Apache Axis1（ <=1.4版本） RCE

orich1 / 2019-07-03 08:43:00 / 浏览数 7605 安全技术 漏洞分析 顶(1) 踩(0)

前言

2019.6.16 发出了一则漏洞预警：
https://www.qdcert.com.cn/index/news_detail/W1BZRDEYCh0cDRkcGw

最近两天刚好在学习WebService相关知识，这个 axis
组件就是一个SOAP引擎，提供创建服务端、客户端和网关SOAP操作的基本框架，没见过这类漏洞，抱着学习的心态研究下

触发流程

第一次请求：
org.apache.axis.transport.http.AxisServlet#doPost
->
org.apache.axis.utils.Admin#processWSDD
->
org.apache.axis.AxisEngine#saveConfiguration

第二次请求：
org.apache.axis.transport.http.AxisServlet#doPost
->
freemarker.template.utility.Execute#exec
->
java.lang.Runtime#exec(java.lang.String)

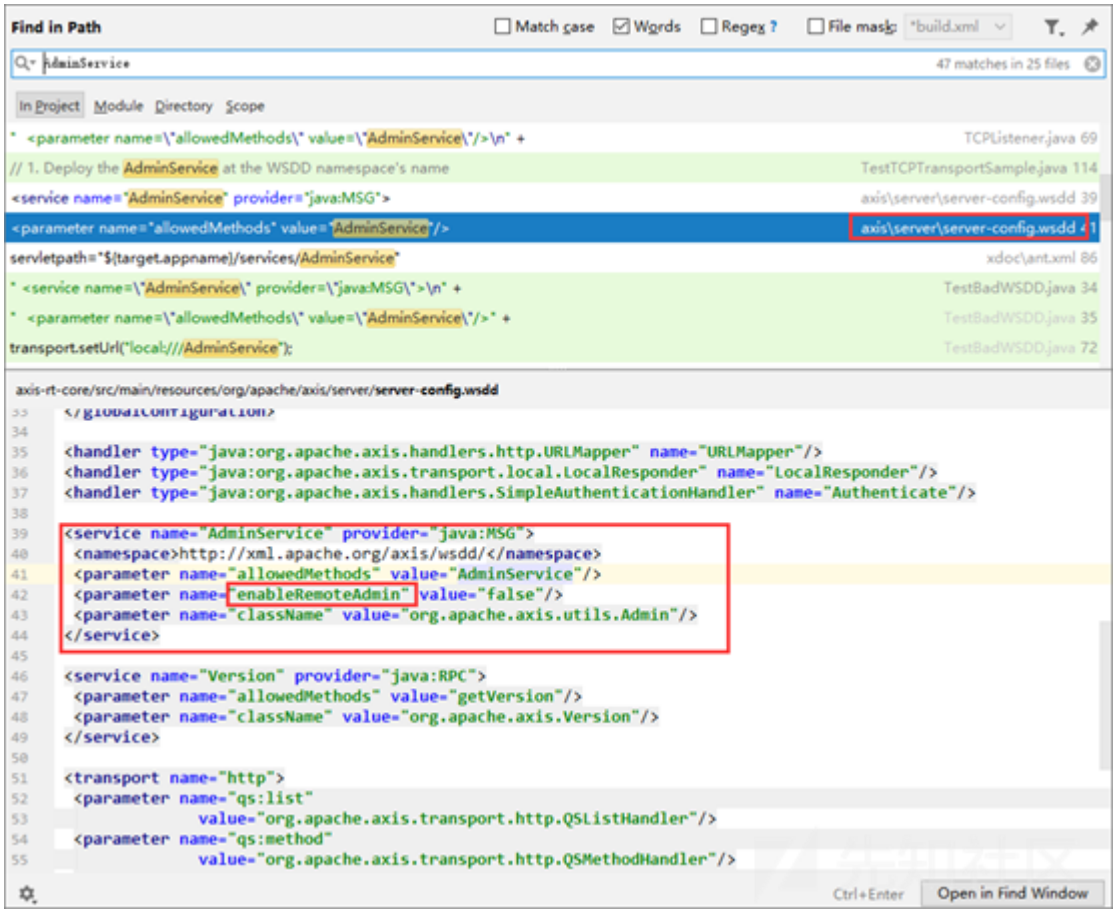
分析过程

本地环境： jdk1.8_191 、Apache Axis1 1.4 、 Tomcat6

看了几个预警，大多描述的是 “Apache Axis 中的 FreeMarker 组件/插件存在相关漏洞”，我找了半天，没见着 FreeMarker 和 Axis
的配合使用呀emmm，这就让我更好奇了，因为之前也有过 FreeMarker 模板注入 bypass 的文章（ https://xz.aliyun.com/t/4846 ）
难道是未授权访问到了 FreeMarker的模板解析流程？

