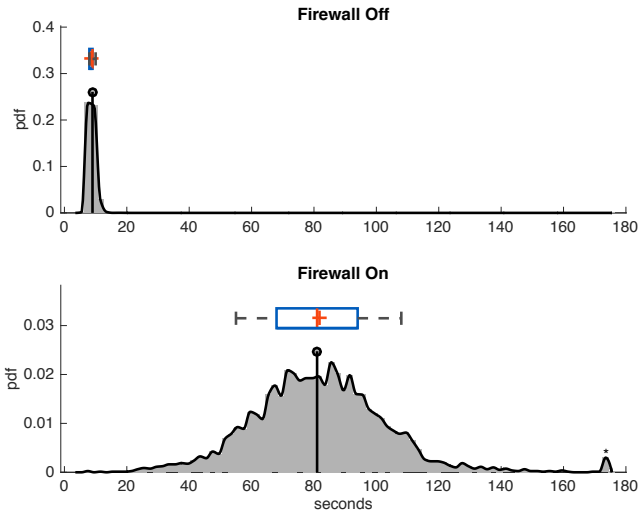
Het verschil met de vorige test is de toevoeging van een aantal parameters en het weglaten van de PHP factor. Omdat het een lege pagina is, moet PHP niets uitrekenen zodat de focus op de transactietijd ligt.

De tool is opgeroepen met het volgende commando:

```
1.  ab -n4000 -c20 -g /var/log/testab_on.log
    "http://10.133.4.192:90/inject_me.php?name=homer&surname=simpson&postcode=3929"
```

Dit simuleert twintig simultane gebruikers die in totaal vierduizend requests uitvoeren.

De volgende figuur toont de verdeling van de laadtijden.



Figuur 6.5: *Testscenario blanco pagina met parameters. Bovenaan staat de firewall uit, onderaan staat de firewall aan.*

In deze test komt de extra transactietijd duidelijk naar boven. Waar de lege pagina eerst ongeveer tien milliseconden nodig heeft, ligt de laadtijd bij het inschakelen van de firewall rond de tachtig milliseconden.

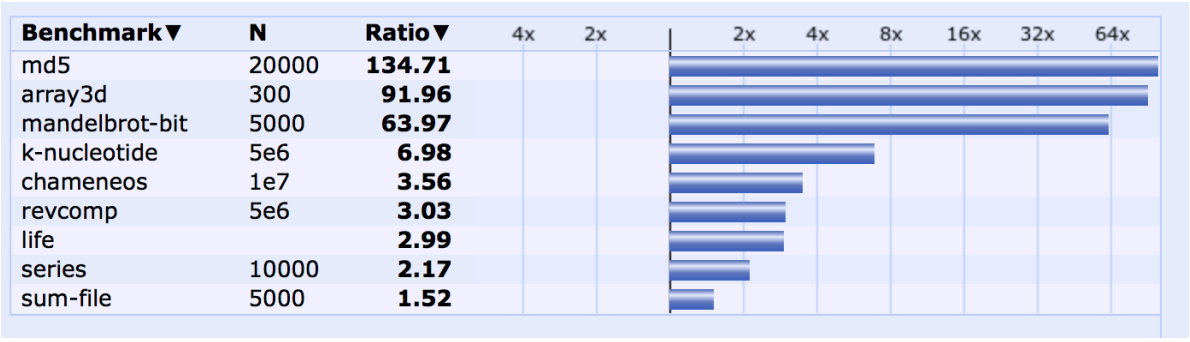
Mediaan verschil	72
Gemiddelde verschil	73.0837

Tabel 6.4: *Resultaten van het testscenario bij een blanco pagina met parameters.*

Deze resultaten liggen in lijn met de vorige testen waar er ook een zestig tot negentig milliseconden bijkomt.

6.4.1.4 Mogelijke optimalisatie

De vorige testresultaten waren nog acceptabel, maar optimalisatie is zeker aanbevolen. Dit is gelukkig mogelijk door in het logscript gebruik te maken van LUAJIT in plaats van de normale LUA-module. Deze "Just In Time" *compiler* verhoogt de prestaties verschillende keren tegenover de normale LUA-*compiler*. Afhankelijk van de gebruikte functies in de scripts kan deze *compiler* de snelheid tientallen keer verbeteren.



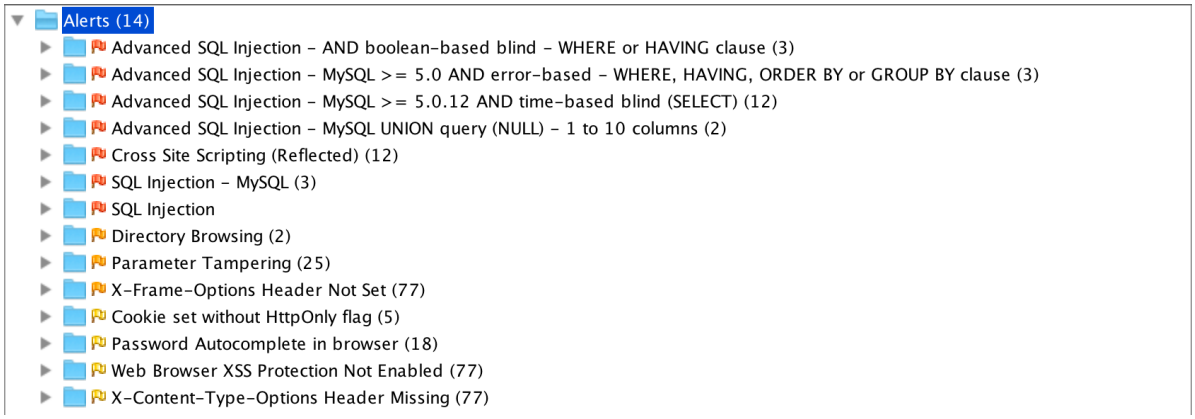
Figuur 6.6: *Verbetering prestaties tegenover de gewone LUA compiler [29].*

6.4.2 Sterkte basisregels bij Modsecurity

Deze test toont het nut en de kracht aan van de basisregels bij Modsecurity. Dit gebeurt praktisch door het scannen van een slechte webapplicatie "Bricks" met de tool OWASP ZAP.

