Logo

Description automatically generated

Assignment (30%) (Exploitation – Task 2)

|  |  |  |
| --- | --- | --- |
| **Module Code** | **:** | CT119-3-3-VAPT (Vulnerability Assessment and Penetration Testing) |
| **Intake Code** | **:** | APD3F2411CS(CYB) |
| **Lecturer Name** | **:** | TS. DR. MANIMEGALAI A/P RAJENDERAN |
| **Date Assigned** | **:** | 1st January |
| **Date Submitted** | **:** | 26th March |

|  |  |
| --- | --- |
| **Student TP number** | **Full Name** |
| TP071012 | Abdulrahman Gamil Mohammed Ahmed |

Contents

[1.0 INTRODUCTION 4](#_Toc193632204)

[2.0 Vulnerability Scanning 5](#_Toc193632205)

[2.1 Reconnaissance, Enumeration, and Exploiting Port Knocking 5](#_Toc193632206)

[2.2 Service Enumeration and Vulnerability Discovery 13](#_Toc193632207)

[3.0 Exploitation of the System 17](#_Toc193632208)

[3.1 SQLinjection 17](#_Toc193632209)

[3.3 Privilege Escalation via Command Injection 23](#_Toc193632210)

[3.4 Establishing Persistent Access 26](#_Toc193632211)

[3.5 Privilege Escalation to Root 32](#_Toc193632212)

[4.0 COUNTERMEASURES PROPOSED 36](#_Toc193632213)

[5.0 CONCLUSION 37](#_Toc193632214)

[6.0 REFERENCES 38](#_Toc193632215)

Table of Figures

[1 IP discovery 5](#_Toc193632138)

[2 OS nmap scan 5](#_Toc193632139)

[3 Advanced nmap scanning 6](#_Toc193632140)

[4 UDP nmap scanning 6](#_Toc193632141)

[5 Unicornscan scan 7](#_Toc193632142)

[6 SNMPwalk command for information gathering 7](#_Toc193632143)

[7 knocking ports 545 232 1876 8](#_Toc193632144)

[8 masscan scanning 8](#_Toc193632145)

[9 Nmap scanning for open ports 9](#_Toc193632146)

[10 ports 22&23 info 10](#_Toc193632147)

[11 port 5800 info 10](#_Toc193632148)

[12 ports 10009&10010 info 10](#_Toc193632149)

[13 Trying access via SSH 11](#_Toc193632150)

[14 knocking ports 555 423 1800 11](#_Toc193632151)

[15 masscan for open ports 12](#_Toc193632152)

[16 nmap scanning for port 8 12](#_Toc193632153)

[17 accessing port 8 web page 13](#_Toc193632154)

[18 Credentials entering 13](#_Toc193632155)

[19 Find sub directory & web page 14](#_Toc193632156)

[20 accessing Bobs web hosting admin portal 15](#_Toc193632157)

[21 Invalid credentials 16](#_Toc193632158)

[22 Using SQLmap to bypass the login 17](#_Toc193632159)

[23 Finding CRITICAL vulnerability 17](#_Toc193632160)

[24 Using SQLmap to exploit 17](#_Toc193632161)

[25 Finding databases in the system 18](#_Toc193632162)

[26 username & password hash discovery 18](#_Toc193632163)

[27 Crack the SHA1 hash 19](#_Toc193632164)

[28 Accessing admin portal 20](#_Toc193632165)

[29 Webmaster info 21](#_Toc193632166)

[30 Finding hideen index.php page 22](#_Toc193632167)

[31 Accessing live chat page 23](#_Toc193632168)

[32 Capture web request 24](#_Toc193632169)

[33 Result of command injection “whoami” 25](#_Toc193632170)

[34 Result of command injection “sudo+-l” 25](#_Toc193632171)

[36 View pen tester public SSH id 26](#_Toc193632172)

[37 Edit special char to URL encoded 26](#_Toc193632173)

[Check the permission of .ssh directory 38 27](#_Toc193632174)

[39 Chmod of .ssh directory to 777 27](#_Toc193632175)

[Double check permissions 40 27](#_Toc193632176)

[41 Chmod to 666 28](#_Toc193632177)

