Lab V – Session Hijacking Attack and Protection

CPS 499-02/592-02

Software/Language Based Security

Fall 2020

Dr. Phu Phung

Evan Krimpenfort

Code:

https://github.com/Krimpenfort23/autumn-2020/tree/master/cps_499/lab-5

Task I:

a. Login Modification

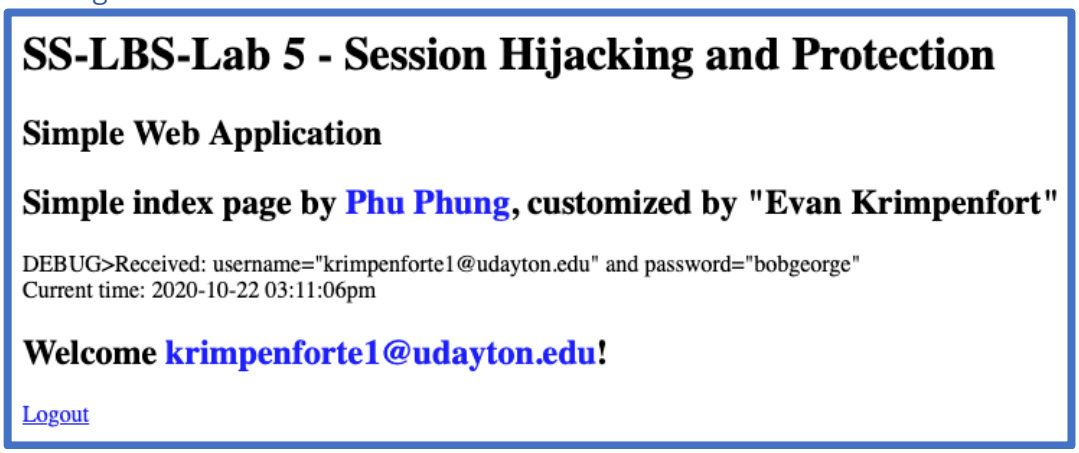

Figure 1: changing login credentials

b. Observing HTTP Reponses/Requests With Cookies – First Time

i. Any cookie information in the request?

Request Headers (509 B) Raw

```
POST /lab5/index.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:81.0) Gecko/2010
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 55
Origin: http://192.168.56.101
DNT: 1
Connection: keep-alive
Referer: http://192.168.56.101/lab5/login.php
Upgrade-Insecure-Requests: 1
```

Figure 2: HTTP Request

There is no cookie information here in Figure 2 because there is no cookie to get from the server. The server right now does not have a cookie for this site. So, there is nothing to grab.

ii. Any cookie information in the response?

```
Response Headers (474 B)                                    Raw
HTTP/1.1 200 OK
Date: Sat, 24 Oct 2020 22:54:02 GMT
Server: Apache/2.4.18 (Ubuntu)
Set-Cookie: PHPSESSID=qr2hg7000qf5q0ot1811j2qsm1; expires=Sun, 25-Oct-2020 22
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 372
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

Figure 3: HTTP Response

However, in Figure 3, we can see that there is cookie information under a *Set-Cookie* call because the server now wants to associate that site with some sort of ID. We can see here that the cookie is "*Set-Cookie: PHPSESSID=qr2hg7000qf5q0ot1811j2qsm1; expires=Sun, 25-Oct-2020 22:54:02 GMT; Max-Age=86400; path=/"*.

c. Observing HTTP Reponses/Requests With Cookies – Second Time

i. Any cookie information in the request?

```
Request Headers (407 B)                                     Raw
GET /lab5/index.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:81.0) Gecko/2010
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Cookie: PHPSESSID=qr2hg7000qf5q0ot1811j2qsm1
Upgrade-Insecure-Requests: 1
```

Figure 4: HTTP Request After

In figure 4, there is cookie information. This is because the server has the cookie and it got that cookie from the previous HTTP response with the *Set-Cookie* call. The Cookie seen in the request was *"Cookie: PHPSESSID=qr2hg7000qf5q0ot1811j2qsm1"* which is the same cookie as before. Since the browser has that ID and so does the server, that check allows the HTTP Request to ask for that cookie information and see if there's anything there.

ii. Any cookie information in the response?

