CSCI 157 – Computer Security

Assignment 1: CSRF and XSS

Due: 9/18/2023 11:59 PM

Instructions:

There are two sections in this assignment. For Section 1, answer all questions completely. For Section 2 (two tasks), execute the code (using files from Assignment1_Files.zip) and answer all questions.

The final submission is a single report (PDF file). You need to submit a detailed report, with:

Section 1 - Detailed answers and observations.

Section 2 – Screenshots, to describe what you have done and what you have observed. You also need to provide explanations to the observations that are interesting or surprising. Also list the important code snippets followed by explanation. **Simply attaching code or screenshot without any explanation will not receive points.**

Section 1

40 points – 5 points each.

1. Assume the following conditions:

   a) Bob has an active session on a banking website which lacks safety features to separate same-site and cross-site requests.

   b) Samy uses the same website and while sending $100 to Bob observes the following http request using LiveHTTPHeaders extension:

```
http://www.bestbankever.com/action/send?fromID=20&toID=30&amount=100
GET /action/send?fromID=20&toID=30&amount=100
Host: www.bestbankever.com
User-Agent: Mozilla
//omitted lines
Cookie: bbe=abcdefghijklmno
Connection: keep-alive
```

   Samy decides to trick Bob by making Bob send $500 to Samy. To achieve this, Samy first sets up a website and fills it with code to generate a forged GET request.
   Complete (replacing the blanks) the following, such that Samy's attack succeeds:

   a) Samy's website/webpage URL : _____

   b) Code inside Samy's webpage (only outline provided):

```
<html>
<body> <img src="_____" alt="image" width = "1" height="1" / >
</body>
</html>
```

c) Message that Samy must send to Bob (assume Bob has an active session on the bank site):

"Hi Bob, checkout this funny video _____ .
Hilarious stuff, if this link doesn't work I'll send you another one. Let me know. "

2. Using LiveHTTPHeader, we find out that the following POST request is used to send an HTTP request to www.example.com to delete a page owned by a user (only the owner of page can delete the page).

```
http://www.example.com/delete.php
POST /delete.php HTTP/1.1
Host: www.example.com
...
Content-Length: 8
pageid=5
```

Construct a simple malicious web page, so when a victim visits this web page, a forged request will be launched against www.example.com to delete a page belonging to the victim.

3. What are the differences between XSS and CSRF attacks?

4. Consider the following CSP :

```
<?php
  $cspheader = "Content-Security-Policy:".
               "default-src 'self';".
               "script-src 'self' 'nonce-1rA2345' 'example.com'".
               "";
  header($cspheader);
?>
```

Which of the following blocks of code (1-6) will be executed?

```
<html>
<script type="text/javascript" nonce="1rA2345">
  ... JavaScript Code ...                                    ❶
</script>

<script type="text/javascript" nonce="2rB3333">
  ... JavaScript Code ...                                    ❷
</script>

<script type="text/javascript">
  ... JavaScript Code ...                                    ❸
</script>

<script src="script.js"> </script>                           ❹

<script src="https://example.com/script2.js"> </script>      ❺

<button onclick="alert('hello')">Click me</button>           ❻
</html>
```

5. How can same-site cookie help prevent CSRF attacks?
6. How can we use secret tokens to prevent CSRF attacks, and why does it work?
7. Can we simply ask browsers not to attach any cookie for cross-site requests? What would be the effect of this approach?
8. Secret tokens are used to protect against CSRF, can they also prevent XSS attacks?

Section 2

60 points – 30 points each.

It may be necessary to clear cached data in the browser between different tasks!

Task1: CSRF Attack

A CSRF attack involves a victim user, a trusted site, and a malicious site. The victim user holds an active session with a trusted site while visiting a malicious site. The malicious site injects an HTTP request for the trusted site into the victim user session, causing damages.