[42 Double check authorized\_keys file permission 28](#_Toc193632178)

[Write the public ID into authorized\_keys file 43 28](#_Toc193632179)

[44 view the public id of the pen tester in the authorized\_keys file 29](#_Toc193632180)

[45 Chmod to 700 29](#_Toc193632181)

[46 Chmod to 600 29](#_Toc193632182)

[47 Accessing the system via SSH 30](#_Toc193632183)

[48 Accessing the Terminal 31](#_Toc193632184)

[49 View system info 32](#_Toc193632185)

[50 Search of exploitation script 32](#_Toc193632186)

[51 gcc the script and transfer it to targeted system 33](#_Toc193632187)

[52 Execute the exploitation 34](#_Toc193632188)

[53 Gaining Root Access 35](#_Toc193632189)

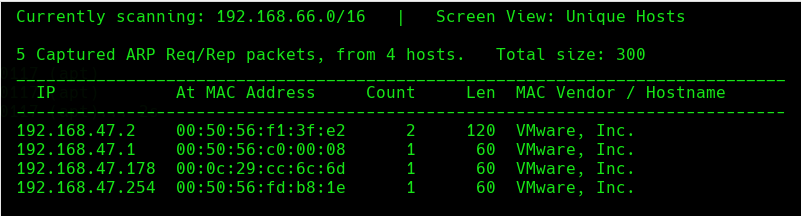
# 1.0 INTRODUCTION

This report describes a penetration test carried out on an Ubuntu 4.4 x86\_64 server for ‎a cloud hosting company after a suspected security breach in which the company ‎feared the presence of a backdoor. Testing was performed using provided credentials, ‎milton:thelaststraw, in order to assess the security of the system before its official ‎launch. The testing here aimed at exploiting and identifying the vulnerabilities at the server, whether there was already a backdoor or not and how it was to be accessed, root access to assess the extent of the compromise, and recommendations on security to reduce the risk of future breaches.

# 2.0 Vulnerability Scanning

## 2.1 Reconnaissance, Enumeration, and Exploiting Port Knocking

**Identifying the Server IP Address**



1 IP discovery

The penetration test begun by identifying the IP address of the target server using ‎NetDiscover, scanning the network to find active devices. The address of the server in question is ‎IP: 192.168.47.178 on which the succeeding testing was based.‎

**Conducting Nmap Enumeration**

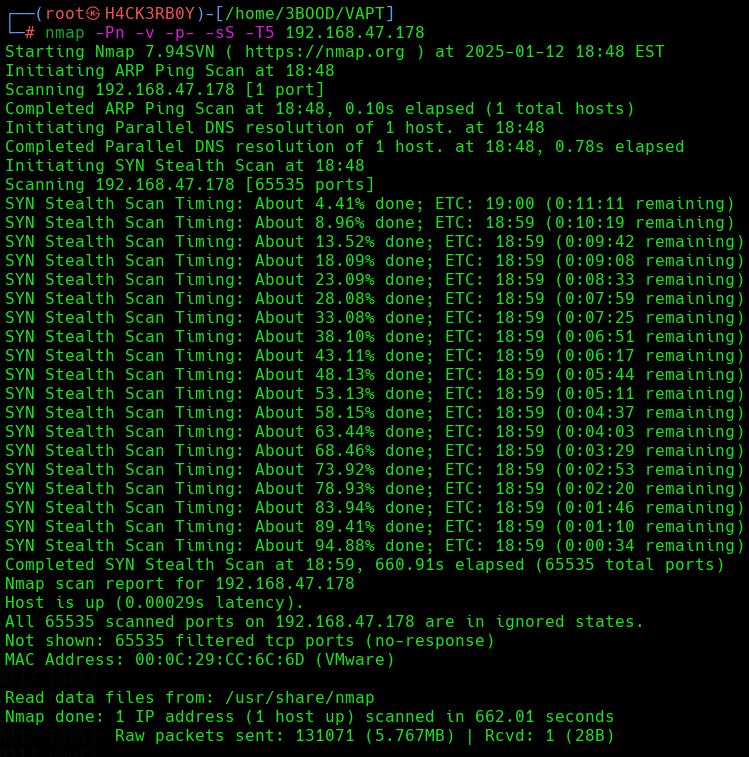
A computer screen with green text

Description automatically generated

2 OS nmap scan

Nmap enumeration was done against the target ‎server to discover open ports and services: nmap -p- 192.168.47.178. No results were returned for open ports, which may be an indication of very tight security or external accesses that have been restricted, which calls for additional investigation.‎

