



Labra 7

Ryhmä 3

Sami Koivisto

Eino Puttonen

Jussi-Pekka Rantala

Markku Sutinen

Jukka Virtanen

Harjoitustyö

Huhtikuu 2024

Tieto- ja viestintätekniikan tutkinto-ohjelma (AMK)

Sisältö

1	Johdanto	4
2	Teoria.....	4
2.1	Web Application Firewall (WAF)	4
3	Toteutus	5
3.1	Setting up	5
3.2	Testing and hardening.....	8
4	Johtopäätökset.....	17
	Lähteet	19

Kuviot

Kuvio 1.	ZIP -tiedoston lataaminen	5
Kuvio 2.	mysql -asennus.....	6
Kuvio 3.	htaccess.....	6
Kuvio 4.	htaccess muokkaus	7
Kuvio 5.	Unzip & mysql + chmod -komennot	7
Kuvio 6.	annoy.js	8
Kuvio 7.	xss.php modifioiminen	9
Kuvio 8.	cookieien vastauksessa	9
Kuvio 9.	SQL-injektio kirjautumisessa	10
Kuvio 10.	modsecurity loki.....	10
Kuvio 11.	modsecurity paranoia	11
Kuvio 12.	login.php kovennuksen jälkeen.....	12
Kuvio 13.	xss.php kovennuksen jälkeen.....	12
Kuvio 14.	modsecurity loki koventamisen jälkeen.....	12
Kuvio 15.	hacked.php upload.....	13
Kuvio 16.	Liitoksen määrittely.....	14
Kuvio 17.	Luodaan oma sääntö, joka estää tiedoston lataamisen	15
Kuvio 18.	Todennus, että sääntö toimii	16
Kuvio 19.	Kiroilu onnistuu	16

Kuvio 20. Todennus että sääntö toimii	17
---	----

1 Johdanto

Seitsemännessä harjoitustyössä tutustuimme web-palveluiden suojaamiseen. Pääpainopiste lab-rassa on ModSecurity palomuuuri ja siihen tehtävät turvallisuussäännöt. Harjoituksen aikana suori-tamme erilaisia testejä ja katsomme miten säännöt purevat niitä vastaan.

2 Teoria

2.1 Web Application Firewall (WAF)

Web Application Firewall (WAF) eli suomeksi verkkosovelluksen palomuuuri on palomuuuri joka suo-jaa verkkosovelluksia ja API:ja monitoroimalla, suodattamalla ja estämällä haitallista verkkoliiken-nettä ja sovellustason hyökkäyksiä kuten sivustojen välinen komentosarjahyökkäys (XSS), evästei-den manipulointi, SQL-injektio ja hajautettu palvelunestohyökkäys (DDoS). OSI-mallin 7. kerroksessa eli sovelluskerroksessa toimiva verkkosovelluksen palomuuuri keskittyy liikenteeseen verkkosovellusten ja internetin välillä havaiten ja vastaten haitallisiin pyyntöihin ennen kuin verk-kosovellukset ja verkkopalvelimet hyväksyvät pyynnöt ja näin ollen tarjoaa yrityksille ja heidän asi-akkailleen olennaista turvaa. (What Is a WAF? | Web Application Firewall Explained 2024.)

Verkkosovelluksen palomuurit voidaan erotella niiden toimintatavan perusteella. Blocklist WAF:t ovat suunniteltu estämään tiettyjä päätepisteitä tai liikennetyyppejä ja sallimaan kaikki muut. Al-lowlist WAF:t toimivat joissain määrin päinvastoin Blocklist WAF:iin verrattuna estäen kaiken lii-kenteen oletusarvoisesti ja sallien vain nimenomaisesti hyväksytyn liikenteen. Allowlist WAF:ja pi-detään turvallisempina, koska ne minivoimat riskin siitä, että haitallinen liikenne väistää puolustuksen virheellisesti määritettyjen palomuurisääntöjen vuoksi. (What Is a WAF? | Web Ap-plication Firewall Explained 2024.)

Ne voidaan myös luokitella niiden käyttöönottomallin perusteella eli verkkopohjaiseen, isäntäpoh-jaiseen ja pilvipohjaiseen. Verkkopohjainen WAF toimii verkkoinfrastruktuureissa kuten kytkimillä, jotka sijaitsevat sovellusten ja internetin välissä. Isäntäpohjaiset WAF:t on sijoitettuna samoihin

