

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)

Patient Records System

Your User ID
' or 1=1;

Your Password

Patient Surname
' or 1=1;

Search

Figure 1: SQL Injection login page

Patient Records System

Patient Details

Surname	Forename	Date of Birth	GP Identifier	Treated For
Davison	Peter	1942-04-12	4	Lung cancer
Baird	Joan	1927-05-08	17	Osteoarthritis
Stevens	Susan	1989-04-01	2	Asthma
Johnson	Michael	1951-11-27	10	Liver cancer
Scott	Ian	1978-09-15	15	Pneumonia

Home

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 attempts, 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 attempts 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.

```
(root@ docker-desktop)-[/opt/wordlists]
# hydra -l aps -P /opt/wordlists/rockyou.txt -u -f docker.for.mac.host.internal -s 8080 http-post-form "[:username=^USER^&password=^PASS^:are not valid"
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2023-11-29 11:41:34
[WARNING] Restorefile (you have 10 seconds to abort... (use option -I to skip waiting)) from a previous session found, to prevent overwriting, ./hydra.restore
[DATA] max 16 tasks per 1 server, overall 16 tasks, 14344398 login tries (l:1/p:14344398), ~896525 tries per task
[DATA] attacking http-post-form://docker.for.mac.host.internal:8080/:username=^USER^&password=^PASS^:are not valid
[STATUS] 617.00 tries/min, 617 tries in 00:01h, 14343789 to do in 387:28h, 8 active
[8080][http-post-form] host: docker.for.mac.host.internal login: aps password: abcd1234
[STATUS] attack finished for docker.for.mac.host.internal (valid pair found)
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2023-11-29 11:44:08
```

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 `PreparedStatement` does. Using `PreparedStatement` 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>