Wordpress-WAF

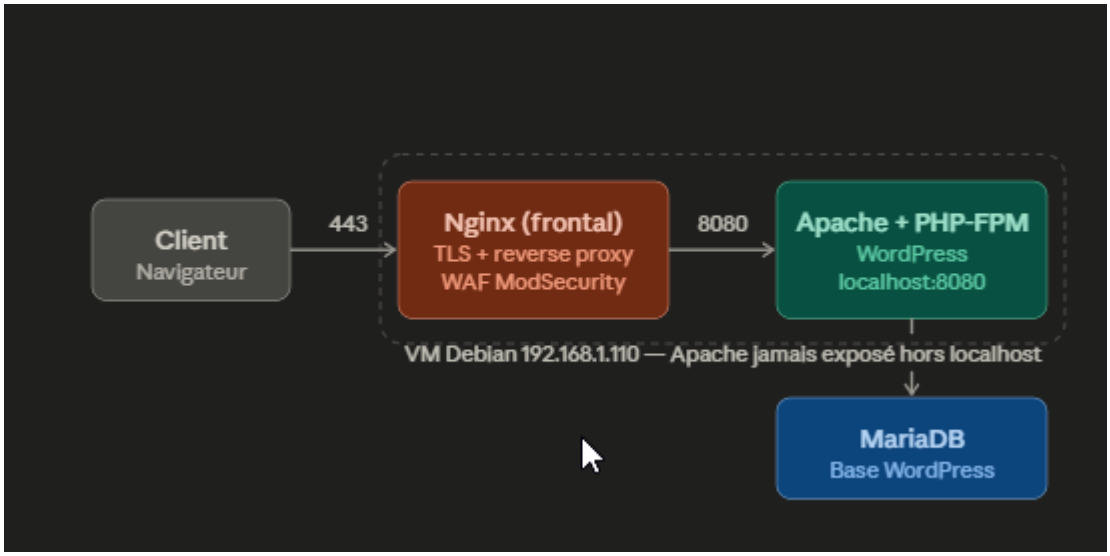
WAF signifie Web Application Firewall.

Contexte du projet (E5)

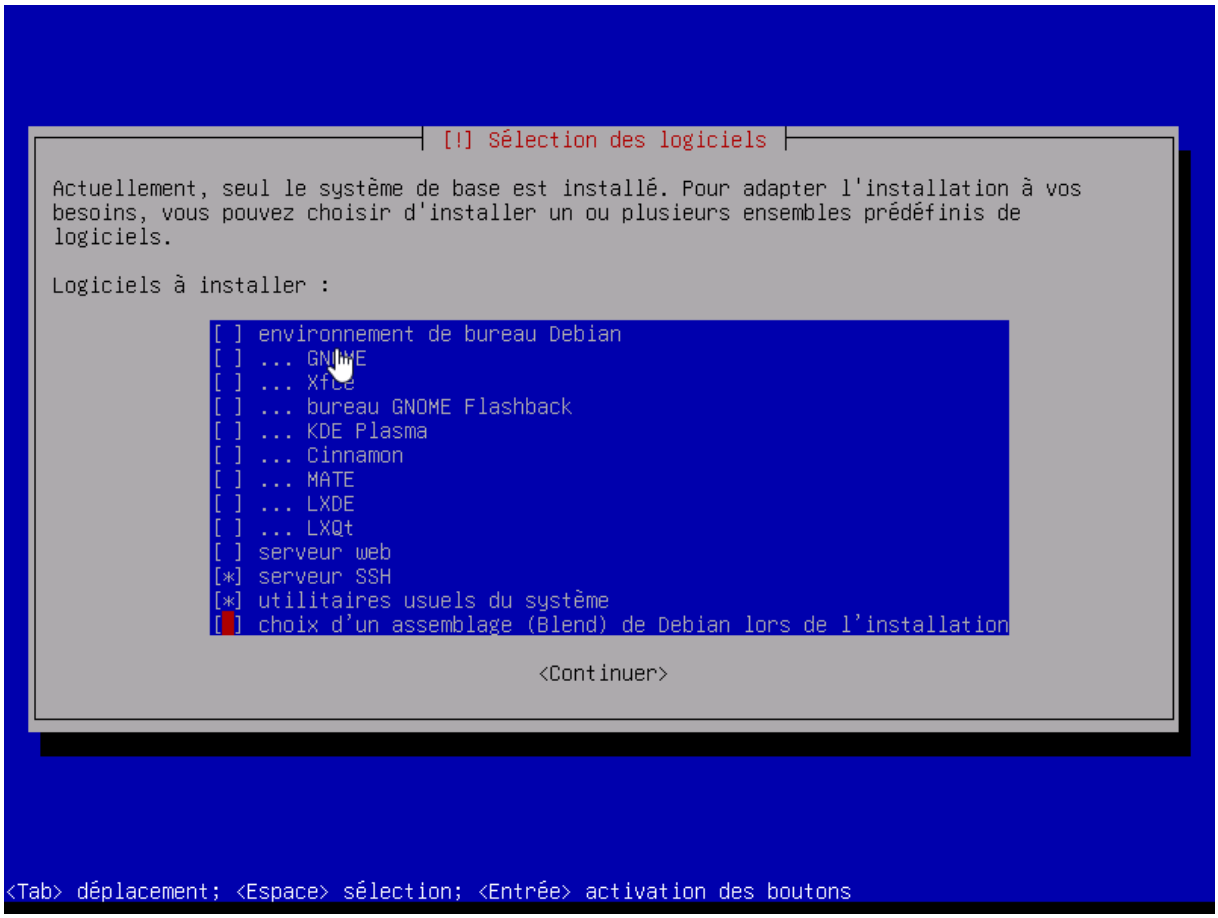
Hébergement d'un site WordPress derrière une architecture défensive en couches : Nginx en frontal (reverse proxy + terminaison TLS), Apache confiné en local comme backend, et ModSecurity (WAF) avec le jeu de règles OWASP CRS pour filtrer les attaques applicatives.