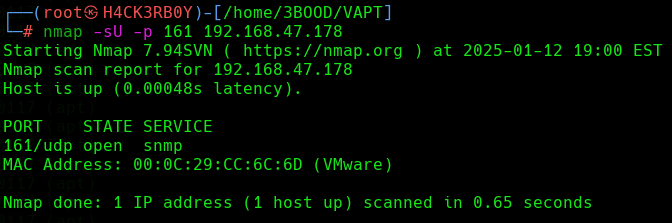
**Advanced Nmap Scanning**



3 Advanced nmap scanning

Perform an advanced Nmap scan: nmap -Pn -v -p- -sS -T4 192.168.47.178 to bypass host discovery and perform a full TCP SYN scan. Even after scanning all 65,535 ports, the results showed that all ports were filtered, which means some kind of security is up and running, like a firewall or intrusion prevention system blocking the incoming traffic.

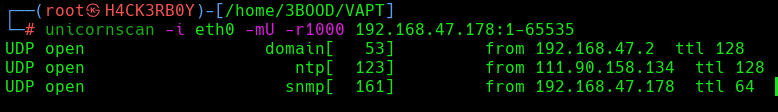
**UDP Port Scanning**



4 UDP nmap scanning

Running a UDP scan via nmap -sU -p 161 192.168.47.178 showed that port 161 was open over UDP and running an SNMP service. This is significant, since an improperly configured SNMP could disclose critical system information, configuration settings, or sensitive data and may provide an open door for attack.

**Utilizing Unicornscan for Advanced Packet Analysis**



5 Unicornscan scan

unicornscan -i eth0 -mU -r1000 192.168.47.178:1-65535 confirmed port 161 UDP ‎as ‎active, thereby confirming the earlier results from Nmap about SNMP. Cross verifying ‎results through different tools has given accuracy to penetration testing.‎

**Using SNMPwalk for Information Gathering**

A computer screen with green text

Description automatically generated

6 SNMPwalk command for information gathering

snmpwalk -Os -c public -v1 192.168.47.178 Times when the output showed an email and a number sequence: 545 232 1876, probably for authentication. Here is how risky the inappropriate configuration of SNMP services can be.‎

**Port Knocking with the Knock Tool**

A screen shot of a computer

Description automatically generated

7 knocking ports 545 232 1876

This was where it was found that the server was using Port Knocking for secured access. Using the Knock tool, the right sequence of port numbers were sent out with the command, knock -v 192.168.47.178 545 232 1876. This indeed triggered a port knocking mechanism that most probably opened specific ports for further access. Now, it was time to scan once more and find which ports or services might have been opened.

**Conducting a Comprehensive Scan with Masscan**

A screenshot of a computer screen

Description automatically generated

8 masscan scanning

Immediately following the activation of the port knocking mechanism, several newly opened ports were identified from a Masscan: masscan 192.168.47.178 ‎-‎rate‎=1000 -p1-65535,U:1-65535 -e eth0

• SSH: Remote access via port 22

• Port 23: Telnet-insecure remote access

• Lan 161 (SNMP): Network management

• Ports 5800, 10007-10010, 2048, 4096: Probably custom services‎

**Reconfirming Open Ports with Nmap**

A computer screen shot of a computer program

Description automatically generated

9 Nmap scanning for open ports

In the wake of the Masscan port discovery, a full detailed Nmap scan, nmap 192.168.47.178, revealed active services:‎

• SSH on port 22: allows for secure remote access

• Telnet on port 23: allows for insecure remote access

• Port 2048: custom service

• Port 5800: possibly VNC over HTTP

• Ports 10009, 10010: probably proprietary services

**Detailed Service Scanning with Nmap**

A computer screen with green text

Description automatically generated

10 ports 22&23 info

A screen shot of a computer

Description automatically generated

11 port 5800 info

A computer screen with green text

Description automatically generated

12 ports 10009&10010 info

With the targeted Nmap, the scan identified service versions and possible vulnerabilities: nmap -p22,23,2048,5800,10009,10010 -sV -sC ‎‎192.168.47.178. This displayed a potential backdoor on port 23, considered a major security concern since Telnet is insecure. This again underlines the need to replace Telnet with secure alternatives like SSH.‎