Het inschakelen van de basisregels zou namelijk beschermen tegen de "OWASP-top ten" en deze test kijkt of dit in de praktijk ook waar is.

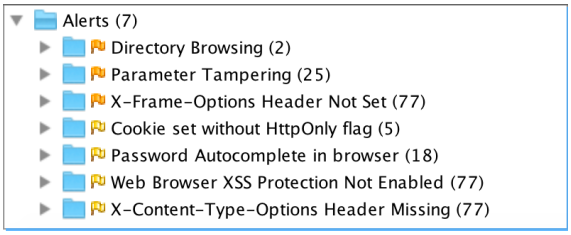
Als eerste is de kwetsbare applicatie getest met de Modsecurity basisregels uitgeschakeld.



Figuur 6.7: *Gevonden beveiligingsrisico's bij een uitgeschakelde firewall door OWASP ZAP.*

In de resultaten komen zaken als SQL-injectie en Cross Site Scripting (XSS) voor. Dit zijn twee beveiligingsrisico's die voorkomen in de "OWASP-top ten".

De resultaten van het scannen met alle ingeschakelde basisregels, zijn veel beter. Merk op dat één regel ("modsecurity_crs_21_protocol_anomalies") uitgeschakeld moest zijn omdat de firewall de scanningtool detecteerde.



Figuur 6.8: *Overzicht van de gevonden beveiligingsrisico's bij een ingeschakelde firewall met de basisregels geactiveerd. De gebruikte scanningtool is OWASP ZAP.*

Door het inschakelen blijven er enkel nog lichte waarschuwingen over. Deze waarschuwingen gaan eerder over aanbevelingen in de configuratie van de webserver en de applicatie. Door bijvoorbeeld het configuratiebestand van Apache aan te passen zijn een aantal van deze waarschuwingen verholpen. Zaken als "X-frame-options header" en "Webbrowser XSS protection" in de *headers* zorgen ervoor dat in geval van een XSS-kwetsbaarheid in de applicatie, de *browser* van de bezoeker deze aanval toch tegenhoudt. Het is een extra beveiligingsmaatregel en zeker een goede aanbeveling om onbekende of toekomstige XSS-kwetsbaarheden tegen te houden.

Een andere aanbeveling van de tool is het uitschakelen van "password autocomplete" in de applicatie. Dit is niet direct een risico, maar eerder een aanbeveling voor de ontwikkelaars zodat de beveiliging bij de browser van de gebruiker beter is.

6.4.3 Pentesten van de applicatie

Natuurlijk is de ontwikkelde applicatie ook uitvoerig getest. Met OWASP ZAP was er één SQL-injectie gevonden. Dit was vreemd omdat in de PHP-code altijd PDO is gebruikt. De volgende code toont de fout die deze SQL-injectie mogelijk maakte.

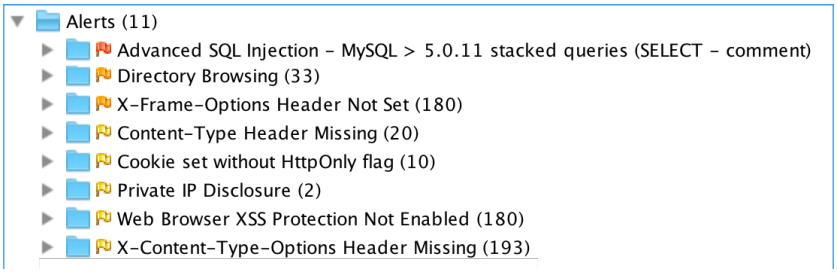
```
1.  $n=$_GET['last_n_parameters'];
2.  …
3.  $sql="SELECT * FROM (SELECT * FROM ".$_SESSION["host_requests_table"]." INNER JOIN
    ".$_SESSION["host_parameters_table"]." ON
    ".$_SESSION["host_parameters_table"].".request_id =
    ".$_SESSION["host_requests_table"].".unique_id ORDER BY request_id) sub ORDER BY
    unique_id DESC limit $n";
4.  …
```

PDO biedt geen ondersteuning voor sleutelparameters als "ORDER BY". Dit was tijdelijk opgelost door de parameter ($n) rechtstreeks in de SQL-query te steken. Helaas werd deze variabele niet gevalideerd waardoor een SQL-injectie langs deze parameter kon plaatsvinden.

Dit is nu opgelost door een korte validatie te gebruiken.

```
1.   if(!ctype_digit($n)){
2.      die("tampering detected");
3.  }
```

De rest van de waarschuwingen zijn toe te wijten aan de configuratie van de webserver. De gebruiker installeert de implementatie op een bestaande webserver, waardoor de gebruiker zijn eigen serverconfiguratie heeft. De configuratie van de webserver is overal anders en het was dan ook nutteloos om hier tijd in te steken in de testopstellingen.



Figuur 6.9: *Overzicht van de gevonden beveiligingsrisico's in de applicatie zelf. De gebruikte scanningtool is OWASP ZAP.*

Als laatste is er nog de "Private IP Disclosure" opmerking wat eigenlijk een *false positive* is. In het web paneel kan het logboek geraadpleegd worden, maar dit bevat dus ook IP-adressen. Dit heeft de scantool opgevangen en gerapporteerd als het uitlekken van een IP-adres.

7 TOEKOMSTIGE VERBETERINGEN EN FUNCTIES

Tijdens het testen zijn er verschillende *bugs* naar boven gekomen, maar ook voorstellen voor enkele toekomstige verbeteringen en functies.

7.1 Installatiescript

Er bestaat al een procedure om een webserver te configureren met de zelflerende WAF, maar een installatiescript is interessant voor nieuwe gebruikers die onmiddellijk aan de slag willen.

Een *shell script* kan de files downloaden van de projectwebsite, de server volledig configureren en een uniek wachtwoord voor de webapplicatie aanmaken waarmee de gebruiker direct kan inloggen. Dit uniek wachtwoord is tegenwoordig een security standaard wat beschermt tegen automatische web scanners die inloggen op admin-panelen met de standaard wachtwoorden.