Objectif : exposer une seule porte d'entrée durcie, chiffrer les échanges et bloquer les attaques du Top 10 OWASP avant qu'elles n'atteignent l'application.

Cible : environnement type CHU, où un service web exposé doit être protégé et tracé.



Backend WordPress sur Debian 13



Mise à jour des paquets

On met le système à jour, puis on installe la stack : Apache (serveur web local), MariaDB (base de données), PHP-FPM (exécution PHP en processus séparé) et les extensions PHP requises par WordPress.

sudo apt update && sudo apt upgrade -y

sudo apt install -y apache2 mariadb-server php-fpm php-mysql php-curl php-gd php-xml php-mbstring php-zip php-intl libapache2-mod-fcgid

```
talos@WP-WAF:~$ sudo apt install -y apache2 mariadb-server php-fpm php-mysql php-curl php-gd php-xml p
hp-mbstring php-zip php-intl libapache2-mod-fcgid
```

Créer la base et l'utilisateur WordPress

WordPress stocke tout son contenu en base. On crée une base dédiée et un utilisateur dont les droits sont limités à cette base : si le compte fuite, il ne touche que WordPress, pas le reste du SGBD.

sudo mysql -u root -p

CREATE DATABASE wordpress CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci; CREATE USER 'talos'@'localhost' IDENTIFIED BY '1002';

GRANT ALL PRIVILEGES ON wordpress.* TO 'talos'@'localhost';
FLUSH PRIVILEGES;
EXIT;

```
MariaDB [(none)]> CREATE USER 'talos'@'localhost' IDENTIFIED BY '1002';
Query OK, 0 rows affected (0,001 sec)

MariaDB [(none)]> GRANT ALL PRIVILEGES ON wordpress.* TO 'talos'@'localhost';
Query OK, 0 rows affected (0,001 sec)

MariaDB [(none)]> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0,000 sec)

MariaDB [(none)]> EXIT;
```

Faire écouter Apache sur 127.0.0.1:8080 uniquement, en éditant le fichier /etc/apache2/ports.conf
Apache ne doit plus être joignable directement depuis l'extérieur. On le confine en local (127.0.0.1) : seul Nginx, en frontal, pourra lui parler. C'est le principe du backend caché derrière le proxy.