**Attempting SSH Connection and Discovering a New Lead**

A screenshot of a computer screen

Description automatically generated

13 Trying access via SSH

An attempt was made to connect to the server via SSH using the provided ‎credentials. This has not succeeded, because the given credentials did not give access ‎to it.‎

A screen shot of a computer

Description automatically generated

14 knocking ports 555 423 1800

The phone number that was found, 555-423-1800, from the scan above was tried as a port ‎knocking sequence with the Knock tool: knock -v 192.168.47.178 555 423 1800. ‎This was an attempt to activate some kind of mechanism that would disclose certain ‎hidden or forbidden ports. If successful, this could provide new opportunities for further ‎testing and exploitation and reinforce the importance of testing all clues for possible ‎system access.‎

A computer screen with text and numbers

Description automatically generated

15 masscan for open ports

After the port knocking sequence (555, 423, 1800), a rescan revealed the closure of the previous ports and showed only ports 22 ‎‎(SSH) and 8 open, indicating increased security rules. ‎Port 8 was likely serving a certain purpose and should be studied closely for ‎exploitation purposes.‎

A screenshot of a computer

Description automatically generated

16 nmap scanning for port 8

The Nmap targeted scan showed, on port 8, an abnormally running ‎HTTP service: nmap -p 8 -sV -sC 192.168.47.178. HTTP service normally runs on port 80 or 8080, so it might be an administrative web application. The version of the identified service and ran default scripts to identify possibly vulnerabilities. It opens new opportunities in the field of exploitation, looking at the version if the software is outdated, looking at exposed directories, and or insecure configurations.‎

## 2.2 Service Enumeration and Vulnerability Discovery

A screenshot of a computer

Description automatically generated

17 accessing port 8 web page

A screenshot of a computer

Description automatically generated

18 Credentials entering

Finally, by using http://192.168.47.178:8 and having credentials of milton:thelaststraw, that webpage opened for access. Essentially, the port 8 of HTTP service accepts these credentials valid. Since an administrator has just successfully logged on to this host, this website probably has backdoors or key sensitive information administratively. Additional steps would be required to reach the ‎web interface, which is vulnerable to insecure configurations, exposed data, ‎or features that might lead to privilege scalation. ‎A screenshot of a computer

Description automatically generated

19 Find sub directory & web page

On the "Milton's To-Do List" page, there was a link leading to /breach3/index.php, which might indicate a web application vulnerable to certain attacks. The ‎http://192.168.47.178:8/breach3/index.php‎ page can be visited to know more about its structure and possible weaknesses. The source code of this page will be analysed in order to find ‎vulnerabilities such as SQL injection.‎

A screenshot of a computer

Description automatically generated

20 accessing Bobs web hosting admin portal

After the failed login, examine the page source code for hidden clues, such as ‎hardcoded credentials or vulnerable scripts. Try SQL injection and test for other ‎issues like insecure password reset or session handling to find further ‎vulnerabilities.‎

A login screen shot

Description automatically generated

21 Invalid credentials

After some unsuccessful password attempts, it certainly seems that the authentication of ‎‎/breach3/index.php is more difficult. The next thing to check would be vulnerabilities such as ‎SQL injection. Injecting malicious input into the login fields, for example, the payload ‎‎' OR '1'='1 might trick the application to skip its authentication. Testing this would come in the form of various crafted inputs, depending on system weaknesses. One could also continue with HTTP request methods or use fuzzing tools to uncover the presence of other bypass techniques.‎

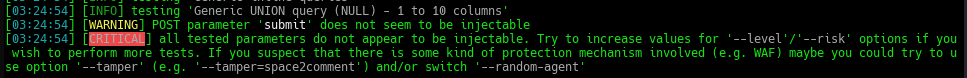
# 3.0 Exploitation of the System

## 3.1 SQLinjection

A screenshot of a computer screen

AI-generated content may be incorrect.

