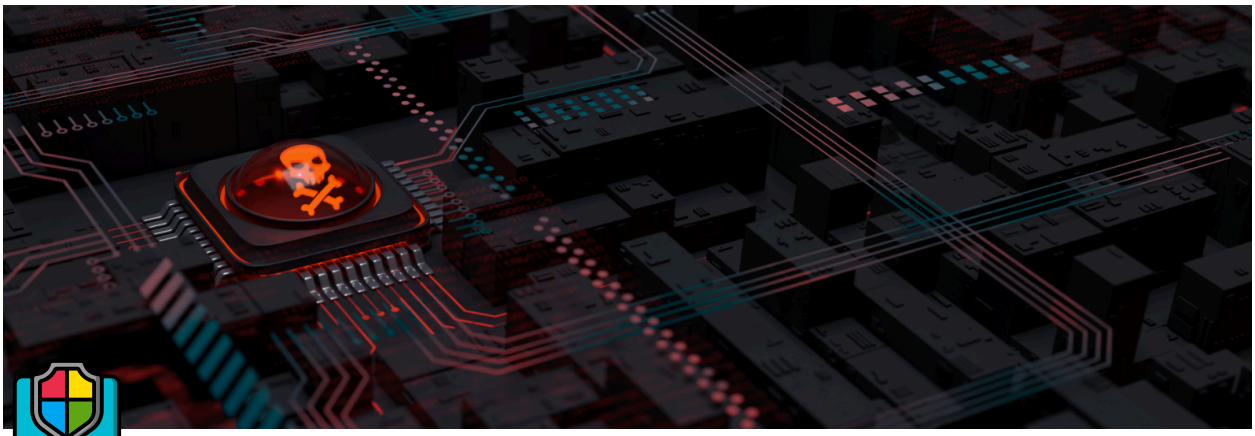


Sécurisation Avancée d'une Application Web sur Serveur Nginx

Objectif

L’objectif de ce projet est de **sécuriser un serveur Nginx exposé à Internet** en mettant en place plusieurs mécanismes de protection et de supervision. L’approche adoptée repose sur **trois axes principaux** :

1️⃣ **Chiffrement des communications** → Mise en place de HTTPS avec un certificat SSL/TLS pour sécuriser les échanges entre les clients et le serveur.

2️⃣ **Protection avancée contre les attaques web** → Intégration du pare-feu applicatif **ModSecurity (WAF)** pour filtrer et bloquer les tentatives d’attaques telles que les injections SQL, les attaques XSS et autres menaces courantes.

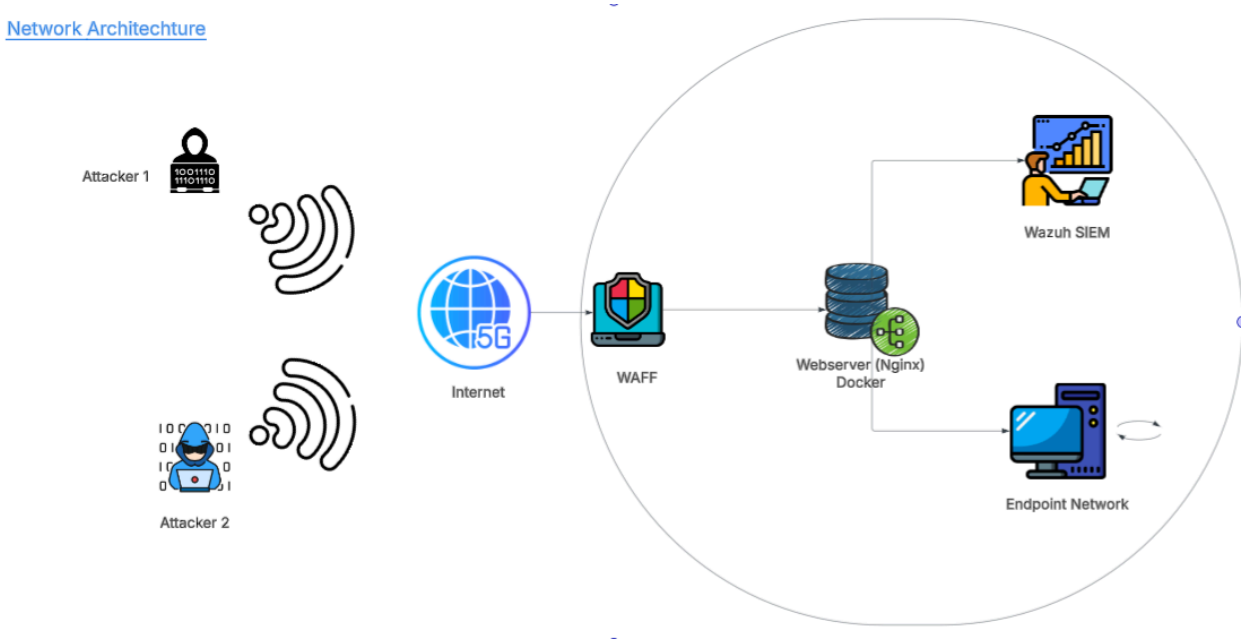
3️⃣ **Supervision et analyse des événements de sécurité** → Déploiement de **Wazuh SIEM** pour la collecte, l’analyse et la détection des incidents de sécurité en temps réel à partir des logs de Nginx et ModSecurity.