sudo nano /etc/apache2/ports.conf

```
# If you just change the port or add more port
# have to change the VirtualHost statement in
# /etc/apache2/sites-enabled/000-default.conf

Listen 127.0.0.1:8080

<IfModule ssl_module>
        Listen 443
</IfModule>

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

Activer PHP-FPM avec Apache

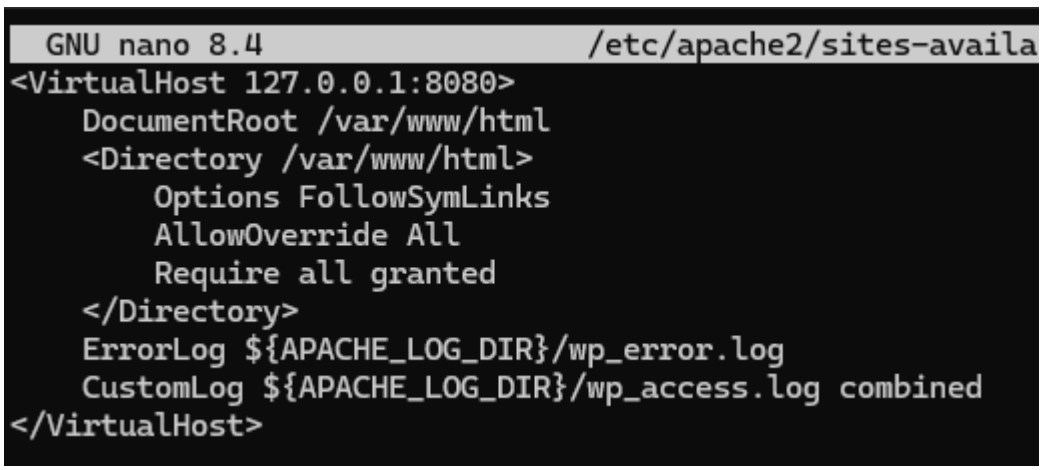
Cette étape charge la configuration qui dit à Apache comment se connecter au socket UNIX ou TCP de PHP-FPM.

sudo a2enmod proxy_fcgi setenvif (module qui permet de parler à PHP-FPM)
sudo a2enconf $(ls /etc/apache2/conf-available/ | grep php | grep fpm | sed 's/.conf//')
sudo a2dismod php* 2>/dev/null Désactive l'ancien module PHP (mod_php)
sudo systemctl restart apache2

On télécharge et installe WordPress
On récupère la dernière version officielle, on la place dans la racine web, puis on fixe le propriétaire (www-data) et des permissions restrictives (750 dossiers, 640 fichiers) : le serveur lit, personne d'autre n'écrit.

cd /tmp
wget https://wordpress.org/latest.tar.gz
tar -xzf latest.tar.gz
sudo cp -r wordpress/* /var/www/html/
sudo chown -R www-data:www-data /var/www/html/
sudo find /var/www/html/ -type d -exec chmod 750 {} \;
sudo find /var/www/html/ -type f -exec chmod 640 {} \;

Configuration du VirtualHost avec sudo nano /etc/apache2/sites-available/wordpress.conf
Le VirtualHost indique à Apache quel dossier servir et sur quel port répondre (8080, en local uniquement).



sudo nano /etc/apache2/ports.conf

Activation du site
On désactive le site par défaut et on active celui de WordPress, puis on charge le module rewrite pour les URL propres (permaliens).

sudo a2dissite 000-default.conf

```
sudo a2ensite wordpress.conf
sudo a2enmod rewrite
sudo systemctl restart apache2
```

```
talos@WP-WAF:~$ curl -I http://127.0.0.1:8080
HTTP/1.1 200 OK
Date: Thu, 28 May 2026 17:27:07 GMT
Server: Apache/2.4.67 (Debian)
Last-Modified: Thu, 28 May 2026 16:28:32 GMT
ETag: "29cf-652e33816a7aa"
Accept-Ranges: bytes
Content-Length: 10703
Vary: Accept-Encoding
Content-Type: text/html
```

Nginx en frontal (reverse proxy)

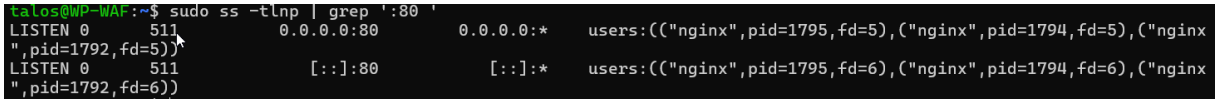
Nginx devient le seul point d'entrée réseau. Il reçoit le trafic et le transmet à Apache en local.

sudo apt install -y nginx

Vérification des ports en écoute :

On vérifie qui écoute sur le port 80 avant de basculer : on veut Nginx devant et Apache replié en local.

sudo ss -tlnp | grep ':80 '



Création du VirtualHost Nginx (reverse proxy)

On définit le point d'entrée public : Nginx reçoit les requêtes et les relaie vers Apache sur 127.0.0.1:8080 (proxy_pass).

```
server {
    listen 80;
    server_name _;

    client_max_body_size 64M;