22 Using SQLmap to bypass the login



23 Finding CRITICAL vulnerability

A screenshot of a computer screen

AI-generated content may be incorrect.

24 Using SQLmap to exploit

A screenshot of a computer program

Description automatically generated

25 Finding databases in the system

A screenshot of a computer program

Description automatically generated

26 username & password hash discovery

The sqlmap tool was run with an extended command:‎

sqlmap -r login\_pack.txt --dbs --batch --dump --level=5 --risk=3 --‎tamper=equaltolike

•‎ --level=5: Increases depth, scanning all parameters and headers.‎

•‎ --risk=3: Automatically enables the most intrusive payloads, increasing chances of getting good exploitation.‎

•‎ --tamper=equaltolike: Replaces = with LIKE in SQL queries to bypass ‎filters.‎

The scan showed five databases and extracted critical information, which included:‎

•‎ The admin username and hashed password.‎

A screenshot of a computer

Description automatically generated

27 Crack the SHA1 hash

Following the extraction of the SHA1 hashed password of the admin account with sqlmap, the next ‎step was to crack it and retrieve the plaintext password. An online hash-breaking ‎service was used by submitting the SHA1 hash: randomhash‎.

A screenshot of a computer

Description automatically generated

28 Accessing admin portal

After having access to the admin dashboard, he got detailed information about the system clients: their names, statuses, values, and URLs that were connected with them. Such information provides a good overview of how the system works and might be attacked, showing which clients or areas to focus your attention on during further testing or exploitation.

A screenshot of a computer

Description automatically generated

29 Webmaster info

In the admin dashboard, the details of a client were found: Samir Negheenanajar, ‎including email and name. Well, good information, but nothing that leads directly to ‎vulnerabilities. It might be useful in the future for social engineering or targeted attacks. ‎Next, explore other clients or resources for potential weaknesses.‎

A screenshot of a computer program

Description automatically generated

30 Finding hideen index.php page

The discovery of /breach3/thebobscloudhostingllc/index.php opens the door for ‎further exploitation. Further steps include checking for SQL injection, file inclusion, ‎or command injection; inspecting source code for any vulnerability; and manipulating ‎requests using Burp Suite. Moreover, it could be possible to disclose critical vulnerabilities ‎by searching for configuration issues such as unprotected admin panels or insecure file ‎uploads. These actions are in order to find vulnerabilities for further access.‎

## 3.3 Privilege Escalation via Command Injection

A screenshot of a computer

Description automatically generated

31 Accessing live chat page

On the page /breach3/thebobscloudhostingllc/index.php, there was a live chat ‎support interface where the user had to fill in his name, email, and message ‎(subject) ‎to send a question to the support team. It would be a good opportunity to test how the server processes and handles the user input.‎

A screenshot of a computer

Description automatically generated

32 Capture web request

The captured HTTP request from the live chat form is sent to Burp Suite's ‎Repeater tool for manual manipulation. We should try to cause a vulnerability based on changing some inputs like name, email, or ‎message fields.‎

A screenshot of a computer

Description automatically generated

33 Result of command injection “whoami”

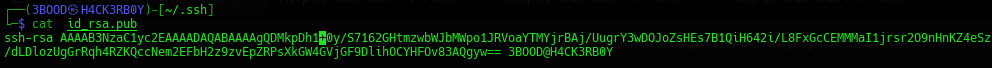
A screenshot of a computer

Description automatically generated

34 Result of command injection “sudo+-l”

The whoami command showed that injected commands run under "Samir." It means sudo -l ‎that Samir can use chmod with sudo privileges without a password. That provides the ways to‎ elevate privileges by either changing file permissions or creating a script with elevated ‎rights. Now, it's time to exploit this and get complete control of the system.‎

## 3.4 Establishing Persistent Access



36 View pen tester public SSH id



37 Edit special char to URL encoded

For this to work, the pentester's public key needs to be added ‎to ‎the server's authorized\_keys file. Here is how to:‎