Technologies utilisées

- **Nginx :** Serveur Web

- **Reverse Proxy**
- Docker : C’est le conteneur facilitant le déploiement et la migration de notre application
- **OpenSSL :** Génération et gestion des certificats SSL/TLS
- **OWASP Core Rule Set (CRS)** (Règles avancées pour ModSecurity)
- **ModSecurity :** Web Application Firewall (WAF)Règles de sécurité avancées pour ModSecurity
- **Wazuh SIEM :**Supervision des logs et détection des menaces

Architecture



Réalisation du projet

1. Installation de Nginx

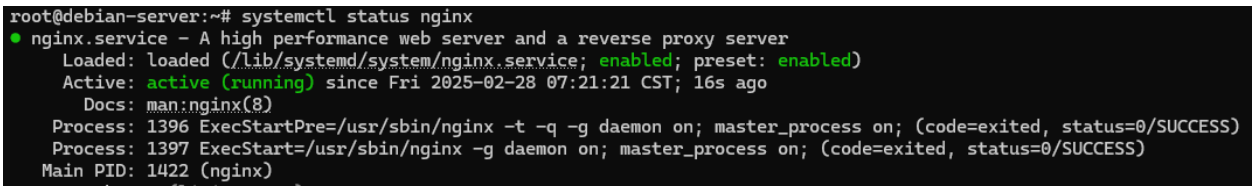
Notre serveur est un os Debian 12, une fois prêt, on va le mettre à jour et installer Nginx:

```
apt update && apt upgrade -y
```

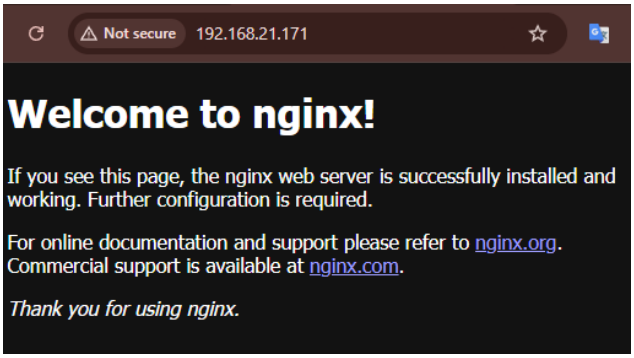
```
apt install nginx -y
```

Une fois que c'est installé on peut regarder le statut :

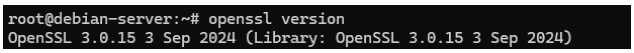
```
systemctl status nginx
```



Pour faire un test on va dans le navigateur et on saisi l'adresse IP de notre serveur : http://192.168.21.171



2. Ensuite on vérifie que OpenSSL est bien installé car par défaut il est présent sur debian



- On va générer un certificat auto-signé : Un certificat SSL/TLS contient des informations permettant de sécuriser la communication entre un client et un serveur via **HTTPS**. Dans le cas d'un certificat **auto-signé**, c'est l'administrateur qui signe lui-même son certificat, au lieu de le faire valider par une **autorité de certification** comme **Let's Encrypt, GlobalSign, DigiCert**, etc.

