Booster Dose

Problem Type

SQL Injection, Privilege Escalation, Authentication Bypass, UNION SQL Injection, Hash Collision Attack

---

Solution

Step 1 - Investigation:

Inspecting source code of webpage reveals a fishy `/records` page.

```
<div class="collapse navbar-collapse" id="navbar">
  <div class="navbar-nav">
    <a class="nav-item nav-link" id="home" href="/">Home</a>
    <a class="nav-item nav-link" id="login" href="/login">Login</a>
    <a class="nav-item nav-link" id="records" href="/records">Records</a>
  </div>
</div>
```

Step 2 - UNION SQL Injection:

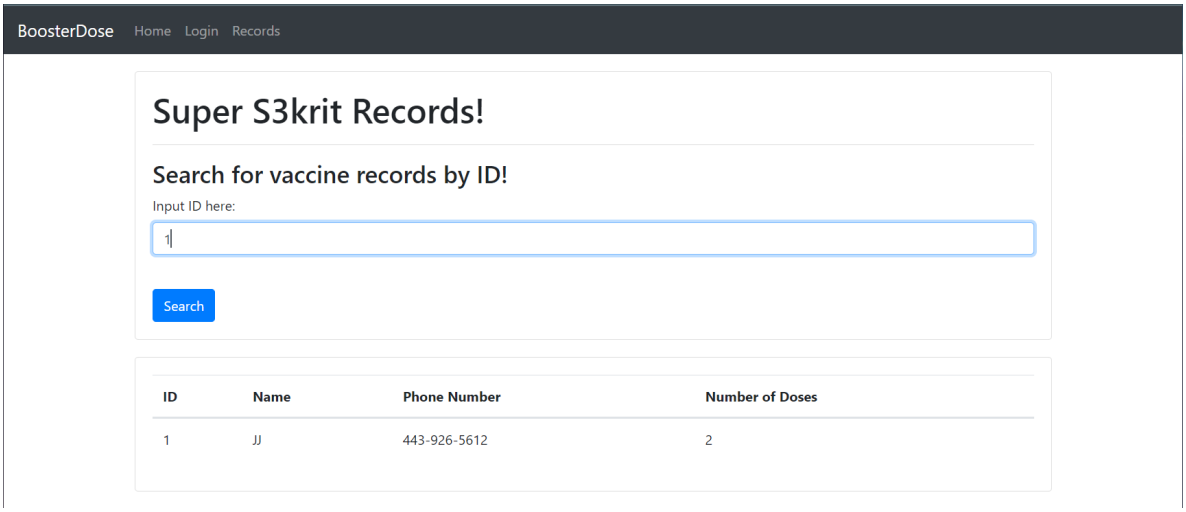
Searching of Records Database at `~/records` is vulnerable to Union based SQL Injection. Through the `UNION SELECT` operator, we are able to pipe a second `SELECT` SQL Query, and we can use this second select to leak valuable information on the database.

We first see how many columns there are in the current table through enumeration:

- `1' UNION SELECT 1,2,3,4,5;--` returns false (an error)
- `1' UNION SELECT 1,2,3,4;--` returns true

Hence, we deduce that in our current table that we are querying, there are 4 columns.

