

TP IDS

CYBERSÉCURITÉ

Mohamed BOUCHENGUOUR
POLYTECH UNICE
MASTER 2 CYBERSÉCURITÉ



Table des matières

Étape 1 : Mise en place du pare-feu.....	2
Étape 2 : Mise en place d'un reverse proxy et du chiffrement TLS	5
Étape 3 : Intégration du Web Application Firewall	10
Étape 4 : Mise en place du système de détection d'intrusion	14
Étape 5 : Mise en place d'un Système de Prévention d'intrusion	17
Détection des connexions SSH non autorisées.....	22
Détection de scans XMAS	24
Détection de l'accès au fichier acquisitions.md	26
Prévention de l'accès à la page "about"	28
Détection des attaques de type DoS	30

Note : J'ai réalisé ce TP sur 3 machines différentes (mon pc fixe, mon pc portable et le pc fixe chez mes parents) car j'étais chez mes parents durant les vacances. Certaines fois, l'ip de la vm change entre 192.168.56.12 et 192.168.56.4.

Étape 1 : Mise en place du pare-feu

La première étape consiste à configurer les règles de base pour sécuriser les communications entrantes et sortantes sur la machine. L'objectif est de bloquer par défaut tout trafic non autorisé afin de ne permettre que les connexions essentielles et nécessaires.

Définition des politiques par défaut

Les politiques par défaut pour les chaînes INPUT, OUTPUT, et FORWARD ont été configurées à DROP. Cela garantit que tout paquet non explicitement autorisé est bloqué :

```
sudo iptables -P INPUT DROP
sudo iptables -P OUTPUT DROP
sudo iptables -P FORWARD DROP
```

Autorisation des connexions SSH avec la machine hôte

Pour permettre les connexions SSH entrantes depuis la machine hôte (adresse 192.168.56.1) vers la machine virtuelle, les règles suivantes ont été définies :

```
sudo iptables -t filter -A INPUT -p tcp -s 192.168.56.1 --dport 22 -j ACCEPT
```

```
sudo iptables -t filter -A OUTPUT -p tcp --sport 22 -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
```

Autorisation des requêtes DNS

Pour garantir la résolution correcte des adresses réseau (via DNS), les règles suivantes ont été mises en place :

```
sudo iptables -t filter -A OUTPUT -p udp --dport 53 -j ACCEPT
sudo iptables -t filter -A INPUT -p udp --sport 53 -j ACCEPT
```

Autorisation du trafic ICMP

Les règles suivantes ont été configurées pour autoriser le trafic ICMP, permettant des diagnostics réseau, tels que les pings :

```
sudo iptables -t filter -A OUTPUT -p icmp -j ACCEPT
sudo iptables -t filter -A INPUT -p icmp -j ACCEPT
```

Autorisation du trafic DHCP

Pour les configurations réseau utilisant le protocole DHCP, les règles suivantes ont été mises en place pour permettre la communication :

```
sudo iptables -t filter -A OUTPUT -p udp --dport 67:68 -j ACCEPT
```

```
sudo iptables -t filter -A INPUT -p udp --dport 67:68 --sport 67:68 -j ACCEPT
```

Autorisation du trafic Loopback

Pour garantir le bon fonctionnement des services locaux, les connexions sur l'interface loopback ont été autorisées :

```
sudo iptables -t filter -A INPUT -i lo -j ACCEPT
sudo iptables -t filter -A OUTPUT -o lo -j ACCEPT
```

Autorisation du trafic HTTP et HTTPS

Pour permettre l'accès à Internet (sites web), les règles suivantes ont été ajoutées pour les connexions HTTP et HTTPS :

- Pour HTTP (port 80) :

```
sudo iptables -t filter -A OUTPUT -p tcp --dport 80 -j ACCEPT
sudo iptables -t filter -A INPUT -p tcp --sport 80 -m conntrack --ctstate
RELATED,ESTABLISHED -j ACCEPT
```

- Pour HTTPS (port 443) :

```
sudo iptables -t filter -A OUTPUT -p tcp --dport 443 -j ACCEPT
sudo iptables -t filter -A INPUT -p tcp --sport 443 -m conntrack --ctstate
RELATED,ESTABLISHED -j ACCEPT
```

Ces règles permettent à la machine de se connecter à des sites via HTTPS, en autorisant uniquement le trafic entrant lié aux connexions établies.

Enregistrement des règles iptables

Pour garantir que les règles définies sont persistantes après un redémarrage, elles ont été sauvegardées :

```
sudo iptables-save -f /etc/iptables/rules.v4
```

Logging des paquets avec iptables

Pour surveiller les connexions, les paquets traversant la table FILTER peuvent être logués. Cette fonctionnalité a été activée et les requêtes sont enregistrées dans /var/log/syslog grâce à la règle suivante :

```
sudo iptables -t filter -I INPUT 1 -m state --state NEW -j LOG --log-prefix "[IPTABLES]: " --log-ip-options
```

Exemple de test avec une connexion SSH

Lors de la création d'une connexion SSH, il est possible de suivre les logs pour observer les tentatives de connexion et leur traitement par iptables. La commande suivante permet de surveiller les logs en temps réel :

```
sudo tail -f /var/log/syslog
```

Exemple de log observé :

```
Oct 22 06:41:54 ids-polytech kernel: [IPTABLES]: IN=enp0s8 OUT=  
MAC=08:00:27:91:d7:a0:0a:00:27:00:00:0b:08:00 SRC=192.168.56.1 DST=192.168.56.4  
LEN=52 TOS=0x00 PREC=0x00 TTL=128 ID=14684 DF PROTO=TCP SPT=24379  
DPT=22 WINDOW=64240 RES=0x00 SYN URGP=0
```

Explication : Ce log montre une tentative de connexion SSH entrante détectée par iptables. L'adresse source est 192.168.56.1 (la machine hôte) et la destination est 192.168.56.4 (la machine virtuelle), sur le port 22 (port standard pour SSH). Le paquet est un paquet TCP avec le flag SYN activé, indiquant qu'il s'agit de l'initiation d'une nouvelle connexion. Le log contient également d'autres détails tels que l'interface réseau utilisée (**enp0s8**), la longueur du paquet (**LEN=52**), et d'autres paramètres techniques (**TTL**, **TOS**, etc.) liés à l'état du paquet au moment de la connexion.

Étape 2 : Mise en place d'un reverse proxy et du chiffrement TLS

Le conteneur Juice Shop exposait un service à protéger, accessible au port 3000. Pour permettre l'accès à ce service, les règles iptables suivantes ont été appliquées afin de sécuriser le trafic entrant et sortant lié à ce port :

```
sudo iptables -t filter -A INPUT -p tcp --dport 3000 -j ACCEPT
```

```
sudo iptables -t filter -A OUTPUT -p tcp --sport 3000 -m conntrack --ctstate  
RELATED,ESTABLISHED -j ACCEPT
```

Ces règles permettent les connexions entrantes vers le service sur le port 3000 et autorisent les réponses aux connexions établies, garantissant une communication sécurisée.

Création de l'autorité de certification (CA)

Pour garantir un chiffrement TLS sécurisé, une autorité de certification (CA) interne a été créée à l'aide de l'outil easy-rsa.

Initialisation de l'infrastructure PKI :

```
make-cadir certifications
```

```
cd certifications
```

```
./easyrsa init-pki
```

```
./easyrsa build-ca
```

make-cadir certifications crée un répertoire contenant les fichiers nécessaires pour gérer les certificats.

./easyrsa init-pki initialise la structure PKI requise.

./easyrsa build-ca génère la clé privée et le certificat de la CA. Un mot de passe (poly) a été défini pour protéger cette clé.

Création et signature d'un certificat pour le site :

```
./easyrsa gen-req serveur.fr nopass
```

```
./easyrsa sign-req server serveur.fr
```

La première commande génère une demande de certificat (CSR) pour serveur.fr sans protection par mot de passe. La deuxième commande signe la CSR à l'aide de la CA interne, validant ainsi le certificat pour le domaine spécifié. Cela permet au serveur d'établir des connexions sécurisées reconnues par la CA interne.

Informations trouvées dans le certificat :

- **Numéro de série** : 0c:37:87:12:19:39:07:3f:aa:0c:2f:f3:ae:d7:08:86 - Permet d'identifier et de tracer chaque certificat individuellement, utile pour la gestion de la sécurité.
- **Émetteur** : CN=ca-BOUCHENGUOUR.org - Confirme l'autorité qui a émis le certificat, validant son authenticité.
- **Période de validité** : 27 Oct 2024 à 30 Jan 2027 - Indique la durée pendant laquelle le certificat peut être utilisé en toute sécurité.
- **Algorithme de signature** : sha256WithRSAEncryption - Garantit une signature robuste pour éviter toute altération du certificat.
- **Utilisation des clés** : TLS Web Server Authentication - Spécifie que le certificat est utilisé pour l'authentification des serveurs web.
- **Nom commun (CN)** : serveur.fr - Associe le certificat au domaine sécurisé par celui-ci.

Le nom commun (CN) serveur.fr signifie que ce certificat est destiné à sécuriser les connexions pour le domaine serveur.fr.

Mise en place du reverse proxy Apache

Pour rediriger et sécuriser le trafic web, un reverse proxy Apache avec TLS a été configuré.

Activation des modules Apache nécessaires :

```
sudo a2enmod ssl
sudo a2enmod proxy proxy_http
```

- ssl est activé pour permettre le chiffrement des connexions.
- proxy et proxy_http permettent la redirection du trafic web.

Création du répertoire pour les certificats :

```
sudo mkdir /etc/apache2/ssl
```

Ce répertoire contient les fichiers nécessaires pour la configuration SSL.

Copie des certificats et de la clé privée :

```
sudo cp certifications/pki/issued/serveur.fr.crt /etc/apache2/ssl/
sudo cp certifications/pki/private/serveur.fr.key /etc/apache2/ssl/
```

Les fichiers générés sont déplacés pour permettre leur utilisation dans la configuration Apache.

