第四十六期 《Log4j2 高风险漏洞的来龙去脉》

Log4j2 高风险漏洞

安全漏洞

CVE-2021-44228

时间 - 20211126

描述

Apache Log4j2 2.0-beta9 到 2.12.1 和 2.13.0 到 2.15.0 JNDI 功能在配置、日志消息和参数中使用，不能防止攻击者控制的 LDAP 和其他 JNDI 相关端点。 当启用消息查找替换时，可以控制日志消息或日志消息参数的攻击者可以执行从 LDAP 服务器加载的任意代码。 从 log4j 2.15.0 开始，默认情况下已禁用此行为。 从版本 2.16.0 开始，此功能已完全删除。 请注意，此漏洞特定于 log4j-core，不会影响 log4net、log4cxx 或其他 Apache 日志服务项目。

详情

https://cve.mitre.org/cgi-bin/cvename.cgi?name=2021-44228

CVE-2021-45046

时间 - 20211214

描述

发现 Apache Log4j 2.15.0 中针对 CVE-2021-44228 的修复在某些非默认配置中不完整。 当日志配置使用非默认模式布局和上下文查找（例如，$${ctx:loginId}）或线程上下文映射模式（%X、%mdc 或 %MDC）使用 JNDI 查找模式制作恶意输入数据，从而导致拒绝服务 (DOS) 攻击。 默认情况下，Log4j 2.15.0 尽最大努力将 JNDI LDAP 查找限制为 localhost。 Log4j 2.16.0 通过删除对消息查找模式的支持和默认禁用 JNDI 功能来修复此问题。

详情

https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-45046

测试代码

https://github.com/cckuailong/Log4j_CVE-2021-45046

修复方案

CVE-2021-44228 修复方案

方案一：使用安全产品隔离非授权服务器 IP

推荐指数：*

方案二：前端（网关）对请求参数进行特殊过滤

推荐指数：*

方案三：删除风险类 - org.apache.logging.log4j.core.lookup.JndiLookup

推荐指数：**

Spring Boot 不太实现 FAT JAR

方案四：配置禁用 log4j2 lookup

推荐指数：**

1. 设置日志输出 Pattern 格式

2.7以及以上的版本，在 %msg 占位符后面添加 {nolookups}：

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
    <Appenders>
        <Console name="Console"
target="SYSTEM_OUT">
            <PatternLayout pattern="%-5level -
%msg{nolookups}%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Root level="error">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>
</Configuration>
```

2. JVM 系统属性

-Dlog4j2.formatMsgNoLookups=true

3. log4j2.component.properties 配置文件

log4j2.component.properties 中添加：

```
log4j2.formatMsgNoLookups=true
```

4. 环境变量

LOG4J_FORMAT_MSG_NO_LOOKUPS=true

方案五：升级 JDK 版本

推荐指数：***

Oracle JDK >= 11.0.1、8u191、7u201、6u211

`com.sun.jndi.rmi.object.trustURLCodebase` "true" -> "false"

`com.sun.jndi.ldap.object.trustURLCodebase` "true" -> "false"

方案六：升级 Log4j 2.16.0 +

方案七：Java Security 控制远程代码执行

方案八：通过 ClassPath 下的 jndi.properties 文件 java.naming.factory.url.pkgs 的 package前缀

方案九：修改全局的 javax.naming.spi.InitialContextFactoryBuilder

通过 Java 设置，如下：

```
NamingManager.setInitialContextFactoryBuilder(new
FileSystemInitialContextFactoryBuilder());
```

CVE-2021-45046 修复方案

方案一

升级 Log4j 2.16.0

生效条件

Oracle JDK < 11.0.1、8u191、7u201、6u211

`com.sun.jndi.rmi.object.trustURLCodebase` "true"

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

原理分析

Log4j2

特性 - Lookups

JDNI Lookup

https://logging.apache.org/log4j/2.x/manual/lookups.html#JndiLookup

org.apache.logging.log4j.core.lookup.JndiLookup

`${jndi:ldap://127.0.0.1:1099/Exploit}`

jndi -> org.apache.logging.log4j.core.lookup.JndiLookup

JNDI ->

- ldap://127.0.0.1:1099/Exploit
  - com.sun.jndi.url.ldap.ldapURLContextFactory
- rmi://127.0.0.1:1099/Exploit
  - com.sun.jndi.url.rmi.rmiURLContextFactory
- file:///${user.home}/Exploit
  - com.sun.jndi.url.file.fileURLContextFactory

jndi.properties

