Apache Batik XXE—【CVE-2017-5662】

orich1 / 2018-01-10 09:26:03 / 浏览数 4261 技术文章 技术文章 顶(2) 踩(0)

漏洞简介

cve-2017-5662：batik < 1.9

Description
In Apache Batik before 1.9, files lying on the filesystem of the server which uses batik can be revealed to arbitrary users who send maliciously formed SVG files. The file types that can be shown depend on the user context in which the exploitable application is running. If the user is root a full compromise of the server - including confidential or sensitive files - would be possible. XXE can also be used to attack the availability of the server via denial of service as the references within a xml document can trivially trigger an amplification attack.

apache batik

Batik是Batik SVG Toolkit或Batik Java SVG
Toolkit的简称，一个基于Java的应用程序或小应用的工具集，意图将SVG格式用于多种目的，如察看，主控或操纵。该项目的目标是让开发一套核心模块，以及实现高度可

调用流程

分析过程

网上除了那一段描述之外啥也没了，那我们就自己分析

emm，这个组件可以操作svg，那么看看svg

很显眼的几个字，使用XML格式定义图形，svg里用xml格式.....xxe

那么就先把batik_1.8源码包下好，由于不那么新，有些依赖没能自动加载上，所以用idea的时候需要手动加上几个包（需要ant编译）
加上 xmlgraphics-commons 、 fop-transcoder-allinone 、 jacl 、 jython 应该就可以正常跑起来了

现在要做的就是，让它真正跑起来，先随便加载一个svg试一试
额，运气比较好，他写了gui的
路径如下：

直接执行 void main 就行

在 batik-batik-1_8\samples 路径下有不少svg，随便跑一个，成功解析了

那么我们现在自己构造一个svg，去试试

比如这样的

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE ANY SYSTEM "http://192.168.204.142:8888/test">
<svg>orich1</svg>
```

收到了连接请求，这就说明xxe是存在的，并且应该就是在解析svg文件时触发的

那么现在我们有两个选择，要么打断点使用payload调试，要么直接看代码静态跟进
虽然调试的方式速度快很多，但是我觉得静态跟进更亲切一点，也能更清楚看见调用流程中发生了什么

那么就从 void main 开始看

跟进Main的构造函数（其实就是当前类的构造函数）

代码太长，不全帖
它首先判断了操作系统平台，还有一大堆的对话框初始化，接着就是设置各种Canvas的Listener，说实话看到这里有点发慌，不知道能不能继续跟出来

但是看到这个构造函数最后的时候，发现它是这么操作的

我们不管其他的，可以肯定svgInitializationURI是一个字符串，然后被带进了loadSVGDocument函数里，从函数名和参数名可以猜测一下，他有一个初始化用的svg文件，

不过不放心，我们先去看看这个被加载的路径是啥 SVG_INITIALIZATION

恩，现在十分确定，它初始化时就加载了一个init.svg文件，那么就应该跟进这个 loadSVGDocument 函数

做了一些简单url解析和判断，然后又给 stopThenRun 传递了一个匿名内部类，按理说这里我们是应该跟进 stopThenRun
看看详情的，但是仔细看一下它在这里重写的run函数

内容大致是对 DocumentLoader 的一些初始化处理，我们知道这个 run 是肯定会被执行的，被执行的时候才在做 DocumentLoader 的初始化，那么就说明 stopThenRun 里面肯定还没有对svg进行解析，所以我们可以直接跟进 startDocumentLoader 函数

如上图，它就在 JSVGComponnent 类中，这里我们可以直接判断，documentLoader 可以开启线程，那么就去找他的 run 函数，看看做了啥

首先找到 documentLoader 的类型

跟进 SVGDocumentLoader，里面刚好有个 run 函数

其他部分都是catch，就不贴出来了
继续跟进 loadDocument 函数，因为我们只需要知道这个初始化svg是如何被解析的

开始创建 Document 了，继续跟进 createSVGDocument
发现跟进了 SVGDocumentFactory 接口，不过没事，看一下类关系就好

恩，只有一个 SAXSVGDocumentFactory 实现了 SVGDocumentFactory 接口，那么 createSVGDocument 就在里面没跑了

继续跟进 createDocument

```
/**
* Creates a SVG Document instance.
* This method supports gzipped sources.
* @param uri The document URI.
* @exception IOException if an error occured while reading the document.
*/
public Document createDocument(String uri) throws IOException {
   ParsedURL purl = new ParsedURL(uri);

   InputStream is = purl.openStream
      (MimeTypeConstants.MIME_TYPES_SVG_LIST.iterator());
   uri = purl.getPostConnectionURL();

   InputSource isrc = new InputSource(is);

   // now looking for a charset encoding in the content type such
   // as "image/svg+xml; charset=iso8859-1" this is not official
   // for image/svg+xml yet! only for text/xml and maybe
   // for application/xml
   String contentType = purl.getContentType();
   int cindex = -1;
   if (contentType != null) {
       contentType = contentType.toLowerCase();
       cindex = contentType.indexOf(HTTP_CHARSET);
   }

   String charset = null;
   if (cindex != -1) {
       int i                 = cindex + HTTP_CHARSET.length();
       int eqIdx = contentType.indexOf('=', i);
       if (eqIdx != -1) {
           eqIdx++; // no one is interested in the equals sign...