Configuration du site Apache : Le fichier /etc/apache2/sites-available/serveur.fr.conf a été configuré comme suit :

```

<VirtualHost *:8000>
    SSLEngine on
    SSLCertificateFile /etc/apache2/ssl/serveur.fr.crt
    SSLCertificateKeyFile /etc/apache2/ssl/serveur.fr.key
    <Location />
        ProxyPass http://127.0.0.1:3000/
        ProxyPassReverse http://127.0.0.1:3000/
    </Location>
    <Location /premium>
        ProxyPass http://127.0.0.1:10000/
        ProxyPassReverse http://127.0.0.1:10000/
    </Location>
    ErrorLog /var/log/apache2/error.example.com.log
    CustomLog /var/log/apache2/access.example.com.log combined
</VirtualHost>

```

- La directive 'SSLEngine on' active le chiffrement SSL/TLS.
- Les directives 'ProxyPass' et 'ProxyPassReverse' redirigent le trafic vers les services internes.

Activation du site configuré :

```
sudo a2ensite serveur.fr.conf
```

Ajout du port d'écoute 8000 dans le fichier de configuration Apache :

Un port d'écoute supplémentaire (8000) a été ajouté au fichier /etc/apache2/ports.conf pour correspondre à la configuration du VirtualHost définie précédemment, permettant au serveur Apache d'accepter les connexions entrantes sur ce port. Cela garantit que les requêtes dirigées vers le port 8000 sont correctement redirigées via le reverse proxy configuré.



```

GNU nano 7.2 /etc/apache2/ports.conf *
# If you just change the port or add more ports here, you will likely also
# have to change the VirtualHost statement in
# /etc/apache2/sites-enabled/000-default.conf

Listen 80
Listen 8000

<IfModule ssl_module>
    Listen 443
</IfModule>

<IfModule mod_gnutls.c>
    Listen 443
</IfModule>

```

Redémarrage du service Apache pour appliquer les modifications :

```
sudo systemctl restart apache2
```

Configuration de l'accès au domaine et test du reverse proxy

Ajout du nom de domaine serveur.fr au fichier /etc/hosts pour correspondre à l'IP 192.168.56.4 :

```
sudo nano /etc/hosts
```



```

GNU nano 7.2 /etc/hosts *
127.0.0.1 localhost
127.0.1.1 ids-polytech
192.168.56.4 serveur.fr
# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

```

Cette étape garantit que les requêtes vers serveur.fr pointent vers l'IP locale configurée, permettant de tester les communications via le domaine spécifié.

Test d'accès sécurisé avec curl : Depuis la machine hôte, la commande suivante a été exécutée :

`curl -v https://serveur.fr:8000/premium`

```

poly@ids-polytech:~$ curl -v https://serveur.fr:8000/premium
* Host serveur.fr:8000 was resolved.
* IPv6: (none)
* IPv4: 192.168.56.12
* Trying 192.168.56.12:8000...
* Connected to serveur.fr (192.168.56.12) port 8000
* ALPN: curl offers h2,http/1.1
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* CAfile: /etc/ssl/certs/ca-certificates.crt
* CAPath: /etc/ssl/certs
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):
* TLSv1.3 (IN), TLS handshake, Certificate (11):
* TLSv1.3 (OUT), TLS alert, unknown CA (560):
* SSL certificate problem: unable to get local issuer certificate
* Closing connection
curl: (60) SSL certificate problem: unable to get local issuer certificate
More details here: https://curl.se/docs/sslcerts.html

curl failed to verify the legitimacy of the server and therefore could not
establish a secure connection to it. To learn more about this situation and
how to fix it, please visit the web page mentioned above.

```

Une erreur indiquant *SSL certificate problem: unable to get local issuer certificate* est apparue. Cela est dû au fait que le certificat de l'autorité de certification n'était pas reconnu par le système local.

Ajout de l'autorité de certification au store local :

```

sudo cp certifications/pki/ca.crt /usr/local/share/ca-certificates/
sudo update-ca-certificates

```

Cela a permis de reconnaître le certificat et d'établir une connexion sécurisée avec le serveur.

Vérification de l'accès sécurisé

Une nouvelle tentative avec curl :

`curl -v https://serveur.fr:8000/premium`

```

poly@ide-polytech:~$ curl -v https://serveur.fr:8000/premium
* Host serveur.fr:8000 was resolved.
* IPv6: (none)
* IPV4: 192.168.56.12
* Trying 192.168.56.12:8000...
* Connected to serveur.fr (192.168.56.12) port 8000
* ALPN: curl offers h2,http/1.1
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* CAfile: /etc/ssl/certs/ca-certificates.crt
* CApath: /etc/ssl/certs
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):
* TLSv1.3 (IN), TLS handshake, Certificate (11):
* TLSv1.3 (IN), TLS handshake, CERT verify (15):
* TLSv1.3 (IN), TLS handshake, Finished (20):
* TLSv1.3 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.3 (OUT), TLS handshake, Finished (20):
* SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384 / X25519 / RSASSA-PSS
* ALPN: server accepted http/1.1
* Server certificate:
* subject: CN=serveur.fr
* start date: Oct 27 13:01:18 2024 GMT
* expire date: Jan 30 13:01:18 2027 GMT
* subjectAltName: host "serveur.fr" matched cert's "serveur.fr"
* issuer: CN=ca-BOUCHENGOUR.org
* SSL certificate verify ok:
* Certificate level 0: Public key type RSA (2048/112 Bits/secBits), signed using sha256withRSAEncryption
* Certificate level 1: Public key type RSA (2048/112 Bits/secBits), signed using sha256withRSAEncryption
* using HTTP/1.x
> GET /premium HTTP/1.1
> Host: serveur.fr:8000
> User-Agent: curl/8.5.0
> Accept: */*
*
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
* old SSL session ID is stale, removing
< HTTP/1.1 200 OK
< Date: Sun, 27 Oct 2024 13:05:12 GMT
< Server: Apache/2.4.58 (Ubuntu)
< Content-Length: 71
*
* Connection 80 to host serveur.fr left intact
<html>=<title>SS DEMO</title>=</head>=<body>=</body>=</html>poly@ide-polytech:~$ |

```

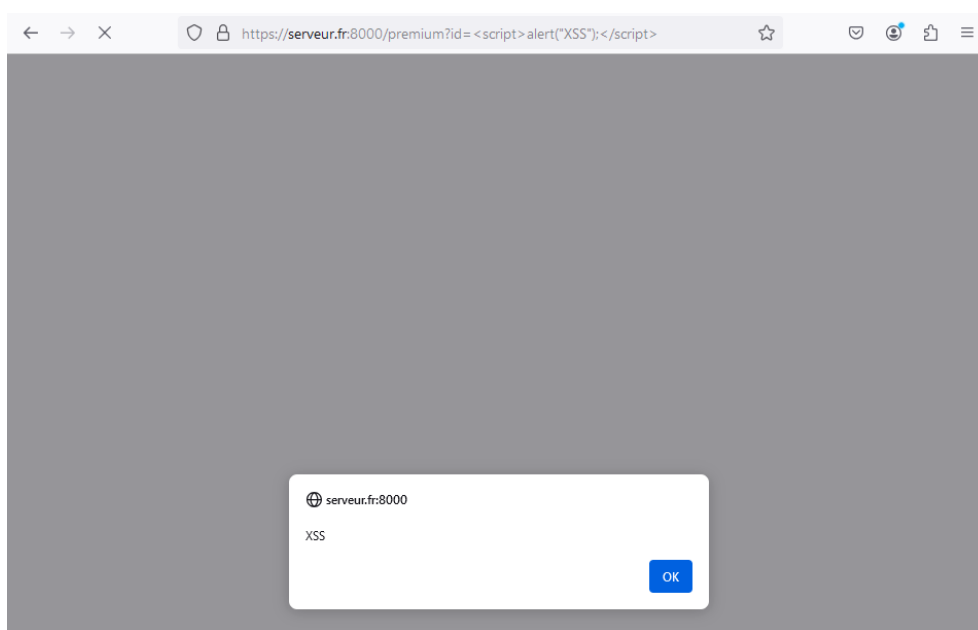
A prouvé que le certificat était désormais reconnu, garantissant une connexion sécurisée sans erreurs liées à l'authentification du certificat. Aucun besoin de régénérer le certificat, prouvant que l'ajout de la CA au store local était suffisant pour résoudre le problème initial.

Etape 3 : Intégration du Web Application Firewall

Pour protéger l'application web contre les attaques de type injection, un Web Application Firewall (WAF) a été mis en place. Le WAF utilisé est ModSecurity, qui est le module officiel intégré à Apache2. Ce dernier inspecte les requêtes HTTP/S pour détecter les tentatives d'attaques basées sur les règles configurées, protégeant ainsi la couche application du modèle OSI.

Accès initial et démonstration de la vulnérabilité XSS : L'accès à l'URL suivante a été effectué pour démontrer une vulnérabilité :

`https://serveur.fr:8000/premium?id=<script>alert("XSS");</script>`



Cette URL a montré une exécution de code JavaScript dans le navigateur, déclenchant un alerte affichant "XSS". Cela a révélé une vulnérabilité à une attaque de type Cross-Site Scripting (XSS), permettant l'injection et l'exécution de scripts malveillants. Une telle vulnérabilité pourrait être exploitée par un attaquant pour afficher des alertes, voler des cookies ou injecter d'autres scripts nuisibles.

Installation de ModSecurity

Pour remédier à cette vulnérabilité, ModSecurity a été installé et configuré. Les étapes suivantes ont été suivies :

Vérification de l'installation actuelle :

La commande suivante a permis de vérifier la présence de ModSecurity :

ls /etc/apache2/mods-enabled/

```
poly@ids-polytech:~$ ls /etc/apache2/mods-enabled/
access_compat.load  authz_core.load  deflate.load  mime.load  proxy.load  socache_shmcb.load
alias.conf          authz_host.load  dir.conf     mpm_event.conf  proxy_http.load  ssl.conf
alias.load          authz_user.load  dir.load     mpm_event.load  reqtimeout.conf  ssl.load
auth_basic.load     autoindex.conf  env.load     negotiation.conf  reqtimeout.load  status.conf
authn_core.load     autoindex.load  filter.load  negotiation.load  setenvif.conf    status.load
authn_file.load     deflate.conf     mime.conf    proxy.conf      setenvif.load
```

L'absence des fichiers security2.conf et security2.load indique que ModSecurity n'était pas encore installé.