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/
```

La commande ci-dessus stipule que : x509 : Génère un certificat auto-signé, nodes : Pas de chiffrement de la clé privée (nécessaire pour Nginx), days 365 : Durée de validité du certificat, newkey rsa:2048 : Crée une clé RSA de 2048 bits.

On suis juste les informations , mais on ne renseigne que l'adresse IP dans la partie Common name :

```
Common Name (e.g. server FQDN or YOUR name) []:192.168.21.171
```

3. Nous passons à la Génération d'un groupe **Diffie-Hellman (DH)** : Il permet l'échange sécurisé de clés entre un client et un serveur sans qu'un attaquant puisse intercepter ou reconstruire ces clés. Cela renforce la sécurité de la connexion HTTPS.

```
openssl dhparam -out /etc/nginx/dhparam.pem 4096
```

Cette commande va durer plusieurs minutes en fonction de la puissance du CPU (25+mins chez moi) en attendant faites un tour pour en savoir plus sur Diffie-Hellman (DH).

4. Ensuite on procède à la **Configuration Nginx pour HTTPS :**

On crée un fichier **/etc/nginx/snippets/self-signed.conf et on mettra les configurations suivant :**

👩🏾‍💻 Configurations

5. Par la suite dans le fichier de configuration de nginx /etc/nginx/sites-available/default on rajoute ce lignes dans la partie server {.....} :

```
listen 443 ssl;
listen [::]:443 ssl;
include snippets/self-signed.conf;
```

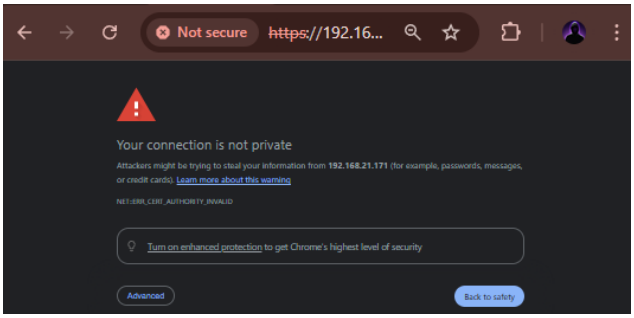
Ceci permet à Nginx d'activer le support de **HTTPS** sur le serveur. Après ça on relaod notre serveur Nginx avec la commande : systemctl reload nginx .

6. Pour continuer on doit réalisez une redirection du trafic HTTP entrant sur le port 80 vers le protocole HTTPS. Pour cela on rajoute ceci dans le fichier de

config nginx /etc/nginx/sites-available/default :

```
server {
    listen 80;
    listen [::]:80;
    server_name _;
    return 301 https://$host$request_uri;
}
```

Une fois fait on redémarre le service systemctl restart nginx .



💡 L'avantage du **Certificat auto-signé est que l**e certificat est signé par l'utilisateur, et non par une Autorité de Certification (CA). L'inconvénient est que les navigateurs affichent un avertissement de sécurité (manque de confiance). La solution est d'utiliser un certificat validé par une CA (ex: Let's Encrypt via Certbot).

Pour obtenir un certificat **validé** par une CA publique comme Let's Encrypt :

```
apt install certbot python3-certbot-nginx -y
certbot --nginx -d nom-domaine.com
```

Cela permet d'avoir un certificat valide **gratuitement**

Installation et configuration de ModSecurity (WAF)

ModSecurity est un **Web Application Firewall (WAF)** qui protège contre diverses attaques.

Installation de ModSecurity

1. Installation des prérequis

```
apt-get -y install apt-transport-https lsb-release ca-certificates curl
```

2. Ajout du dépôt et installation de ModSecurity