We will be attacking a social networking web application using the CSRF attack. The open-source social networking application is called Elgg, which has already been installed in our VM. **Elgg has countermeasures against CSRF, but we have turned them off.**

Information: For this task, we will be using files within the **"Assignment1_Files/CSRF"** folder. To setup for this task, locate the "docker-compose.yml" file in **"Assignment1_Files/CSRF",** run the commands "dcbuild" and "dcup".

Environment Setup:

We will use two websites. The first website is the vulnerable Elgg site accessible at www.seed-server.com. The second website is the attacker's malicious web site that is used for attacking Elgg. This web site is accessible via www.attacker32.com. We use containers to set up the environment.

If you encounter problems when setting up the environment, read the "Common Problems" section of the manual for potential solutions provided in links in assignment 0.

Elgg Web Application

We use an open-source web application called Elgg. Elgg is a web-based social-networking application. It is already set up in the provided container images. We use two containers, one running the web server (10.9.0.5) , and the other running the MySQL database (10.9.0.6). The IP addresses for these two containers are hardcoded in various places in the configuration, **do not change them from the docker-compose.yml file.**

The Elgg container. We host the Elgg web application using the Apache web server. The website setup is included in apache_elgg.conf inside the Elgg image folder. The configuration specifies the URL for the website and the folder where the web application code is stored.

```
<VirtualHost *:80>
      DocumentRoot /var/www/elgg
      ServerName www.seed-server.com
      <Directory /var/www/elgg>
            Options FollowSymlinks
            AllowOverride All
            Require all granted
      </Directory>
</VirtualHost>
```

The Attacker container. We use another container (10.9.0.105) for the attacker machine, which hosts a malicious website. The Apache configuration for this website is listed in the following:

```
<VirtualHost *:80>
      DocumentRoot /var/www/attacker
      ServerName www.attacker32.com
</VirtualHost>
```

Since we need to create web pages inside this container, for convenience, as well as for keeping the pages we have created, we mounted a folder (CSRF/attacker on the hosting VM) to the container's /var/www/attacker folder, which is the DocumentRoot folder in our Apache configuration. Therefore, the web pages we put inside the attacker folder on the VM will be hosted by the attacker's website. We have already placed some code skeletons inside this folder.

DNS configuration. We access the Elgg website, the attacker website, and the defense site using their respective URLs. We need to add the following entries to the /etc/hosts file, so these hostnames are mapped to their corresponding IP addresses. You need to use the root privilege to change this file (using sudo). It should be noted that these names might have already been added to the file due to some other containers. If they are mapped to different IP addresses, the old entries must be removed.

```
10.9.0.5     www.seed-server.com
10.9.0.5     www.example32.com
10.9.0.105   www.attacker32.com
```

MySQL database. Containers are usually disposable, so once it is destroyed, all the data inside the containers are lost. We do want to keep the data in the MySQL database, so we do not lose our work when we shutdown our container. To achieve this, we have mounted the mysql data folder on the host machine (it will be created after the MySQL container runs once) to the /var/lib/mysql folder inside the MySQL container. This folder is where MySQL stores its

database. Therefore, even if the container is destroyed, data in the database is still kept. If you do want to start from a clean database, you can remove this folder:

```
$ sudo rm -rf mysql_data
```

User accounts. We have created several user accounts on the Elgg server; the user name and passwords are given in the following.

```
---------------------------
UserName | Password
---------------------------
admin | seedelgg
alice | seedalice
boby | seedboby
charlie | seedcharlie
samy | seedsamy
---------------------------
```

Questions:

a) (10 points) Samy (a user in the social network) wants Alice (another user in the social network) to say "**CSCI 157**" in her profile (malicious activity).

Use the CSRF attack to achieve that goal (as Samy).

One way to do the attack is to send a message to Alice's Elgg account, hoping that Alice will click the URL inside the message. This URL will lead Alice to your (i.e., Samy's) malicious web site www.attacker32.com, where you can launch the CSRF attack. The objective of your attack is to modify the victim's profile.