    location / {
        proxy_pass http://127.0.0.1:8080;
        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;
        proxy_read_timeout 300;
    }
}
```

Activation du site et désactivation de la configuration par défaut

Lien symbolique pour activer la conf, suppression du site par défaut pour ne garder qu'un seul point d'entrée, puis nginx -t pour valider la syntaxe avant rechargement.

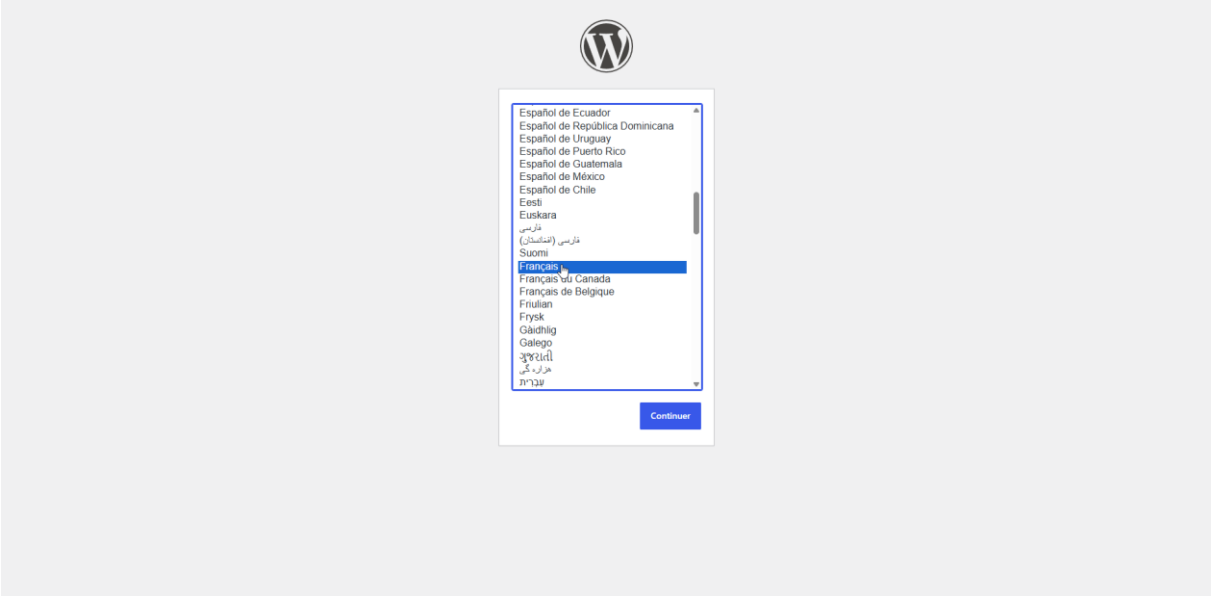
sudo ln -s /etc/nginx/sites-available/wordpress /etc/nginx/sites-enabled/ (lien symbolique)
sudo rm /etc/nginx/sites-enabled/default
sudo nginx -t

```
talos@WP-WAF:~$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

sudo systemctl restart nginx

On va maintenant indiquer à WordPress qu'il est derrière le proxy
Derrière un reverse proxy en TLS, WordPress doit savoir que la requête d'origine est en HTTPS. Sinon il génère des liens en HTTP et casse la navigation (boucles de redirection).

sudo nano /var/www/html/wp-config.php



Vous devez saisir ci-dessous les détails de connexion à votre base de données. Si vous ne les connaissez pas, contactez votre hébergeur.

Nom de la base de données

wordpress

Le nom de la base de données avec laquelle vous souhaitez utiliser WordPress.

Identifiant

talos

Votre identifiant MySQL.

Mot de passe

••••

Afficher

Votre mot de passe de base de données.

Adresse de la base de données

localhost

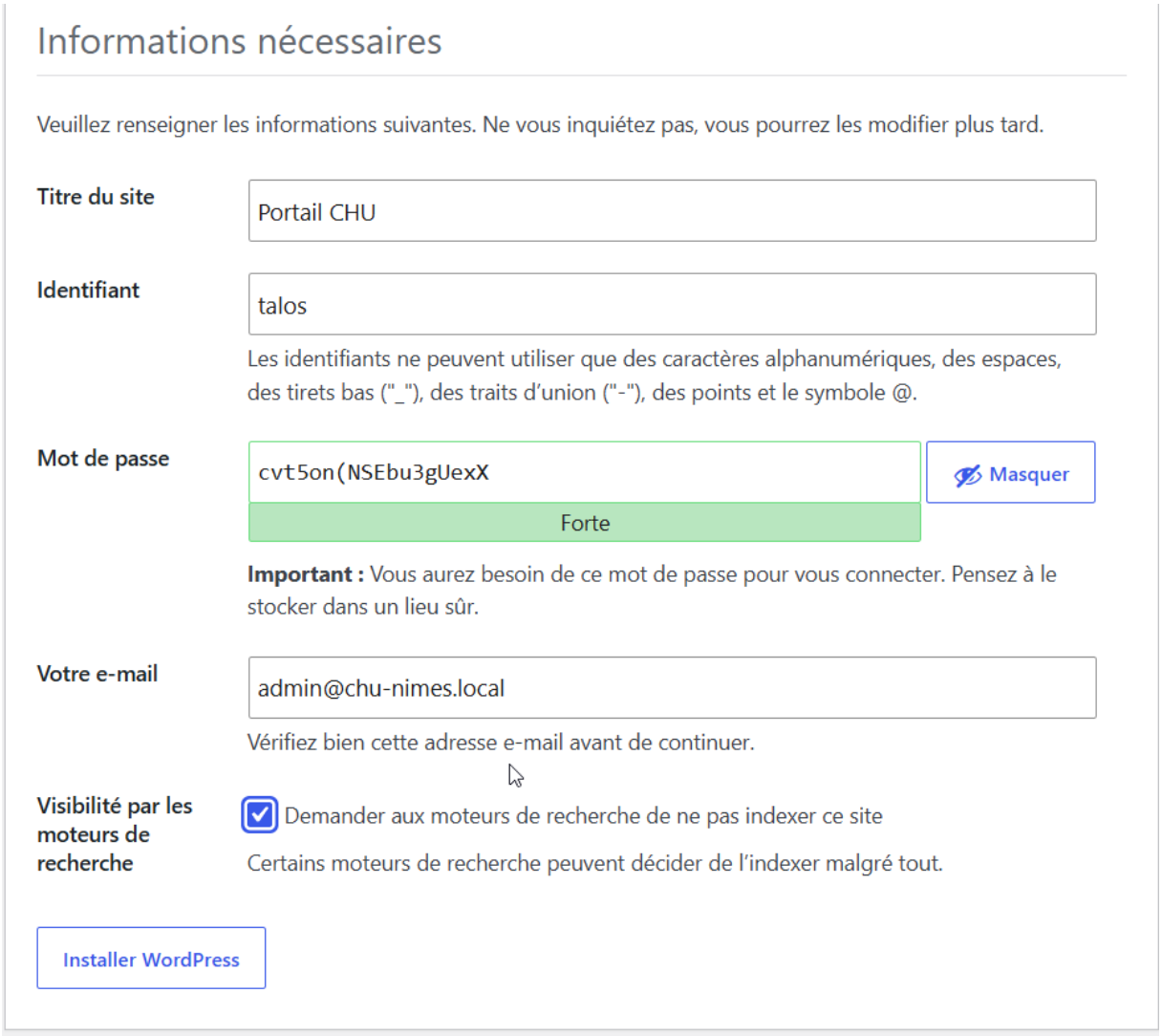
Si `localhost` ne fonctionne pas, demandez cette information à l'hébergeur de votre site.