```
wget -qO - https://modsecurity.digitalwave.hu/archive.key | tee /etc/apt/trusted.
```

3. On créer le fichier `/etc/apt/sources.list.d/dwmodsec.list` avec le contenu suivant:

```
deb http://modsecurity.digitalwave.hu/debian/ bookworm main
deb http://modsecurity.digitalwave.hu/debian/ bookworm-backports main
```

En suite on installe ModSecurity :

```
apt-get update
apt-get install libnginx-mod-http-modsecurity -y
```

Configuration de ModSecurity

1. Activer ModSecurity dans la directive `server {}` de Nginx :

```
modsecurity on;
modsecurity_rules_file /etc/nginx/modsecurity_includes.conf;
```

2. Modifier `/etc/nginx/modsecurity_includes.conf` en décommentant la ligne commenté de tel sorte que les deux lignes ne sont pas commentés.

3. Dans le fichier `/usr/share/modsecurity-crs/owasp-crs.load` on modifie les directives IncludeOptionnal par Include.
4. Par défaut, ModSecurity détecte mais ne bloque pas les requêtes, pour modifier cela, on active le mode blocage en modifiant `/etc/nginx/modsecurity.conf` et en y remplaçant SecRuleEngine DetectionOnly à On. Ensuite on relance nginx systemctl restart nginx .

ModSecurity : WAF

- **WAF** : Web Application Firewall.
- **Fonctionnalités** :
  - Analyse des requêtes HTTP pour détecter les attaques (SQLi, XSS, etc.).
  - Blocage des requêtes malveillantes.
  - Journalisation des incidents.

Attaques bloquées par ModSecurity

- Injections SQL.
- Cross Site Scripting (XSS).
- Command Injection.
- Scans de vulnérabilités.
- Requêtes malformées.

5. Pour tester si notre WAF fonctionne, dans le PC attaquant on crée une attaque grâce à cette commande :

```
curl -k GET "https://192.168.21.171/index.php?param=<script>alert('XSS')</script
```

NB : C’est l’IP de mon serveur.

Après avoir lancer la commande chez l'attaquant, on vérifie les logs de notre WAF ModSecurity sur notre serveur. Si le WAF fonctionne, la requête sera bloquée (code 403) et un message d'erreur s'affichera comme ceci :

```
root@debian-hacker:~# curl -k GET "https://192.168.21.171/index.php?param=<script>alert('XSS')</script>"
curl: (6) Could not resolve host: GET
<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.22.1</center>
</body>
</html>
```

Il bloque bien l'attaque et si on regarde en temps réel les logs on aura :