Installation de ModSecurity :

La commande suivante a été utilisée pour installer ModSecurity :

sudo apt install libapache2-mod-security2

Pour vérifier que l'installation a été correctement effectuée, la commande suivante a été exécutée :

ls /etc/apache2/mods-enabled/

```
poly@ids-polytech:~$ ls /etc/apache2/mods-enabled/
access_compat.load  authz_host.load  dir.load      negotiation.conf  security2.conf  status.conf
alias.conf          authz_user.load  env.load      negotiation.load  security2.load  status.load
alias.load          autoindex.conf  filter.load   proxy.conf        setenvif.conf   unique_id.load
auth_basic.load     autoindex.load  mime.conf     proxy.load        setenvif.load
authn_core.load     deflate.conf     mime.load     proxy_http.load   socache_shmcb.load
authn_file.load     deflate.load     mpm_event.conf  reqtimeout.conf  ssl.conf
authz_core.load     dir.conf        mpm_event.load  reqtimeout.load  ssl.load
```

Les fichiers security2.conf et security2.load sont présents, indiquant que ModSecurity est bien installé.

Activation de ModSecurity :

Le fichier de configuration recommandé a été copié et configuré pour activer le WAF :

sudo cp /etc/modsecurity/modsecurity.conf-recommended /etc/modsecurity/modsecurity.conf

Ensuite, la valeur de SecAuditEngine a été modifiée à On pour activer la journalisation :

sudo nano /etc/modsecurity/modsecurity.conf

```
# -- Audit log configuration -----

# Log the transactions that are marked by a rule, as well as those that
# trigger a server error (determined by a 5xx or 4xx, excluding 404,
# level response status codes).
#
SecAuditEngine On
SecAuditLogRelevantStatus "^(?:5|4(?:?!04))"
```

La section SecAuditEngine a été changée pour activer la journalisation complète des tentatives suspectes.

Redémarrage d'Apache :

Pour appliquer les modifications, Apache a été redémarré :
sudo systemctl restart apache2

Test de la sécurité après activation du WAF

La même requête a été exécutée :

curl "https://serveur.fr:8000/premium?id=<script>alert('XSS');</script>"

Observation des logs :

Les logs de ModSecurity dans /var/log/apache2/modsec_audit.log révèlent une détection de tentative d'injection XSS sur une requête envoyée depuis 192.168.56.1 vers 192.168.56.12:8000 avec le paramètre id=<script>alert('XSS');</script>. Plusieurs règles ont été déclenchées pour détecter et alerter sur cette tentative d'injection XSS :

- La règle **941100** a détecté l'injection XSS via libinjection.

```
Message: Warning, detected XSS using libinjection. [file "/usr/share/modsecurity-crs/rules/REQUEST-941-APPLICATION-ATTACK-XSS.conf"] [line "56"] [id "941100"] [msg "XSS Attack Detected via libinjection"] [data "Matched Data: XSS data found within ARGS:id: <script>alert(\x22XSS\x22);</script>"] [severity "CRITICAL"] [ver "OWASP_CRS/3.3.5"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-xss"] [tag "paranoia-level/1"] [tag "OWASP_CRS"] [tag "capec/1000/152/242"]
```

- La règle **941110** a identifié la présence du tag <script> dans le paramètre.

```
Message: Warning, Pattern match "(?i)(?<[^\s\S]*>[^\s\S]*?)" at ARGS:id. [file "/usr/share/modsecurity-crs/rules/REQUEST-941-APPLICATION-ATTACK-XSS.conf"] [line "83"] [id "941110"] [msg "XSS Filter - Category I: Script Tag Vector"] [data "Matched Data: <script> found within ARGS:id: <script>alert(\x22XSS\x22);</script>"] [severity "CRITICAL"] [ver "OWASP_CRS/3.3.5"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-xss"] [tag "paranoia-level/1"] [tag "OWASP_CRS"] [tag "capec/1000/152/242"]
```

- La règle **941160** a détecté une injection HTML à travers un tag <script>, vérifiant la présence de code JavaScript potentiellement malveillant.

```
Message: Warning, Pattern match "(?i)(?<[^\s\S]*>[^\s\S]*?)" at ARGS:id. [file "/usr/share/modsecurity-crs/rules/REQUEST-941-APPLICATION-ATTACK-XSS.conf"] [line "280"] [id "941160"] [msg "No Script XSS InjectionChecker: HTML Injection"] [data "Matched Data: <script> found within ARGS:id: <script>alert(\x22XSS\x22);</script>"] [severity "CRITICAL"] [ver "OWASP_CRS/3.3.5"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-xss"] [tag "paranoia-level/1"] [tag "OWASP_CRS"] [tag "capec/1000/152/242"]
```

Ces alertes ont entraîné un **score d'anomalie élevé**, atteignant **15**, comme indiqué par les règles **949110** et **980130**, qui classent la requête comme **critique**.

```
Message: Warning, Operator GE matched 5 at TX:anomaly_score. [file "/usr/share/modsecurity-crs/rules/REQUEST-949-BLOCKING-EVALUATION.conf"] [line "94"] [id "949110"] [msg "Inbound Anomaly Score Exceeded (Total Score: 15)"] [severity "CRITICAL"] [ver "OWASP_CRS/3.3.5"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-generic"]
Message: Warning, Operator GE matched 5 at TX:inbound_anomaly_score. [file "/usr/share/modsecurity-crs/rules/RESPONSE-980-CORRELATION.conf"] [line "92"] [id "980130"] [msg "Inbound Anomaly Score Exceeded (Total Inbound Score: 15 - SQLi=0,XSS=15,RFI=0,LFI=0,RCE=0,PHPI=0,HTTP=0,SESS=0): individual paranoia level scores: 15, 0, 0, 0"] [ver "OWASP_CRS/3.3.5"] [tag "event-correlation"]
Apache-Error: [file "apache2_util.c"] [line 275] [level 3] [client 192.168.56.1] ModSecurity: Warning, detected XSS using libinjection. [file "/usr/share/modsecurity-crs/rules/REQUEST-941-APPLICATION-ATTACK-XSS.conf"] [line "56"] [id "941100"] [msg "XSS Attack Detected via libinjection"] [data "Matched Data: XSS data found within ARGS:id: <script>alert(\x22XSS\x22);</script>"] [severity "CRITICAL"] [ver "OWASP_CRS/3.3.5"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-xss"] [tag "paranoia-level/1"] [tag "OWASP_CRS"] [tag "capec/1000/152/242"] [hostname "serveur.fr"] [uri "/premium"] [unique_id "Z5c5ganK-9MosCKd0B5HxQAAEB"]
```

Test avec le port 10000

Lors de l'exécution de la requête sur le port 10000, aucune détection n'a eu lieu dans les logs. Cela s'explique par le fait que l'accès direct au service sur le port 10000 contourne le WAF, qui est configuré pour filtrer le trafic passant par le reverse proxy sur le port 8000. Le WAF étant positionné en amont, il ne peut contrôler que les connexions qui transitent par le port où il est actif. Pour résoudre ce problème et garantir que toutes les requêtes, y compris celles destinées au port 10000, passent par le WAF, une redirection des requêtes entrantes sur ce port vers le port 8000 a été mise en place. Cette redirection assure que toutes les

connexions passent par le filtre de sécurité du WAF, renforçant ainsi la sécurité globale de l'application.

Redirection avec iptables pour forcer le passage par le WAF

La règle suivante redirige les requêtes destinées au port 10000 vers le port 8000 :

```
sudo iptables -t nat -A PREROUTING -p tcp --dport 10000 -j REDIRECT --to-port 8000
sudo iptables-save -f /etc/iptables/rules.v4
```

Dans les logs, on peut observer que les requêtes initialement destinées au port 10000 sont redirigées vers le port 8000, où elles sont traitées par le WAF. Cette redirection assure que toutes les tentatives de connexion effectuées via le port 10000 passent d'abord par le filtrage du WAF sur le port 8000. Cela permet de centraliser l'analyse et le traitement des requêtes, garantissant que chaque communication, quelle que soit son origine, soit inspectée par le WAF avant d'atteindre le service cible.

```
--bf5d0850-A--
[27/Oct/2024:16:35:49.269662 +0000] Zx5r5anK-9MosCKd085M6wAAAFQ 192.168.56.1 64317 192.168.56.12 8000
--bf5d0850-B--
GET /premium?id=%3Cscript%3Ealert(%22XSS%22);%3C/script%3E HTTP/1.1
Host: serveur.fr:10000
```

Etape 4 : Mise en place du système de détection d'intrusion

Pour cette étape, l'objectif est de configurer Snort pour détecter des activités réseau spécifiques, notamment les tentatives de connexion SSH, et de consigner les alertes correspondantes.

Installation de Snort

Pour commencer, Snort a été installé à l'aide de la commande suivante :

```
sudo apt install snort
```

Snort est un système de détection d'intrusion (IDS) qui surveille le trafic réseau pour identifier des comportements suspects ou des tentatives d'attaques selon des règles configurées.

Ajout d'une règle pour détecter les connexions SSH

Afin de configurer Snort pour détecter les connexions SSH, une règle personnalisée a été ajoutée au fichier `/etc/snort/rules/local.rules`. Cette règle détecte et journalise toute tentative de connexion au port 22 (le port par défaut pour SSH).

Pour cela, la ligne suivante a été insérée dans le fichier de configuration :

```
sudo nano /etc/snort/rules/local.rules
```

Puis, la règle suivante a été ajoutée :

```
alert tcp any any -> any 22 (msg: "Tentative de connexion SSH détectée"; sid:10000001; rev:1; content:"SSH");
```

Explication de la règle :

- `alert tcp` : Indique qu'une alerte doit être déclenchée sur les connexions TCP.
- `any any -> any 22` : Désigne toutes les adresses IP sources et ports sources vers toutes les adresses IP destinations sur le port 22 (SSH).
- `msg: "Tentative de connexion SSH détectée"` : Message affiché lorsque la règle est déclenchée.
- `sid:10000001` : Identifiant unique pour cette règle personnalisée.
- `rev:1` : Révision de la règle.
- `content:"SSH"` : Recherche la présence de "SSH" dans le contenu des paquets.

Test de la détection des connexions SSH

Pour tester cette règle, Snort a été lancé avec la commande suivante :

```
sudo snort -A console -q -u snort -g snort -c /etc/snort/snort.conf -i enp0s8
```

Cette commande exécute Snort en mode console, affichant les alertes en temps réel pour chaque activité détectée sur l'interface réseau spécifiée (enp0s8).