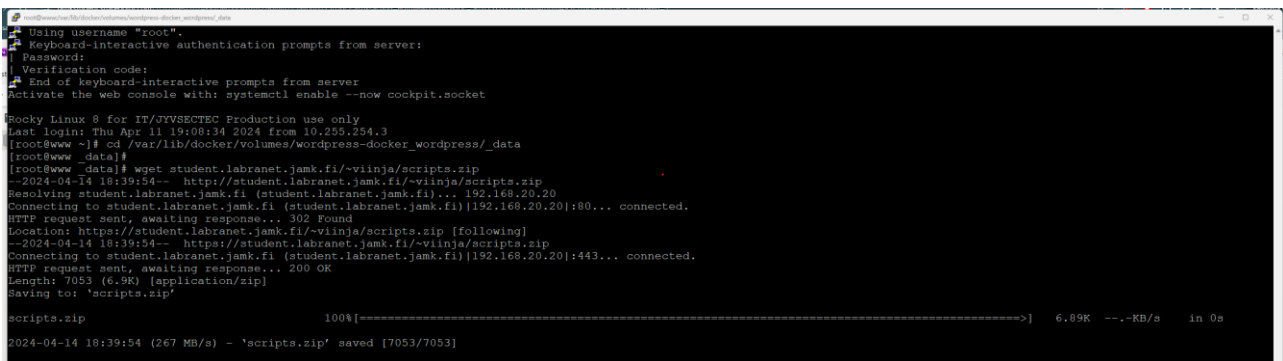
palvelimiin missä verkkosovellukset sijaitsevat. Ne käyttävät käyttöjärjestelmätason suodatusta suodattaakseen liikennettä, joka kulkee verkkosovelluksiin. Pilvipohjainen WAF suojelee pilvessä sijaitsevia sovelluksia. Se integroituu pilvessä oleviin virtuaalisiin verkkopalveluihin tai kuormantasaajiin suodattamaan verkkoliikennettä. (What Is a WAF? | Web Application Firewall Explained 2024.)

3 Toteutus

3.1 Setting up

Aivan ensimmäisenä otimme yhteyden kotikoneelta PuTTY:llä www -palvelimelle, jossa lähdimme tekemään seuraavia askelia. Ensimmäiseksi siirryimme kansioon missä wordpress sijaitsee, jonka jälkeen lataimme zip -tiedoston ohjeen mukaisesti (Ks. Kuvio 1)

- `cd /var/lib/docker/volumes/wordpress-docker_wordpress/_data`
- `wget student.labranet.jamk.fi/~viinja/scripts.zip`



```

Using username "root".
Keyboard-interactive authentication prompts from server:
Password:
Verification code:
End of keyboard-interactive prompts from server
Activate the web console with: systemctl enable --now cockpit.socket

Rocky Linux 8 for IT/JYVSECTEC Production use only
Last login: Thu Apr 11 19:08:34 2024 from 10.255.254.3
[root@www ~]# cd /var/lib/docker/volumes/wordpress-docker_wordpress/_data
[root@www _data]#
[root@www _data]# wget student.labranet.jamk.fi/~viinja/scripts.zip
--2024-04-14 18:39:54-- http://student.labranet.jamk.fi/~viinja/scripts.zip
Resolving student.labranet.jamk.fi (student.labranet.jamk.fi)... 192.168.20.20
Connecting to student.labranet.jamk.fi (student.labranet.jamk.fi)[192.168.20.20]:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://student.labranet.jamk.fi/~viinja/scripts.zip [following]
--2024-04-14 18:39:54-- https://student.labranet.jamk.fi/~viinja/scripts.zip
Connecting to student.labranet.jamk.fi (student.labranet.jamk.fi)[192.168.20.20]:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 7053 (6.9K) [application/zip]
Saving to: 'scripts.zip'

scripts.zip                               100%[=====] 6.89K --KB/s in 0s
2024-04-14 18:39:54 (267 MB/s) - 'scripts.zip' saved [7053/7053]
  
```

Kuvio 1. ZIP -tiedoston lataaminen

Tämän jälkeen asensimme mysql isäntäkoneeseen (Ks. Kuvio 2)

```
[root@www_data]# yum install mysql
Last metadata expiration check: 3:25:23 ago on Sun 14 Apr 2024 03:15:13 PM EEST.
Dependencies resolved.
Package Architecture Version Repository Size
Installing:
mysql x86_64 8.0.36-1.module+el8.9.0+1729+481e3b0b.0.1 appstream 14 M
Installing dependencies:
mariadb-connector-c-config noarch 3.1.11-2.el8_3 appstream 14 k
mysql-common x86_64 8.0.36-1.module+el8.9.0+1729+481e3b0b.0.1 appstream 137 k
Enabling module streams:
mysql 8.0
Transaction Summary
Install 3 Packages
Total download size: 15 M
Installed size: 73 M
Is this ok [y/N]: y
Downloading Packages:
(1/3): mariadb-connector-c-config-3.1.11-2.el8_3.noarch.rpm 115 kB/s | 14 kB 00:00
(2/3): mysql-common-8.0.36-1.module+el8.9.0+1729+481e3b0b.0.1.x86_64.rpm 869 kB/s | 137 kB 00:00
(3/3): mysql-8.0.36-1.module+el8.9.0+1729+481e3b0b.0.1.x86_64.rpm 31 MB/s | 14 MB 00:00
Total 22 MB/s | 15 MB 00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
Preparing : mariadb-connector-c-config-3.1.11-2.el8_3.noarch 1/1
Installing : mariadb-connector-c-config-3.1.11-2.el8_3.noarch 1/3
Installing : mysql-common-8.0.36-1.module+el8.9.0+1729+481e3b0b.0.1.x86_64 2/3
Installing : mysql-8.0.36-1.module+el8.9.0+1729+481e3b0b.0.1.x86_64 3/3
Running scriptlet: mysql-8.0.36-1.module+el8.9.0+1729+481e3b0b.0.1.x86_64 3/3
Verifying : mariadb-connector-c-config-3.1.11-2.el8_3.noarch 1/3
Verifying : mysql-8.0.36-1.module+el8.9.0+1729+481e3b0b.0.1.x86_64 2/3
Verifying : mysql-common-8.0.36-1.module+el8.9.0+1729+481e3b0b.0.1.x86_64 3/3
Installed:
mariadb-connector-c-config-3.1.11-2.el8_3.noarch mysql-8.0.36-1.module+el8.9.0+1729+481e3b0b.0.1.x86_64 mysql-common-8.0.36-1.module+el8.9.0+1729+481e3b0b.0.1.x86_64
Complete!
[root@www_data]#
```