Préfixe des tables

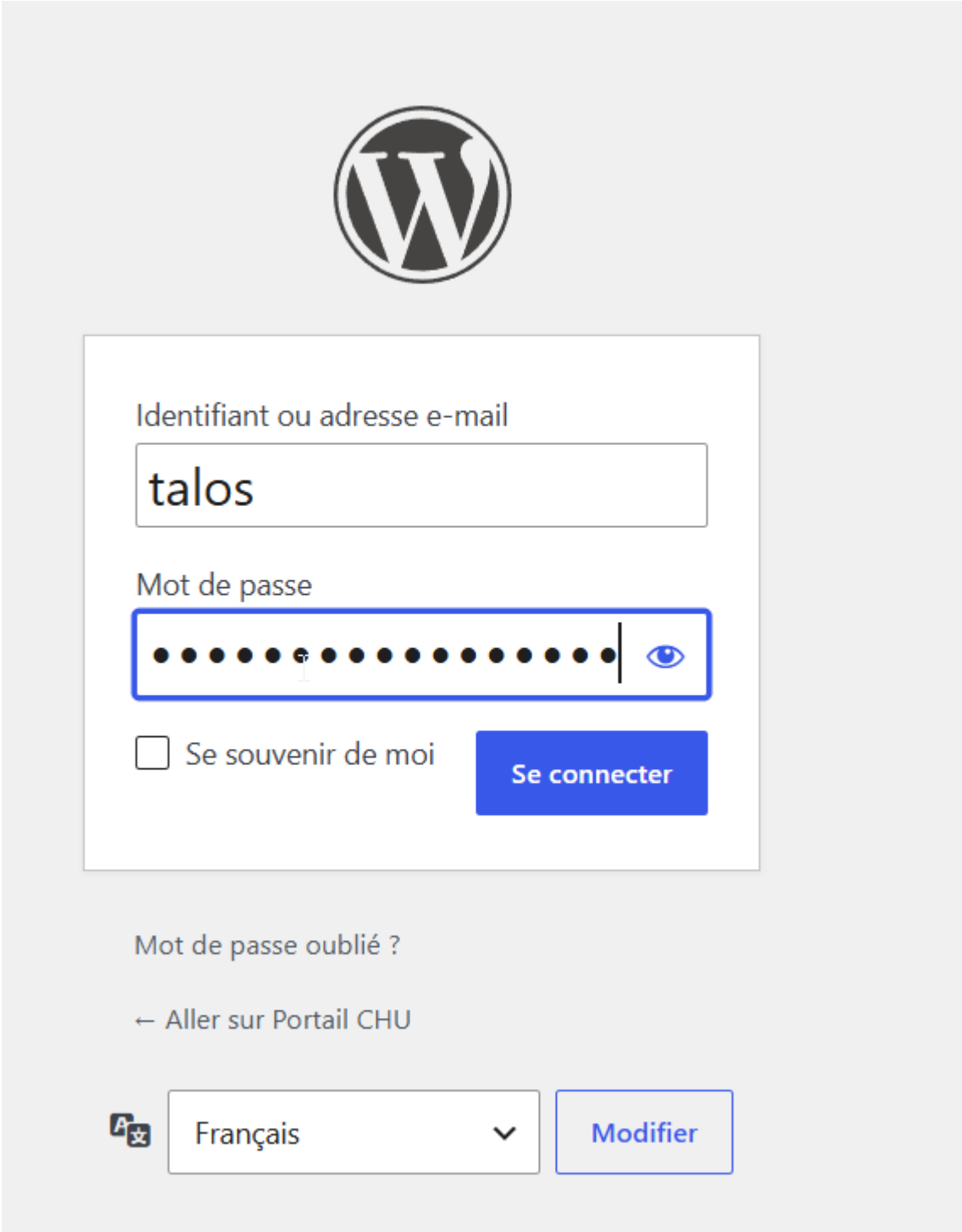
wp_

Si vous souhaitez faire tourner plusieurs installations de WordPress sur une même base de données, modifiez ce réglage.