Lorsqu'une tentative de connexion SSH a été effectuée, Snort a généré l'alerte suivante :

```
poly@ids-polytech:~$ sudo snort -A console -q -u snort -g snort -c /etc/snort/snort.conf -i enp0s8
[sudo] password for poly:
10/27-17:01:56.203193  [*] [1:10000001:1] Tentative de connexion SSH détectée [*] [Priority: 0] {TCP} 192.168.56.1:49228 -> 192.168.56.12:22
```

Cette alerte indique qu'une tentative de connexion SSH a été détectée depuis l'adresse IP 192.168.56.1 vers 192.168.56.12 sur le port 22. Les informations incluent :

- **Timestamp** : Date et heure de l'alerte.
- **Priorité** : Niveau de priorité de l'alerte (ici, priorité 0).
- **Détails TCP** : Informations sur l'adresse source, l'adresse de destination, et le port utilisé

Lorsqu'une tentative de connexion SSH est effectuée depuis des adresses IP autres que 192.168.56.1, aucune alerte n'est générée par Snort. Cela s'explique par le fait que le pare-feu bloque ces tentatives de connexion avant même que le trafic ne puisse atteindre Snort. Ainsi, les paquets sont filtrés par le pare-feu et ne sont pas analysés par Snort.

Création d'un service pour Snort

Pour automatiser le démarrage et l'arrêt de Snort, un service systemd a été créé. Le fichier suivant a été édité :

```
sudo nano /etc/systemd/system/alert-snort.service
```

Contenu du fichier :

```
[Unit]
Description=XSS Server Demo
Wants=network-online.target
After=network-online.target
[Service]
Type=simple
ExecStart=snort -A fast -q -u snort -g snort -c /etc/snort/snort.conf --pid-path /run/snort/ -i enp0s8
ExecStop=kill -INT -F /run/snort/snort_enp0s8.pid
[Install]
WantedBy=default.target
```

Ce service permet de contrôler Snort via systemctl, en simplifiant son démarrage et son arrêt.

Pour activer le service :

```
sudo systemctl enable alert-snort
```



```
poly@ids-polytech:~$ sudo systemctl enable alert-snort
Created symlink /etc/systemd/system/default.target.wants/alert-snort.service → /etc/systemd/system/alert-snort.service.
```

Pour démarrer le service :

```
sudo systemctl start alert-snort
```

Pour afficher le status du service :

```
sudo systemctl status alert-snort
```

```
poly@ids-polytech:~$ sudo systemctl status alert-snort
● alert-snort.service - XSS Server Demo
   Loaded: loaded (/etc/systemd/system/alert-snort.service; enabled; preset: enabled)
   Active: active (running) since Sun 2024-10-27 17:10:46 UTC; 22s ago
     Main PID: 6702 (snort)
        Tasks: 2 (limit: 2277)
       Memory: 79.9M (peak: 95.4M)
          CPU: 370ms
      CGroup: /system.slice/alert-snort.service
              └─6702 snort -A fast -q -u snort -g snort -c /etc/snort/snort.conf --pid-path /run/snort/ -i enp0s8

Oct 27 17:10:46 ids-polytech systemd[1]: Started alert-snort.service - XSS Server Demo.
```

Les logs d'alerte générés par Snort sont disponibles dans le fichier /var/log/snort/alert, offrant une vue détaillée des activités détectées sur le réseau.

```
poly@ids-polytech:~$ tail -f /var/log/snort/alert
11/10-19:15:25.332707  [**] [1:1917:6] SCAN UPnP service discover attempt [**] [Classification: Detection of a Network Scan] [Priority: 3] {UDP} 192.168.56.1:60
114 -> 239.255.255.250:1900
11/10-20:07:30.051378  [**] [1:527:8] BAD-TRAFFIC same SRC/DST [**] [Classification: Potentially Bad Traffic] [Priority: 2] {UDP} 0.0.0.0:68 -> 255.255.255.255:
67
11/10-20:07:52.040795  [**] [1:100000122:1] COMMUNITY WEB-MISC mod_jrun overflow attempt [**] [Classification: Web Application Attack] [Priority: 1] {TCP} 192.1
68.56.1:63027 -> 192.168.56.4:8000
11/10-21:10:13.584817  [**] [1:1917:6] SCAN UPnP service discover attempt [**] [Classification: Detection of a Network Scan] [Priority: 3] {UDP} 192.168.56.1:60
114 -> 239.255.255.250:1900
11/10-21:10:16.585018  [**] [1:1917:6] SCAN UPnP service discover attempt [**] [Classification: Detection of a Network Scan] [Priority: 3] {UDP} 192.168.56.1:60
114 -> 239.255.255.250:1900
11/10-21:10:19.585053  [**] [1:1917:6] SCAN UPnP service discover attempt [**] [Classification: Detection of a Network Scan] [Priority: 3] {UDP} 192.168.56.1:60
114 -> 239.255.255.250:1900
11/10-18:59:21.373167  [**] [1:100000122:1] COMMUNITY WEB-MISC mod_jrun overflow attempt [**] [Classification: Web Application Attack] [Priority: 1] {TCP} 192.1
68.56.1:61166 -> 192.168.56.4:8000
11/11-13:06:35.636865  [**] [1:10000001:1] Tentative de connexion SSH détectée [**] [Priority: 0] {TCP} 192.168.56.1:53231 -> 192.168.56.4:22
11/11-14:36:58.935922  [**] [1:527:8] BAD-TRAFFIC same SRC/DST [**] [Classification: Potentially Bad Traffic] [Priority: 2] {UDP} 0.0.0.0:68 -> 255.255.255.255:
67
11/11-15:06:22.988528  [**] [1:10000001:1] Tentative de connexion SSH détectée [**] [Priority: 0] {TCP} 192.168.56.1:57451 -> 192.168.56.4:22
276
```

Etape 5 : Mise en place d'un Système de Prévention d'intrusion

Les systèmes de prévention d'intrusion (IPS) permettent d'analyser et de bloquer les paquets en fonction des attaques détectées, contribuant ainsi à stopper les menaces.

Configuration de ModSecurity en mode bloquant

Pour commencer, ModSecurity a été configuré en mode bloquant en modifiant le paramètre SecRuleEngine :

```
sudo nano /etc/modsecurity/modsecurity.conf
```

La ligne suivante a été modifiée pour activer le mode de blocage :
SecRuleEngine On

```
GNU nano 7.2 /etc/modsecurity/modsecurity.conf
# -- Rule engine initialization -----
# Enable ModSecurity, attaching it to every transaction. Use detection
# only to start with, because that minimises the chances of post-installation
# disruption.
#
SecRuleEngine On
```

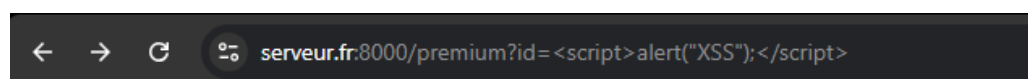
Cela permet à ModSecurity de bloquer activement les attaques au lieu de simplement les signaler. Un redémarrage du service Apache est nécessaire pour appliquer les modifications :

```
sudo systemctl restart apache2
```

Test de la prévention des attaques XSS

Une tentative d'exploit XSS similaire à l'étape précédente a été effectuée en accédant à l'URL suivante :

[https://serveur.fr:8000/premium?id=<script>alert\('XSS'\);</script>](https://serveur.fr:8000/premium?id=<script>alert('XSS');</script>)



Forbidden

You don't have permission to access this resource.

Apache/2.4.58 (Ubuntu) Server at serveur.fr Port 8000

Ce résultat démontre que l'accès à la ressource a été bloqué par le WAF, empêchant ainsi l'exécution de l'attaque XSS et protégeant l'application de manière efficace.

Installation et configuration de Fail2ban

Pour renforcer la sécurité, Fail2ban a été installé. Cet outil surveille les logs de divers services et peut bloquer les adresses IP qui présentent des comportements suspects :

```
sudo apt install fail2ban
```

Configuration d'un Jail pour SSH

Pour détecter les échecs de connexions SSH, un jail a été configuré afin de détecter les échecs de connexions. Pour suivre les recommandations de fail2ban, un fichier `/etc/fail2ban/jail.local` a été créé et on saisit les lignes suivantes :

```
[sshd]
port=ssh
enabled=true
filter=sshd
findtime=1h
bantime=15s
maxretry=3
banaction = iptables-multiport
```

Explications :

- **port=ssh** : Spécifie que ce jail surveille le service SSH.
- **enabled=true** : Active ce jail pour qu'il prenne effet.
- **filter=sshd** : Utilise le filtre sshd de Fail2Ban pour analyser les logs du service SSH et détecter les tentatives de connexion échouées.
- **findtime=1h** : Indique la durée (1 heure) pendant laquelle les tentatives échouées seront comptabilisées.
- **bantime=15s** : Spécifie la durée pendant laquelle une IP sera bannie après avoir dépassé le nombre maximal de tentatives autorisées (ici, 15 secondes).
- **maxretry=3** : Définit le nombre maximal de tentatives échouées avant de déclencher le bannissement (ici, 3 tentatives).
- **banaction=iptables-multiport** : Réalise le bannissement à l'aide de iptables.

Pour garantir que le bannissement se réalise dans iptables, le paramètre `banaction = iptables-multiport` a été ajouté dans la configuration. Sans ce paramètre, le bannissement est tout de même effectué, mais il est géré par un autre mécanisme interne à Fail2Ban, ce qui rend sa vérification plus difficile via les règles du pare-feu.

Pour que les modifications prennent effet, Fail2Ban a été redémarré :

```
sudo systemctl restart fail2ban
```

Test du bannissement des connexions SSH

Pour vérifier le fonctionnement de Fail2Ban, une connexion SSH a été tentée avec 3 erreurs de mot de passe. Cela a entraîné un bannissement temporaire de 15 secondes :

```
Chain f2b-sshd (1 references)
target     prot opt source                destination            reject-with icmp-port-unreachable
REJECT     all  --  192.168.56.1          anywhere
RETURN     all  --  anywhere              anywhere
```

Lorsqu'une tentative de reconnexion a été effectuée pendant la période de bannissement, la connexion a échoué. Cela est dû au paramètre bantime=15s et au fait que le nombre maximal de tentatives (défini par maxretry=3) a été atteint.