在 ‘/services/FreeMarkerService’ 这个路径上浪费了许多时间，转头看向 ‘/services/AdminService’ 的未授权

全局搜一下，如下图：



完全和预警通告中的描述吻合，跟进 org.apache.axis.utils.Admin#AdminService 查看如下：

```
/**
 * Process a given XML document - needs cleanup.
 */
public Element[] AdminService(Element [] xml)
    throws Exception
{
    log.debug( o: "Enter: Admin::AdminService");
    MessageContext msgContext = MessageContext.getCurrentContext();
    Document doc = process( msgContext, xml[0] );
    Element[] result = new Element[1];
    result[0] = doc.getDocumentElement();
    log.debug( o: "Exit: Admin::AdminService");
    return result;
}
```

稍微了解过 WebService 的带哥可能此时就明白了，之前的那个 service-config.wsdd 配置文件展示的 service 标签就是一个个 WebService 发布的端口，其中 allowedMethods 则是发布的函数，如上图的 AdminService 函数，我们可以通过 SOAP 的方式去调用服务端的 AdminService
函数，需要构造的请求大致如下：

POST /services/AdminService ...
...

他会自动处理我们传递的xml结构参数，上图中 xml[0] 内容是我们完全可控的，跟进 process 函数，如下：

```
public Document process(MessageContext msgContext, Element root)
    throws Exception
{
    // Check security FIRST.

    /** Might do something like this once security is a little more
     * integrated.
    if (!engine.hasSafePassword() &&
        !action.equals("passwd"))
        throw new AxisFault("Server.MustSetPassword",
     "You must change the admin password before administering Axis!",
                            null, null);
     */

    verifyHostAllowed(msgContext);

    String rootNS = root.getNamespaceURI();
    AxisEngine engine = msgContext.getAxisEngine();

    // If this is WSDD, process it correctly.
    if (rootNS != null && rootNS.equals(WSDDConstants.URI_WSDD)) {
        return processWSDD(msgContext, engine, root);
    }

    // Else fault
    // TODO: Better handling here
    throw new Exception(Messages.getMessage( key: "adminServiceNoWSDD"));
}
```

首先调用了 verifyHostAllowd 函数进行验证，其验证内容大致如下：

```
private void verifyHostAllowed(MessageContext msgContext) throws AxisFault {
    /** For now, though - make sure we can only admin from our own
     * IP, unless the remoteAdmin option is set.
     */
    Handler serviceHandler = msgContext.getService();
    if (serviceHandler != null  &&
        !JavaUtils.isTrueExplicitly(serviceHandler.getOption( name: "enableRemoteAdmin"))) {

        String remoteIP = msgContext.getStrProp(Constants.MC_REMOTE_ADDR);
        if (remoteIP != null  &&
            !(remoteIP.equals(NetworkUtils.LOCALHOST) ||
              remoteIP.equals(NetworkUtils.LOCALHOST_IPV6))) {
```

大意是如果服务端该 WebService 端口的 enableRemoteAdmin 属性为 false 的话，则判断当前访问ip，如果是本机访问则放行，如果是远程访问就抛错

继续跟进 processWSDD 函数，如下：

```
protected static Document processWSDD(MessageContext msgContext,
                                      AxisEngine engine,
                                      Element root)
    throws Exception
  {
    Document doc = null ;
    String action = root.getLocalName();
    if (action.equals("passwd")) {
        [...]
    }
    if (action.equals("quit")) {
        [...]
    }
    if ( action.equals("list") ) {
        [...]
    }
    if (action.equals("clientdeploy")) {
        // set engine to client engine
        engine = engine.getClientEngine();
    }
    WSDDDocument wsddDoc = new WSDDDocument(root);
    EngineConfiguration config = engine.getConfig();
```

```
        if (config instanceof WSDDEngineConfiguration) {
            WSDDDeployment deployment =
                ((WSDDEngineConfiguration)config).getDeployment();
            wsddDoc.deploy(deployment);
        }
        engine.refreshGlobalOptions();
        engine.saveConfiguration();
        doc = XMLUtils.newDocument();
        doc.appendChild( root = doc.createElementNS("", "Admin" ) );
        root.appendChild( doc.createTextNode( Messages.getMessage("done00") ) );
        return doc;
    }
```