Kuvio 2. mysql -asennus

Tästä menimme ohjeen mukaisesti editoimaan .htaccess tiedostoa, johon meidän piti lisätä aivan viimeiseksi "Options +indexes". (Ks. Kuvio 3 & Kuvio 4)

```
Complete!
[root@www_data]# nano .htaccess
[root@www_data]#
```

Kuvio 3. htaccess

```

GNU nano 2.9.8 .htaccess

# BEGIN WordPress
# The directives (lines) between "BEGIN WordPress" and "END WordPress" are
# dynamically generated, and should only be modified via WordPress filters.
# Any changes to the directives between these markers will be overwritten.

# END WordPress
Options +indexes

```

Kuvio 4. htaccess muokkaus

Tiedoston muokkaamisen jälkeen ohjeessa sanottiin unzipata scripts.zip tiedosto, jonka jälkeen meidän piti lisätä tietoa tietokantaan, joka tapahtui skriptit -kansiossa, jossa ajettiin seuraava mysql -komento → "mysql -h localhost -P 3306 --protocol=tcp -u root -proot66 < logintest.sql". Tämän lisäksi meidän piti vielä antaa oikeudet, että voimme ladata tiedostoja upload -kansioon. Tämä tapahtui chmod 777 -komennolla, ja pääsimme käsiksi 10.4.0.11/scripts -sivustoon, josta seuraavissa kappaleissa lisää. (Ks. Kuvio 5)

```

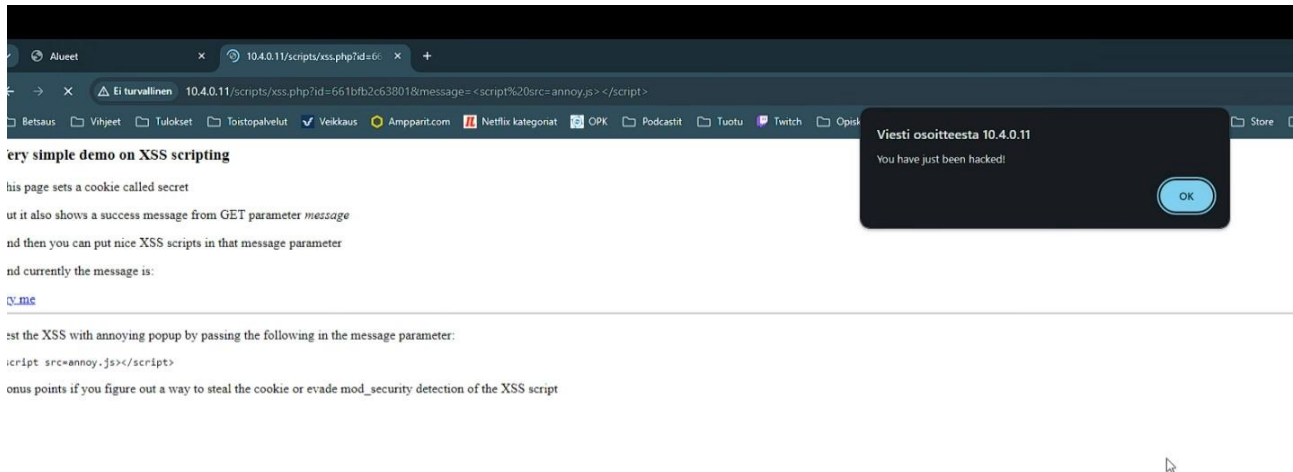
[root@www_data]# unzip scripts.zip
Archive:  scripts.zip
  creating: scripts/
  inflating: scripts/upload.php
  inflating: scripts/logintest.sql
  inflating: scripts/db-connect-test.php
  inflating: scripts/xss.php
  inflating: scripts/annoy.js
  creating: scripts/uploads/
  inflating: scripts/uploads/installation_status.json
  inflating: scripts/login.php
  inflating: scripts/testecho.php
  creating: scripts/swear/
  inflating: scripts/swear/testecho.php
[root@www_data]# cd scripts/
[root@www_scripts]# ls -la
total 32
drwxr-xr-x. 4 root root 168 Nov 29 2022 .
drwxr-xr-x. 6 33 tape 4096 Apr 14 18:44 ..
-rw-r--r--. 1 root root 133 Nov 29 2022 annoy.js
-rw-r--r--. 1 root root 619 Nov 29 2022 db-connect-test.php
-rw-r--r--. 1 root root 1257 Nov 29 2022 login.php
-rw-r--r--. 1 root root 409 Nov 29 2022 logintest.sql
drwxr-xr-x. 2 root root 26 Nov 29 2022 swear
-rw-r--r--. 1 root root 405 Nov 29 2022 testecho.php
-rw-r--r--. 1 root root 597 Nov 29 2022 upload.php
drwxr-xr-x. 2 root root 30 Nov 29 2022 uploads
-rw-r--r--. 1 root root 783 Nov 29 2022 xss.php
[root@www_scripts]# mysql -h localhost -P 3306 --protocol=tcp -u root -proot66 < logintest.sql
mysql: [Warning] Using a password on the command line interface can be insecure.
[root@www_scripts]# chmod 777 uploads
[root@www_scripts]#