Omwille van de verschillende Linux distributies en verschillende configuraties op een server is het zeer ingewikkeld om hier een universeel installatiescript voor te maken.

7.2 Database optimalisatie

7.2.1 Initialisatie

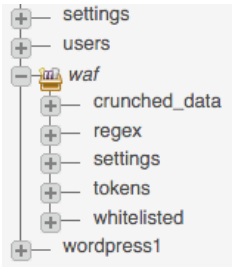
Bij het importeren van alle databases zou het installatiescript de wachtwoorden moeten aanpassen. Daarnaast zou het ook een aantal basiszaken zoals installatielocaties en andere parameters moeten instellen.

7.2.2 MySQL gebruikersrechten

Elke databasegebruiker moet de minimale gebruikersrechten hebben en dit enkel voor de databases die zijn gebruikt door deze databasegebruiker. Dit zorgt ervoor dat een aanvaller andere databases niet kan uitlezen en aanpassen als de webapplicatie is gehackt. Daarnaast is het ook wenselijk om in te stellen welke commando's een databasegebruiker mag gebruiken. Het commando "DROP" is een heel destructief commando, het verwijdert namelijk een tabel en zou dus bij de meeste toepassingen niet nodig zijn.

7.2.3 Naamgeving databases

De implementatie gebruikt een aantal databases met verschillende namen. Het is eigenlijk beter als deze databases voor de WAF een *prefix* krijgen zoals "waf_". De database "users" zou beter "waf_users" heten zodat deze veelgebruikte naam terug beschikbaar is voor de gebruiker.



Figuur 7.1: *Prefixen worden automatisch gebundeld in PHPMyAdmin.*

7.3 Afval opruimen

Dit gebeurt via een functie die oude en tijdelijke bestanden verwijdert. De *watchfolders* bevatten bijvoorbeeld allemaal bestanden die de daemon hebben aangestuurd en niet meer nodig zijn na gebruik. Het zou ook wenselijk zijn om na een bepaald tijdsinterval deze bestanden te verwijderen. Deze zijn toch niet meer van nut en nemen enkel maar plaats in.

De database bevat ook *tokens* die de functie moet opruimen.

Als laatste kan de functie ook de database die de opgeslagen *requests* van de webserver bevat opruimen om plaats te besparen bij drukke websites.

7.4 Andere webservers

Voorlopig draait deze implementatie enkel op Apache. Maar in principe is het ook mogelijk om na enkele aanpassingen dit werkend te krijgen op NGINX of Microsoft IIS.

BESLUIT

Met het toenemend aantal aanvallen op websites en applicaties is het aangewezen om een firewall te gebruiken. Niet alleen om klantendata te beschermen, maar ook om diensten waarvan het bedrijf afhankelijk is, draaiend te houden.

Modsecurity is een effectieve oplossing, maar vereist een zekere hoeveelheid technische kennis.

Met deze masterproef is aangetoond dat Modsecurity ook toegankelijk kan zijn voor personen met een beperktere security achtergrond. Een frisse en eenvoudige interface zorgt ervoor dat beginners snel hun weg kunnen terugvinden, maar ook dat experts snel bepaalde zaken kunnen instellen. Het aanvalsoverzicht geeft gebruikers een duidelijk beeld vanwaar de aanvallen komen en welk type aanvallen dit zijn. De applicatie geeft deze informatie grafisch weer in tabellen en diagrammen waardoor deze informatie snel kan opgenomen worden door gebruikers.

Het concept van de *daemon* die administratorhandelingen gaat uitvoeren op commando van de web interface is heel interessant. Hierdoor kan de webserver in een veilige configuratie draaien en toch commando's uitvoeren die normaal enkel door *root* gebruikers kunnen uitgevoerd worden. Het is niet alleen snel, maar ook vooral veilig wat net de topprioriteit is van deze aanpak.

Bij de prestatietesten is het duidelijk dat deze implementatie extra resources vraagt. Voor webapplicatie firewalls zijn een zestigtal milliseconden nog acceptabel, zeker als dit drastisch verminderd kan worden door LUAJIT te gebruiken. Uit de testen komt ook naar boven dat de basisregels van Modsecurity in staat zijn om aanvallen op kritieke kwetsbaarheden tegen te houden. Dit geeft al een basisbescherming nog voor het zelflerende gedeelte van de firewall zijn werk heeft gedaan.

In de toekomst kan de applicatie nog verder uitgebouwd worden om meer functies te kunnen uitvoeren en op meerdere platformen te kunnen draaien. Het potentieel is zeker aanwezig.

LITERATUURLIJST

[1] B. Barrett, „Hack Brief: Hackers Are Holding an LA Hospital's Computers Hostage," 16 02 2016. [Online]. Available: http://www.wired.com/2016/02/hack-brief-hackers-are-holding-an-la-hospitals-computers-hostage/.

[2] L. Bershidsky, „Russians Have Learned How to Hack Power Grids," 7 01 2016. [Online]. Available: http://www.bloombergview.com/articles/2016-01-07/russians-have-learned-how-to-hack-power-grids. [Geopend 04 02 2016].

[3] The Telegraph, „Chinese hackers seized 'gold mine' of information about US spies and army personnel," 13 06 2015. [Online]. Available: http://www.telegraph.co.uk/news/worldnews/northamerica/usa/11672451/Chinese-hackers-seized-gold-mine-of-information-about-US-spies-and-army-personnel.html. [Geopend 22 03 2016].

[4] D. Doe, „Hackers have breached Goldcorp, a Canadian gold-mining firm," 27 04 2016. [Online]. Available: http://www.dailydot.com/politics/goldcorp-hack-data-dump/. [Geopend 2 05 2016].

[5] Akamai, May 2015. [Online]. Available: https://www.stateoftheinternet.com/downloads/pdfs/resources-web-security-2015-web-app-attack-stats-ponemon-infographic.pdf. [Geopend 29 Februari 2016].

[6] Strategic Cyber, LLC, „Homepage," 22 2 2016. [Online]. Available: https://www.cobaltstrike.com/.