1. ‎‎‎‎Create an SSH key pair on the attacker's machine:‎

ssh-keygen -t rsa

1. ‎‎URL encode the public key (id\_rsa.pub) being sure special characters like + ‎are properly encoded (for example, + becomes %2B).‎
2. ‎‎ Construct a command that appends the encoded public key to the authorized\_keys ‎file:‎

echo <encoded\_public\_key> >> /home/Samir/.ssh/authorized\_keys‎

1. Use the command injection vulnerability and inject the command into a vulnerable parameter, like the searcher parameter.‎

After adding the key, pentester can then connect to the server using private key via SSH without password authentication to attain persistent access.‎

A screenshot of a computer

Description automatically generated

Check the permission of .ssh directory 38

Before adding the SSH public key to the authorized\_keys file, let's check the ‎permissions of the.ssh directory. From the result of the ls -ld.ssh‎ command, it shows drwx------; this means only the ‎owner has access-which is Samir-and it is secure, but allows him to modify the ‎directory.‎

Check if .ssh contains the authorized\_keys file. If not, create it and add the pentester's SSH public key using echo <encoded\_public\_key> >> ‎‎/home/Samir/.ssh/authorized\_keys, making sure the permission is chmod 600 ‎‎/home/Samir/.ssh/authorized\_keys so that nobody else can access it.‎

Since Samir owns the directory, the following steps can be taken to grant SSH access ‎using the tester's private key to maintain persistent access to the server.‎



39 Chmod of .ssh directory to 777

A screen shot of a computer

Description automatically generated

Double check permissions 40

Permissions of the.ssh ‎directory were temporally changed with:‎ To embed the SSH public key into the authorized\_keys file

sudo -u thebobs chmod 777 /home/thebobs/.ssh

This will give read, write, and execute permissions to everyone, thereby making the ‎directory writable. It allows the pentester to add the public key to the ‎authorized\_keys file. This is a huge security risk and should be done only temporarily. Once the key is added, the permissions must be restored back to a secure ‎setting of 700 to block unauthorized access.‎



41 Chmod to 666

A screenshot of a computer

Description automatically generated

42 Double check authorized\_keys file permission

To allow writing to the authorized\_keys file with the pentester's SSH public key, the permission of the file was changed via:‎

sudo -u thebobs chmod 666 /home/thebobs/.ssh/authorized\_keys

This gave read and write permissions to the owner, group, and others, respectively, making the ‎file writable. This is not secure, but it is a temporary setup to add the public key for passwordless SSH ‎access. Once the key addition is done, it needs to be changed back to 600 for the file ‎to be secured. This is a very important step that ensures that the pentester will have persistent SSH access to the server.‎



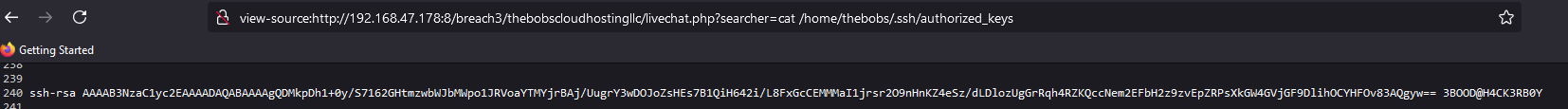
Write the public ID into authorized\_keys file 43

With the authorized\_keys file temporarily set to 666, the pentester added his SSH public ‎key with the following command:‎

echo <encoded\_public\_key> >> /home/thebobs/.ssh/authorized\_keys

Here, the key was URL-encoded to handle special characters, such as + to %2B, so that it would execute properly. The >> operator appended the public key to the file without overwriting any content that was already there. After adding it, the pentester could connect to the server with their private key and establish passwordless SSH for persistence.