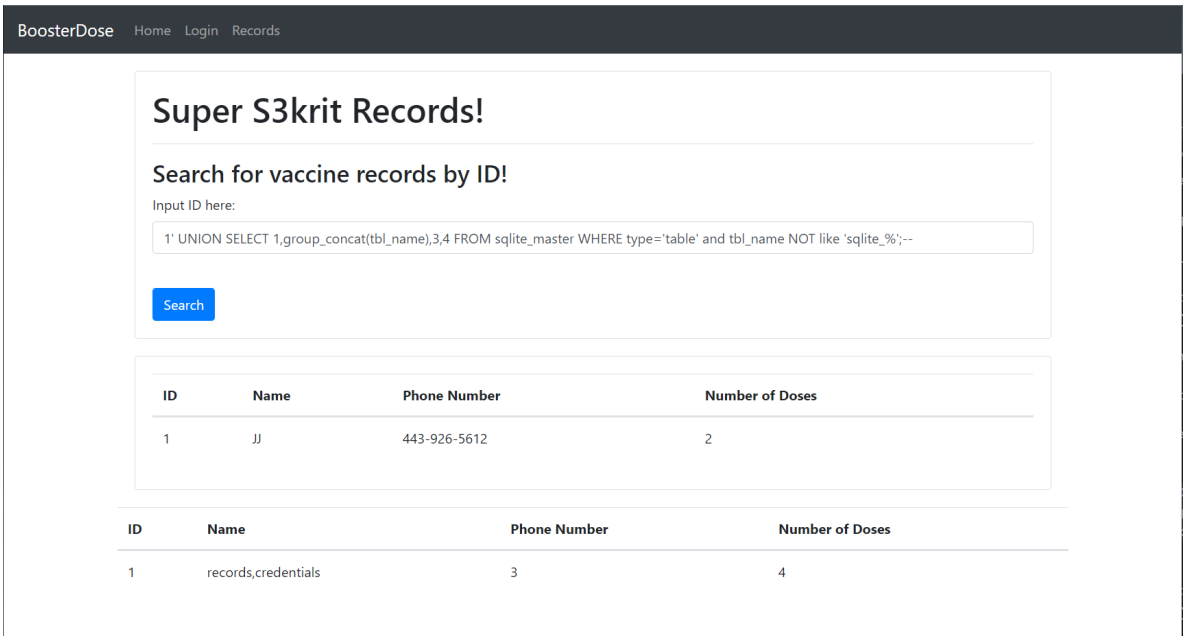
After querying for the standard entries, we see column 1 is ID, while column 2 is `name`, which is most likely a `TEXT` SQL Column.



With this `TEXT` column, we can craft and inject a `UNION SELECT` payload to print the table names in the database inside column 2.

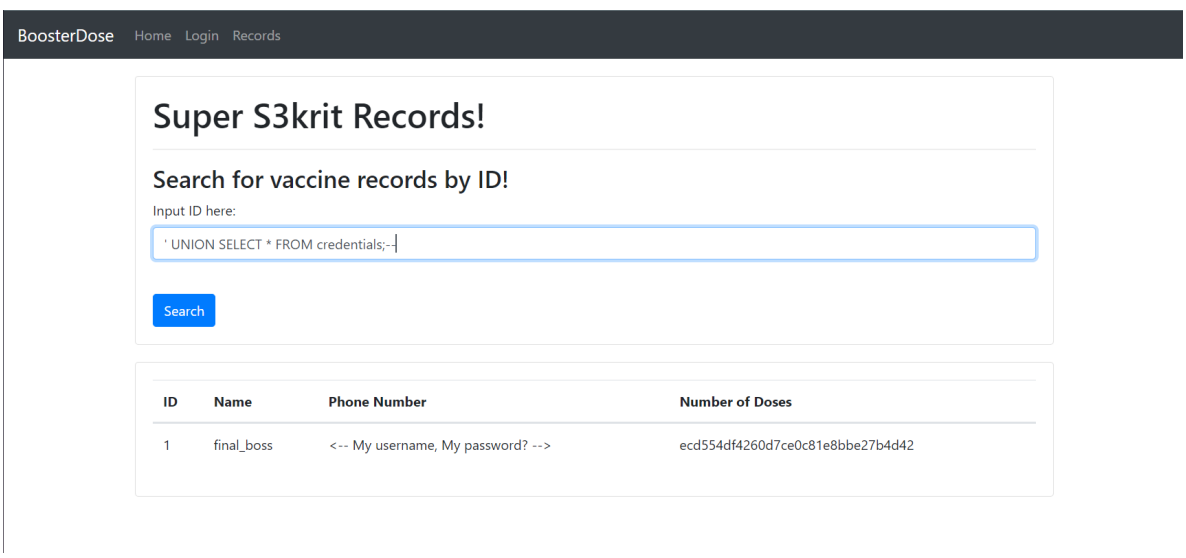
We inject:

```
1' UNION SELECT 1,group_concat(tbl_name),3,4
FROM sqlite_master WHERE type='table' and tbl_name NOT like 'sqlite_%';--
```



We obtain the table names - `records` and `credentials`. Let's find out what's inside the `credentials` table! It looks suspicious!

