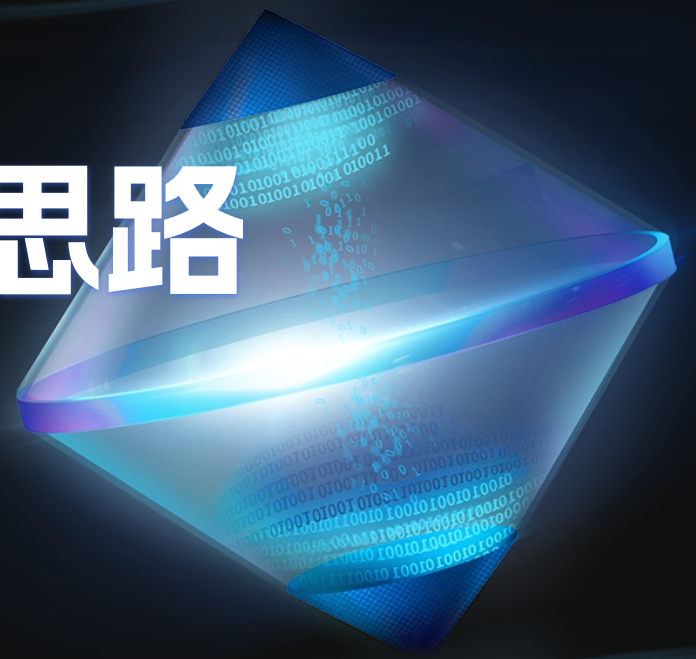
KCon

归源 · 智变

Returning to the source · Intelligent transformation

WebLogic 漏洞挖掘思路



演讲人：丁立洁（thiscodecc）

个人介绍

- 丁立洁（thiscodecc）
- 墨云科技安全研究员
- 多次收到IBM、VMware、Oracle、OpenJDK 等厂商公开致谢
- 近两年收到Oracle WebLogic 14个CVE编号
- https://github.com/thiscodecc

KCon

归源·智变

Returning to the source · Intelligent transformation

目录 / CONTENTS

WebLogic简介

WebLogic是Oracle公司出品的一个Java应用服务器，是一个基于JAVAEE架构的中间件，用于开发、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。

WebLogic基于序列化数据传输的协议

- WebLogic默认开启T3和IIOP协议,这两个协议都是基于序列化数据传输的协议,这两个协议从2015年到2023年Oracle官方修复了38个严重漏洞。
- WebLogic会把反序列化漏洞中的**关键类**加入到黑名单列表中,以此来修复反序列化漏洞。

WebLogic防御措施

```
public class FilteringObjectInputStream extends ObjectInputStream {
    private volatile Class expectedType;
    private volatile Class[] expectedTypes;

    protected FilteringObjectInputStream() throws IOException {...}

    public FilteringObjectInputStream(InputStream is) throws IOException {...}

    protected FilteringObjectInputStream(InputStream is, boolean setFilter) throws IOExcep

    protected Class<?> resolveClass(ObjectStreamClass descriptor) throws ClassNotFoundExce
        this.checkLegacyBlacklistIfNeeded(descriptor.getName());
        Class cls = super.resolveClass(descriptor);
        this.validateReturnType(cls);
        return cls;
    }
```

重写 resolveClass 方法来过滤

具体的黑名单列表

```
DEFAULT_LIMITS = new String[]{"maxdepth=100"};
DEFAULT_BLACKLIST_PACKAGES = new String[]{"org.apache.commons.collections.functors", "com.sun.org.apache.xalan.inte
DEFAULT_BLACKLIST_CLASSES = new String[]{"org.codehaus.groovy.runtime.ConvertedClosure", "org.codehaus.groovy.runti
DEFAULT_WLS_ONLY_BLACKLIST_PACKAGES = new String[]{"com.tangosol.internal.util.invoke", "com.tangosol.internal.util
DEFAULT_WLS_ONLY_BLACKLIST_CLASSES = new String[]{"com.tangosol.util.extractor.ReflectionExtractor", "com.tangosol.
```

如何绕过黑名单列表?

