COMP3911 Secure Computing
Coursework 2

Name	Username	Student Id
Michael Wiciak	ed203mw	201491794
Lai Ting Yeung	sc21lty	201542796
Savan Dosanjh	sc21srsd	

1 Analysis of Flaws

- Passwords stored as plain text
- SQL Injection Vulnerability
- Susceptible to brute force attacks (no limit to login attempts)

1.1 Passwords stored as plain text

Nature of the flow How it was discovered Screenshot of the flow

1.2 SQL Injection Vulnerability

How it was discovered

SQL Injection is a common flaw when not enough care is taken when treating user inputs. Describe it more (reference)

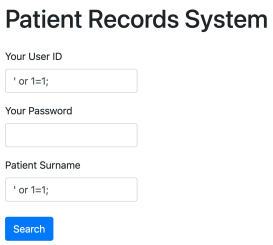

Figure 1: SQL Injection login page

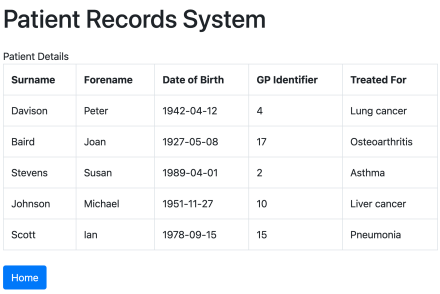

Figure 2: SQL Injection result

As seen in Figure 1, simply by putting username as `' or 1=1;`, effectively bypassing authentication.

Also in Figure 1, putting the same magic code into the patient surname input field also bypasses search, returning all patient's details within the database as seen in Figure 2.

In the Java code `select * from user where username='%s' and password='%s'`, the magic code `' or 1=1;` simply closes the first string, and the or effectively always evaluates to true, bypassing the simple user authentication.

1.3 Susceptible to brute force attacks (no limit to login attempts)

One could easily try commonly used usernames and passwords to put into the form. The application has no limits on login attemps, which makes it prone to this method, making it a security flaw.

How is was discovered

To test this theory, we used the `Hydra` [1] login cracker to see if the application is susceptible to brute force attacks, `Hydra` attemps to login using the provided username and a password list. As seen in Figure 3, we managed to crack one of the user's password in 3.5 minutes.

```
hydra -l aps -P /opt/wordlists/rockyou.txt \
  -f localhost -s 8080 \
  http-post-form "/:username=^USER^&password=^PASS^:are not valid"
```

Code 1: Command for running Hydra

In Code 1, we call `hydra` with the following options:

- `-l aps` specifies user `aps`
- `-P /opt/wordlists/rockyou.txt` specifies which password list to use, in this case, the rockyou password list [2]
- `-f localhost -s 8080` to specify where the server is

- `http-post-form "/:username=^USER^&password=^PASS^:are not valid"` to specify we want to use the `http-post-form` module and post the form to `/` with form data `username` and `password`, we also want Hydra to know if the page shows text “are not valid”, then the login failed.

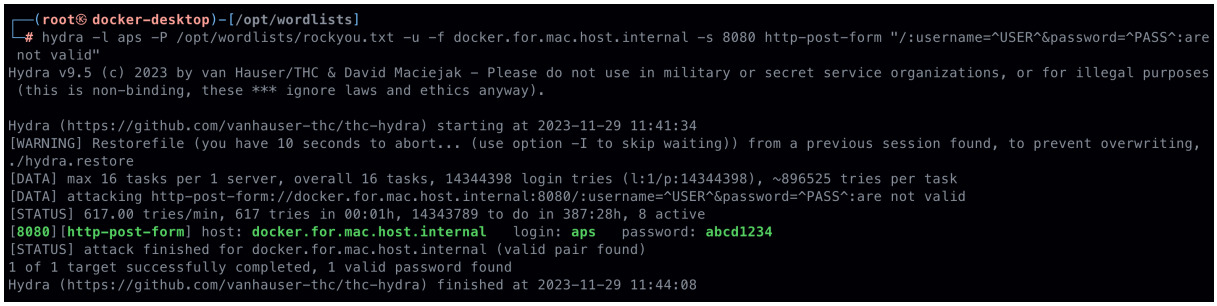

Figure 3: Output from running hydra

2 Fixes Implemented

2.1 Fixing weak password storage

What was changed How does it fix the problem

2.2 Fixing SQL injection vulnerability

To fix the SQL injection vulnerability, instead of using `String.format(...)`, we now use `Connection.prepareStatement(...)`

`String.format` does not escape SQL keywords but `PrepareStatement` does. Using `PrepareStatement` stops SQL injection since user input is escaped.

```
// Instead of...
String query = String.format(QUERY, userInput);
try (Statement stmt = database.createStatement()) {
  ResultSet results = stmt.executeQuery(query);
  // work with results...
}

// We now use...
try (PreparedStatement stmt = database.prepareStatement(QUERY)) {
  stmt.setString(1, userInput);
  ResultSet results = stmt.executeQuery();
  // work with results...
}
```

Code 2: Fixing SQL injection

Applying the generic changes in Code 2 to all database queries, SQL injection is no longer possible.

2.3 Fixing login brute force attack

What was changed How does it fix the problem

References

[1] K., “hydra | Kali Linux Tools”. [Online]. Available: https://www.kali.org/tools/hydra/

[2] W. Burns, “Common Password List (rockyou.txt)”. [Online]. Available: https://www.kaggle.com/datasets/wjburns/common-password-list-rockyoutxt