```

Kuvio 5. Unzip & mysql + chmod -komennot

3.2 Testing and hardening

Aloitimme testaamalla xss.php -skriptiä. Ohjeistuksen mukaisesti annoimme message-parametrille arvoksi annoy.js -skriptin ja totesimme että se tuli ajetuksi (kts. Kuvio 6).



Kuvio 6. annoy.js

Bonustehtävänä tutkimme cookien varastamista. Alkuun huomasimme ettei xss.php vaikuttanut asettavan lainkaan cookieta. Tämä todettiin selaimen developer-työkaluilla sekä tutkimalla wget-komennoilla requestille palaavan responsen headereita. Tutkittiin xss.php-skriptiä ja kokeiltiin siirtää cookien asettaminen aivan skriptin alkuun, ennen kuin muuta dokumenttia kirjoitetaan ulos (Kts. Kuvio 7).

```
GNU nano 2.9.8 xss.php
<?php
$cookie_id = $_GET['id'];
setcookie("secret",$cookie_id);

echo "<h3>Very simple demo on XSS scripting</h3>\n";
echo "<p>This page sets a cookie called secret</p>\n";

echo "<p>But it also shows a success message from GET parameter <i>message</i></p>\n";
echo "<p>And then you can put nice XSS scripts in that message parameter</p>\n";

$message = $_GET['message'];
```


Kuvio 7. xss.php modifioiminen

Muutoksen jälkeen saatiin cookie responseen mukaan ja päästiin tutkimaan sen varastamista (kts. Kuvio 8).

```
[root@www scripts]# wget -S -O - http://localhost/scripts/xss.php
--2024-04-21 15:46:28-- http://localhost/scripts/xss.php
Resolving localhost (localhost)... ::1, 127.0.0.1
Connecting to localhost (localhost)|::1|:80... connected.
HTTP request sent, awaiting response...
HTTP/1.1 200 OK
Date: Sun, 21 Apr 2024 12:46:28 GMT
Server: Apache/2.4.53 (Debian)
X-Powered-By: PHP/7.4.30
Vary: Accept-Encoding
Content-Length: 605
Content-Type: text/html; charset=UTF-8
Set-Cookie: secret=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; Max-Age=0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Length: 605 [text/html]
Saving to: 'STDOUT'
```

Kuvio 8. cookien vastauksessa

Tehtiin annoy.js -skriptistä versio joka lisää popupiin myös cookien arvon. Tämä annettiin xss.php-skriptille messages-parametrin arvona, kuten alkuperäinen annoy.js. Tuloksena saimme cookien arvon näkymään popup-ikkunassa. Todellisiin hakkerointitarpeisiin tulisi skripti saada vielä toimitamaan arvo jollain sopivalla menetelmällä hakkerille, mutta sitä emme lähteneet enää kehittämään.

