COMP SCI 7406 - Secure Programming
Web Site

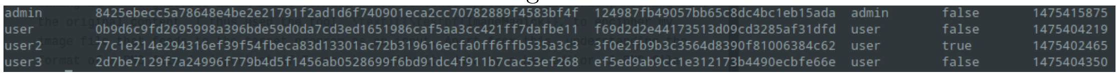
Hsiao Yu Chiang (a1675750)

October 3, 2016

1 Structure

Before deeply talking about concepts behind the design, I am going to simply introduce the structure of my application. Except for the basic required files, the file ”process.php” plays a significantly important role in the Web site. What it does is to deal with all requests sent by the browsers through HTML form with its tag and inputs. In addition to this, there are two javascript files as data transmission(user.js) and SHA-1 implementation(sha1.js), which will be discussing later. For my database file(user.db), I created a table named users in which contains *username, password, token, role, active and last* of its column; the active records whether a users still is on active, while the last is written as the duration of active. The following shows what the data actually records.

Figure 1:



```
admin    8425ebecc5a78648e4be2e21791f2ad1d6f740901eca2cc70782889f4583bf4f  124987fb49057bb65c8dc4bc1eb15ada  admin    false    1475415875
user     0b9d6c9fd2695998a396bde59d0da7cd3ed1651986caf5aa3cc421ff7dafbe11  f69d2d2e44173513d09cd3285af31dfd  user     false    1475404219
user2    77c1e214e294316ef39f54fbeca83d13301ac72b319616ecfa0ff6ffb535a3c3  3f0e2fb9b3c3564d8390f81006384c62  user     true     1475402465
user3    2d7be7129f7a24996f779b4d5f1456ab0528699f6bd91dc4f911b7cac53ef268  ef5ed9ab9cc1e312173b4490ecbfe66e  user     false    1475404350
```

2 Password Management

As a log-in system, it is important to carry out how it manage user's passwords and protect them not being cracked by attackers. According to the research [1], the one of most important reason to increasing passwords vulnerability is poor password practice. These poor passwords are easy to be guessed, and even some of attackers might use the brute force in the way that trying thousands of different password to log into a system to obtain personal information without any privileges.

In order to help users to create strong enough password, my log-in system focuses users to create a password that at least the length of eight characters and at least contains one numeric, one lowercase and one uppercase. Apart from the initial accounts that I created previously, the late users must follow the requirements while registering their account, otherwise the system will not be able to allow them to register new one. In addition, the research [1] suggests that it is better to avoid passwords that contain user's ID, therefore the same password as username will also not be allowed on my designed system.

On the other hand, I designed a way that could effectively protect passwords being cracked, even not worrying about if the passwords are stolen from the database. In this case, a password passed by the server and stored in the data are different. What I did is that when the password is passing from the client site to the server side, the password is hashed by SHA-1 generated an almost-unique 160-bit (20-byte) signature for a text [2]. Before the incoming password stored in the database, I used the regular expression to check whether the password is encrypted by SHA-1. If the password is valid, it prepares to store into the database. After that, I encrypted the password again by using the PHP built-in function with SHA-256, which randomly generates 40-byte texts, and written the 40-byte hashed password to the database.

It can be seen that from the database screen-shot(Fig.1) the password of the second column is stored as 40-byte long text. This way may protect those passwords being decrypted when attacker try to recover the password stolen from the database.

3 Session Management

As the HTTP is stateless, it will not track users status while they are surfing other applications. A web session is a sequence of HTTP request and response transactions with the same users. The session provides the ability to establish variables, which apply to each and every interaction a user has with the web application for the duration of the session. In my design, there are two most important components when it comes to implementing session management:

- Session-authenticated
- Session-token

The purpose of session-authenticated is to identify a user whether has the authentication to view a restricted page and manage other user's profiles. It is used to verify a user's role as user or admin, which is also justified by the user's role after they registered in the database. In addition, the session-token is similar as the session-ID but I did not actually use it in my log-in system. Instead, the session-token is generated by randomly hex string with 16 bytes longer size, that simply takes from digital number *0-9* to alphabetical order *a-f* it can be seen in the *user.js and process.php.*

```
var $hex = "0123456789abcdef";
```

Based on the Open Web Application Security project's statement[3], the session-token, or said session-id, must be longer enough at least 128bits(16 bytes) to prevent brute force attacks, where an attacker can go through the whole range of ID values and verify the extension of a valid session.

When a user registered successfully on the Web site, the first I will do is to give it as 16 bytes longer hex string generated by the random function in the *user.js.* After the token bypassed to the server side, the token will again be hashed; in this way, I extended the length of string to 32 bytes and the store it with the username to the database. The reason for why doing hash twice is the same as the password management. It could avoid the token being cracked in the middle attack or the token being stolen from the database as both of them are encrypted by the different cryptographic hash algorithm.

How can both of session-authenticated and session-token work together on the Web site ? In my design, when a authenticated user has been logged in the system and is going to the visit a webpage via a link, the global variable session-authenticated will be refreshed as false, and the server side will verify this user whether is active and verify his old token at the same time. If the user is valid, the session-authenticated set up as true and the server will issue a new token for this user and store it into the database and wait for next interaction.

4 Web Site Security

As a Web site is usually revealed on the Internet and the Public, its security is necessary to be considered when we design our own site. Based on my design, the log-in system might cover the following two topics:

- SQL Injection
- Code Injection

When SQL is used to display data on a web page, it is common to let web users input their own information. For example, in my case, the log-in system is required users input their account and password, while in the admin page will list all users who have been registered on my system. Also because SQL statements are text only, it is easy to dynamically change SQL statements. In normal case, most of SQL will be written in that code as below:

```
$data = 1;
$query          = "SELECT * FROM users where id= $data";
```

However, it is highly risk to be injected a little piece of computer code, like :

```
$data = 1; DROP TABLE users;
```

That will destroy a table in the database! This one of situations calls SQL injection. SQL injection is a technique where malicious users can inject SQL commands into an SQL statement via web page input. The injected SQL commands can alter SQL statement and compromise the security of a web application. To avoid SQL injection, we need to either stop writing dynamic queries and/or prevent user supplied input which contains malicious SQL from affecting the logic of the executed query [4]. In my case, I used SQL preparation statements with bind values rather than directly writing queries. Therefore, when a query will be issued, the statement in my program looks like:

```
$stmt = $db->prepare("SELECT * FROM users where username = ?")
$stmt->bindValue(1, $username, SQLITE3_TEXT);
$stmt->execute();
```

Although the code is similar to the above, the difference is that the query sent to the database is without any data. Instead, the data is sent by the bindValue function that is completely separated from the query itself.

In terms of code injection, it is possible that an attacker may fill in invalid data through the form with its input. For instance, the text field of username may inject some illegible characters that could cause the database on unsafety. I thus programmed a function which is the ability to filter out specifics characters such as *"'", "//" and ";"*.

References

[1] Shirley Gaw and Edward W. Felten. 2006. Password management strategies for online accounts. In Proceedings of the second symposium on Usable privacy and security (SOUPS '06). ACM, New York, NY, USA, 44-55. DOI=http://dx.doi.org/10.1145/1143120.1143127

[2] SHA-1 Cryptographic Hash Algorithm, http://www.movable-type.co.uk/scripts/sha1.html

[3] The Open Web Application Security Project, Session Management Cheat Sheet, https://www.owasp.org/index.php/Session_Management_Cheat_Sheet.html

[4] The Open Web Application Security Project, SQL Injection, https://www.owasp.org/index.php/SQL_Injection