[7] A. D. Cid, „RevSlider Vulnerability Leads To Massive WordPress SoakSoak Compromise," 15 12 2014. [Online]. Available: https://blog.sucuri.net/2014/12/revslider-vulnerability-leads-to-massive-wordpress-soaksoak-compromise.html.

[8] D. Bass, „Six Things You Need to Know About ATMs and the Windows XP-ocalypse," *Bloomberg,* 2014.

[9] Shodan, „Shodan is the world's first search engine for Internet-connected devices.," 2013. [Online]. Available: https://www.shodan.io/. [Geopend 26 01 2016].

[10] Cisco, „Cisco 2016 Annual Security Report," Cisco, 2016.

[11] M. Ciampa, Security Awareness: Applying Practical Security in Your World, 4th Edition red., Western Kentucky, 2014, p. 304.

[12] L. O. M. a. E. C. Nicolas Falliere, „W32.Stuxnet Dossier," 02 2011. [Online]. Available: https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf. [Geopend 05 02 2016].

[13] P. Mavrommatis, „Protecting people across the web with Google Safe Browsing," 12 03 2015. [Online]. Available: https://googleblog.blogspot.be/2015/03/protecting-people-across-web-with.html. [Geopend 08 02 2016].

[14] C. Wueest, „Underground black market: Thriving trade in stolen data, malware, and attack services," 20 11 2015. [Online]. Available: http://www.symantec.com/connect/blogs/underground-black-market-thriving-trade-stolen-data-malware-and-attack-services. [Geopend 08 02 2016].

[15] T. Hunt, „Check if you have an account that has been compromised in a data breach," 2016. [Online]. Available: https://haveibeenpwned.com/.

[16] OWASP, „OWASP Top 10," The OWASP Foundation, 2013.

[17] I. Ristic, Modsecurity handbook, Development Version (revision 629) red., J. G. - Risti, Red., London: Feisty Duck Limited, 2015, p. 379.

[18] Modsecurity, „What Can ModSecurity Do?," [Online]. Available: https://www.modsecurity.org/about.html. [Geopend 09 02 2016].

[19] J. Graham-Cumming, „CloudFlare's new WAF: compiling to Lua," 23 08 2013. [Online]. Available: https://blog.cloudflare.com/cloudflares-new-waf-compiling-to-lua/.

[20] Trustwave Holdings, Inc, „Reference Manual," 2016. [Online]. Available: https://github.com/SpiderLabs/ModSecurity/wiki/Reference-Manual.

[21] Trustwave Holdings, „Trustwave," 2016. [Online]. Available: https://ssl.trustwave.com/web-application-firewall.

[22] OWASP, „The free and open software security community," 2016. [Online]. Available: https://www.owasp.org/index.php/Main_Page.

[23] CVE Details, „The ultimate security vulnerability datasource," 2016. [Online]. Available: https://www.cvedetails.com/.

[24] Materialize, „A modern responsive front-end framework based on Material Design," 2016. [Online]. Available: http://materializecss.com/.

[25] The PHP Group, „Deprecated features in PHP 5.5.x," 2016. [Online]. Available: http://php.net/manual/en/migration55.deprecated.php.

[26] PortSwigger, 2016. [Online]. Available: https://portswigger.net/burp/.

[27] w3techs, „Usage of content management systems for websites," 02 05 2016. [Online]. Available: http://w3techs.com/technologies/overview/content_management/all.

[28] Digitalocean, „Simple Cloud Computing, Built for Developers.," 2016. [Online]. Available: https://www.digitalocean.com/.

[29] M. Pall, „Performance: x86/x64,” 2016. [Online]. Available: http://luajit.org/performance_x86.html. [Geopend 21 April 2016].

[30] Symantec, „Internet Security Threat Report Apendices,” 5 April 2015. [Online]. Available: https://www4.symantec.com/mktginfo/whitepaper/ISTR/21347931_GA-internet-security-threat-report-volume-20-2015-appendices.pdf. [Geopend 28 Februari 2016].

[31] Akamai, „State of the internet,” [Online]. Available: https://www.stateoftheinternet.com. [Geopend 25 Februari 2016].

[32] Trustwave Spiderlabs, „Modsecurity Open Source Web Application Firewall,” [Online]. Available: https://www.modsecurity.org. [Geopend 26 Januari 2016].

[33] The Apache Software Foundation, „Welcome to The Apache Software Foundation!,” [Online]. Available: http://www.apache.org. [Geopend 22 Februari 2016].

[34] Symantec, „Internet Security Threat Report Apendices,” 2015.

[35] Akamai, „The Cost of Web Application Attacks,” Akamai Technologies, 2015.

BIJLAGE 1: WETENSCHAPPELIJKE PAPER

Executing Root Commands in Web Applications While Maintaining Security Best Practices

Cox Vincent, Cooman Nico, Jans Stijn and Vermeulen Gustaaf

Abstract—**These guidelines allow a developer to configure a web application that can securely execute root commands. An overview of the disadvantages and dangers of running a web application as root provides insight into the importance of this topic. This document will provide a secure solution via a daemon, along with additional security measures to optimize a developer's web application environment.**

Index Terms—**Security, web application, root.**

I. Introduction

MOST web applications are connected to the Internet and can be accessed by anyone, including attackers. Unfortunately, a vast majority of web applications are prone to attack vectors like SQL injection, XSS and various other techniques. Running the web application as root aggravates the impact of a breach and gives the attacker full control over the server as a root user. This does not only affect the web application itself but also the entire server, including other services on the machine. This is why security professionals always prefer to run a web server as a regular user instead of as a root user. However, running as a regular user limits some functionality like execution of scripts or root commands via a web application. This paper shall address these limitations by discussing some options to execute privileged commands in a web application as a regular user.