Envoyer



Le mot de passe fort : cvt5on(NSEbu3gUexX



Nginx termine le TLS et transmet la requête à Apache **en HTTP** (sur 127.0.0.1:8080)

```
/* Add any custom values between this line and the "stop editing" line. */

if (isset($_SERVER['HTTP_X_FORWARDED_PROTO']) && $_SERVER['HTTP_X_FORWARDED_PROTO'] === 'https') {
    $_SERVER['HTTPS'] = 'on';
}

/* That's all, stop editing! Happy publishing. */
```

sudo ls -la /var/www/html/wp-config.php

```
talos@WP-WAF:~$ sudo grep -E "DB_NAME|DB_USER|DB_HOST" /var/www/html/wp-config.php
define( 'DB_NAME', 'wordpress' );
define( 'DB_USER', 'talos' );
define( 'DB_HOST', 'localhost' );
```

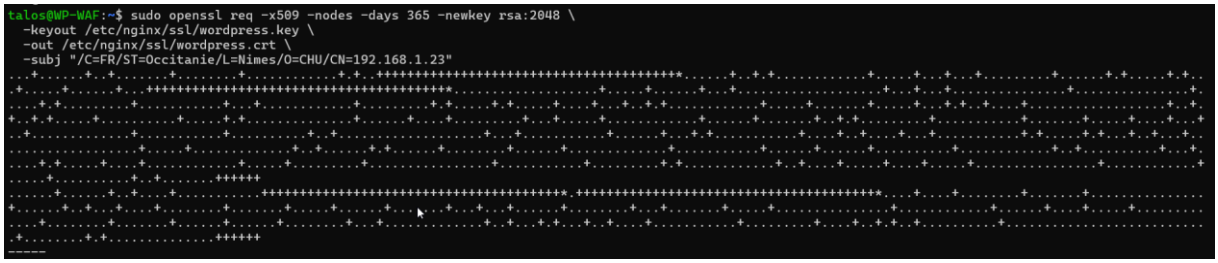
wp-config.php contient les identifiants de la base en clair. C'est le fichier le plus sensible de l'installation.

Durcissement des permissions
On restreint wp-config.php au strict minimum (lecture par www-data uniquement) : c'est le fichier qui contient les accès à la base, il ne doit jamais être lisible par tous.

```
talos@WP-WAF:~$ sudo chmod 640 /var/www/html/wp-config.php
talos@WP-WAF:~$ sudo chown www-data:www-data /var/www/html/wp-config.php
talos@WP-WAF:~$ sudo ls -la /var/www/html/wp-config.php
-rw-r----- 1 www-data www-data 3741 29 mai   15:54 /var/www/html/wp-config.php
talos@WP-WAF:~$
```

Génération du certificat auto-signé
Pour activer HTTPS sans autorité de certification, on génère un certificat auto-signé. En production réelle, il serait remplacé par un certificat d'une CA interne ou publique.

```
sudo mkdir -p /etc/nginx/ssl
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
  -keyout /etc/nginx/ssl/wordpress.key \
  -out /etc/nginx/ssl/wordpress.crt \
  -subj "/C=FR/ST=Occitanie/L=Nimes/O=CHU/CN=192.168.1.23"
```



Reconfiguration du VirtualHost Nginx pour le TLS
On bascule Nginx en écoute sur le 443, on charge le certificat et la clé, et on force la redirection HTTP vers HTTPS.

```
sudo nano /etc/nginx/sites-available/wordpress
```

```
 GNU nano 8.4                                                    /etc/nginx
# Redirection HTTP → HTTPS
server {
    listen 80;
    server_name _;
    return 301 https://$host$request_uri;
}

# Serveur HTTPS (reverse proxy + TLS)
server {
    listen 443 ssl;
    server_name _;

    ssl_certificate     /etc/nginx/ssl/wordpress.crt;
    ssl_certificate_key /etc/nginx/ssl/wordpress.key;

    # Durcissement TLS de base
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;

    client_max_body_size 64M;

    location / {
        proxy_pass http://127.0.0.1:8080;
        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;
        proxy_read_timeout 300;
    }
}
```

On teste et recharge Nginx
nginx -t valide la configuration avant tout reload : on ne recharge jamais une conf non testée en production.

sudo nginx -t

```
talos@WP-WAF:~$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

sudo systemctl reload nginx

Mise à jour de l'URL dans :
On aligne les URL de WordPress sur https:// pour rester cohérent avec le TLS terminé par Nginx.

sudo nano /var/www/html/wp-config.php