```
root@debian-server:~# tail -f /var/log/nginx/error.log

2025/02/28 10:51:12 [error] 2367#2367: *3 [client 192.168.21.172] ModSecurity: Access denied with code 403 (phase 2). Ma
tched "Operator `Ge' with parameter `5' against variable `TX:ANOMALY_SCORE' (Value: `18' ) [file "/usr/share/modsecurity
-crs/rules/REQUEST-949-BLOCKING-EVALUATION.conf"] [line "81"] [id "949110"] [rev ""] [msg "Inbound Anomaly Score Exceede
d (Total Score: 18)"] [data ""] [severity "2"] [ver "OWASP_CRS/3.3.7"] [maturity "0"] [accuracy "0"] [tag "application-m
ulti"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-generic"] [hostname "192.168.21.171"] [uri "/index.php
"] [unique_id "174076147235.610140"] [ref ""], client: 192.168.21.172, server: _, request: "GET /index.php?param=<script
>alert('XSS')</script> HTTP/1.1", host: "192.168.21.171"
```

Notre WAF fonctionne bien. Par la suite on va tester sur une application codé expressément avec les failles.

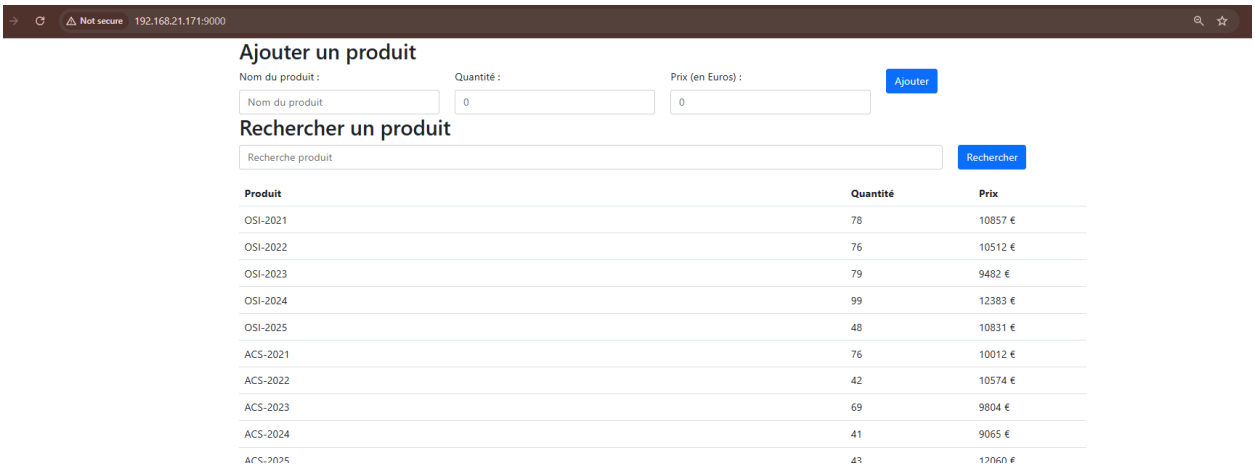
Déploiement de l'application test

L'application en question se trouve sur le Docker hub. Donc au préalable, ayez déjà docker d'installer sur le serveur.

1. On va pull l'image de notre application

```
docker pull scarface05/sec-waf-wazuh
Ensuite:
docker run -d -p 9000:80 scarface05/sec-waf-wazuh:1.0
```

Et si on va sur un navigateur, on saisi : http://192.168.21.171:9000 on aura ceci :



2. Maintenant que l'application est ok, on doit configurer un Reverse Proxy : Elle permet de rediriger les requêtes HTTP reçues par Nginx vers un autre serveur (ou un conteneur Docker, une API, un service backend. Notre cas sera vers un conteneur docker.

- Récupérons d'abord l'IP de notre conteneur grâce à la commande :

```
docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' <I
```

Dans notre cas l'IP est 172.17.0.2 .

Dans la configuration de nginx, on rajoute les lignes suivante dans la partie location :

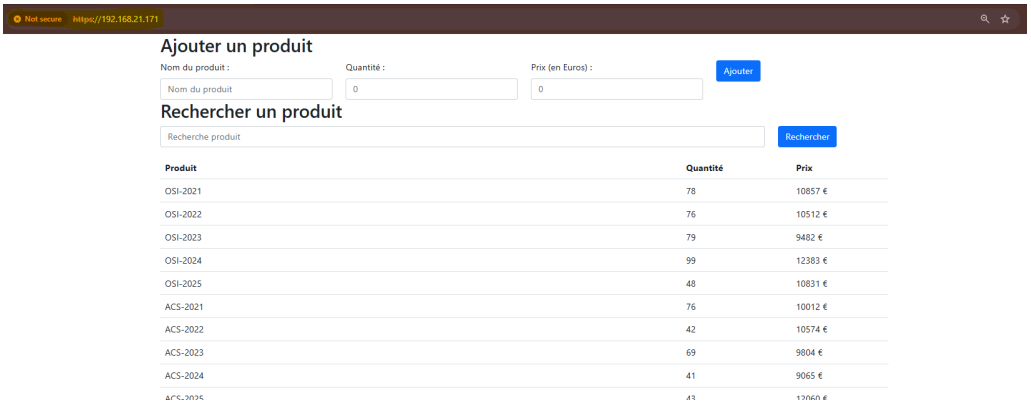
```
location / {
        proxy_pass http://172.17.0.2:80;  # Redirige vers le conteneur Docker
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
```

On active la configuration et on redémarre nginx.