上述代码的几个 if 我都省略了，因为那不重要，不过 action 也是我们完全可控的，但是这几个if中的功能代码根本不痛不痒，不能做到 rce 的效果，此时我们能够完全掌控的就是 root 变量，它被带入了 WSDDDocument 构造函数中，跟进去如下：

```
public WSDDDocument(Element e) throws WSDDException
{
    doc = e.getOwnerDocument();
    if (ELEM_WSDD_UNDEPLOY.equals(e.getLocalName())) {
        undeployment = new WSDDUndeployment(e);
    } else {
        deployment = new WSDDDeployment(e);
    }
}
```

诶，又对当前标签名作了一次判断，而且看见了 undeployment 和 deployment 这种和 WebService 密切相关的变量名，undeployment 不感兴趣，继续跟进 WSDDDeployment 构造函数：

```
public WSDDDeployment(Element e)
            throws WSDDException {
        super(e);
        [...]
        elements = getChildElements(e, ELEM_WSDD_SERVICE);
        for (i = 0; i < elements.length; i++) {
            try {
                WSDDService service = new WSDDService(elements[i]);
                deployService(service);
            } catch (WSDDNonFatalException ex) {
                // If it's non-fatal, just keep on going
                log.info(Messages.getMessage("ignoringNonFatalException00"), ex);
            } catch (WSDDException ex) {
                // otherwise throw it upwards
                throw ex;
            }
        }
        [...]
    }
```

还记得之前的 server-config.wsdd 文件对 service 的描述内容吗，如下：

```
<service name="AdminService" provider="java:MSG">
<namespace>http://xml.apache.org/axis/wsdd/</namespace>
<parameter name="allowedMethods" value="AdminService"/>
<parameter name="enableRemoteAdmin" value="ture"/>
<parameter name="className" value="org.apache.axis.utils.Admin"/>
</service>
```

如上配置代码，是直接用 service 标签包裹的，所以我们在 WSDDDeployment 构造函数中直接瞄准 service 关键词，上面贴出来的代码中，ELEM_WSDD_SERVICE 意义如下：

```
public static final String ELEM_WSDD_CHAIN = "chain";
public static final String ELEM_WSDD_SERVICE = "service";
public static final String ELEM_WSDD_TRANSPORT = "transport";
public static final String ELEM_WSDD_GLOBAL = "globalConfiguration";
```

首先，这个类型为 Element 的 e 变量是我们从客户端传递的，所以完全可控，在WSDDDeployment 构造函数中，只要 e 的子节点中含有 service 标签就将其还原成 WSDDService 对象，并且调用 deployService 函数，一路跟进到 WSDDService 的 deployToRegistry 函数中，如下：

```
public void deployToRegistry(WSDDDeployment registry)
{
    registry.addService(this);

    // Register the name of the service as a valid namespace, just for
    // backwards compatibility
    registry.registerNamespaceForService(getQName().getLocalPart(),  service: this);

    for (int i = 0; i < namespaces.size(); i++) {
        String namespace = (String) namespaces.elementAt(i);
        registry.registerNamespaceForService(namespace,  service: this);
    }

    super.deployToRegistry(registry);
}
```

基本上就没有过多的操作了，我们先停一停，首先这个初始化过程仅仅是做了注册操作，虽然我们构造的参数被注册了，但是目前还没有下一步触发的地方，回到org.apach
中

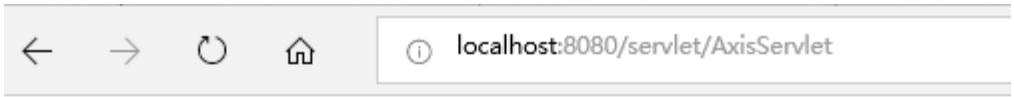
```
WSDDDocument wsddDoc = new WSDDDocument(root);
EngineConfiguration config = engine.getConfig();
if (config instanceof WSDDEngineConfiguration) {
    WSDDDeployment deployment =
        ((WSDDEngineConfiguration)config).getDeployment();
    wsddDoc.deploy(deployment);
}
engine.refreshGlobalOptions();

engine.saveConfiguration();
```