That should be followed by changing the file permissions to 600 and the.ssh ‎directory to 700 in order to secure the system.‎



44 view the public id of the pen tester in the authorized\_keys file

In order to ensure that the SSH public key had been correctly inserted in the authorized\_keys ‎file, the pentester used the following command:‎

cat /home/thebobs/.ssh/authorized\_keys

This confirmed the key was correctly appended and that it matched the expected ‎value, hence ensuring the correct setup for SSH access. With this key in place, the ‎pentester will now be able to establish a secure, passwordless SSH connection.‎

After this confirmation, the next step will be to restore the secure permissions: 600 for ‎the authorized\_keys file and 700 for the.ssh directory, so that unauthorized ‎access or modifications will not occur.‎



45 Chmod to 700

After confirming that the SSH public key had been added to the ‎authorized\_keys file, the pentester secured the .ssh directory by restoring its ‎permissions with:‎

sudo -u thebobs chmod 700 /home/thebobs/.ssh‎

This command ensures that only the owner, thebobs, has full access: read, write, ‎execute, while others are denied. It uses sudo -u thebobs to execute as ‎the thebobs user and ensures that the.ssh directory is secure. This step prevents unauthorized changes to SSH configurations while maintaining the pentester's passwordless access by using their private key.‎



46 Chmod to 600

The last step in securing SSH was to restore the permissions of the ‎authorized\_keys file using:‎

sudo -u thebobs chmod 600 /home/thebobs/.ssh/authorized\_keys‎

This command ensures that only the user thebobs can read and write to the ‎file, in order to prevent unauthorized changes. It executes as ‎thebobs with sudo -u thebobs and uses chmod 600 to restrict access. With the file's permissions, along with earlier tweaks, this maintains a secure SSH configuration and preserves passwordless ‎access.‎

A screenshot of a computer

Description automatically generated

47 Accessing the system via SSH

After adding the SSH public key and changing permissions, the following was done: establishing an SSH connection using the following command:‎

ssh -i id\_rsa thebobs@192.168.47.178 -o PubkeyAcceptedKeyTypes=ssh-rsa

* ssh initiates an SSH connection.‎
* ‎-i id\_rsa: Connect using the private key for authentication.‎
* thebobs@192.168.47.178: Connects as thebobs to the server.‎
* ‎-o PubkeyAcceptedKeyTypes=ssh-rsa

Gained passwordless SSH access, allowing for further exploration and post-exploitation activities. This is a very important point in the test, where secure server access is achieved.A screenshot of a computer

Description automatically generated

48 Accessing the Terminal

After getting SSH access, an interactive shell was spawned with Python's pty ‎module using:‎

import pty

pty.spawn("/bin/bash")‎

This will create a pseudo-terminal session and spawn a Bash shell for easier navigation.

## 3.5 Privilege Escalation to Root



49 View system info

In the initial phases of scanning, it was identified that the server is running Ubuntu 4.4 on x86\_64 architecture. So, with the gathered information, privilege escalation will be done in order to obtain root access to the system. Ubuntu 4.4 may be an old version and probably contains unpatched vulnerabilities or misconfigurations which can be exploited to obtain escalated privileges.

A screen shot of a computer

Description automatically generated

50 Search of exploitation script

First, the searchsploit ‎tool was used to identify a privilege escalation exploit for the Ubuntu 4.4 machine, with the command: searchsploit linux kernel ubuntu 4.4 privilege ‎escalation. This led to the Dirty COW exploit, CVE-2016-5195, as a potentially ‎vulnerable one. This exploit attacks a race condition in the kernel's copy-on-write ‎mechanism and escalates a local user to root. The exploit was ‎downloaded with searchsploit -m 43418, or alternatively from the Exploit Database. It had to be downloaded, then compiled and executed on the target system in order to escalate privileges to root, underlining the importance of utilizing known vulnerabilities for privilege escalation during penetration testing.

A screenshot of a computer screen

Description automatically generated

51 gcc the script and transfer it to targeted system

First, the pentester downloaded the exploit file 43418.c and compiled it into an ‎executable on his local machine using the command gcc 43418.c -static -o 43418‎ because GCC and root access were not available on the target server. The compiled exploit was encoded in Base64 to avoid character alteration during ‎transfer using the following command: base64 43418 > 43418\_enc. Thus, a Base64 encoded file was copied to the ‎target server, decoded back to its original binary with the base64 -d ‎‎43418\_enc > 43418, a way for the exploit to be safe in transfer for execution ‎on the target system.

‎A screenshot of a computer screen

Description automatically generated

52 Execute the exploitation

The Base64 encoded payload was uploaded to the targeted server, having been decoded in the first step by using a command cat 43418\_enc | base64 -d > 43418. Subsequent verification of that file showed via the command: file. After that was performed, an attempt was done with setting execute permission: chmod +x 43418. A final touch and execution of exploit itself through a simple:./43418 Dirty COW could be exploited hence privilege escalation: root access totally controlled.