           // The patch had ',' as the terminator but I suspect
           // that is the delimiter between possible charsets,
           // but if another 'attribute' were in the accept header
           // charset would be terminated by a ';'.  So I look
           // for both and take to closer of the two.
           int idx     = contentType.indexOf(',', eqIdx);
           int semiIdx = contentType.indexOf(';', eqIdx);
           if ((semiIdx != -1) && ((semiIdx < idx) || (idx == -1)))
               idx = semiIdx;
           if (idx != -1)
               charset = contentType.substring(eqIdx, idx);
           else
               charset = contentType.substring(eqIdx);
           charset = charset.trim();
           isrc.setEncoding(charset);
       }
   }
```

```
   isrc.setSystemId(uri);

   SVGOMDocument doc = (SVGOMDocument) super.createDocument
       (SVGDOMImplementation.SVG_NAMESPACE_URI, "svg", uri, isrc);
   doc.setParsedURL(new ParsedURL(uri));
   doc.setDocumentInputEncoding(charset);
   doc.setXmlStandalone(isStandalone);
   doc.setXmlVersion(xmlVersion);

   return doc;
}
```

最主要的代码：

再次调用了一个 createDocument 函数，然后才对返回的 SVGOMDocument 类型做一些设置，不过这个 createDocument 是父类 SAXDocumentFactory 中的，跟进去

```
/**
* Creates a Document.
* @param is  The document input source.
* @exception IOException if an error occured while reading the document.
*/
protected Document createDocument(InputSource is)
   throws IOException {
   try {
       if (parserClassName != null) {
           parser = XMLReaderFactory.createXMLReader(parserClassName);
       } else {
           SAXParser saxParser;
           try {
               saxParser = saxFactory.newSAXParser();
           } catch (ParserConfigurationException pce) {
               throw new IOException("Could not create SAXParser: "
                       + pce.getMessage());
           }
           parser = saxParser.getXMLReader();
       }

       parser.setContentHandler(this);
       parser.setDTDHandler(this);
       parser.setEntityResolver(this);
       parser.setErrorHandler((errorHandler == null) ?
                              this : errorHandler);

       parser.setFeature("http://xml.org/sax/features/namespaces",
                         true);
       parser.setFeature("http://xml.org/sax/features/namespace-prefixes",
                         true);
       parser.setFeature("http://xml.org/sax/features/validation",
                         isValidating);
       parser.setProperty("http://xml.org/sax/properties/lexical-handler",
                          this);
       parser.parse(is);
   } catch (SAXException e) {
       Exception ex = e.getException();
       if (ex != null && ex instanceof InterruptedIOException) {
           throw (InterruptedIOException)ex;
       }
       throw new SAXIOException(e);
   }

   currentNode  = null;
   Document ret = document;
   document     = null;
   doctype      = null;
   locator      = null;
   parser       = null;
   return ret;
}
```

值得我们关注的是这一块

is 变量就是svg文件的内容，在 parse 之前，有两块值得注意
`setEntityResolver` 和 `setFeature`
一般对于组件来说，这两者都可以用来防御xxe

setFeature 如上图很明显，并没有对 `doctype、entities` 做设置
那么我们看看 setEntityResolver ，它传入的是 this ，别忘了现在是在父类的 createDocument 函数里，这个 this 应该是子类本身，也就是 SAXSVGDocumentFactory 类

那么这个 setFeature 有啥用呢，它是会将 resolveEntity 函数指定为传入参数类的 resolveEntity 函数，这个函数将会影响解析xml时对外部资源的请求

看看它的定义：

具体就是，该函数返回用户自定义的InputSource实例，如果该方法返回null，
则XML解析引擎默认为外部实体为URL，并使用该URL创建InputSource。
如果重写的时候直接返回了一个 new InputSorce ，那么是直接忽略所有外部URL申请的

那么我们现在去看看 SAXSVGDocumentFactory 类中重写的 resolveEntity 函数

函数体略长，就不贴出来了，就看看它返回啥

在函数最后，默认是返回 null 的
这里解释一下，publicId 和 systemId 分别代表 PUBLIC 和 SYSTEM 标识
如下图
PUBLIC：

SYSTEM：

so，我们的payload里面，根本没用到PUBLIC标识，所以直接返回 null 了

在这之后，就对 svg 进行了解析，其 xml 内容通过构造，可以造成 xxe

batik：https://baike.baidu.com/item/SVG/63178?fr=aladdin
cve 描述：http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5662
batik简单使用：http://blog.csdn.net/lfsf802/article/details/40628547
java xxe 防御：http://blog.csdn.net/raintungli/article/details/53486383

上一篇：CoffeeMiner：劫持WIF... 下一篇：Spectre攻击分析
1. 1 条回复


mozi 2018-04-11 16:21:19

厉害了

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