```
java.naming.factory.initial =
jndi.file.FileSystemInitialContextFactory
```

org.apache.logging.log4j.core.lookup.Interpolator#Interpolator(java.util.Map<java.lang.String,java.lang.String>):

```
        // JNDI
        try {
            // [LOG4J2-703] We might be on
Android
            strLookupMap.put(LOOKUP_KEY_JNDI,

 Loader.newCheckedInstanceOf("org.apache.logging.
log4j.core.lookup.JndiLookup", StrLookup.class));
        } catch (final LinkageError | Exception
e) {
            handleError(LOOKUP_KEY_JNDI, e);
        }
```

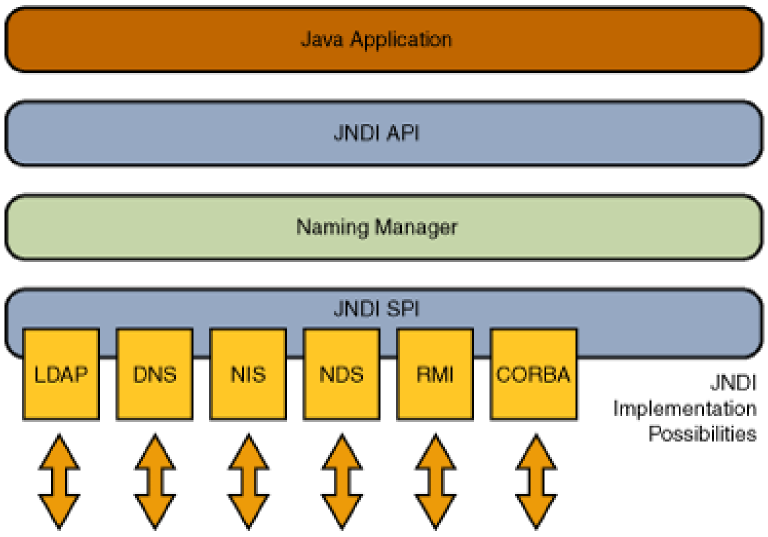
RMI - Remote Method Invocation

JNDI - Java Naming and Directory Interface

参考文档

官方文档：https://docs.oracle.com/javase/tutorial/jndi/overview/index.html

架构



JNDI 提供 SPI 为底层实现作统一抽象，上层应用使用 JNDI API 进行资源统一的查找模式。

分发包（Packaging）

JDK 自带部分实现：

- Lightweight Directory Access Protocol (LDAP)
- Common Object Request Broker Architecture (CORBA) Common Object Services (COS) name service

- Java Remote Method Invocation (RMI) Registry
- Domain Name Service (DNS)

协议实现通常存放在 `com.sun.jndi.url` 包下。API 包存放在：

- javax.naming
- javax.naming.directory
- javax.naming.ldap
- javax.naming.event
- javax.naming.spi

特性

- 组件容器
- 容器配置
  - 如 Context#getEnvironment() 方法
- 容器生命周期
  - 如 Context#close() 方法
- 上下文层次性
  - javax.naming.Context#createSubcontext 方法
- 别名方法
  - Context#lookupLink 方法
- 时间/监听器

核心接口

javax.naming.Context

特性分类

- 查找
    - javax.naming.Context#lookup(javax.naming.Name)
- 注册
    - javax.naming.Context#bind(javax.naming.Name, java.lang.Object)
    - javax.naming.Context#rebind(javax.naming.Name, java.lang.Object)
- 注销
    - javax.naming.Context#unbind(javax.naming.Name)
- 列表
    - javax.naming.Context#list(javax.naming.Name)

类比 Spring 实现

组件	JNDI	Spring Framework
上下文	javax.naming.Context	org.springframework.beans.factory.BeanFactory
组件名	javax.naming.Name	Bean 名称（String 类型）
组件名和类型	javax.naming.NameClassPair	BeanDefinitionHolder
组件名与组件对选哪个	javax.naming.Binding	BeanDefinition 和 Bean 对象
配置	Hashtable getEnvironment()	类似于 PropertySource

javax.naming.spi.ObjectFactory

接口定义