一、执行了从bytes到ObjectInputStream的二次反序列化过程
二、用一个类似功能的class
三、不同package下的class
四、自定义序列化过程的调用链
五、在反序列化的过程中调用了黑名单列表的class
六、不同的weblogic发行版本黑名单列表不一样
七、一条新的调用链
八、第三方jar没有更新到最新版本
九、后反序列化漏洞

一、执行了从bytes到ObjectInputStream的二次反序列化过程

StreamMessageImpl.class

Decompiled .class file, bytecode version: 49.0 (Java 5)

CVE-2016-0638

修复前

```
849 @   public void readExternal(ObjectInput var1) throws IOException, ClassNotFoundException {
850         super.readExternal(var1);
851         byte var2 = var1.readByte();
852         byte var3 = (byte)(var2 & 127);
853         if (var3 >= 1 && var3 <= 3) {
854             switch(var3) {
855             case 1:
856                 this.payload = (PayloadStream)PayloadFactoryImpl.createPayload((InputStream)var1);
857                 BufferInputStream var4 = this.payload.getInputStream();
858                 ObjectInputStream var5 = new ObjectInputStream(var4);
```

修复后

```
850 @   public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
851         super.readExternal(in);
852         byte unmaskedVersion = in.readByte();
853         byte vrsn = (byte)(unmaskedVersion & 127);
854         if (vrsn >= 1 && vrsn <= 3) {
855             switch(vrsn) {
856             case 1:
857                 this.payload = (PayloadStream)PayloadFactoryImpl.createPayload((InputStream)in);
858                 InputStream is = this.payload.getInputStream();
859                 ObjectInputStream ois = new FilteringObjectInputStream(is);
```

新的ObjectInputStream类没有黑名单列表

MarshalledObject.class

Attaching...

CVE-2016-3510

修复前

```
43      public Object readResolve() throws IOException, ClassNotFoundException, ObjectStreamException {
44          if (this.objBytes == null) {
45              return null;
46          } else {
47              ByteArrayInputStream var1 = new ByteArrayInputStream(this.objBytes);
48              ObjectInputStream var2 = new ObjectInputStream(var1);
49              Object var3 = var2.readObject();
50              var2.close();
51              return var3;
```

修复后

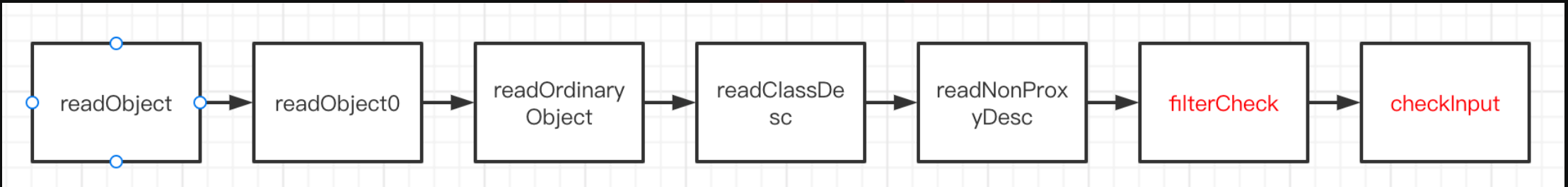
```
public Object readResolve() throws IOException, ClassNotFoundException, ObjectStreamException {
    if (this.objBytes == null) {
        return null;
    } else {
        ByteArrayInputStream bin = new ByteArrayInputStream(this.objBytes);
        FilteringObjectInputStream in = new FilteringObjectInputStream(bin)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

在支持JEP290的JDK中无法利用这种方法

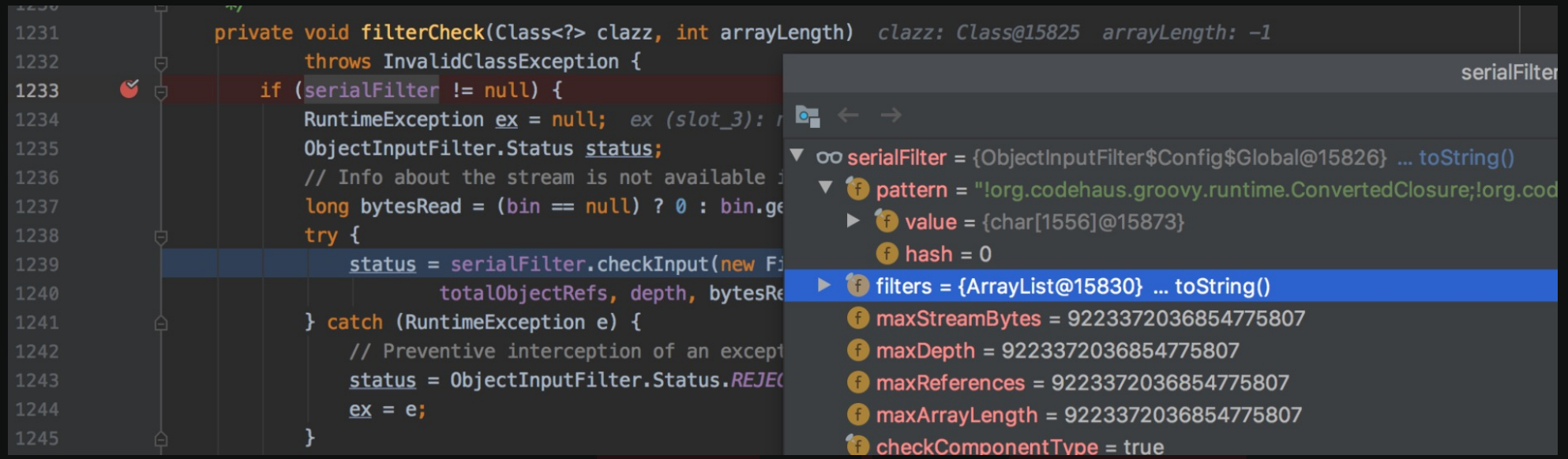
jep290机制用来解决不安全的反序列化风险

- 自定义过滤器
- 全局过滤器
- 内置过滤器

其核心就是提供了一个ObjectInputFilter接口,在进行反序列化操作的时候,会调用该接口的checkInput方法,JDK自带的sun.misc.ObjectInputFilter$Config$Global实现了该接口,需要自定义过滤器的话实现该接口,然后在调用ObjectInputStream.setInternalObjectInputFilter赋值自定义的filter即可。



在JDK 8u121中开始支持JEP290特性