如上，我们控制的 service 已经在 wsddDoc 中注册了，最后会调用一此 engine.refreshGlobalOptions 和 saveConfiguration
，这意味着刚刚的初始化操作很有可能会被保存进配置文件中，那么回想下 axis 的 WebService
操作似乎确实是由配置文件控制，那么到下次访问该服务的时候，是不是会刷新我们自己注册的 service 呢？

测试流程

先下载 Apache Axis 1.4 版本的源码或者是安装包
源码在GitHub上有，安装包在：http://apache.fayea.com/axis/axis/java/1.4/

然后启动项目，直接访问 localhost:8080/servlet/AxisServlet



And now... Some Services

- AdminService *(wsdl)*
  - AdminService
- Version *(wsdl)*
  - getVersion

当前只有两个 WebService 端口，在 WEB-INF/server-config.wsdd 中也只有两个 service 标签，如下：

```
<handler name="Authenticate" type="java:org.apache.axis.handlers.SimpleAuthenticationHandler
<service name="AdminService" provider="java:MSG">
 <parameter name="allowedMethods" value="AdminService"/>
 <parameter name="enableRemoteAdmin" value="ture"/>
 <parameter name="className" value="org.apache.axis.utils.Admin"/>
 <namespace>http://xml.apache.org/axis/wsdd/</namespace>
</service>
<service name="Version" provider="java:RPC">
 <parameter name="allowedMethods" value="getVersion"/>
 <parameter name="className" value="org.apache.axis.Version"/>
</service>
<transport name="http">
```

（其 enableRemoteAdmin 已经被我改成 ture，默认为 false ）

那么此时我们随便构造一个POST包，发过去看看能不能新建一个 WebService 端口

访问刚刚的网页：

已经含有报错，此时看看server-config.wsdd 文件中的 service 标签：

```
<handler name="LocalResponder" type="java:org.apache.axis.transport.local.LocalResponder"/>
<handler name="Authenticate" type="java:org.apache.axis.handlers.SimpleAuthenticationHandler"/>
<service name="a" provider="java:RPC">
 <parameter name="allowedMethods" value="*"/>
 <parameter name="className" value="b"/>
</service>
<service name="AdminService" provider="java:MSG">
 <parameter name="allowedMethods" value="AdminService"/>
 <parameter name="enableRemoteAdmin" value="ture"/>
 <parameter name="className" value="org.apache.axis.utils.Admin"/>
 <namespace>http://xml.apache.org/axis/wsdd/</namespace>
</service>
<service name="Version" provider="java:RPC">
 <parameter name="allowedMethods" value="getVersion"/>
 <parameter name="className" value="org.apache.axis.Version"/>
</service>
<transport name="http">
 <requestFlow>
```

如上图，确实是我们自己添加的内容，那么访问一下 /services/a?wsdl

虽然报错了，但是这表明，我们在 POST 自定义配置过后，是可以立刻生效的，这估计也是 Asix 基于配置文件驱动有关。

那么我们现在可以干什么，这就需要对 WebService 有一定的了解了，可以将 WebService
看作一个个微型服务端口，只要有相关配置，可以做到单个函数级别的调用。那么在我们完全可控配置文件的情况下，这就意味着我们可以调用"任意函数"

不能完全做到随心所欲，需要满足 WebService 的规则，比如 端口类 需要有一个 public 的无参构造函数，还要需要当前 ClassLoader
能够找得到指定类，最后也是最麻烦的则是指定函数的参数匹配，没有详细研究这个老框架，不知道复杂类型的传参是否允许

利用

现在我们知道可以在有限条件下调用任意函数，那么如何利用？还有和通告中的 freemarker 有啥关系？

在腾讯云的预警中找到了一句：

Apache AXIS中的freemarker组件中调用template.utility.Execute类时存在远程命令执行攻击

查看 Execute 类：

```java
public class Execute implements TemplateMethodModel {
   private static final int OUTPUT_BUFFER_SIZE = 1024;

   public Object exec(List arguments) throws TemplateModelException {
      StringBuffer aOutputBuffer = new StringBuffer();
      if (arguments.size() < 1) {
         throw new TemplateModelException("Need an argument to execute");
      } else {
         String aExecute = (String)((String)arguments.get(0));

         try {
            Process exec = Runtime.getRuntime().exec(aExecute);
            InputStream execOut = exec.getInputStream();

            try {
               Reader execReader = new InputStreamReader(execOut);
               char[] buffer = new char[1024];

               for(int bytes_read = execReader.read(buffer); bytes_read > 0; bytes_read = execReader.read
                  aOutputBuffer.append(buffer, 0, bytes_read);
               }
            } finally {
               execOut.close();
            }
         } catch (IOException var13) {
            throw new TemplateModelException(var13.getMessage());
         }

         return aOutputBuffer.toString();
      }
   }
}
```