Une règle iptables a été ajoutée pour placer l'IP source (192.168.56.1 dans cet exemple) dans la chaîne f2b-sshd. Cette règle utilise l'action REJECT avec un message d'erreur de type icmp-port-unreachable, empêchant toute connexion SSH de cette IP pendant la durée du bannissement.

```
poly@ids-polytech:~$ sudo tail -f /var/log/fail2ban.log
[sudo] password for poly:
2024-10-27 22:58:29,247 fail2ban.filter [1978]: INFO [sshd] Found 192.168.56.1 - 2024-10-27
22:58:28
2024-10-27 22:58:31,967 fail2ban.filter [1978]: INFO [sshd] Found 192.168.56.1 - 2024-10-27
22:58:31
2024-10-27 22:58:34,994 fail2ban.filter [1978]: INFO [sshd] Found 192.168.56.1 - 2024-10-27
22:58:34
2024-10-27 22:58:35,356 fail2ban.actions [1978]: NOTICE [sshd] Ban 192.168.56.1
2024-10-27 22:58:50,363 fail2ban.actions [1978]: NOTICE [sshd] Unban 192.168.56.1
2024-10-27 22:58:59,493 fail2ban.filter [1978]: INFO [sshd] Found 192.168.56.1 - 2024-10-27
22:58:59
2024-10-27 22:59:05,993 fail2ban.filter [1978]: INFO [sshd] Found 192.168.56.1 - 2024-10-27
22:59:05
2024-10-27 22:59:13,036 fail2ban.filter [1978]: INFO [sshd] Found 192.168.56.1 - 2024-10-27
22:59:12
2024-10-27 22:59:13,237 fail2ban.actions [1978]: NOTICE [sshd] Ban 192.168.56.1
2024-10-27 22:59:27,477 fail2ban.actions [1978]: NOTICE [sshd] Unban 192.168.56.1
```

Les logs de Fail2Ban montrent les tentatives échouées et le bannissement de l'IP source via le fichier /var/log/fail2ban.log.

Ajout de la Configuration du Jail pour ModSecurity

Pour configurer un jail spécifique à ModSecurity, les lignes suivantes ont été ajoutées au fichier /etc/fail2ban/jail.local :

```
[modsec]
enabled = true
port = http,https, 8000
filter = modsec
action = iptables-multiport[name=ModSec, port="http,https, 8000"]
logpath = /var/log/apache2/modsec_audit.log
bantime = 15s
maxretry = 3
backend = auto
```

Explications :

- **enabled** = true active ce jail pour qu'il soit pris en compte par Fail2Ban.
- **port** = http,https,8000 spécifie que ce jail surveille les ports HTTP (80), HTTPS (443) et 8000.
- **filter** = modsec indique le filtre à utiliser pour analyser les logs générés par ModSecurity.
- **action** = iptables-multiport assure que le bannissement se fait via iptables pour bloquer l'accès aux ports spécifiés.
- **logpath** = /var/log/apache2/modsec_audit.log définit le chemin des logs de **ModSecurity** que Fail2Ban surveillera pour détecter les comportements à risque.
- **bantime** = 15s spécifie la durée pendant laquelle une IP sera bannie après avoir dépassé le nombre maximal de tentatives autorisées.
- **maxretry** = 3 définit le nombre de tentatives autorisées avant de déclencher le bannissement.
- **backend** = auto permet à Fail2Ban de choisir automatiquement la méthode de lecture des logs, garantissant une compatibilité et une efficacité optimales, même si la méthode d'accès aux fichiers ou leur format évoluent.

Création du Filtre pour ModSecurity

Un filtre spécifique a été créé pour permettre à Fail2Ban de surveiller les logs de ModSecurity. Le fichier /etc/fail2ban/filter.d/modsec.conf a été configuré avec les lignes suivantes :

[Definition]

failregex = .*? \[client <HOST>\] ModSecurity: Access denied with code 403\s

ignoreregex =

Explications :

- failregex définit le motif de recherche que Fail2Ban utilisera pour identifier les tentatives suspectes.
 - Ici, la ligne .*? \[client <HOST>\] ModSecurity: Access denied with code 403\s indique que Fail2Ban surveillera les logs pour les messages où ModSecurity a rejeté une requête avec un code d'accès refusé (403).
 - <HOST> représente l'adresse IP du client qui a tenté l'accès et sera utilisé par Fail2Ban pour identifier et bannir l'IP.
- ignoreregex est laissé vide, ce qui signifie qu'il n'y a pas de motifs à ignorer pour ce filtre.

Ce filtre permet de détecter les tentatives bloquées par ModSecurity, telles que les attaques XSS ou autres comportements malveillants, et de déclencher une action de bannissement via Fail2Ban.

Redémarrage de Fail2Ban

Pour que les modifications du jail et du filtre soient prises en compte, Fail2Ban doit être redémarré :

```
sudo systemctl restart fail2ban
```

Test du Bannissement avec XSS

Pour tester l'efficacité de cette configuration, une tentative d'exploiter la faille XSS a été réalisée trois fois via l'URL suivante :

```
https://serveur.fr:8000/premium?id=<script>alert("XSS");</script>
```

Après trois tentatives, l'IP source a été automatiquement bannie. Cela empêche toute nouvelle requête vers le site depuis cette IP pendant la durée spécifiée dans le bantime.

```
2024-11-11 17:00:03,051 fail2ban.filter [559]: INFO [modsec] Found 192.168.56.1 - 2024-11-11 17:00:02
2024-11-11 17:00:04,677 fail2ban.filter [559]: INFO [modsec] Found 192.168.56.1 - 2024-11-11 17:00:04
2024-11-11 17:00:05,165 fail2ban.filter [559]: INFO [modsec] Found 192.168.56.1 - 2024-11-11 17:00:05
2024-11-11 17:00:05,277 fail2ban.actions [559]: NOTICE [modsec] Ban 192.168.56.1
```

Détection des connexions SSH non autorisées

Rappel de la règle demandée

Déclencher une alerte avec le message "Tentative de connexion SSH depuis une machine non autorisée" pour toute tentative de connexion SSH depuis une machine autre que celle de l'administrateur.

Mise en place de la règle

Pour implémenter cette détection, nous utilisons Snort. La règle suivante est à insérer dans **/etc/snort/rules/local.rules** :

```
alert tcp !192.168.56.1 any -> any 22 (msg: "Tentative de connexion SSH depuis une machine non autorisée"; sid:10000002; rev:1; content:"SSH")
```

Détails de la règle :

- **Protocol** : tcp car SSH fonctionne sur ce protocole.
- **Source** : !192.168.56.1 cible toute adresse IP source sauf celle de l'administrateur (192.168.1.1).
- **Source Port** : any, car les connexions SSH peuvent venir de n'importe quel port source.
- **Destination** : any pour cibler toutes les adresses de destination.
- **Destination Port** : 22, le port par défaut du service SSH.
- **Message d'alerte** : Lorsqu'une tentative de connexion non autorisée est détectée, l'alerte "Tentative de connexion SSH depuis une machine non autorisée" sera générée.
- **Content** : "SSH" pour détecter les paquets ayant du contenu spécifique au SSH.

Justification du choix

Cette règle est conçue pour identifier de manière ciblée les tentatives de connexion SSH en provenance d'IP non autorisées. Elle génère des alertes instantanées, facilitant une surveillance proactive et rapide des tentatives de connexion SSH potentiellement non sécurisées.

Preuve de fonctionnement

Pour la configuration actuelle, les règles d'iptables sont définies pour autoriser uniquement les connexions SSH provenant de l'IP administrative spécifiée (192.168.56.1). Ainsi, toute tentative de connexion SSH depuis une IP non autorisée est bloquée directement par iptables, empêchant Snort d'intercepter et de traiter ces requêtes.

Pour démontrer le fonctionnement de notre règle Snort, nous avons temporairement assoupli ces règles d'iptables, permettant toutes les connexions au port SSH. Cette modification garantit que Snort puisse détecter et générer une alerte lors d'une tentative de connexion depuis une IP non autorisée, prouvant ainsi l'efficacité de la règle.

Étapes de Validation

1. Assouplissement des règles d'iptables

Nous avons utilisé la commande suivante pour autoriser toutes les connexions entrantes au port SSH (22) : `sudo iptables -I INPUT -p tcp --dport 22 -j ACCEPT`

Cette règle permet à toutes les IP d'accéder au port SSH sans restriction, ce qui permet à Snort de surveiller les tentatives de connexion SSH depuis des machines non autorisées et de générer une alerte

2. Lancement de la tentative de connexion non autorisée

Une tentative de connexion SSH a été effectuée depuis une machine non autorisée. Cette action a déclenché l'alerte configurée, affichant le message « Tentative de connexion SSH depuis une machine non autorisée », prouvant ainsi l'efficacité de la règle (voir capture d'écran ci-dessous).

```
10/28-18:50:28.875686  [**] [1:10000002:1] Tentative de connexion SSH depuis une machine non autorisée [**] [Priority: 0] {TCP} 192.168.56.25:56077 -> 192.168.56.12:22
10/28-18:50:28.875686  [**] [1:10000001:1] Tentative de connexion SSH détectée [**] [Priority: 0] {TCP} 192.168.56.25:56077 -> 192.168.56.12:22
poly@ids-polytech:~$
```

3. Rétablissement des règles d'iptables initiales

Après validation, nous avons restauré les règles de sécurité initiales pour restreindre l'accès SSH aux seules machines autorisées en supprimant la règle temporaire :

`sudo iptables -D INPUT -p tcp --dport 22 -j ACCEPT`

Cette étape assure que seuls les administrateurs peuvent à nouveau accéder au service SSH.

La capture d'écran ci-dessous montre bien l'alerte déclenchée, avec le message « Tentative de connexion SSH depuis une machine non autorisée ». Cela confirme que la règle fonctionne correctement et détecte les connexions SSH non autorisées, tout en assurant un niveau de sécurité renforcé une fois les règles initiales rétablies.

Détection de scans XMAS

Rappel de la règle demandée

Déclencher une alerte avec le message "Tentative de scan XMAS" lorsque la technique de scan XMAS est utilisée par un attaquant, identifiable via l'outil Nmap.

