Elinor Holt

Prof. Goldberg

CS 357

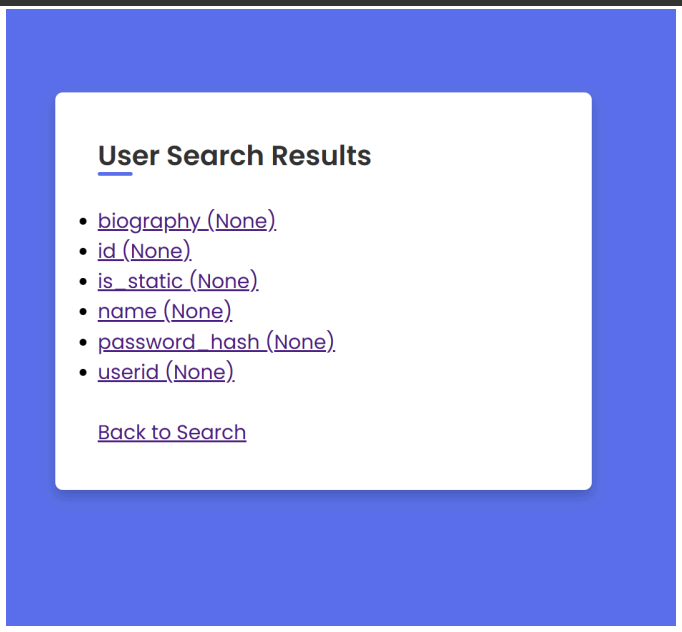
Nov 1, 2024

Challenge 7: Proper SHA256, but Still Vulnerable

I. Recon:

A. I found that the columns differed from the previous challenge. I used the following SQL injection in the search field to retrieve column names within the members table:

```
unique" UNION SELECT name, NULL FROM pragma_table_info('members'); -
```

B. 

C. The instructions hinted that reversing the hash for the admin password was not possible. This suggested that the solution likely involved SQL injection rather than brute-forcing or hash cracking. I figured that one of two types of SQL injection might work:

1. Either an SQL injection on the login form to bypass authentication by making the query evaluate to TRUE.
2. Or, an SQL injection on another form that could modify the admin's password hash directly.

D. After testing all the forms for SQL injection, I found that the registration form was vulnerable to SQL injection.

II. Exploit Instructions (Automated in code):

A. Since the registration form is vulnerable to SQL injection, I guessed at how the backend might be structured during account creation. When a new account is created, the backend might execute an INSERT INTO statement like this:

```
INSERT INTO members (name, user_id, password_hash)
VALUES (<name>, <username>, <password>);
```

B. Since the <name> input is not sanitized, it makes it possible to inject additional SQL statements. I found a way to add an UPDATE query after the INSERT statement, which modifies the admin's password_hash directly. By using the known hash for the password "password" (5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8), I could set the admin's password hash to this value. It's also important that this exploit matches the backend's expected format, with three values (for name, username, and password). Since the first past of the INSERT INTO statement is set up to receive three values, it will cause an error if we do not provide them, however the values we use do not matter as long as there is not already a member with the same username.

C. Also, when we submit the SQL query, the two other fields on the register page must be filled out. These forms are probably checked by the HTML.

D. I used this SQL statement in the “name” field:

```
souljaboy" , "tellem" , "crankthat" ); UPDATE members SET password_hash =
"5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8" WHERE userid =
"admin"; --
```

E. This results in the following SQL commands being executed on the backend:

```
INSERT INTO members (name, user_id, password_hash)
VALUES (“souljaboy” , “tellem” , “crankthat” );
UPDATE members SET password_hash =
“5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8”
WHERE userid = “admin”; -- , , );
```

F. The injected UPDATE statement overwrites the password_hash for the admin user, setting it to the hash for “password”.

G. Now that the admin’s password_hash has been updated, we can log in with username “admin” and password “password.”

III. References:

A. https://sqlite.org/forum/info/0ae33e6c45c10fc699ccc9682b12c4660d4aafc6b12179c8c1938a99c3b493f5

B. https://emn178.github.io/online-tools/sha256.html

IV. People I Worked With:

A. Gilbert Hong