Seuraavaksi siirryimme testaamaan SQL-injektiota ja virheellistä login-syötettä. Kokeilimme tehtävänannon salasanalla useita erilaisia skenaarioita ja totesimme että riippumatta tunnuksesta, pääsemme kirjautumaan onnistuneesti (kts. Kuvio 9).


```

root@www:~/wordpress-docker
GNU nano 2.9.8                                docker-compose.yml
---
version: '3'
services:
  modsecurity:
    container_name: modsecurity
    image: owasp/modsecurity-crs:apache-alpine
    environment:
      PROXY: 1
      BACKEND: http://wordpress
      PORT: "8080"
      SSL_PORT: "443"
      METRICS_ALLOW_FROM: All
      PARANOIA: 4
      ANOMALY_INBOUND: 20
      PROXY_SSL_CERT: /etc/letsencrypt/live/www.group3.ttc60z.vle.fi/fullchain.pem
      PROXY_SSL_CERT_KEY: /etc/letsencrypt/live/www.group3.ttc60z.vle.fi/privkey.pem
    volumes:
      - /etc/letsencrypt:/etc/letsencrypt/

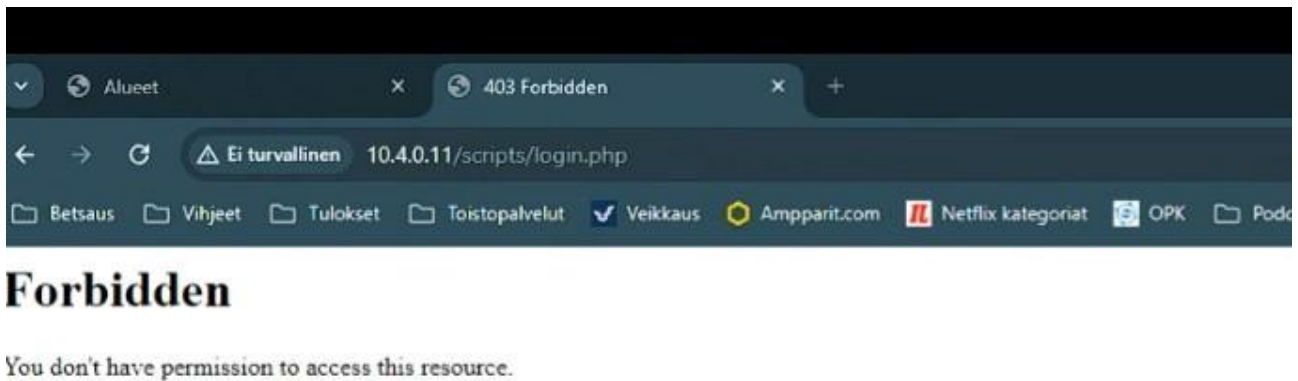
    ports:
      - "80:8080"
      - "8443:443"
    depends_on:
      - wordpress

  wordpress:
    container_name: wordpress
    image: custom/wordpress
    build: .
    env_file: .env
    volumes:
      - wordpress:/var/www/html
    ports:
      - "8080:80"
    depends_on:
      - db

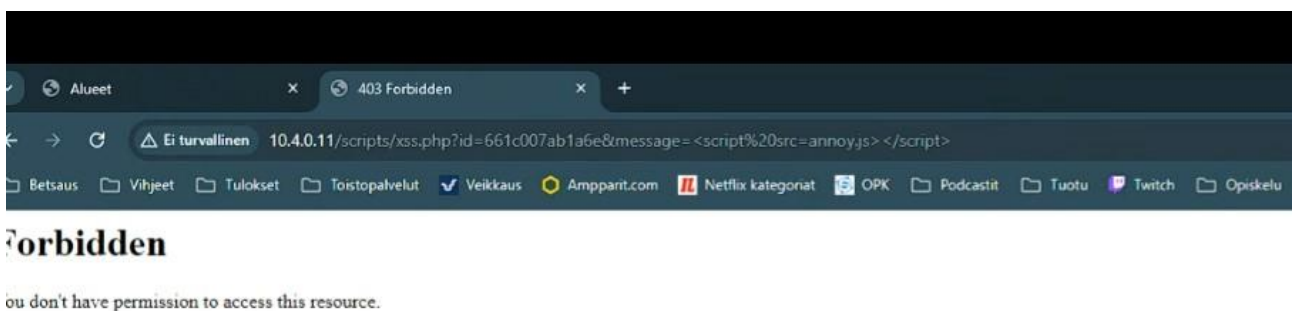
```

Kuvio 11. modsecurity paranoia

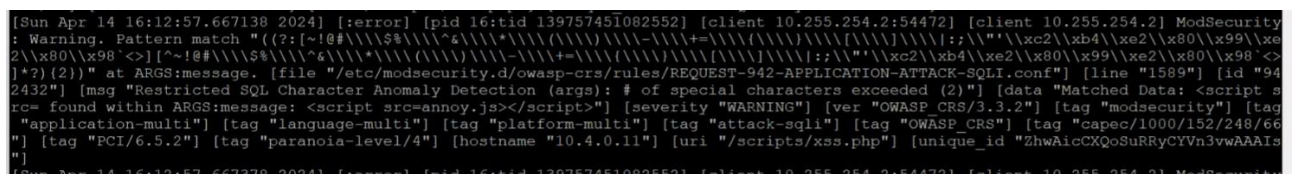
Koventamisen jälkeen ei skriptien ajaminen onnistunut enää aiemmalla tavalla (kts Kuvio 12 ja Kuvio 13). Samoin modsecurityn lokista voidaan todeta koventamisen vaikutus (kts. Kuvio 14).