```
Response Headers (361 B)                                    Raw
HTTP/1.1 200 OK
Date: Sat, 24 Oct 2020 23:14:06 GMT
Server: Apache/2.4.18 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 367
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

Figure 5: HTTP Response After

In figure 5, we can see that there is no cookie information and that's because the server doesn't need any cookie information anymore from the browser. From now on, the Requests will ask for the cookie to verify any change in the server that need to be shown in the browser and the responses will remain cookieless.

Task II: Session Hijacking Attack

a. Performing the Attack

i. Steal the Cookie inside of the SEED VM

Figure 6: Getting the Cookie.

In figure 6, you can see that the Cookie is "*PHPSESSID=uf9otmqf2jnrcq1nl8pb5h7a07*".

ii. The Attacker Side

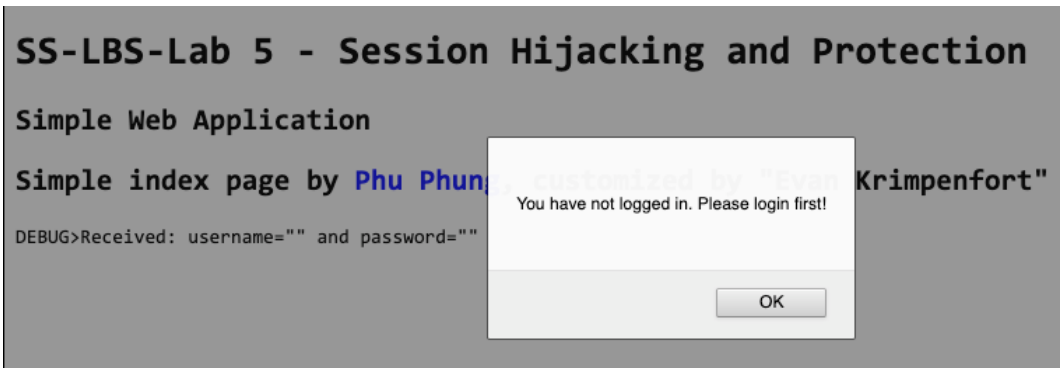

Figure 7: To the Desktop: Can't log in now

iii. Perform the Attack

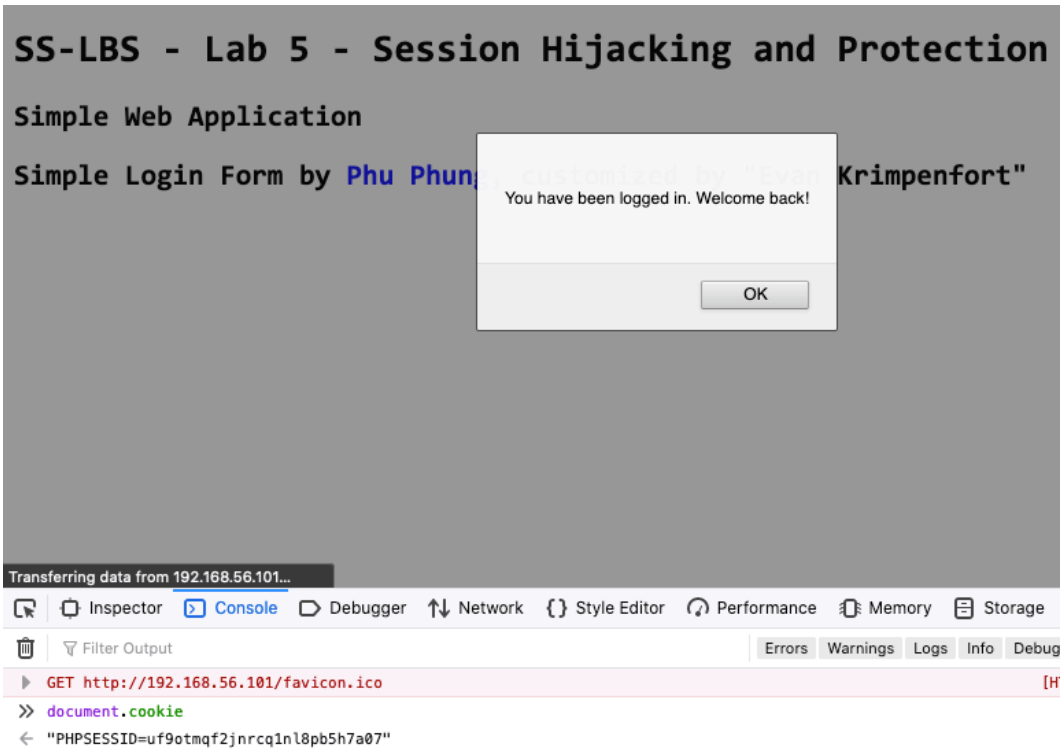

Figure 8: Attack executed

b. Attack Explanation

This attack happens because I stole the ID that was associated with a session of a successful login. So, when the browser sends an HTTP request with the cookie information, the information received back was making the "logged" variable true so that the user could log in. Because the attacker was able to steal the cookie and place it into their own browser, that was the only reason this could happen.

