登录

Apache Log4j 反序列化分析—【CVE-2017-5645】

orich1 / 2018-01-08 09:12:00 / 浏览数 4833 技术文章 技术文章 顶(0) 踩(0)

---

apache Log4j 组件漏洞描述：

```
CVE-2017-5645: Apache Log4j socket receiver deserialization vulnerability

Severity: High

CVSS Base Score: 7.5 (AV:N/AC:L/Au:N/C:P/I:P/A:P)

Vendor: The Apache Software Foundation

Versions Affected: all versions from 2.0-alpha1 to 2.8.1

Description: When using the TCP socket server or UDP socket server to
receive serialized log events from another application, a specially crafted
binary payload can be sent that, when deserialized, can execute arbitrary
code.

Mitigation: Java 7+ users should migrate to version 2.8.2 or avoid using
the socket server classes. Java 6 users should avoid using the TCP or UDP
socket server classes, or they can manually backport the security fix from
2.8.2: <https://git-wip-us.apache.org/repos/asf?p=logging-log4j2.
git;h=5dcc192>
```

Log4j简介

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
Log4j■Apache■■■■■■■■■■■■■Log4j■■■■■■■■■■■■■■■■■■■■■■■■■■GUI■■■■■■■■■■■■■■NT■■■■■■■UNIX Syslog■■■■■■■■■■■■■■■■■
Log4j■■■■■■■■■■■■■■■■ System.out ■■■■■■■■Java■■■■■■■■■■■■■■■■■■■■■■■■spring■hibernate■struts■■■■■■■■■■■■■■■■■■■
```

流程图

TCP

UDP

漏洞分析

```
■■■java■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

`TCP or UDP socket server classes`，那么就去全文搜一下tcp udp socket server 相关关键字

其实是有对应的 TcpSocketServer 和 UdpSocketServer 的类

TCP

那么我们直接去查看TcpSocketServer
先看一下类结构

CommandLineArguments类是自定义处理命令行参数用的
SocketHandler类用于处理客户端连接
然后就是三个构造函数，接着`create*SocketServer`函数用于创建各类服务端，main用于直接运行的，extract用于创建一个`ServerSocket`并返回
run函数存在是因为 `TcpSocketServer`
继承于`AbstractSocketServer`，而这个抽象类继承了Runable接口。在`TcpSocketServer`类中，run函数用于接受客户端连接并且交于SocketHandler处理连接
shutdown函数作用是清理并关闭线程

现在，我们从创建一个Tcp的远程日志服务开始跟起，在源码中，有TcpSerializedSocketServerTest类向我们展示了这一操作，虽然它的功能仅仅是用于测试..

在TcpSerializedSocketServerTest中，只有一个构造函数和createLayout函数，剩下的两个如上图所示，其中createLayout函数并没有具体的代码，所以和构造函数一样我
@BeforeClass注解表示该函数用于测试类实例化前执行的函数，并且针对所有测试函数，它只会执行一次，就像static块一样
@AfterClass注解表示测试类实例化后只会执行一次
我们直接看setupClass函数，它先是获取了一下Log的上下文然后就调用了TcpSocketServer的`createSerializedSocketServer`函数，传入的是一个int，跟过去看看啥

在`AbstractSocketServerTest`函数里

继续跟进getNextAvailable，就不贴出来了，说明一下功能，就是从1100端口开始计算，返回一个当前的端口中空闲着的最小端口号

我们先不急着跟进createSerializedSocketServer函数，继续往下看，它调用了TcpSocketServer的startNewThread函数，这个函数具体实现在AbstractSocketS

直接start了，看一下 server.startNewThread的server是啥类型的

因为是TcpSocketServer的对象，所以start后执行就是TcpSocketServer里的run函数

现在我们跟进createSerializedSocketServer函数看看，注意形参是int类型的

参数和返回值都说清楚了，int就是用于监听的端口，返回一个新的socket server，新建的TcpSocketServer带入了ObjectInputStreaLogEventBridge对象

当Test类拿到TcpSocketServer对象后，就会执行其run函数，我们看一下详细内容
函数体有点长，就贴代码不贴图了

```
/**
* Accept incoming events and processes them.
*/
@Override
public void run() {
   final EntryMessage entry = logger.traceEntry();
   while (isActive()) {
       if (serverSocket.isClosed()) {
           return;
       }
       try {
           // Accept incoming connections.
           logger.debug("Listening for a connection {}...", serverSocket);
           final Socket clientSocket = serverSocket.accept();
           logger.debug("Acepted connection on {}...", serverSocket);
           logger.debug("Socket accepted: {}", clientSocket);
           clientSocket.setSoLinger(true, 0);

           // accept() will block until a client connects to the server.
           // If execution reaches this point, then it means that a client
           // socket has been accepted.

           final SocketHandler handler = new SocketHandler(clientSocket);
           handlers.put(Long.valueOf(handler.getId()), handler);
           handler.start();
       } catch (final IOException e) {
           if (serverSocket.isClosed()) {
               // OK we're done.
               logger.traceExit(entry);
               return;
           }
           logger.error("Exception encountered on accept. Ignoring. Stack trace :", e);
       }
   }
   for (final Map.Entry<Long, SocketHandler> handlerEntry : handlers.entrySet()) {
       final SocketHandler handler = handlerEntry.getValue();
       handler.shutdown();
       try {
           handler.join();
       } catch (final InterruptedException ignored) {
           // Ignore the exception
       }
   }
   logger.traceExit(entry);
}
```