Kuvio 12. login.php kovenituksen jälkeen

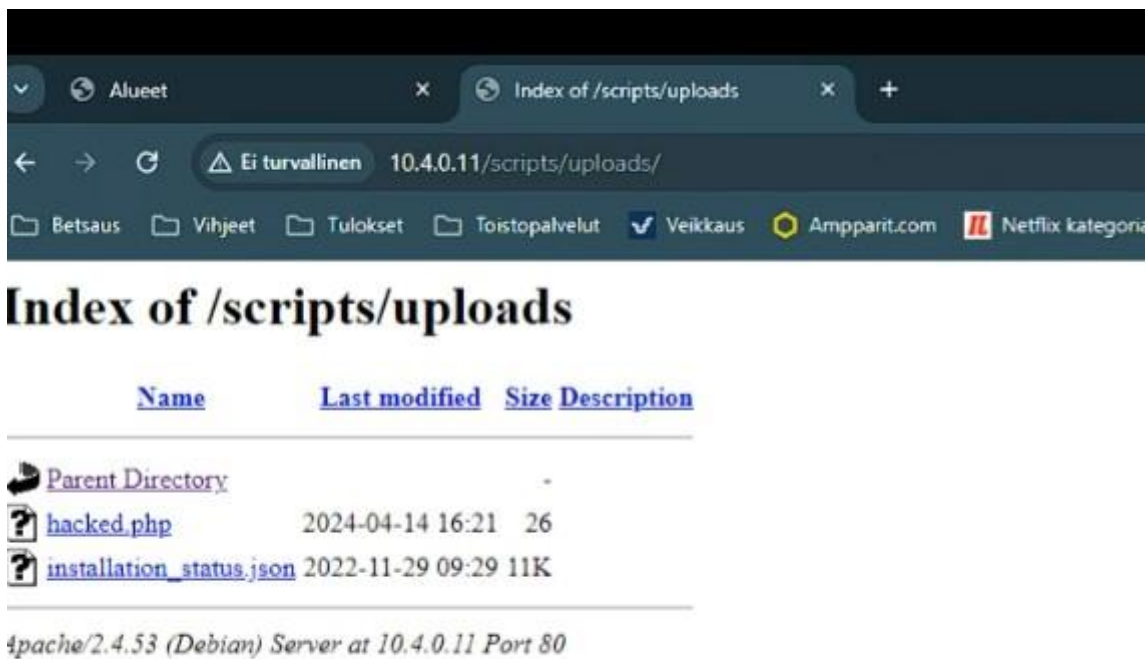


Kuvio 13. xss.php kovenituksen jälkeen



Kuvio 14. modsecurity loki koventamisen jälkeen

Seuraavaksi toteutimme hacked.php -skripti lataamisen koneelle onnistuneesti (kts. Kuvio 15).



Kuvio 15. hacked.php upload

Seuraavaksi pyrimme estämään ettei .php päätteistä tiedostoa pystytä suorittamaan. Tämä tapahtuu luomalla host- koneelle modsecuritylle sääntö, joka tämän estää. Tehdään kansio ownrules komennolla "mkdir /ownrules". Tämän jälkeen kopioidaan nykyiset säännöt modsecurity Docker-kontista paikalliseen tiedostojärjestelmän komennolla "docker cp modsecurity:/etc/modsecurity.d/owasp-crs/rules /ownrules/". Seuraavaksi päivitetään Docker-compose.yml tiedoston konfiguraatiota lisäämällä uusi liitos (mount) -sääntö (ks. Kuvio 16). Tällä liitoksella määritetään liittämisen paikallisesta kansioista (/ownrules/rules/) ModSecurity sääntöjen hakemistoon (/etc/modsecurity.d/owasp-crs/rules/) Docker-ympäristössä. Kun mounttaus on määritetty Docker-compose tiedostossa, se tarkoittaa sitä, että hakemisto, joka sisältää omat ModSecurity-säännöt liitetään ModSecurity sääntöjen hakemistoon ja tämä mahdollistaa omien sääntöjen käyttämisen ModSecurityn kanssa ja niiden käyttämisen verkkosivustojen suojaamiseen.

```

---
version: '3'
services:
  modsecurity:
    container_name: modsecurity
    image: owasp/modsecurity-crs:apache-alpine
    environment:
      PROXY: 1
      BACKEND: http://wordpress
      PORT: "8080"
      SSL_PORT: "443"
      METRICS_ALLOW_FROM: All
      PARANOIA: 4
      ANOMALY_INBOUND: 20
      PROXY_SSL_CERT: /etc/letsencrypt/live/www.group3.ttc60z.vle.fi/fullchain.pem
      PROXY_SSL_CERT_KEY: /etc/letsencrypt/live/www.group3.ttc60z.vle.fi/privkey.pem
    volumes:
      - /etc/letsencrypt/live/www.group3.ttc60z.vle.fi/fullchain.pem:/etc/letsencrypt/live/www.group3.ttc60z.vle.fi/fullchain.pem
      - /ownrules/rules:/etc/modsecurity.d/owasp-crs/rules/
    ports:
      - "80:8080"
      - "8443:443"
    depends_on:
      - wordpress

  wordpress:
    container_name: wordpress
    image: custom/wordpress
    build: .
    env_file: .env
    volumes:
      - wordpress:/var/www/html
    ports:
      - "8080:80"
    depends_on:
      - db

  db:

```

Kuvio 16. Liitoksen määrittely

Päivitetään Docker komponentit, jotka on siis määritelty edellä mainitussa tiedostossa komennolla "docker compose up -d". Up- komento luo ja käynnistää tiedostossa määritellyt palvelut. -d valitsin käynnistää palvelut taustalla.