Mise en place de la règle

Pour implémenter cette détection, nous utilisons Snort. La règle suivante est ajoutée dans le fichier `/etc/snort/rules/local.rules` :

```
alert tcp any any -> any any (flags: FPU; msg: "Tentative de scan XMAS"; sid:10000003; rev:1;)
```

Détails de la règle :

- **Protocol** : tcp car le scan XMAS utilise TCP pour analyser la réponse des ports.
- **Source et Destination** : any, pour couvrir toutes les adresses IP sources et destinations.
- **Flags** : FPU pour identifier les paquets XMAS, caractérisés par les indicateurs TCP FIN, PSH et URG actifs simultanément.
- **Message d'alerte** : Lorsqu'un scan XMAS est détecté, l'alerte "Tentative de scan XMAS" est déclenchée.
- **SID et Révision** : sid:10000003; rev:1; permet d'identifier cette règle de manière unique.

Justification du choix

Cette règle est spécifiquement configurée pour détecter les scans XMAS en raison des indicateurs TCP uniques utilisés par cette technique. En alertant immédiatement sur ces scans, elle permet de réagir rapidement aux tentatives de reconnaissance réseau.

Preuve de fonctionnement

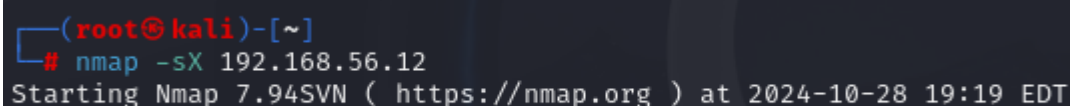
Pour démontrer le bon fonctionnement de cette règle, nous avons exécuté un scan XMAS en utilisant Nmap depuis une machine distante.

Étapes de Validation :

1. Exécution du scan XMAS

Nous avons utilisé la commande suivante pour lancer le scan XMAS :

```
nmap -sX 192.168.56.12
```



```
(root@kali)-[~]  
# nmap -sX 192.168.56.12  
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-10-28 19:19 EDT
```

Cette commande envoie des paquets avec les drapeaux FIN, PSH et URG activés, caractéristiques d'un scan XMAS.

2. Vérification de l'alerte Snort

L'alerte configurée dans Snort a été déclenchée avec le message "Tentative de scan XMAS", comme illustré dans la capture d'écran ci-dessous, confirmant que Snort détecte efficacement cette technique de scan.

```
10/28-23:20:18.911717  [**] [1:10000003:1] Tentative de scan XMAS [**] [Priority: 0] {TCP} 192.168.56.3:52571 -> 192.168.56.12:1984
10/28-23:20:18.911717  [**] [1:1228:7] SCAN nmap XMAS [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.56.3:52571 -> 192.168.56.12:1984
```

La capture d'écran ci-dessous montre bien l'alerte générée par Snort avec le message « Tentative de scan XMAS », confirmant ainsi que notre règle est correctement configurée et fonctionne comme attendu. Cette alerte prouve que Snort est capable de détecter efficacement les scans XMAS, offrant une protection supplémentaire contre les tentatives de reconnaissance non autorisées sur le réseau.

Détection de l'accès au fichier acquisitions.md

Pour permettre la communication avec le conteneur Juice Shop dans le réseau docker0, nous ajoutons ces règles car le pare-feu actuel bloque tout sauf ce qui est explicitement autorisé. docker0 utilise l'adresse réseau 172.17.0.0/16, mais pour couvrir l'accès, nous autorisons l'ensemble de la plage 172.16.0.0/12 :

```
sudo iptables -A OUTPUT -d 172.16.0.0/12 -j ACCEPT
sudo iptables -A INPUT -s 172.16.0.0/12 -j ACCEPT
sudo iptables-save -f /etc/iptables/rules.v4
```

Rappel de la règle demandée

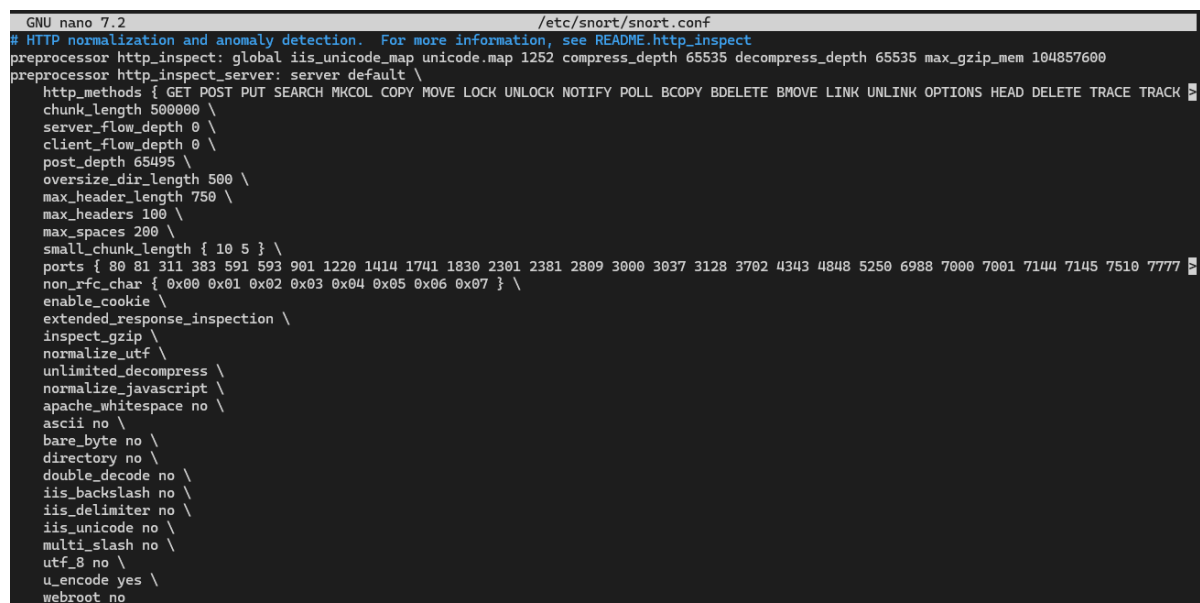
Déclencher une alerte avec le message "Tentative de récupération de fichier confidentiel" pour toute tentative d'accès au fichier **acquisitions.md** situé dans un répertoire sensible du site.

Mise en place de la règle

Pour implémenter cette détection, nous utilisons Snort. La règle suivante est ajoutée dans **/etc/snort/rules/local.rules** :

```
alert tcp any any -> any 3000 (msg: "Tentative d'accès non autorisé au fichier confidentiel acquisitions.md"; sid:10000004; content:"ftp/acquisitions.md"; http_uri; nocase;)
```

Le port 3000 n'est pas un port reconnu de base par snort comme étant un port HTTP. Pour pouvoir utiliser http_uri, nous devons l'ajouter dans preprocessor http_inspect_server de snort dans le fichier /etc/snort/snort.conf



```
GNU nano 7.2 /etc/snort/snort.conf
# HTTP normalization and anomaly detection.  For more information, see README.http_inspect
preprocessor http_inspect: global iis_unicode_map unicode.map 1252 compress_depth 65535 decompress_depth 65535 max_gzip_mem 104857600
preprocessor http_inspect_server: server default \
  http_methods { GET POST PUT SEARCH MKCOL COPY MOVE LOCK UNLOCK NOTIFY POLL BCOPY BDELETE BMOVE LINK UNLINK OPTIONS HEAD DELETE TRACE TRACK \
  chunk_length 500000 \
  server_flow_depth 0 \
  client_flow_depth 0 \
  post_depth 65495 \
  oversize_dir_length 500 \
  max_header_length 750 \
  max_headers 100 \
  max_spaces 200 \
  small_chunk_length { 10 5 } \
  ports { 80 81 311 383 591 593 901 1220 1414 1741 1830 2301 2381 2809 3000 3037 3128 3702 4343 4848 5250 6988 7000 7001 7144 7145 7510 7777 } \
  non_rfc_char { 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 } \
  enable_cookie \
  extended_response_inspection \
  inspect_gzip \
  normalize_utf \
  unlimited_decompress \
  normalize_javascript \
  apache_whitespace no \
  ascii no \
  bare_byte no \
  directory no \
  double_decode no \
  iis_backslash no \
  iis_delimiter no \
  iis_unicode no \
  multi_slash no \
  utf_8 no \
  u_encode yes \
  webroot no
```

Détails de la règle :

- **Protocol** : tcp pour surveiller les requêtes HTTP.

- **Source et Destination** : any pour cibler toutes les adresses IP source et destination, car l'accès peut provenir de n'importe quel utilisateur.
- **Destination Port** : 3000, car le service web de Juice Shop fonctionne sur ce port.
- **Message d'alerte** : En cas de tentative d'accès, une alerte avec le message "*Tentative d'accès non autorisé au fichier confidentiel acquisitions.md*" sera générée.
- **Content** : La chaîne "/ftp/acquisitions.md" est utilisée pour détecter la requête spécifique vers le fichier confidentiel.
- **http_uri** : Cet opérateur assure que la règle analyse uniquement l'URI de la requête HTTP.
- **nocase** : Cette option rend la détection insensible à la casse, pour une détection plus fiable.

Justification du choix

Cette règle identifie toute tentative d'accès au fichier **acquisitions.md**, qui contient des informations confidentielles. Elle est conçue pour alerter immédiatement si un utilisateur tente de consulter ce fichier, renforçant ainsi la sécurité et la surveillance des accès à des ressources sensibles.

Preuve de fonctionnement

Pour valider le bon fonctionnement de cette règle, nous avons visité le lien suivant : <https://serveur.fr:8000/ftp/acquisitions.md> pour simuler une tentative d'accès non autorisé.