II. Advantages of not running a web server as root

A. Extra layer of security

A lot of web applications and websites suffer from the vulnerabilities listed in the OWASP Top Ten [3]. The Open Web Application Security Project (OWASP) [4] is a worldwide not-for-profit charitable organization that focuses on improving the security of software. Their mission is to make software security transparent, so that individuals and organizations across the world can take informed decisions about how to counter software security risks. The OWASP Top Ten provides a broad consensus about the most critical web application security flaws. A few examples from this list are SQL injection, XSS and broken session management.
While a breach through any of the above techniques can be very dangerous on its own, the risk escalates drastically when the web application is running with root privileges. If an attacker manages to upload a web shell in a web application that is running as root, he or she can browse the entire system of the server and modify files including system files, databases and passwords.
This would also affect all the other services and web applications, because the attacker would be able to access and edit everything.
Running as a regular user would limit the impact of a breach to the web application; the entire system with all its services and applications would not be impacted.

B. Limiting collateral damage caused by bugs

A bug or misconfiguration in an application could possibly delete folders, even those outside the application folder, when the application is running as root. This is not necessarily caused by attackers. The fault can be attributed to a programmer who made a mistake despite having good intentions.
Running as a regular user with the right permissions would prevent deletion or modification of files outside the web application folder.

C. Limiting the impact of zero-day exploits or state hackers

A zero-day vulnerability (also known as zero-hour or 0-day) is a previously undisclosed vulner-

N. Cooman and S. Jans are with The Security Factory, Schelle, Belgium, e-mail: info@thesecurityfactory.be.

G. Vermeulen is with the Department of Electrical Engineering (ESAT) TC, KU Leuven, Technology Campus Geel, Belgium, e-mail: staf.vermeulen@kuleuven.be

ability in computer software that hackers can exploit to adversely affect computer programs, data, additional computers or even a whole network.
It is known as a ‘zero-day’ because once the flaw becomes known, the software’s writer has zero days to plan or suggest any mitigation against its exploitation (for example, by recommending workarounds or by issuing patches).
Especially advanced hackers and state-sponsored hackers have several zero-day exploits in their arsenal. If a service suffers from a zero-day while running as root, the entire machine can be controlled.

III. Executing root commands in a secure and controlled way using a daemon

The most effective and secure method in such a situation is to use a daemon running as root. The web server triggers the daemon to do certain predefined actions as root. This paper focuses on a bash daemon, but the principles are applicable for all languages.

A. Security measurements

The following guidelines must be followed for the daemon concept to be fruitful.

1) Hardcode commands that can be executed: Never get the input from a web application (or normal user account) to be executed as a command via a root account. It may be a tempting option because one does not have to hard code each command. However, if an attacker breaches the web application, he or she can inject commands which will be executed as root. This would make the use of a daemon pointless.

2) Use a case function in the daemon: A case function is a great way to let the daemon know what command needs to be run. This improves the security because commands are limited to a controlled and predefined set of commands.

3) Use escaping and sanitization: A great example demonstrating the importance of this problem involves variables used without quotes in bash [6]. Assume that a developer wants to echo the following string:

```
testString="home/*"
```

To achieve this, the developer uses the following command:

```
echo "$testString"
```

This will output the intended and original string. However, the output is different when the developer uses the same command without quotes:

```
echo $testString
```

This yields a very interesting result:

```
"/home/vincentcox"
```

It displays the user account, which is not what the developer expects. Always use escaping and sanitizing on passed variables in the daemon (and applications in general).

B. Triggering the daemon

”Inotifywait” efficiently waits for file changes using Linux’s ”Inotify” interface. It is very suitable for shell scripts because the package can be installed easily. It can either exit once an event occurs, or continue to execute and output events as they occur. This method gives an almost immediate response when a developer wants to trigger the daemon.
For example, one can create a directory called ‘triggerdaemon’ in the web application file structure. After this, the developer can create a daemon using Inotifywait locked on the new folder. It is possible to write a file to that folder in PHP, thus triggering the daemon. Listing 1 in the appendix shows an example code for a daemon.

IV. Alternatives

A. Using a cronjob

This method may sound very appealing because most beginners are familiar with cronjobs. However, the response time is limited because the smallest interval for a cronjob is one minute. To pass variables towards the root script, one can use a database like MySQL. Remember to escape and sanitize Bash code in the daemon script.

V. Optimisation

A. Whitelist the actions performed by the daemon

This is a very effective security measure because when an injection occurs at the daemon level, the damage is limited to the permitted commands.
To achieve this, a new user has to be created with a sudo configuration. An example with visudo:

```
peter, %operator ALL= /sbin/, /usr/sbin, /
    usr/local/apps/check.pl
```

Do not use this to loosen security by permitting all actions!

B. Chrooting

Chroot is an operation that changes the apparent root directory for the current running process and its children. A program that runs in such a modified environment cannot access files and commands outside the environmental directory tree. This modified environment is called a *chroot jail*.
A developer can limit the daemon to certain directories. This will tighten the overall security and it prevents the daemon from modifying system and configuration files.

C. DOS protection

If an attacker is able to breach the web interface, it means he or she has access to the watchfolder of the daemon and possibly also the database. If the daemon or script gets triggered at a high pace (by writing to the watchfolder or by inserting requests in the database), this would cause a Denial Of Service (DOS). For example, an attacker can trigger the deamon to let the web server restart at a high pace. Each restart command will take the server offline for a few seconds and a high pace of restart commands would take the server offline for a long time. This can be prevented by using tokens and timestamps.
The script or daemon can check if the timestamp from the trigger request is in a range from the current time and if the token is valid. This can be done by counting the trigger requests made in the last X minutes. If this count exceeds a threshold, the admin can be notified. A lot of failed checks indicate that a serious bug has occurred, or that an attacker has breached the web interface and is trying to DOS the system.

VI. Conclusion

Performing root actions triggered by a web application can be very time-consuming and difficult to implement in a secure way. In mission critical systems, these extra security measures are worth the effort because the time to recover from a root breach can be many times higher than the time taken to put these measures in place.
Moreover, name damage and leaked information about clients may also lead to a huge financial impact on a company.