一进来就是一个循环，先判断了socket是否关闭，如果没有的话，就接受从客户端传递的数据，如果已经关闭则退出循环，退出循环后做一些清理工作。

我们来看看接受客户端的情况，如下图

serverSocket就是根据之前传入的int端口号来新建的一个ServerSocket对象。注意红框里，将clientSocket作为参数实例化SocketHandler类，然后放入handlers中，ha

我们去看看这个 SocketHandler 具体做了什么

构造函数里，获取了 socket 中的数据流后，传入了 logEventInput 的wrapStream中，看看logEventInput

是一个LogEventBridge类型的，还记得在之前的createSerializedSocketServer函数中的新建ObjectInputStreamLogEventBridge对象吗，这里的logEventI

那么我们跟进 wrapStream函数

发现有AbstractLogEventBridge和ObjectInputStreamLogEventBridge实现了，这里我们当然选择跟进ObjectInputStreamLogEventBridge类中

就将传入的inputStream用ObjectInputStream包装一下然后返回了
那么这个SocketHandler中的inputStream就是一个ObjectInputStream对象了

我们接着看SocketHandler的run函数

直接将inputStream带入了logEventInput的logEvents函数中
跟进去看看

肯定是ObjectInputStreamLogEventBridge里的logEvents

如上图，inputStream传入logEvents后，直接调用了readObject函数，这里就触发了反序列化

UDP

查看UdpSocketServer
先看结构

与TcpSocketServer的结构类似，create*SocketServer函数用于创建接受不同类型数据的Socket Server

我们直接去看他的run函数

其他操作都很正常，接收数据后，提取二进制流赋值给bais，然后带入wrapStream处理成ObjectInputStream，然后传入ObjectInputStreamLogEventBridge中的

一些其他的尝试

在TCP和UDP中，我们注意到关键点都在于这个logEventsInput的类型，如果它是ObjectInputStreamLogEventBridge类型，那么在后面调用logEvents函数的时
这个logEvents函数通过名字就可以判断出，它是用于将接收到的数据做一下日志记录的，所以在其他流程中，也仅仅是为这个记录操作提供前提条件，仅仅是将数据接收

那么logEventsInput如果是其他类型呢，其他类型的LogEventBridge又会对接收到的数据如何进行处理，在处理过程中会不会有问题存在？

我们先去看看LogEventBridge的子类有哪些（ logEvent由LogEventBridge定义 ）

LogEventBridge本身是一个接口，AbstractLogEventBridge是LogEventBridge的抽象类，所以我们的关注点在于
*StreamLogEventBridge，之前的反序列化是由于ObjectInputStreamLogEventBridge触发的，那么我们去看看
InputStreamLogEventBridge的logEvents函数
XmlInputStreamLogEvnetBridge和JsonInputStreamLogEventBridge并没有实现logEvents函数

```
@Override
public void logEvents(final InputStream inputStream, final LogEventListener logEventListener) throws IOException {
   String workingText = Strings.EMPTY;
   try {
       // Allocate buffer once
       final byte[] buffer = new byte[bufferSize];
       String textRemains = workingText = Strings.EMPTY;
       while (true) {
           // Process until the stream is EOF.
           final int streamReadLength = inputStream.read(buffer);
           if (streamReadLength == END) {
               // The input stream is EOF
               break;
           }
           final String text = workingText = textRemains + new String(buffer, 0, streamReadLength, charset);
           int beginIndex = 0;
           while (true) {
               // Extract and log all XML events in the buffer
               final int[] pair = getEventIndices(text, beginIndex);
               final int eventStartMarkerIndex = pair[0];
               if (eventStartMarkerIndex < 0) {
                   // No more events or partial XML only in the buffer.
                   // Save the unprocessed string part
                   textRemains = text.substring(beginIndex);
                   break;
               }
               final int eventEndMarkerIndex = pair[1];
               if (eventEndMarkerIndex > 0) {
                   final int eventEndXmlIndex = eventEndMarkerIndex + eventEndMarker.length();
                   final String textEvent = workingText = text.substring(eventStartMarkerIndex, eventEndXmlIndex);
                   final LogEvent logEvent = unmarshal(textEvent);
```

```
                    logEventListener.log(logEvent);
                    beginIndex = eventEndXmlIndex;
                } else {
                    // No more events or partial XML only in the buffer.
                    // Save the unprocessed string part
                    textRemains = text.substring(beginIndex);
                    break;
                }
            }
        }
    } catch (final IOException ex) {
        logger.error(workingText, ex);
    }
}
```

主要代码如下图

先在字符数组中截取特定标签之间的字符，然后传入 unmarshal 函数进行反序列化操作，不过这里的反序列化是由jackson框架操作

unmarshal函数：

但是里面的ObjectMapper没有调用enableDefaultTyping函数...

InputStreamLogEventBridge可以解析json和xml，json的反序列化没有找到利用点，而解析xml也是直接解析的
`<Event></Event>`或是`<Event`标签里的数据，技术水平限制了我的想象，不知道代码中还有啥处理问题了....

漏洞通报：http://seclists.org/oss-sec/2017/q2/78
log4j简介：https://www.cnblogs.com/shanheyongmu/p/5650632.html
log4j-core源码分析：http://www.blogjava.net/DLevin/archive/2012/06/28/381667.html

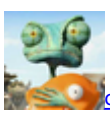
上一篇：一步一步 Pwn RouterOS... 下一篇：一步一步 Pwn RouterOS...

1. 2 条回复


cike 2018-01-12 13:39:35

附带一个github上面的 R C E

0 回复Ta


orich1 2018-01-14 21:49:48

@cike emmmmmm,这个rce靠commons-colloection的呀，我记得log4j不带的....求教难道可以用这个CommonsCollections5 直接打过去吗

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板