Tämän jälkeen luomme oman säännön (ks. Kuvio 17), joka määrittää polkuun /ownrules/rules/MyCustomRule.conf. Säännön selitys:

ModSecurity-sääntö on jaettu kahteen osaan ja se suoritetaan, kun http-pyyynnön tiedostonimi vastaa ehtoja. Tämä tarkastaa, että http-pyyynnön REQUEST_FILENAME -kenttä sisältää merkkijonon upload.php. Jos sääntö täyttyy, sääntö suorittaa seuraavat toimet:

- id: 5000: Tunnistaa säännön numerolla 50000
- chain: Merkitsee, että sääntö on osa sääntöketjua ja että se liitetään seuraavaan sääntöön

- deny: kieltää pyynnön
- log: kirjaa viestin lokiin
- msg: Määrittää viestin, joka kirjoitetaan lokiin, jos sääntö laukeaa

Toinen osa. SecRule FILES @rx .*\.php\$. Tämä osa tarkistaa, että http-pyyntöön tiedostot vastaavat säännöllistä lauseketta (regex) eli .php\$. Tämä tarkoittaa, että tiedostonimen lopussa on .php. Tämä toinen osa ei tee mitään, mutta se liitetään osaksi ensimmäistä osaa, jolloin molempien ehtojen täytyy täyttyä ennen kuin sääntöketju laukeaa.

Yhteensä nämä kaksi sääntöä estävät PHP-tiedoston lataamisen, jos http-pyyntöön tiedostonimi on upload.php



```
root@wordpress-docker:~# nano /ownrules/rules/MyCustomRule.conf
GNU nano 2.9.8 /ownrules/rules/MyCustomRule.conf

SecRule REQUEST_FILENAME "upload.php" "id:'50000',chain,deny,log,msg:'Tried to upload a PHP file'"
SecRule FILES "@rx .*\.php$" [REDACTED]
```

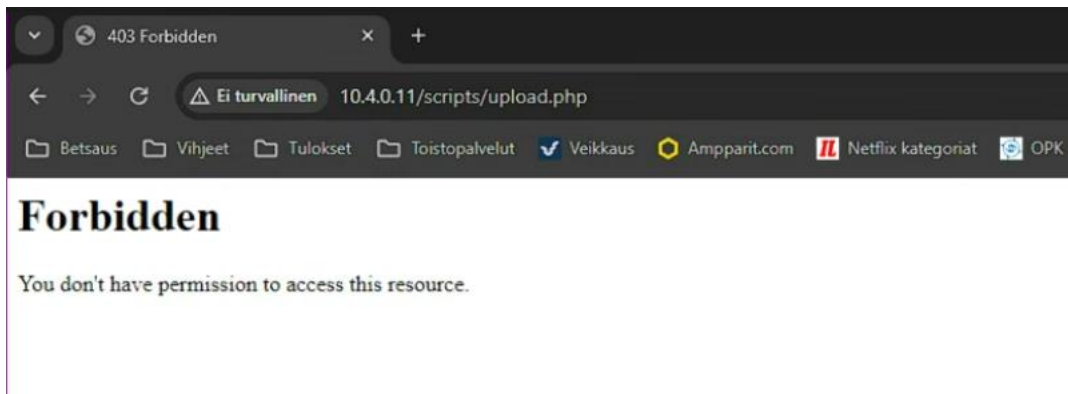
Kuvio 17. Luodaan oma sääntö, joka estää tiedoston lataamisen

Tämän jälkeen vielä suoritetaan seuraavat komennot ennen testaamista.

docker stop

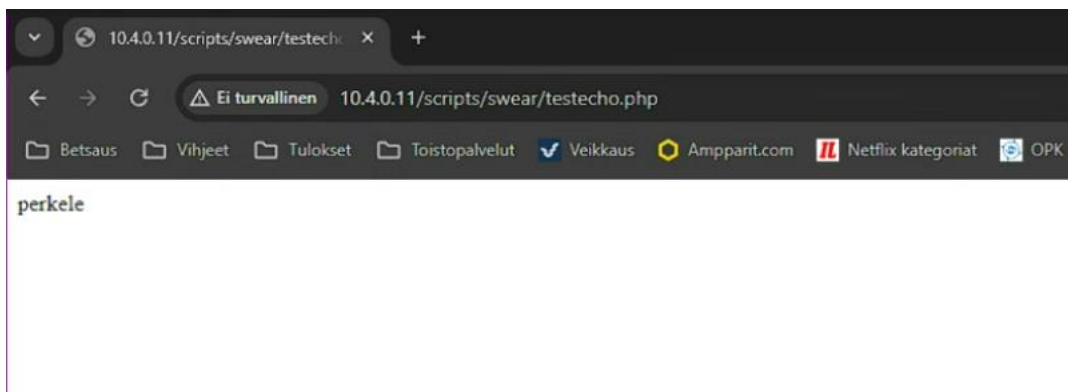
docker start

Suoritetaan testi ja todetaan, että sääntö toimii (ks. Kuvio 18).