HINT: Modify the editprofile.html JavaScript (in Assignment1_Files/CSRF) to carry out this attack. The modified file will have to be placed/copied on to the attacker's site (using "docker cp" command) for it to work.

b) (20 points) Visit http://example32.com/ which is setup on one of the containers to demonstrate SameSite cookies. Three cookies will be set on your browser, cookie-normal, cookie-lax, and cookie-strict. There are two links on example32.com, one points to example32.com and the other points to attacker32.com, through each site, three different types of requests can be sent to www.example32.com/showcookies.php . Briefly explain your observations (with screenshots) on the cookies attached to each different type of request from both these links on example32.com.

NOTE: In your submission, explain any investigations you did to succeed in the attack. (For example, how to get Alice's user id, how to observe, analyze and form the fake post request)

RUN COMMAND "dcdown" between these tasks to shutdown containers of CSRF and start containers of XSS. The files used are different.

It may be necessary to clear cached data in the browser between different tasks!

Task2: XSS Attack and Content Security Policy

Information: For this task, we will be using files within the **"Assignment1_Files/XSS"** folder. To setup for this task, locate the "docker-compose.yml" file in **"Assignment1_Files/XSS",** run the commands "dcbuild" and "dcup".

DNS Setup:

We have set up several websites. They are hosted by the container 10.9.0.5. We need to map the names of the web server to this IP address. Please add the following entries to /etc/hosts. You need to use the root privilege to modify this file:

```
10.9.0.5  www.example32a.com
10.9.0.5  www.example32b.com
10.9.0.5  www.example32c.com
10.9.0.5  www.example60.com
10.9.0.5  www.example70.com
```

Configuration files and CSP:

Inside the Assignment1_Files/XSS/image-www docker image folder, there is a file called **apache_csp.conf.** It defines five websites, which share the same folder, but they will use different files in this folder. The example60 and example70 sites are used for hosting JavaScript code. The example32a, example32b, and example32c are the three websites that have different CSP configurations.

In the experiment, you need to modify this Apache configuration file (**apache_csp.conf**). If you make a modification directly on the file inside the image folder, you need to rebuild the image and restart the container, so the change can take effect.

You can also modify the file while the container is running. The downside of this option is that in order to keep the docker image small, we have only installed a very simple text editor called nano inside the container. It should be sufficient for simple editing. If you do not like it, you can always add an installation command to the **Dockerfile** to install your favorite command-line text editor. On the running container, you can find the configuration file **apache_csp.conf** inside the /etc/apache2/sites-available folder.

After making changes, you need to restart the Apache server for the changes to take effect:

```
# service apache2 restart
```

Web Pages required for Experiment:

The example32a, example32b, example32c, servers host the same web page **index.html**, which is used to demonstrate how the CSP policies work. In this page, there are six areas, area1 to area6. Initially, each area displays "Failed". The page also includes six pieces of JavaScript

code, each trying to write "OK" to its corresponding area. If we can see OK in an area, that means, the JavaScript code corresponding to that area has been executed successfully; otherwise, we would see Failed. There is also a button on this page. If it is clicked, a message will pop up, if the underlying JavaScript code gets triggered.

Questions:

a) (15 points) In class we discussed about a self-propagating version of XSS attack through which we modified the brief-description section of victim's profile (adding "Samy is my hero" to profile). Design a similar self-propagating version of XSS attack, such that victim automatically adds Samy as a friend and also becomes a carrier of the malicious XSS code. Eg: If Alice looks at Samy's profile, Alice adds Samy as a friend and also carries the worm on Alice's profile. Sequentially, When Bob looks at Alice's profile, Bob also adds Samy as a friend and gets a copy of the worm on Bob's profile.
Demonstrate that your self-propagating worm works as desired. Include your worm code in the report.

b) (15 points) Change the server configuration on example32b (modify the Apache configuration), so Areas 5 and 6 display OK. Include your modified configuration in the report.

Note: The steps to be performed after modifying the configuration file are detailed above under section "Configuration Files and CSP".