A screenshot of a computer screen

Description automatically generated

53 Gaining Root Access

After the execution of./43418, it was able to trigger the Dirty COW vulnerability and provided root access to the server. This privilege escalation allowed the pentester to take full control over the system and perform a number of actions like editing files, changing configurations, and viewing sensitive data.

Successful privilege escalation to root was considered the completion of the penetration test. It also highlighted the need to patch known vulnerabilities and ensure secure ‎configurations that prevent such exploits.‎

# 4.0 COUNTERMEASURES PROPOSED

Some key countermeasures are recommended to improve the security of the system based on the identified vulnerabilities from the penetration test. First, regular updates and patch management are very important. Since one of the key vectors was the Dirty COW exploit, CVE-2016-5195, keeping the system updated consistently, especially for kernel vulnerabilities, will reduce the possibility of exploitation significantly. (Asadi, Scandariato, & Sabelfeld, 2017)

Insecure services, such as Telnet, should not be used. Whenever possible, shutdown and replace them with more secure alternatives. Example SSH. This is because telnet transmits in plain text. This presents an open vulnerability when intercepted. SSH provides encryption, along with robust authentication of the communication including key-based log in. The unnecessary services need to be turned off. (Smith & Johnson 2018)

Moreover, to harden SSH configuration for secure remote access, root logins should be disabled, access should be granted to specific IP addresses, and key-based authentication should be enforced. In addition, file permissions on critical files like authorized\_keys should be set appropriately to prevent unauthorized access. Smith & Johnson, 2018 ‎

Another key measure is a different approach in the principle of least privilege. That is, limiting users' permissions to the bare minimum required for them to do their work. Limiting sudo privileges and periodic auditing of user permissions will reduce the attack surface of a system (Wang, Li & Chen, 2019).

Protection against SQL injection and other common attacks finally can be achieved by proper input validation, use of prepared statements, and periodic vulnerability assessment. This will help mitigate the attack on web interfaces that may lead to sensitive data exposure or entry points into the system.‎

# 5.0 CONCLUSION

The other important pentest on Ubuntu 4.4 x86\_64 found several critical ‎‎vulnerabilities such as privilege escalation by means of Dirty COW, CVE-‎‎2016-‎‎5195, ‎insecure services like Telnet. Possible entry points include SQL injection and ‎command ‎injection on the web applications, using various tools such as searchsploit, ‎Nmap, and ‎sqlmap. In fact, it is because of these weaknesses that attackers gain privileges and unauthorized ‎access.‎ These findings have pointed out the need for periodic patching, securing of remote ‎access, and ‎proper configuration of sensitive files and services. Such vulnerabilities ‎‎will be minimized in the future with updated software, secure SSH settings, and web ‎‎application hardening. Finally, proactive security-related measures will pay the way forward ‎for better system resiliency against breaches in the future.‎

# 6.0 REFERENCES

* Badonnel, R., Koch, R., Drasar, M., Pras, A., Eiseler, V., Stiemert, L., Seeber, S., Tuncer, D., Charalambides, M., & Rodosek, G. D. (2018). Report on the 10th International Conference on Autonomous Infrastructure, Management, and Security (AIMS 2016). *Journal of Network and Systems Management*, *26*(4), 1101–1109. https://doi.org/10.1007/s10922-018-9460-5
* Xu, J., Liang, S., Liu, S., Qiao, L., Sun, S., Deng, Q., & Zhu, H. (2017). Passively Mode-Locked Quantum-Well Laser With a Saturable Absorber Having Gradually Varied Bandgap. *IEEE Photonics Technology Letters*, *29*(11), 889–892. https://doi.org/10.1109/lpt.2017.2693408
* Hadi Habibzadeh, Cem Kaptan, Tolga Soyata, Burak Kantarci, & Azzedine Boukerche. (2019). Smart City System Design. *ACM Computing Surveys*, *52*(2), 1–38. https://doi.org/10.1145/3309545

‌

-- word count: 4107--