Kuvio 18. Todennus, että sääntö toimii

Seuraavaksi estetään huonon kielen käyttäminen. Esimerkiksi kirjoilun. Testataan ensimmäiseksi meneekö kiroilu läpi ennen säännön luomista (ks. Kuvio 19).



Kuvio 19. Kiroilu onnistuu

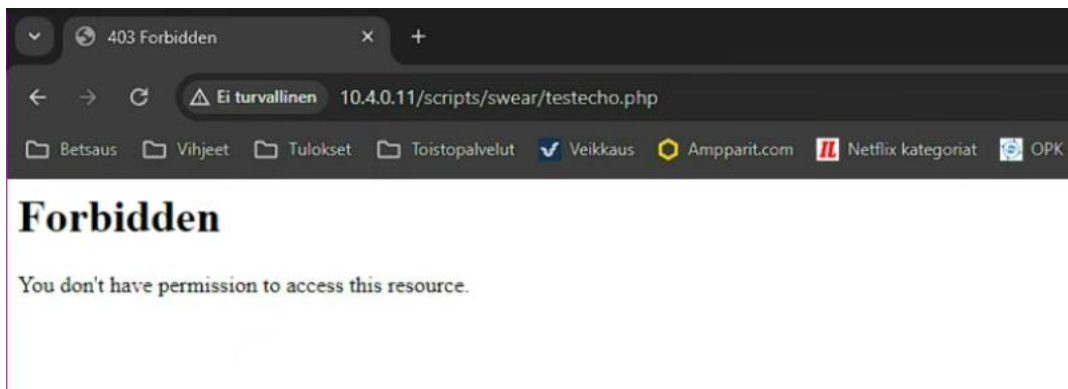
Luodaan uusi sääntö `/ownrules/rules/MyCustomRule2.conf` ja laitetaan sinne seuraava sääntö:

```
SecRule REQUEST_FILENAME "testecho.php" "id:'50001',chain,deny,log,msg:'SWEARING'"
```

```
SecRule REQUEST_BODY "@rx (?i:(perkele|rolex))"
```


Tämä lauseke etsii ilmaisut perkele ja rolex pyynnön rungosta ja ?i -osuus merkitsee, että haku on tehty kirjainkoosta riippuvaksi. Jos ilmaisut löytyvät, sääntö laukeaa ja se kieltää pyynnön. Tämä voi auttaa siihen, että verkkosivustolle ei pysty lisäämään epäasiallista sisältöä.

Todennetaan, että toimii (ks. Kuvio 20).



Kuvio 20. Todennus että sääntö toimii

Selitä mitä ”?i” tarkoittaa?

Vastaus: Sisältää osuuden, joka tekee haun kirjainkoosta riippuvaksi. Tämä siis tarkoittaa, että vaikka ilmaisu olisi kirjoitettu pienellä tai isolla kirjaimella, se vastaa silti hakuun.

4 Johtopäätökset

WAF-tyylinen työkalu todisti hyödyllisyytensä tässä harjoitustyössä. Usein webbikehityksessä, kuten muussakin ohjelmistokehityksessä, on ongelmana puutteellinen näkemys tai ymmärrys kyberturvallisuuden näkökulmasta. Kehittäjän tavoitteena on luoda tarvittavaa toiminnallisuutta, ja oheisvaatimukset kuten hyvät kyberturvallisuuskäytännöt saattavat unohtua tai niitä ei muisteta/tunneta. WAF toimii hyvänä varmistuksena tällaisissa tilanteissa.

Toinen mieleen tullut samansuuntainen seikka liittyy erityisesti webbiservereihin kuten Apacheen. Sen konfigurointimahdollisuudet ovat sen pitkän historian varrella laajentuneet melkoisesti. Monimutkaisuuden lisääntyessä virheiden mahdollisuus kasvaa. Jälleen voidaan ajatella WAFin puolustavan paikkaansa toimiessaan ylimääräisenä suojakerroksena mahdollisten virhekonfiguraatioiden tai huonosti harkittujen oletusarvojen varalta.

Lähteet

What Is a WAF? | Web Application Firewall Explained. 2024. Viitattu 19.4.2024. <https://www.paloaltonetworks.com/cyberpedia/what-is-a-web-application-firewall>