```
nginx -t
systemctl restart nginx
```

```
root@debian-server:~# nginx -t
2025/02/28 11:52:05 [warn] 4708#4708: "ssl_stapling" ignored, issuer certificate not found for certifi
cate "/etc/ssl/certs/nginx.crt"
2025/02/28 11:52:05 [notice] 4708#4708: ModSecurity-nginx v1.0.3 (rules loaded inline/local/remote: 0/
922/0)
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

Une fois que tout est OK, on accède à l'adresse IP de notre serveur http://192.168.21.171 et on aura.



On remarque que le certificat est bien pris en compte.

Maintenant on doit attaquer notre application et voir si le WAF réagi et fait bien son taf.

Sur notre application, on peux ajouter des produits et même rechercher des produits ce qui est idéal pour une attaque XSS et Injection SQL.

Pour le test on va rajouter un produit Clio 4 et on verra qu'il est bien enregistré mais dès lors que l'on insère une balise javascript le WAF la bloque et renvoi une erreur 403 ce qui montre que cela fonctionne très bien comme le montre la vidéo.

attachment:53eb281b-ea3c-4c1e-bba3-e10a8f854463:bandicam_2025-02-28_19-13-11-254.mp4

En conclusion de la première partie, on peut déduire que notre WAF fonctionne au vu des logs de notre serveur web. Pour la suite, on va utiliser un SIEM pour récupérer ces log, les analyser et faire bien d'autre choses.

Pour avoir l'ip de mon conteneur fais ça :

sudo

https://documentation.wazuh.com/current/deployment-options/virtual-machine/virtual-machine.html

3️⃣ Enregistrer l'agent auprès du Wazuh Manager

Sur **ton Wazuh Manager (192.168.21.170)**, récupère la clé de l'agent avec :

```
bash
CopyEdit
/var/ossec/bin/manage_agents
```

- Choisir **"A"** pour ajouter un agent.
- **Nom** : `debian-waf`
- **IP** : `192.168.21.168`
- Une clé sera générée. **Copie-la**.

Sur **ta Debian (192.168.21.168)**, enregistre l'agent avec la clé :

```
bash
CopyEdit
/var/ossec/bin/agent-auth -m 192.168.21.170 -p 1515 -A debian-waf
```

4️⃣ Redémarrer et vérifier l'agent

Démarre l'agent :

```
bash
CopyEdit
systemctl restart wazuh-agent
systemctl enable wazuh-agent
```

Vérifie son statut :

```
bash
CopyEdit
systemctl status wazuh-agent
```

Il doit être **"Active (running)"**.

Ensuite, vérifie la connexion au manager :

```
bash
CopyEdit
/var/ossec/bin/agent_control -l
```

```
root@debian-server:~# systemctl status nginx
● nginx.service - A high performance web server and a reverse proxy server
     Loaded: loaded (/lib/systemd/system/nginx.service; enabled; preset: enabled)
     Active: active (running) since Fri 2025-02-28 07:21:21 CST; 16s ago
       Docs: man:nginx(8)
    Process: 1396 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
    Process: 1397 ExecStart=/usr/sbin/nginx -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
   Main PID: 1422 (nginx)
```

```
root@debian-server:~# tail -f /var/log/nginx/error.log

2025/02/28 10:51:12 [error] 2367#2367: *3 [client 192.168.21.172] ModSecurity: Access denied with code 403 (phase 2). Ma
tched "Operator `Ge' with parameter `5' against variable `TX:ANOMALY_SCORE' (Value: `18' ) [file "/usr/share/modsecurity
-crs/rules/REQUEST-949-BLOCKING-EVALUATION.conf"] [line "81"] [id "949110"] [rev ""] [msg "Inbound Anomaly Score Exceede
d (Total Score: 18)"] [data ""] [severity "2"] [ver "OWASP_CRS/3.3.7"] [maturity "0"] [accuracy "0"] [tag "application-m
ulti"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-generic"] [hostname "192.168.21.171"] [uri "/index.php
"] [unique_id "174076147235.610140"] [ref ""], client: 192.168.21.172, server: _, request: "GET /index.php?param=<script
>alert('XSS')</script> HTTP/1.1", host: "192.168.21.171"
```