SQL injection exercise

Offensive Technologies 2021
Matteo Franzil <matteo.franzil+github@gmail.com>

December 11, 2021

1 Solution

To solve the exercise, I first connected with SSH tunneling to the main server:

```
ssh -L 8118:server.franzil-sqli.offtech:80 otech2af@users.deterlab.net
```

With port forwarding now enabled, I visited the website on my browser and solved the points. All SQL injections shown here use the user ID field, whereas the password one was filled with random gibberish (required by the code, but not used).

1. Show how you can log into a single account without knowing any id numbers ahead of time.

```
( SELECT max(id) FROM accounts LIMIT 1 );--
```

Code 1 Code for the first point.

2. Show how you can log into every account (one at a time) without knowing any id numbers ahead of time.

```
( SELECT id FROM accounts LIMIT 1 OFFSET n);--
```

Code 2 Code for the second point.

3. Make some account (your choice) wire its total balance to the bank with routing number: 314159265 and account number: 271828182845

```
( SELECT id FROM accounts LIMIT 1 OFFSET 20);--
```

Code 3 Code for the third point. With this, we get access to a random account.

We get access to Camille Cantu's account, with id `211`. We can now get access and avoid future password checks with `211` as user ID and `' OR '1'='1` as password. This is required since wiring balances does a password check, and so we need the code not to bother about our lack of knowledge about the password. The screenshot section shows a successful wire transfer with this account.

4. Explain why you can't create a new account or arbitrarily update account balances (or show that you can). Due to how the code is written - more specifically, the `mysqli.query` method, it cannot accept more than a query per statement. This limits the potentials of attacks, and only allows query modification rather than fully query injection. So, in short, no update, no drops, and else. Just to be sure, we can try to inject this:

```
211; update accounts set bal=100000 where id=211;--
```

Code 4 Code for our tentative injection.

This returns a nice `500` error on the Apache2 logs, although it seems that the error may also be related to how the code is written itself. Indeed, we get this:

```
root@server:~# cat /var/log/apache2/error.log
[Fri Oct 01 13:46:24.359863 2021] [php7:error] [pid 11192]
[client 192.168.253.1:58603] PHP Fatal error: Uncaught Error:
Call to undefined method mysqli::error() in /usr/lib/cgi-bin/FCCU.php:37
Stack trace:\n#0 {main}\n thrown in /usr/lib/cgi-bin/FCCU.php on line 37,
referer: http://localhost:8118/cgi-bin/FCCU.php
```

Code 5 Apache server logs.

More on this question can be read in the following section.

2 Code issues and comments

As anticipated in the previous section, the code seems to have some major issues throughout. Some of them are pretty obvious and even pointed by comments: for example, there is a potential for integer overflow during bank transfers and subtraction of funds in the code. The major ones are, of course, related to SQL injections and the lack of a proper usage of prepared statements. In order to make the code really work and output actual issues instead of confused error messages, I removed all `or die(mysql.error())` instances and added `mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT);`, as suggested by https://stackoverflow.com/questions/15318368/mysqli-or-die-does-it-have-to-die/15320411#15320411. Finally, reducing the permissions given to the database user wired to the code would probably be a good idea.

3 Screenshots

Some screenshots are included in this section.

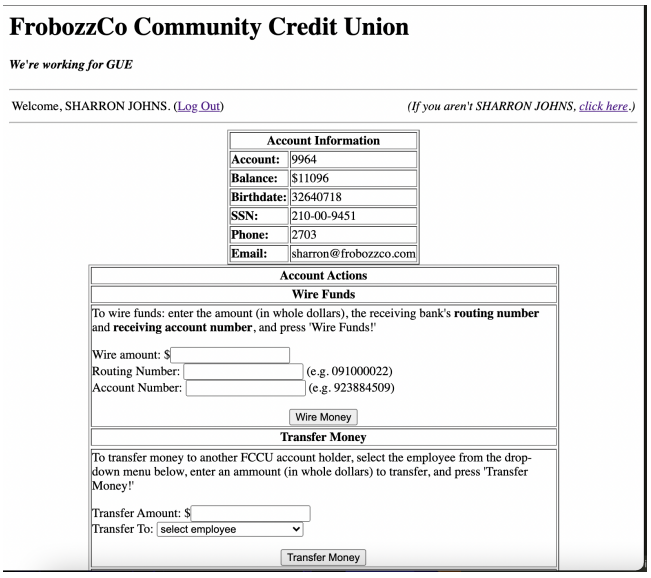

Figure 1 Example of homepage of a random employee.

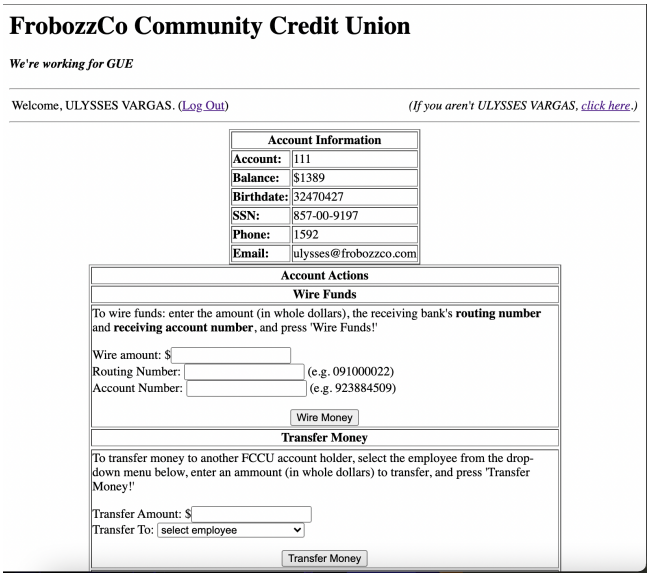

Figure 2 Example of homepage of another random employee.

Wire of $8499 to bank (314159265) account (271828182845) complete.

Figure 3 Successful wire transfer.

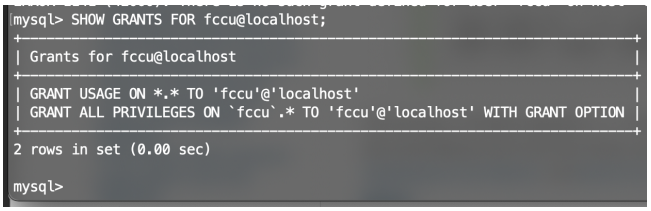

Figure 4 Screenshot of the permissions given to the database user.