```
public interface ObjectFactory {
/**
```

```
 * Creates an object using the location or
reference information
 * specified.
 * <p>
 * Special requirements of this object are
supplied
 * using <code>environment</code>.
 * An example of such an environment property is
user identity
 * information.
 *<p>
 * <tt>NamingManager.getObjectInstance()</tt>
 * successively loads in object factories and
invokes this method
 * on them until one produces a non-null answer.
When an exception
 * is thrown by an object factory, the exception
is passed on to the caller
 * of <tt>NamingManager.getObjectInstance()</tt>
 * (and no search is made for other factories
 * that may produce a non-null answer).
 * An object factory should only throw an
exception if it is sure that
 * it is the only intended factory and that no
other object factories
 * should be tried.
 * If this factory cannot create an object using
the arguments supplied,
 * it should return null.
 *<p>
 * A <em>URL context factory</em> is a special
ObjectFactory that
 * creates contexts for resolving URLs or objects
whose locations
```

```
 * are specified by URLs.  The
<tt>getObjectInstance()</tt> method
 * of a URL context factory will obey the
following rules.
 * <ol>
 * <li>If <code>obj</code> is null, create a
context for resolving URLs of the
 * scheme associated with this factory. The
resulting context is not tied
 * to a specific URL:  it is able to handle
arbitrary URLs with this factory's
 * scheme id.  For example, invoking
<tt>getObjectInstance()</tt> with
 * <code>obj</code> set to null on an LDAP URL
context factory would return a
 * context that can resolve LDAP URLs
 * such as "ldap://ldap.wiz.com/o=wiz,c=us" and
 * "ldap://ldap.umich.edu/o=umich,c=us".
 * <li>
 * If <code>obj</code> is a URL string, create an
object (typically a context)
 * identified by the URL.  For example, suppose
this is an LDAP URL context
 * factory.  If <code>obj</code> is
"ldap://ldap.wiz.com/o=wiz,c=us",
 * getObjectInstance() would return the context
named by the distinguished
 * name "o=wiz, c=us" at the LDAP server
ldap.wiz.com.  This context can
 * then be used to resolve LDAP names (such as
"cn=George")
 * relative to that context.
 * <li>
 * If <code>obj</code> is an array of URL
strings, the assumption is that the
 * URLs are equivalent in terms of the context to
which they refer.
 * Verification of whether the URLs are, or need
to be, equivalent is up
 * to the context factory. The order of the URLs
in the array is
 * not significant.
 * The object returned by getObjectInstance() is
like that of the single
 * URL case.  It is the object named by the URLs.
 * <li>
 * If <code>obj</code> is of any other type, the
behavior of
 * <tt>getObjectInstance()</tt> is determined by
the context factory
 * implementation.
 * </ol>
 *
 * <p>
 * The <tt>name</tt> and <tt>environment</tt>
parameters
 * are owned by the caller.
 * The implementation will not modify these
objects or keep references
 * to them, although it may keep references to
clones or copies.
 *
 * <p>
 * <b>Name and Context Parameters.</b>
&nbsp;&nbsp;&nbsp;
 * <a name=NAMECTX></a>
 *
 * The <code>name</code> and <code>nameCtx</code>
parameters may
 * optionally be used to specify the name of the
object being created.
 * <code>name</code> is the name of the object,
relative to context
 * <code>nameCtx</code>.
 * If there are several possible contexts from
which the object
 * could be named -- as will often be the case --
it is up to
 * the caller to select one.  A good rule of
thumb is to select the
 * "deepest" context available.
 * If <code>nameCtx</code> is null,
<code>name</code> is relative
 * to the default initial context.  If no name is
being specified, the
 * <code>name</code> parameter should be null.
 * If a factory uses <code>nameCtx</code> it
should synchronize its use
 * against concurrent access, since context
implementations are not
 * guaranteed to be thread-safe.
 * <p>
 *
 * @param obj The possibly null object containing
location or reference
 *              information that can be used in
creating an object.
 * @param name The name of this object relative
to <code>nameCtx</code>,
 *              or null if no name is specified.
 * @param nameCtx The context relative to which
the <code>name</code>
 *                parameter is specified, or null
if <code>name</code> is
 *                relative to the default initial
context.
 * @param environment The possibly null
environment that is used in
 *                creating the object.
 * @return The object created; null if an object
cannot be created.
 * @exception Exception if this object factory
encountered an exception
 * while attempting to create an object, and no
other object factories are
 * to be tried.
 *
 * @see NamingManager#getObjectInstance
 * @see NamingManager#getURLContext
 */
    public Object getObjectInstance(Object obj,
Name name, Context nameCtx,
                                    Hashtable<?,?
> environment)
        throws Exception;
}
```

方法参数:

- obj - The possibly null object containing location or reference information that can be used in creating an object
- name - 可能是一个相对于 `nameCtx` 参数的 Name 对象，也可能是 null

- nameCtx - 如果 name 相对于 initial context，则为 null
- environment - 配置对象

> 该接口类似于 Spring Framework org.springframework.beans.factory.ObjectFactory，如果要实现 JNDI 这样方法参数的话，Spring Bean 需要额外实现这些接口：
>
> - name 参数 - BeanNameAware
> - nameCtx 参数 - ApplicationContextAware
> - environment 参数 - EnvironmentAware

标准操作步骤（基于局部 InitialContextFactory 实现）

步骤一: 在 Environment 中设置 InitialContext 服务提供方

比如：