Appendix

Listing 1. Example bash daemon [5].

```
#!/bin/bash
logfile="/var/log/daemon.log"
watchdir_deamon_function1="/var/www/hmtl/
    watchdir_deamon_function1"
trap process_USR1 SIGUSR1
process_USR1() {
echo 'Watchdog aborted! (USR1 SIGNAL)'>>
    $logfile
exit 0
}
deamon_function1(){
while inotifywait --outfile $logfile -e
    create $watchdir_deamon_function1; do
echo "Your function code when triggered">>
    $logfile
done
exit 0
}
me_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}
    " )" && pwd )"
me_FILE=$(basename $0)
cd /
if [ "$1" = "child" ] ; then   # 2. We are
    the child. We need to fork again.
shift
umask 0
exec setsid $me_DIR/$me_FILE refork "$@" </
    dev/null >/dev/null 2>/dev/null &
exit 0
fi
if [ "$1" != "refork" ] ; then # 1. This is
    where the original call starts.
exec $me_DIR/$me_FILE child "$@" &
exit 0
fi
# 3. We have been reforked.
shift
# Spawning the daemons
deamon_function1 &
```

References

[1] I. Ristic, *Modsecurity handbook*. Feisty Duck Limited, 2015.
[2] Cert. Web server security best practices. [Online]. Available: https://www.cert.be/docs/web-server-security-best-practices
[3] OWASP, "Owasp top ten 2013," *The ten most critical web application security risks*, 2013.
[4] ——. The free and open software security community. [Online]. Available: https://www.owasp.org/index.php/Main_Page
[5] S. answer. Best way to make a shell script daemon? [Online]. Available: http://stackoverflow.com/a/29107686
[6] A. Robbins, *Bash pocket reference*, 2016.

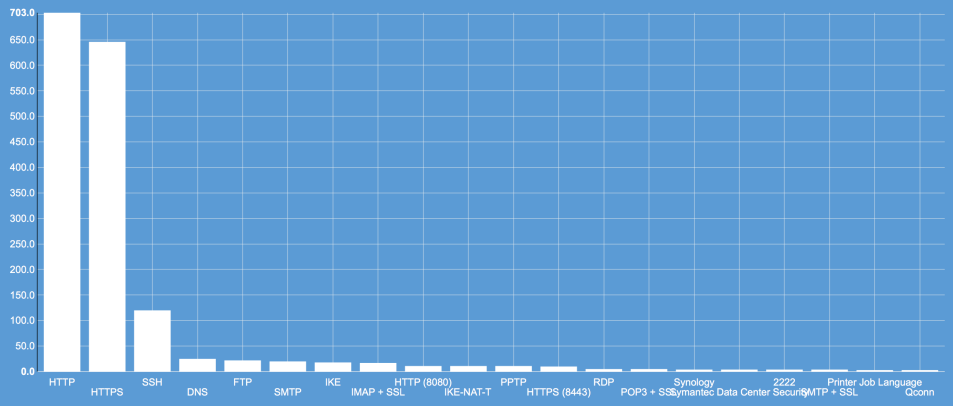
BIJLAGE 2: SHODAN RAPPORT

KULeuven op shodan.io

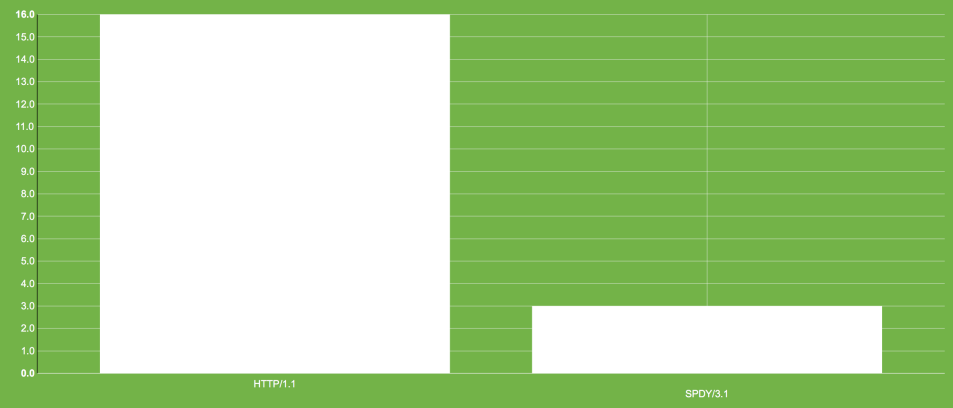
Search for org:"Katholieke Universiteit Leuven" returned 1,688 results on 27-03-2016