Étapes de Validation

1. Accès au fichier

Nous avons saisi l'URL suivante dans le navigateur pour déclencher la règle : <https://serveur.fr:8000/ftp/acquisitions.md>

2. Vérification de l'alerte Snort

Il est désormais nécessaire de lancer Snort sur l'interface **docker0**, contrairement à précédemment où nous utilisions l'interface **enp0s8**. En effet, les requêtes qui arrivent sur **enp0s8** sont chiffrées, ce qui empêche Snort d'analyser le contenu des paquets. Cependant, lorsque Apache2 effectue une requête HTTP vers le conteneur Docker qui contient Juice Shop, les paquets ne sont plus chiffrés, ce qui permet à Snort de les analyser. Il est également nécessaire d'ajouter l'option **-k none** car le conteneur Docker est sur la même machine que le serveur Apache2. L'alerte configurée dans Snort a été déclenchée avec le message 'Tentative d'accès non autorisé au fichier confidentiel acquisitions.md', comme illustré dans la capture d'écran ci-dessous, confirmant que Snort détecte efficacement cet accès sensible.

```
poly@ids-polytech:~$ sudo snort -A console -q -u snort -g snort -c /etc/snort/snort.conf -i docker0 -k none
11/10-18:39:40.473679  [**] [1:100000000:0] Tentative d'accès non autorisé au fichier confidentiel acquisitions.md [**] [Priority: 0] {TCP} 172.17.0.1:49044 -> 172.17.0.2:3000
```

Prévention de l'accès à la page "about"

Rappel de la règle demandée

Bloquer tout accès à la page "about" du site web dès qu'il est détecté.

Mise en place de la règle

Pour implémenter cette règle de prévention, nous avons utilisé ModSecurity, un WAF (Web Application Firewall) intégré dans Apache. La règle suivante a été ajoutée dans le fichier `/etc/modsecurity/modsecurity.conf` pour bloquer l'accès à l'URI contenant le mot-clé "about" :

SecRule REQUEST_URI "about" "id:10000010,phase:1,deny,status:403,msg:'Accès interdit à la page About Us détecté',log"

```
GNU nano 7.2 /etc/modsecurity/modsecurity.conf
# hole for attackers to exploit.
#
SecRequestBodyAccess On

SecRule REQUEST_URI "about" "id:10000010,phase:1,deny,status:403,msg:'Accès interdit à la page About Us détecté',log"
```

Détails de la règle :

- *SecRule* : Déclare une règle de détection et de prévention dans ModSecurity.
- *REQUEST_URI "about"* : Cette règle analyse l'URI de la requête et recherche le terme "about".
- *id:10000010* : Identifiant unique de la règle pour son suivi.
- *phase:1* : La règle est appliquée dans la phase 1 (analyse de la requête).
- *deny* : Bloque l'accès lorsqu'une correspondance est trouvée.
- *status:403* : Renvoie le statut HTTP 403 (Forbidden) pour indiquer que l'accès est refusé.
- *msg* : Message explicatif consigné dans les journaux.
- *log* : Indique que la correspondance doit être consignée.

Justification du choix

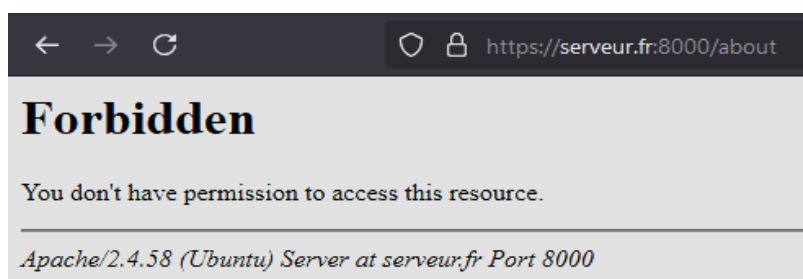
Cette règle garantit que tout accès à l'URI contenant "about" est bloqué et notifié par une entrée dans les journaux de ModSecurity. Cela renforce la sécurité en empêchant l'accès non autorisé à des sections sensibles du site.

Preuve de fonctionnement

Pour valider cette règle, nous avons accédé à l'URL suivante dans le navigateur : <https://serveur.fr:8000/about>

Observations :

1. La page renvoie une réponse **403 Forbidden**, confirmant que l'accès est bloqué.



2. Les journaux de ModSecurity montrent que la règle a été déclenchée, avec les détails de la requête bloquée et le message associé.

```
--a4368504-H--
Message: Access denied with code 403 (phase 1). Pattern match "about" at REQUEST_URI. [file "/etc/modsecurity/modsecurity.conf"] [line "18"] [id "10000010"]
[msg "Acc\xc3\xa8s interdit \xc3\xa0 la page About Us d\xc3\x9ctect\xc3\xa9"]
Apache-Error: [file "apache2_util.c"] [line 275] [level 3] [client 192.168.56.1] ModSecurity: Access denied with code 403 (phase 1). Pattern match "about" at
REQUEST_URI. [file "/etc/modsecurity/modsecurity.conf"] [line "18"] [id "10000010"] [msg "Acc\\\\xc3\\\\xa8s interdit \\\\xc3\\\\xa0 la page About Us d\\\\x
c3\\\\xa9tect\\\\xc3\\\\xa9"] [hostname "serveur.fr"] [uri "/about"] [unique_id "ZzEly3nmPtea-J6m00dC6AAAAABI"]
Action: Intercepted (phase 1)
Stopwatch: 1731269835750032 1063 (- - -)
Stopwatch2: 1731269835750032 1063; combined=83, p1=13, p2=0, p3=0, p4=0, p5=70, sr=0, sw=0, l=0, gc=0
Response-Body-Transformed: Dechunked
Producer: ModSecurity for Apache/2.9.7 (http://www.modsecurity.org/); OWASP_CRS/3.3.5.
Server: Apache/2.4.58 (Ubuntu)
Engine-Mode: "ENABLED"
```

Détection des attaques de type DoS

Rappel de la règle demandée

Mettre en place une détection d'attaque de type DoS (Déni de Service) visant à surcharger le port 8000 du serveur. Créer une règle Snort pour détecter cette attaque et observer les paquets réseau avec Tcpdump. Par la suite, configurer une règle pour bloquer ce type de trafic.

Mise en place de l'attaque DoS avec Hping3

Pour simuler une attaque DoS, l'outil Hping3 a été utilisé pour générer un grand flux de requêtes vers le port 8000 du serveur cible. La commande suivante a été exécutée :

```
sudo hping3 -S -p 8000 --flood 192.168.56.4
```

- **-S** : Définit le drapeau SYN pour simuler de nombreuses requêtes de connexion TCP.
- **-p 8000** : Spécifie que l'attaque cible le port 8000 du serveur.
- **--flood** : Envoie les paquets aussi rapidement que possible pour surcharger le serveur.

Observation

L'attaque génère un flux de paquets qui peut être observé en utilisant Tcpdump pour vérifier le trafic entrant vers le port 8000. Voici la commande utilisée pour capturer les paquets :

```
sudo tcpdump -i enp0s8 port 8000
```

```
17:15:20.326620 IP 192.168.56.5.6392 > serveur.fr.8000: Flags [S], seq 940737151, win 512, length 0
17:15:20.326620 IP 192.168.56.5.6393 > serveur.fr.8000: Flags [S], seq 1939216862, win 512, length 0
17:15:20.326620 IP 192.168.56.5.3079 > serveur.fr.8000: Flags [R], seq 2063249016, win 0, length 0
17:15:20.326620 IP 192.168.56.5.3080 > serveur.fr.8000: Flags [R], seq 748504174, win 0, length 0
17:15:20.326620 IP 192.168.56.5.3081 > serveur.fr.8000: Flags [R], seq 1596479059, win 0, length 0
17:15:20.326620 IP 192.168.56.5.6394 > serveur.fr.8000: Flags [S], seq 2037218633, win 512, length 0
17:15:20.326620 IP 192.168.56.5.6395 > serveur.fr.8000: Flags [S], seq 943028935, win 512, length 0
17:15:20.326620 IP 192.168.56.5.6396 > serveur.fr.8000: Flags [S], seq 1004462373, win 512, length 0
17:15:20.326642 IP serveur.fr.8000 > 192.168.56.5.6392: Flags [S.], seq 72326674, ack 940737152, win 64240, options [mss 1460], length 0
17:15:20.326696 IP serveur.fr.8000 > 192.168.56.5.6393: Flags [S.], seq 4288853123, ack 1939216863, win 64240, options [mss 1460], length 0
17:15:20.326754 IP serveur.fr.8000 > 192.168.56.5.6394: Flags [S.], seq 105313168, ack 2037218634, win 64240, options [mss 1460], length 0
17:15:20.326810 IP serveur.fr.8000 > 192.168.56.5.6395: Flags [S.], seq 3402211119, ack 943028936, win 64240, options [mss 1460], length 0
17:15:20.326863 IP serveur.fr.8000 > 192.168.56.5.6396: Fla^C

8491 packets captured
129399 packets received by filter
107864 packets dropped by kernel
```

De plus, pendant l'attaque DoS, le site devient inaccessible car le serveur est saturé, empêchant toute nouvelle connexion légitime d'être établie.

Détection de l'attaque DoS avec Snort

Pour détecter cette attaque, une règle Snort a été configurée dans /etc/snort/rules/local.rules :

alert tcp any any -> any any (msg: "Tentative d'attaque DoS détectée"; sid:10000005; rev:1; flags:S; threshold: type both, track by_src, count 1000, seconds 10;)

Détails de la règle :

- **Protocol** : tcp, car l'attaque utilise TCP avec des paquets SYN.
- **Source** : any, pour surveiller toutes les adresses IP source.
- **Source Port** : any, car l'attaque peut provenir de différents ports.
- **Destination** : any, pour surveiller toutes les adresses de destination.
- **Destination Port** : any, bien que l'attaque cible spécifiquement le port 8000, nous utilisons any any pour une détection plus large.
- **Flags** : S, pour détecter les paquets avec le drapeau SYN activé.
- **Threshold** : Limite de détection fixée à 1000 paquets SYN par 10 secondes pour une même IP source, générant une alerte en cas de dépassement.

Justification du choix

Cette règle vise à détecter un flux inhabituel de paquets SYN, indicatif d'une tentative de surcharge du serveur (attaque DoS). Elle permet de générer des alertes lorsque le seuil est atteint, offrant une surveillance proactive.