```
Hashtable<String, Object> env = new
Hashtable<String, Object>();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.sun.jndi.ldap.LdapCtxFactory");
```

以上语义是配置 `javax.naming.spi.InitialContextFactory` 实现，比如 `com.sun.jndi.ldap.LdapCtxFactory`：

```
public final class LdapCtxFactory implements
ObjectFactory, InitialContextFactory {
    ...
}
```

如果在应用启动时，每次需要代码配置这个实现，对于功能移植性不友好。是否能够通过一个配置文件来达到 jar（artifact）迁移的目的？答案是可以实现，这个文件存放在 jndi.properties 中。

步骤二：在 Environment 设置 JNDI InitialContext 配置

如：

```
env.put(Context.PROVIDER_URL,
"ldap://ldap.wiz.com:389");
env.put(Context.SECURITY_PRINCIPAL, "joeuser");
env.put(Context.SECURITY_CREDENTIALS,
"joepassword");
```

因为 `javax.naming.spi.InitialContextFactory` 接口在获取 InitialContext 时，能够使用 Environment（Hashtable）对象：

```
public interface InitialContextFactory {


        ...
        public Context
getInitialContext(Hashtable<?,?> environment)
            throws NamingException;
}
```

步骤三：创建 InitialContext 对象

如：

```
Context ctx = new InitialContext(env);
```

可参考项目中的实现：

```
InitialContext context = new InitialContext();
String name = "abc";
Object value = "Hello,World";
Context envContext = (Context)
context.lookup("java:comp/env");
envContext.bind(name, value);

assertEquals(value, envContext.lookup(name));

envContext.unbind(name);
assertNull(envContext.lookup(name));
```

扩展操作步骤（基于全局 InitialContextFactory 实现）

步骤一：通过代码实现 javax.naming.spi.InitialContextFactoryBuilder

如：

```
public class
FileSystemInitialContextFactoryBuilder implements
InitialContextFactoryBuilder {

    @Override
    public InitialContextFactory
createInitialContextFactory(Hashtable<?, ?>
environment) throws NamingException {
        FileSystemInitialContextFactory
initialContextFactory = new
FileSystemInitialContextFactory();
        return initialContextFactory;
    }
}
```

步骤二：关联全局 javax.naming.spi.InitialContextFactoryBuilder 实现

如：

```
NamingManager.setInitialContextFactoryBuilder(new
FileSystemInitialContextFactoryBuilder());
```

其他步骤与“标准操作步骤”一致。

自定义 JNDI 实现

配置化自定义 JNDI 实现

配置方式

- 内部化配置（代码）
- 外部化配置（外部资源）
  - Applet 参数
    - 参考方法：com.sun.naming.internal.ResourceManager#getInitialEnvironment
  - Java System Properties
    - 参考方法：com.sun.naming.internal.VersionHelper#getJndiProperties
  - 应用资源文件（ClassPath 下的 "jndi.properties"）
    - 参考方法：com.sun.naming.internal.ResourceManager#getApplicationResources
    - 可失效（"com.sun.naming.disable.app.resource.files"）

实现步骤

- 实现 javax.naming.spi.InitialContextFactory
- 实现 javax.naming.Context

参考实现：https://github.com/mercyblitz/geekbang-lessons/tree/master/projects/stage-1/middleware-frameworks/my-commons/src/main/java/org/geektimes/commons/jndi/file

全局设置自定义 JNDI 实现

其他实现

Apache Tomcat JNDI 实现

http://tomcat.apache.org/tomcat-8.5-doc/jndi-resources-howto.html

不足

- 缺少丰富组件生命周期管理（实例化）

资源推荐

The JNDI Tutorial

参考资料

[A JOURNEY FROM JNDI/LDAP MANIPULATION TO REMOTE CODE EXECUTION DREAM LAND]

[如何绕过高版本JDK的限制进行JNDI注入利用]