```
/* Add any custom values between this line and the "stop editing" line. */

if (isset($_SERVER['HTTP_X_FORWARDED_PROTO']) && $_SERVER['HTTP_X_FORWARDED_PROTO'] === 'https') {
    $_SERVER['HTTPS'] = 'on';
}
define('WP_HOME', 'https://192.168.1.23');
define('WP_SITEURL', 'https://192.168.1.23');
```

Vérification de la redirection :

```
talos@WP-WAF:~$ curl -sI http://192.168.1.23 | grep -i location
Location: https://192.168.1.23/
talos@WP-WAF:~$
```

Le site est bien en HTTPS

Partie WAF

Le WAF (ModSecurity) inspecte le trafic HTTP applicatif avant qu'il n'atteigne WordPress, pour filtrer les attaques web (injections, XSS, etc.).

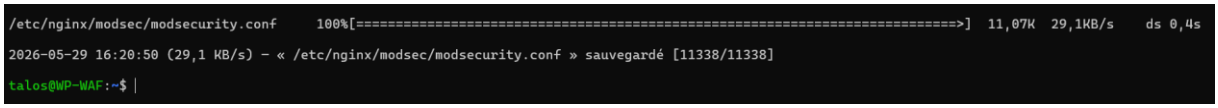
sudo apt install -y libnginx-mod-http-modsecurity

Activation de ModSecurity dans Nginx

Récupération du fichier de conf de ModSecurity

On part de la configuration recommandée par l'éditeur comme base saine.

sudo mkdir -p /etc/nginx/modsec
sudo cp /etc/modsecurity/modsecurity.conf-recommended /etc/nginx/modsec/modsecurity.conf 2>/dev/null || \
  sudo wget -O /etc/nginx/modsec/modsecurity.conf \
  https://raw.githubusercontent.com/owasp-modsecurity/ModSecurity/v3/master/modsecurity.conf-recommended



Création du fichier de configuration :
main.conf est le point d'entrée des règles : il inclut la conf de base, puis plus tard le jeu de règles CRS.

sudo nano /etc/nginx/modsec/main.conf qui contiendra Include /etc/nginx/modsec/modsecurity.conf

Démarrage en mode DetectionOnly
On démarre en observation seule : ModSecurity journalise ce qu'il bloquerait, sans rien bloquer. On valide d'abord l'absence de faux positifs avant de passer en blocage réel.

sudo nano /etc/nginx/modsec/modsecurity.conf

```
# Enable ModSecurity, attaching it to every transaction. Use detection
# only to start with, because that minimises the chances of post-installation
# disruption.
#
SecRuleEngine DetectionOnly
```

On va brancher ModSecurity sur le VirtualHost HTTPS
On active le moteur sur le site et on pointe vers le fichier de règles.

sudo nano /etc/nginx/sites-available/wordpress

On ajoute après server_name
modsecurity on;
modsecurity_rules_file /etc/nginx/modsec/main.conf;

```
# Serveur HTTPS (reverse proxy + TLS)
server {
    listen 443 ssl;
    server_name _;
    modsecurity on;
    modsecurity_rules_file /etc/nginx/modsec/main.conf;
```

```
talos@WP-WAF:~$ sudo nginx -t
2026/05/29 16:45:40 [notice] 3722#3722: ModSecurity-nginx v1.0.3 (rules loaded inline/local/remote: 0/7/0)
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

sudo systemctl reload nginx

Installation de l'OWASP Core Rule Set (CRS)

Le CRS est l'ensemble de règles open-source qui détecte les attaques du Top 10 OWASP.

Vérification si le CRS est packagé sur Debian 13

On privilégie le paquet de la distribution : maintenu et mis à jour en même temps que le système.

```
talos@WP-WAF:~$ apt-cache policy modsecurity-crs
modsecurity-crs:
  Installé : 3.3.7-1+deb13u2
  Candidat : 3.3.7-1+deb13u2
 Table de version :
 *** 3.3.7-1+deb13u2 500
        500 http://deb.debian.org/debian trixie/main amd64 Packages
        100 /var/lib/dpkg/status
     3.3.7-1+deb13u1 500
        500 http://security.debian.org/debian-security trixie-security/main amd64 Packages
```

Branchement du CRS dans /etc/nginx/modsec/main.conf

On inclut le CRS après la conf de base pour que ses règles s'appliquent au trafic.

```
# Moteur ModSecurity de base
Include /etc/nginx/modsec/modsecurity.conf

# Configuration CRS
Include /etc/modsecurity/crs/crs-setup.conf

# Exclusions AVANT les règles principales (dont WordPress)
Include /etc/modsecurity/crs/REQUEST-900-EXCLUSION-RULES-BEFORE-CRS.conf

# Règles principales du CRS
Include /usr/share/modsecurity-crs/rules/*.conf

# Exclusions APRÈS les règles principales
Include /etc/modsecurity/crs/RESPONSE-999-EXCLUSION-RULES-AFTER-CRS.conf
```