Task III: Fix the Session Hijacking Vulnerability

a. Code revision and comparison

```
        if (mockchecklogin($username,$password)){
            $_SESSION["logged"]=TRUE;
            $_SESSION["username"] = $username;
            $_SESSION["browser"] = $_SERVER["HTTP_USER_AGENT"];
            $welcome = "Welcome "; //not previously logged-in
        }else{//failed
            redirect_login('Invalid username/password');
        }
    }else{//no username/password is provided
        //check if the session has NOT been logged in, redirect to the login page
        if ($_SESSION["logged"]!=TRUE) {
            redirect_login('You have not logged in. Please login first!');
        }
        if ($_SESSION["browser"] = $_SERVER["HTTP_USER_AGENT"])
        {
            echo "Session Hijacking is detected.";
            die();
        }
```

Figure 9: Code for the Hijacking protection in the index page

```
    session_start();
    if ($_SESSION["browser"] != $_SERVER['HTTP_USER_AGENT'])
    {
        echo "<script>alert('Session Hijacking is detected.');</script>";
        exit();
    }
```

Figure 10: Code for the Hijacking protection on the login page

b. Attack Prevention

Figure 11: Hijacking on the index page

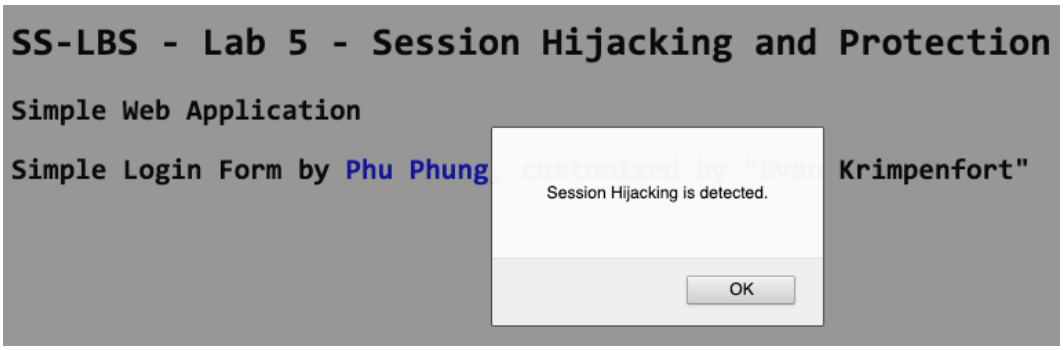

Figure 12: Hijacking on the login page