Preuve de fonctionnement

1. **Lancement de l'attaque DoS avec Hping3** L'attaque a été lancée depuis une machine attaquante vers le port 8000 du serveur.
2. **Détection par Snort** Une alerte a été générée dans les logs de Snort, indiquant la détection de l'attaque DoS avec le message "Tentative d'attaque DoS détectée". Cette alerte confirme que la règle est efficace.

```
poly@ids-polytech:~$ sudo tail -f /var/log/snort/alert
11/11-17:11:53.4927081 ** [1.527:8] BAD-TRAFFIC same SRC/DST ** [Classification: Potentially Bad Traffic] [Priority: 2] [UDP] 0.0.0.0:68 -> 255.255.255.255:67
11/11-17:11:53.450397 ** [1.527:8] BAD-TRAFFIC same SRC/DST ** [Classification: Potentially Bad Traffic] [Priority: 2] [IPv6-ICMP] :: -> ff02::16
11/11-17:11:53.654139 ** [1.527:8] BAD-TRAFFIC same SRC/DST ** [Classification: Potentially Bad Traffic] [Priority: 2] [IPv6-ICMP] :: -> ff02::16
11/11-17:11:53.698148 ** [1.527:8] BAD-TRAFFIC same SRC/DST ** [Classification: Potentially Bad Traffic] [Priority: 2] [IPv6-ICMP] :: -> ff02::1:ff15:31a4
11/11-17:14:56.951228 ** [1.524:8] BAD-TRAFFIC tcp port 0 traffic ** [Classification: Misc activity] [Priority: 3] [TCP] 192.168.56.5:0 -> 192.168.56.4:8000
11/11-17:14:57.313197 ** [1.524:8] BAD-TRAFFIC tcp port 0 traffic ** [Classification: Misc activity] [Priority: 3] [TCP] 192.168.56.5:0 -> 192.168.56.4:8000
11/11-17:14:58.275286 ** [1.524:8] BAD-TRAFFIC tcp port 0 traffic ** [Classification: Misc activity] [Priority: 3] [TCP] 192.168.56.5:0 -> 192.168.56.4:8000
11/11-17:15:21.414549 ** [1.524:8] BAD-TRAFFIC tcp port 0 traffic ** [Classification: Misc activity] [Priority: 3] [TCP] 192.168.56.5:0 -> 192.168.56.4:8000
11/11-17:15:25.538376 ** [1.524:8] BAD-TRAFFIC tcp port 0 traffic ** [Classification: Misc activity] [Priority: 3] [TCP] 192.168.56.5:0 -> 192.168.56.4:8000
11/11-17:15:25.754815 ** [1.524:8] BAD-TRAFFIC tcp port 0 traffic ** [Classification: Misc activity] [Priority: 3] [TCP] 192.168.56.5:0 -> 192.168.56.4:8000
11/11-17:28:38.868725 ** [1.10800006:1] Tentative d'attaque DoS détectée ** [Priority: 0] [TCP] 192.168.56.5:1067 -> 192.168.56.4:8000
11/11-17:28:38.868725 ** [1.10800006:1] Tentative d'attaque DoS détectée ** [Priority: 0] [TCP] 192.168.56.5:1068 -> 192.168.56.4:8000
11/11-17:28:38.868725 ** [1.10800006:1] Tentative d'attaque DoS détectée ** [Priority: 0] [TCP] 192.168.56.5:1069 -> 192.168.56.4:8000
11/11-17:28:38.868725 ** [1.10800006:1] Tentative d'attaque DoS détectée ** [Priority: 0] [TCP] 192.168.56.5:1070 -> 192.168.56.4:8000
11/11-17:28:38.868725 ** [1.10800006:1] Tentative d'attaque DoS détectée ** [Priority: 0] [TCP] 192.168.56.5:1071 -> 192.168.56.4:8000
11/11-17:28:38.869542 ** [1.10800006:1] Tentative d'attaque DoS détectée ** [Priority: 0] [TCP] 192.168.56.5:1072 -> 192.168.56.4:8000
11/11-17:28:38.869542 ** [1.10800006:1] Tentative d'attaque DoS détectée ** [Priority: 0] [TCP] 192.168.56.5:1073 -> 192.168.56.4:8000
11/11-17:28:38.869542 ** [1.10800006:1] Tentative d'attaque DoS détectée ** [Priority: 0] [TCP] 192.168.56.5:1074 -> 192.168.56.4:8000
11/11-17:28:38.869542 ** [1.10800006:1] Tentative d'attaque DoS détectée ** [Priority: 0] [TCP] 192.168.56.5:1075 -> 192.168.56.4:8000
11/11-17:28:38.869542 ** [1.10800006:1] Tentative d'attaque DoS détectée ** [Priority: 0] [TCP] 192.168.56.5:1076 -> 192.168.56.4:8000
11/11-17:28:38.869542 ** [1.10800006:1] Tentative d'attaque DoS détectée ** [Priority: 0] [TCP] 192.168.56.5:1077 -> 192.168.56.4:8000
```

Mise en place du blocage avec Fail2Ban

Pour contrer les attaques DoS détectées par Snort, nous avons configuré Fail2Ban pour bloquer les adresses IP à l'origine du trafic indésirable. Cette solution agit en surveillant les logs de Snort et en bannissant automatiquement les IP qui déclenchent les alertes associées à l'attaque DoS.

Configuration du Jail pour Fail2Ban

Pour configurer le blocage des attaques DoS, nous avons ajouté une section dédiée dans le fichier existant `/etc/fail2ban/jail.local`. Cette section cible les alertes générées par Snort pour les attaques DoS :

```
[snort-dos]
enabled = true
port = 8000
filter = snort-dos
logpath = /var/log/snort/alert
bantime = 60s
findtime = 1m
maxretry = 1
action = iptables-allports[name=SnortDoS]
```

Explications :

- **enabled = true** : Active ce jail pour qu'il prenne effet.
- **port = 8000** : Surveille les événements liés au port 8000, correspondant à la cible de l'attaque DoS.
- **filter = snort-dos** : Utilise le filtre nommé snort-dos pour analyser les alertes de Snort.
- **logpath = /var/log/snort/alert** : Chemin vers le fichier de logs contenant les alertes de Snort.
- **bantime = 60s** : Temps de bannissement d'une IP après détection d'une attaque (60 secondes).
- **findtime = 1m** : Période durant laquelle les tentatives seront comptabilisées (1 minute).
- **maxretry = 1** : Une seule tentative suffit pour déclencher le bannissement.
- **action = iptables-allports** : Bloque l'IP sur tous les ports du système. Cela permet d'empêcher toute interaction de l'IP malveillante, renforçant la sécurité.

Création du filtre pour les alertes Snort

Un filtre Fail2Ban a été configuré dans `/etc/fail2ban/filter.d/snort-dos.conf` :

[Definition]

```
failregex = .*?\[.*?\] Tentative d'attaque DoS détectée .*? \{TCP\} <HOST>:. .*? ->
ignoreregex =
```

Explications :

- **failregex** : Expression régulière qui correspond aux messages d'alerte générés par Snort pour les attaques DoS. Toute alerte contenant "Tentative d'attaque DoS détectée" sera capturée par Fail2Ban.
- **ignoreregex** : Aucune expression ignorée spécifiée.

Preuve de fonctionnement

1. Bannissement de l'IP suite à la détection d'une attaque DoS :

Lorsqu'une attaque DoS est détectée par Snort, Fail2Ban déclenche le bannissement de l'IP malveillante. Comme illustré dans la capture d'écran ci-dessous, Fail2Ban a ajouté une règle dans iptables pour rejeter les paquets provenant de l'IP source, empêchant ainsi toute nouvelle connexion pendant la durée du bannissement.

2. Notification de bannissement par Fail2Ban :

Fail2Ban enregistre également les actions dans ses logs, confirmant le bannissement de l'IP identifiée par Snort comme source de l'attaque DoS.

Preuve de fonctionnement

1. Détection par Snort

Lorsqu'une attaque DoS est lancée, Snort détecte le flux important de paquets et génère une alerte. Cette détection est illustrée dans la capture ci-dessous, où une alerte avec le message "Tentative d'attaque DoS détectée" a été enregistrée dans les logs de Snort.

```
11/12-16:04:03.563621 [**] [1:10000005:1] Tentative d'attaque DoS détectée [**] [Priority: 0] {TCP} 192.168.56.5:18715 -> 192.168.56.4:8000
```

2. Notification de bannissement par Fail2Ban

Une fois l'attaque détectée par Snort, Fail2Ban déclenche le processus de bannissement de l'IP identifiée comme source de l'attaque. Les logs de Fail2Ban montrent une notification indiquant que l'IP attaquante a été bannie, assurant ainsi une protection renforcée.

```
2024-11-12 16:04:03.921 fail2ban.filter [2592]: INFO [snort-dos] Found 192.168.56.5 - 2024-11-12 16:04:03
2024-11-12 16:04:04.075 fail2ban.actions [2592]: NOTICE [snort-dos] Ban 192.168.56.5
```

3. Bannissement de l'IP via iptables

Fail2Ban ajoute une règle dans iptables pour bloquer l'IP source de l'attaque. Comme illustré dans la capture ci-dessous, cette règle rejette toutes les connexions provenant de l'IP bannie, empêchant ainsi toute nouvelle tentative d'attaque pendant la durée du bannissement.

```
Chain f2b-SnortDoS (1 references)
target     prot opt source                destination           reject-with icmp-port-unreachable
REJECT     all  --  192.168.56.5          anywhere
RETURN     all  --  anywhere              anywhere
```

Neutralisation de l'attaque DoS :

La mise en place de Fail2Ban en complément de la détection de Snort permet de neutraliser efficacement les attaques DoS ciblant le serveur. Lorsqu'une attaque est détectée par Snort, l'IP source est immédiatement bannie par Fail2Ban, bloquant ainsi toute nouvelle tentative de connexion via une règle iptables. Cela protège le serveur en limitant le flux de trafic malveillant et empêche l'attaquant de continuer à surcharger les ressources.

Échapper au filtrage de la règle DoS :

Un attaquant peut contourner ce filtre en utilisant diverses techniques telles que :

1. **Réduction du taux d'envoi des paquets** : L'attaquant peut ajuster le flux des paquets pour rester sous le seuil de détection fixé par la règle Snort, rendant la détection plus difficile.
2. **Utilisation d'adresses IP multiples (attaque distribuée)** : Une attaque DDoS utilisant de nombreuses adresses IP peut rendre le blocage par IP inefficace, car chaque IP individuellement peut rester en dessous du seuil d'alerte.
3. **Changement de la signature du trafic** : En modifiant les caractéristiques des paquets (par exemple, en utilisant différents drapeaux TCP), l'attaquant peut éviter la détection basée sur des signatures prédéfinies.