直接将传递进来的参数作为命令执行的参数，而且参数类型是 List ，那么基本可以确定这个应该可以利用（List在xml结构中可以看作Array结构）

不过似乎少个public无参构造函数？查看一下字节码：

```
public class freemarker/template/utility/Execute implements freemarker/tem
    <ClassVersion=48>
    <SourceFile=Execute.java>

    private static final int OUTPUT_BUFFER_SIZE = 1024 (java.lang.Integer

    public Execute() { // <init> //()V
        <localVar:index=0 , name=this , desc=Lfreemarker/template/utility/Ex

        L1 {
            aload0 // reference to self
            invokespecial java/lang/Object.<init>()V
            return
        }
        L2 {
        }
    }

    public exec(java.util.List arg0) throws freemarker/template/TemplateMod
        <localVar:index=6 , name=execReader , desc=Ljava/io/Reader;, sig=nul
```

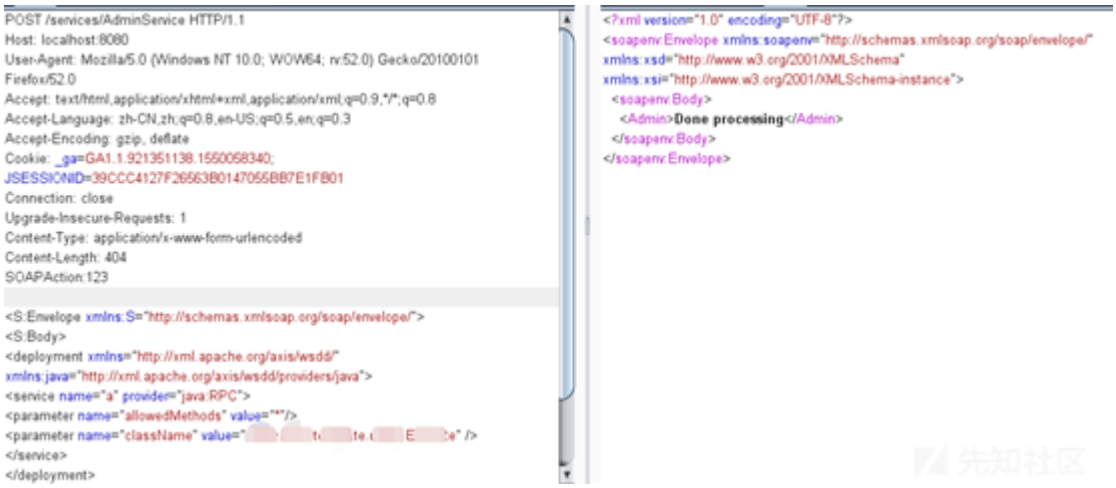
原来只是没有反编译出来而已

那么我们就可以构造两个POST包：

第一个包设置配置文件，将 freemarker.template.utility.Execute 作为一个 WebService 的端口，并且开放 exec 函数，第二个包访问设定的 WebService 端口，并且发送SOAP操作，调用服务端的 freemarker.template.utility.Execute#exec 函数

效果如下：

第一次请求：



第二次请求：

（以上流程我是将freemarker-2.3.23.jar 添加进 WEB-INF/lib 中的，Axis 本身没有这个jar包）
当然也可以在 Axis 自带的 libs 中搜寻利用类，所需条件上文已经阐述过

总结

其实这个漏洞和 freemarker 没有啥必然联系，主要是因为 Axis 的未授权访问，可是默认配置下是不允许远程访问 AdminService 端口的，不过还是学习了 Axis 的处理方式，之前所了解到的 WebService 都是通过注解操作，但其实反过来想想，注解或是配置文件都一样，都需要框架底层自行实现解析流程

上一篇：利用Excel power que... 下一篇：通过异常处理机制实现漏洞利用
1. 2 条回复


grey****@gmail.c 2019-07-03 10:50:38

nice job

0 回复Ta


Deadpool 2019-10-30 20:27:47

大佬，最近刚好挖洞遇到这个，不知道如何，萌新小白，看不懂代码。。。

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