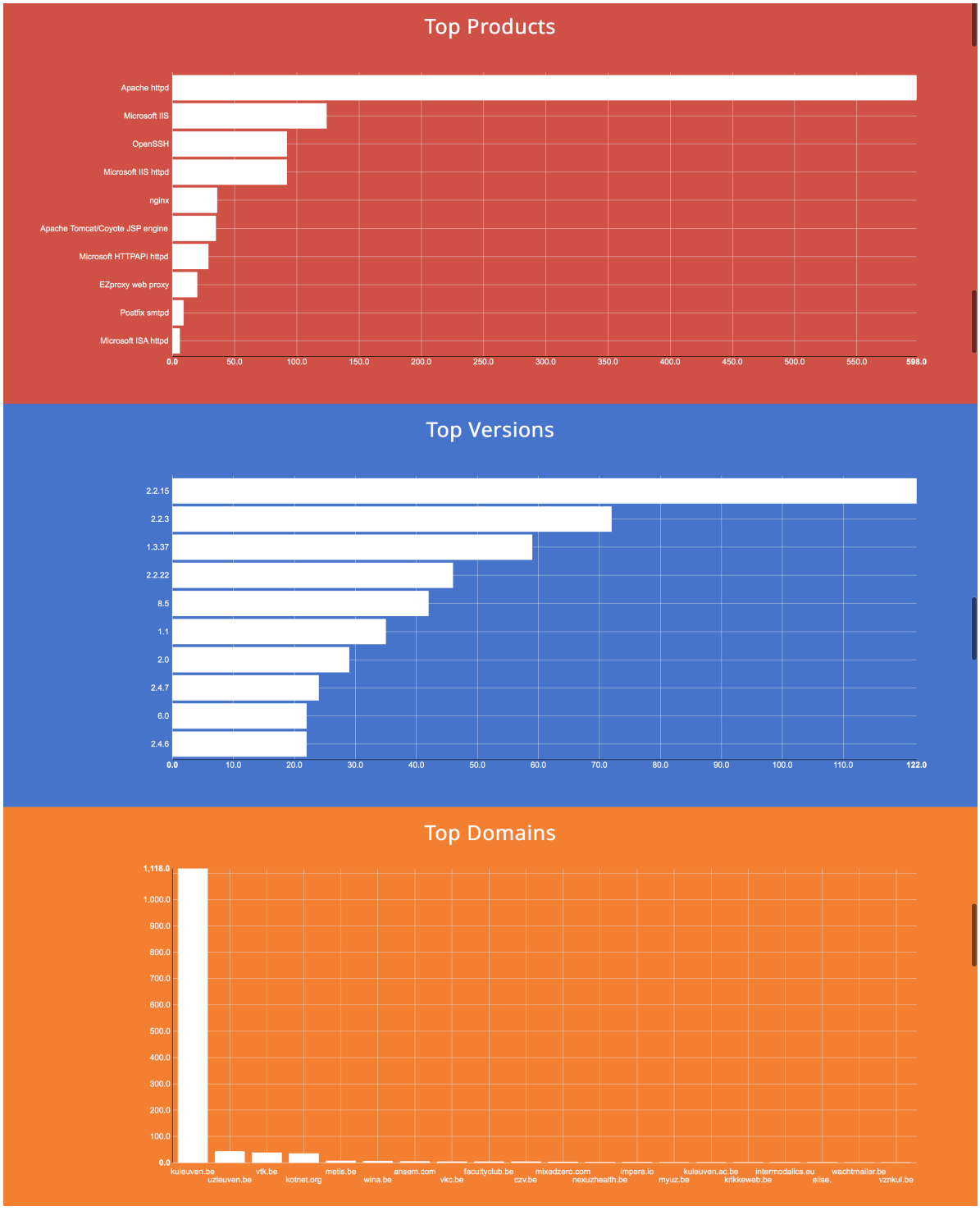
Belgium

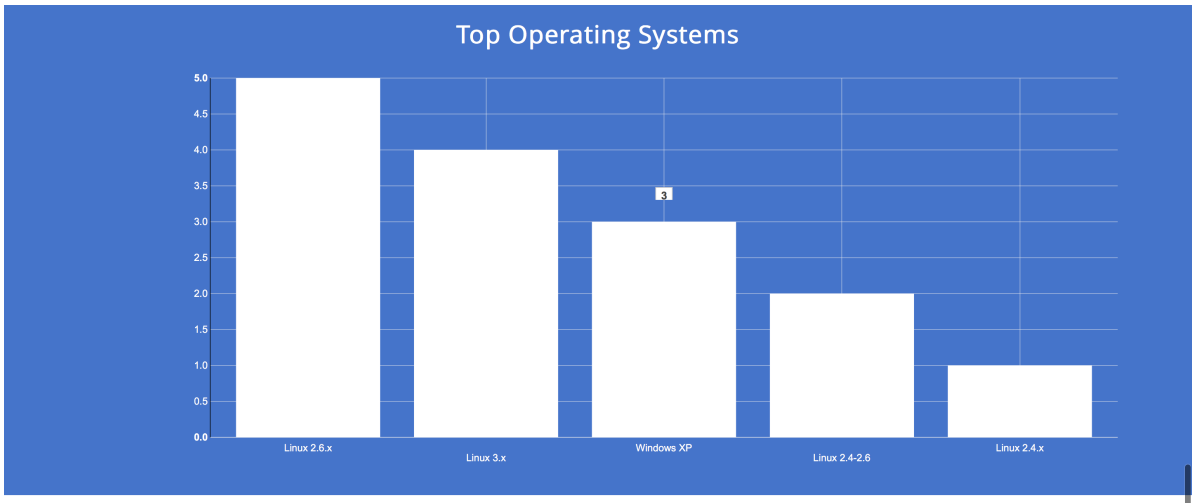
Top Services



Negotiated HTTP Versions




Top Products
Apache httpd
Microsoft IIS
OpenSSH
Microsoft IIS httpd
nginx
Apache Tomcat/Coyote JSP engine
Microsoft HTTPAPI httpd
EZproxy web proxy
Postfix smtpd
Microsoft ISA httpd
0.0
50.0
100.0
150.0
200.0
250.0
300.0
350.0
400.0
450.0
500.0
550.0
598.0
Top Versions
2.2.15
2.2.3
1.3.37
2.2.22
8.5
1.1
2.0
2.4.7
6.0
2.4.6
0.0
10.0
20.0
30.0
40.0
50.0
60.0
70.0
80.0
90.0
100.0
110.0
122.0
Top Domains
1,118.0
1,000.0
900.0
800.0
700.0
600.0
500.0
400.0
300.0
200.0
100.0
0.0
kuleuven.be
uzleuven.be
vtk.be
kotnet.org
metis.be
wina.be
ansem.com
vkc.be
facultyclub.be
czv.be
mixedzero.com
nexuzhealth.be
impera.io
myuz.be
kuleuven.ac.be
krikkeweb.be
intermodalics.eu
elise.
wachtmailer.be
vznkul.be



BIJLAGE 3: EMAIL IVAN RISTIC

```
From:  Ivan Ristić <ivanr@webkreator.com>
Subject: Re: Question about LUA and mod security
Date: 15 Sep 2015 at 20:35:13
To: Vincent Cox <info@vincentcox.com>
```

Ivan Ristić wrote:
It's too much work to research learning and make it production-ready at the same time. Thus, I would focus on the research part first. Once you're happy with the design, if you want, then focus on making it production ready.

If you attempt to do everything at once, you will struggle a lot with boring stuff and won't have enough time to do the really interesting bits, which is the learning.

In particular, trying to support 3 web servers would be a nightmare.

For data storage; you could create your own storage format and dump all the information into a file. That will be possible if you're using a single web server process, because all your data will be in the same place. You only need one function to save your state periodically and another function to load state on server startup.

Another option could be Redis. That would work with multiple processes.

Restarting the web server to activate new rules is not acceptable in production if you care about availability. Just design from the beginning to have your engine auto-reload without interfering with the web server.