Activer les exclusions WordPress dans sudo nano /etc/modsecurity/crs/crs-setup.conf

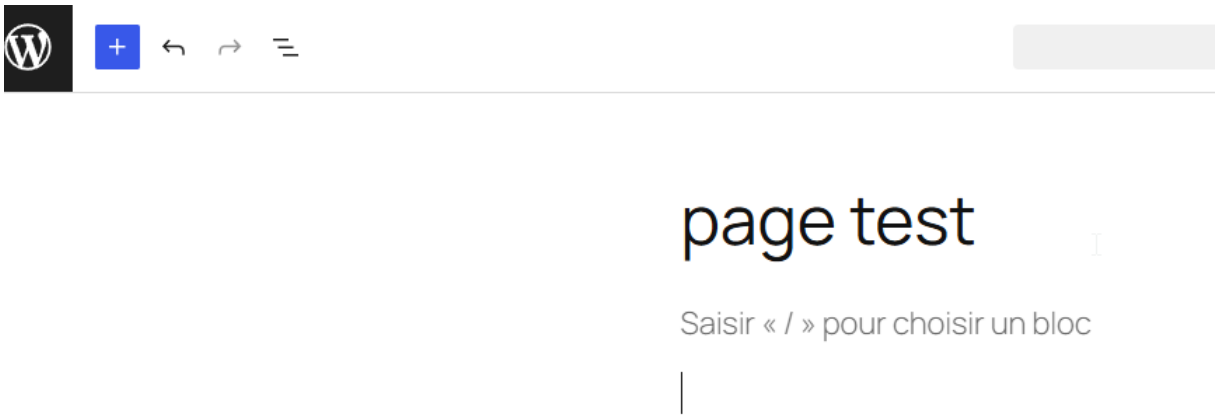
```
#
# Uncomment this rule to change the default:
#
SecAction \
  "id:900000,\
   phase:1,\
   nolog,\
   pass,\
   t:none,\
   setvar:tx.paranoia_level=1"
```

WordPress déclenche naturellement certaines règles (admin, éditeur de blocs). On active les exclusions prévues pour limiter les faux positifs.

Examen des logs avec sudo tail -f /var/log/modsec_audit.log

Création d’une page dans WordPress et vérification des logs

On génère du trafic légitime (création d'une page) et on observe l'audit log pour vérifier le comportement du WAF.



Les logs s’affichent lors de la création de la page

```
---pDztaYPx---F--
HTTP/1.1 200
Access-Control-Allow-Headers: Authorization, X-WP-Nonce, Content-Disposition, Content-MD5, Content-Type
Link: <https://192.168.1.23/wp-json/>; rel="https://api.w.org/"
Content-Type: application/json; charset=UTF-8
Allow: GET, POST
Connection: keep-alive
X-WP-Nonce: d8497280a2
X-Content-Type-Options: nosniff
Access-Control-Expose-Headers: X-WP-Total, X-WP-TotalPages, Link
Date: Fri, 29 May 2026 15:40:46 GMT
Access-Control-Allow-Methods: OPTIONS, GET, POST, PUT, PATCH, DELETE
X-Robots-Tag: noindex
Server: nginx
Access-Control-Allow-Credentials: true
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Cache-Control: no-cache, must-revalidate, max-age=0, no-store, private
Access-Control-Allow-Origin: https://192.168.1.23
Vary: Origin
```

Figure 1: autorisation dans les logs

Bascule en blocage réel

Une fois les faux positifs écartés, on passe SecRuleEngine de DetectionOnly à On : ModSecurity bloque désormais réellement.

Changement de la règle SecRuleEngine : detection → ON

```
# -- 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 DetectionOnly
```

```
# Enable ModSecurity, attaching it to every transaction. Use detection
# only to start with, because that minimises the chances of post-installation
# disruption.
#
SecRuleEngine On
```

Puis sudo nginx -t && sudo systemctl reload nginx

On va tester une injection XSS et vérifier qu'elle est rejetée

Test offensif : la charge XSS doit être rejetée (HTTP 403).

```
talos@WP-WAF:~$ curl -k -I "https://192.168.1.23/?s=<script>alert(1)</script>"
HTTP/1.1 403 Forbidden
Server: nginx
Date: Fri, 29 May 2026 15:45:44 GMT
Content-Type: text/html
Content-Length: 146
Connection: keep-alive
```

Test d'une requête légitime qui passe

Test de non-régression : une requête normale doit continuer à passer. Un WAF utile bloque les attaques sans casser l'usage.

```
talos@WP-WAF:~$ curl -k -I "https://192.168.1.23/"
HTTP/1.1 200 OK
Server: nginx
Date: Fri, 29 May 2026 15:46:57 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Link: <https://192.168.1.23/wp-json/>; rel="https://api.w.org/"
```