We then run `' UNION SELECT * FROM credentials;--` to find `final_boss` as admin username and `ecd554df4260d7ce0c81e8bbe27b4d42` as admin password.



Step 3: Hash Collision Attack:

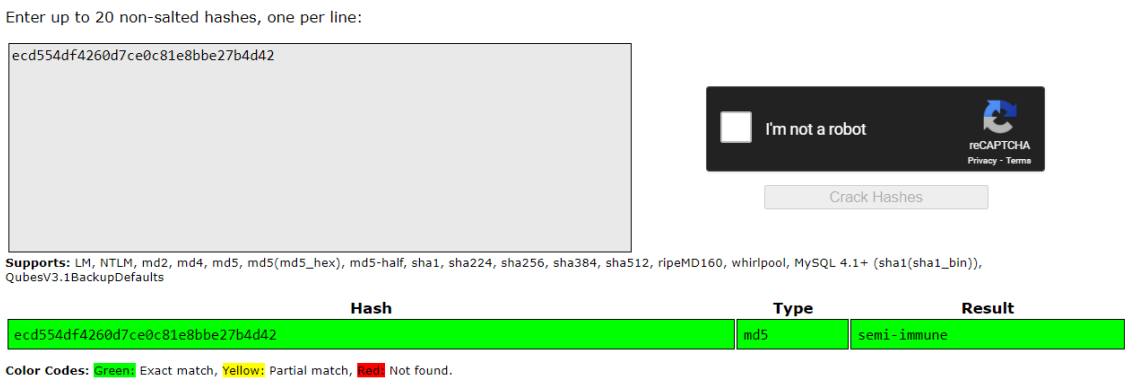
We then visit the login page at `/login`.

We input the credentials:

```
username: final_boss
password: ecd554df4260d7ce0c81e8bbe27b4d42
```

However, the login portal continues to return `Incorrect username or password, try again`. We note that `ecd554df4260d7ce0c81e8bbe27b4d42` is 32 hexadecimal digits long, and resembles that of an MD5 hash.

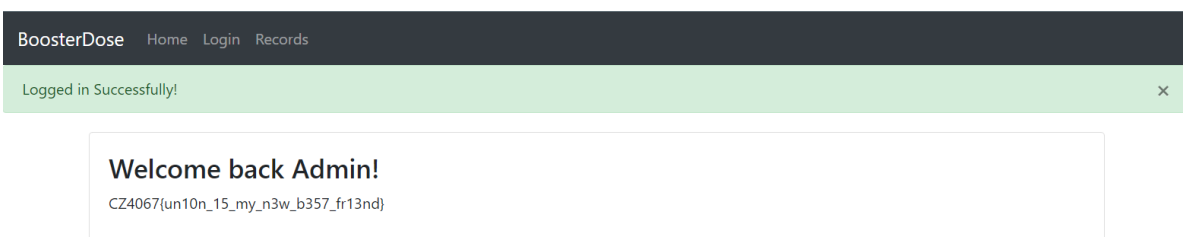
MD5 hashes are vulnerable to hash collision attacks, where tools can be used to find passwords from a wordlist that produces the same hash value. We visit `crackstation.net` and attempt to crack the hash.



We obtain the real password `semi-immune` !

We then input the following credentials into the login portal to obtain the flag:

```
username: final_boss
password: semi-immune
```



Flag

`CZ4067{un10n_15_my_n3w_b357_fr13nd}`