My final advice would be that, if you're serious about this project, build your own brand and avoid tagging onto ModSecurity.

Good luck!

> Dear Ivan Ristic,
>
> Thanks for your input! It really means a lot to get an email directly from you and get advice on these topics. The whole concept is a bit hard to explain on sites like stackoverflow, because a lot is involved and those sites focus on more general topics.
>
> > You could write the whole thing in Lua. It's not necessary to produce ModSecurity rules for blocking – you can block from Lua itself.
> > The only issue there is information sharing, because Apache will typically use more than once process, even with threads. But because your primary goal is research, you could simply deploy with a single Apache process (and many threads) and everything would be in RAM.

How do you suggest to store/edit data from Lua? To get a self-learning model, I have to collect and crunch some data to adjust my filters accordingly. This data needs to persist after the lua script ends.
Because there can be multiple instances of my script (like you said Apache would spawn multiple threads etc) I was thinking to use MySQL, but it would add a "relative big" latency to the script and thus the server in general. I've put my question on StackOverflow because I couldn't find an anwser on the internet; documentation on the Lua API is very rare. `http://stackoverflow.com/questions/32580210/`

> For an alternative approach, consider using OpenResty, which is a combination of Nginx and LuaJit. As with Apache, configure Nginx with a single process to solve the problem of information sharing.

I was planning to make the project work on every webserver which Modsecurity support: Apache, IIS and Nginx. And this approach would only focus on Nginx I think (correct me if I'm wrong).

> No matter what you choose, your entire project will be a single Lua program.

I was thinking of using the Lua API mainly as ßtatistics collector"(grab the parameters of a request etc), and crunch these statistics so I get modsecurity rules which can be placed in the modsecurity config folder.
To "fix"the ïssue"that apache can't edit config folders and restart the server, I have written a Bash Daemon which runs as root. This daemon uses inotifyWait, which watches a folder (where Apache can write to) for changes. If a certain folder changes, it performs a root action like reloading the Apache server so that the Modsecurity rules will be applied.
I know, this makes no sense at first, but let me explain with an example:

---

web folder of the webuserinterface of my project:
`/var/www/html/modsecurityGUI`
watchfolder of the daemon:
`/var/www/html/modsecurityGUI/watchdog/reload_restart_Apache/`
so if a user from the webinterface wants to reload the server, PHP writes a token to the MySQL database and then to a file in
`/var/www/html/modsecurityGUI/watchdog/reload_restart_Apache/`. Because php writes to the watchfolder, the daemon wakes up (because of inotifyWait), and checks if the token matches with the one in the database.
Because the daemon runs as root, it can perform a reload or restart of the apache server.

In this way, the webinterface can perform predefined root actions without running as root itself. It will NEVER execute a command in this watchfolder. Only predefined commands which are linked by the daemon according to the folder being watched.
The token will be generated in such a way that we can verify that the user requesting it has a valid sessionID + an expiration date.
I have to think the token-thing trough, because if this system is flawed, a restart/reload loop can be triggered by an attacker and cause a DOS.
If there is a better way to let the webinterface communicate with a root daemon, please let me know.
I was also planning to use the daemon to load the modsecurity rules in the user interface. So users can use standard rules like "OWASP basic XSS", "OWASP basic SQLinjection" in the webinterface, next to suggested rules by the self learning engine.
Is this whole thing a good idea? Please be so honest as possible, I am trying to make a self-learning WAF which allows the user interact with in a GUI. This is pointless if I'm making a platform which is flawed by design.
What else do you suggest if this was not a good idea?
Because the Lua API is marked as experimental, I'm affraid that some rule filtering-features will not work and break things in the future.
Sorry for all the questions. If you want a compensation for giving me the advice, I can fully understand that!
Kind regards,

Vincent Cox

```
From: Ivan Ristic <ivanr@webkreator.com>
Reply: Ivan Ristic <ivanr@webkreator.com> <mailto:ivanr@webkreator.com>
Date: 14 Sep 2015 at 21:58:31
To: Vincent Cox <info@vincentcox.com>,
Feisty Duck <support@feistyduck.com>
Subject: Re: Question about LUA and mod security
```

Ivan Ristic wrote:

> Hi there,

Hi Vincent,

> I'm a student at the University in Leuven. I'm doing research to make a ßelf learning"WAF with modsecurity.
> ßelf learningïn this way: A form on a website has 3 fields: "name", "last name", "phone number". Most of the time only letters will appear in the argument "nameïn a post request. The ßelf learning"part starts here: it will suggest a modsecurity rule that only allows letters for the "nameärgument.
> After reading the documentation I think LUA can be handy to make custom rules and program interactions. As mentioned before my WAF must be self learning. In order to do that, I have to analyse packets and store the results.
> My questions: 1) Is it smart to use the LUA API of modsecurity for analysing arguments (in post requests etc)? It's quite easy to get the arguments of a post request, but I'm worried about the limitations of this LUA API. The LUA api is experimental, so there is almost no documentation on limitations or possibilities. 2) What rights has the LUA api? Does a called LUA script run as the apache user? Will it able to write to the apache/modsecurity configuration files?

Lua scripts will run as the Apache user. On a well-configured server, only root can modify the configuration files and I wouldn't recommend anything else.

> 3) if your answer is "noön question 1, what else do you suggest?

You could write the whole thing in Lua. It's not necessary to produce ModSecurity rules for blocking – you can block from Lua itself.

The only issue there is information sharing, because Apache will typically use more than once process, even with threads. But because your primary goal is research, you could simply deploy with a single Apache process (and many threads) and everything would be in RAM.

For an alternative approach, consider using OpenResty, which is a combination of Nginx and LuaJit. As with Apache, configure Nginx with a single process to solve the problem of information sharing.

No matter what you choose, your entire project will be a single Lua program.

> Thanks in advance. (I bought your book (proof at the bottom of this mail) and I'm reading trough it.)
>
> Kind regards,
>
> Vincent Cox

FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN
TECHNOLOGIECAMPUS GEEL
Kleinhoefstraat 4
2440 GEEL, België
tel. + 32 14 80 22 40
iiw.geel@kuleuven.be
iiw.kuleuven.be