```
public ObjectInputFilter.Status checkInput(ObjectInputFilter.FilterInfo var1) {  var1: ObjectInputStrea
    if (var1.references() >= 0L && var1.depth() >= 0L && var1.streamBytes() >= 0L && var1.references()
        Class var2 = var1.serialClass();  var2 (slot_2): Class@15931
        if (var2 == null) {
            return ObjectInputFilter.Status.UNDECIDED;
        } else {
            if (var2.isArray()) {...}

            if (var2.isPrimitive()) {...} else {
                Optional var4 = this.filters.stream().map((var1x) -> {  var4 (slot_3): Class@15931
                    return (ObjectInputFilter.Status)var1x.apply(var2);  var2 (slot_2): Class@15931
                }).filter((var0) -> {
                    return var0 != ObjectInputFilter.Status.UNDECIDED;
                }).findFirst();
                return (ObjectInputFilter.Status)var4.orElse(ObjectInputFilter.Status.UNDECIDED);  var4
```

二、用一个类似功能的class

- 从CVE-2020-2555到CVE-2020-14825的绕过方法
- CVE-2020-2555调用链如下

```
gadget:
    BadAttributeValueExpException.readObject()
        com.tangosol.util.filter.LimitFilter.toString()
            com.tangosol.util.extractor.ChainedExtractor.extract()
                com.tangosol.util.extractor.ReflectionExtractor.extract()
                    Method.invoke()
                    ...
                    com.tangosol.util.extractor.ReflectionExtractor.extract()
                    Method.invoke()
                        Runtime.getRuntime.exec()
```

CVE-2020-2555修复方案

修复前

```
public String toString() {
    StringBuilder sb = new StringBuilder("LimitFilter: (");
    sb.append(this.m_filter).append(" [pageSize=").append(this.m_cPageSize).append(", pageNum=").append(this.m_nPage);
    if (this.m_comparator instanceof ValueExtractor) {
        ValueExtractor extractor = (ValueExtractor)this.m_comparator;
        sb.append(", top=").append(extractor.extract(this.m_oAnchorTop)).append(", bottom=").append(extractor.extract(t
    } else if (this.m_comparator != null) {
        sb.append(", comparator=").append(this.m_comparator);
```

修复后

```
public String toString() {
    StringBuilder sb = new StringBuilder("LimitFilter: (");
    sb.append(this.m_filter).append(" [pageSize=").append(this.m_cPageSize).append(", pageNum=").append(this.m_nPage);
    if (this.m_comparator != null) {
        sb.append(", top=").append(this.m_oAnchorTop).append(", bottom=").append(this.m_oAnchorBottom).append(", compar
    }
```

CVE-2020-2883绕过CVE-2020-2555补丁

```
CVE-2020-2555:
    BadAttributeValueExpException.readObject()
        com.tangosol.util.filter.LimitFilter.toString()
            com.tangosol.util.extractor.ChainedExtractor.extract()
                com.tangosol.util.extractor.ReflectionExtractor.extract()
                    Method.invoke()
                    ...
                    com.tangosol.util.extractor.ReflectionExtractor.extract()
                    Method.invoke()
                        Runtime.getRuntime.exec()


CVE-2020-2883:
    PriorityQueue.readObject()
    ...
        ExtractorComparator.compare()
            ChainedExtractor.extract()
                com.tangosol.util.extractor.ChainedExtractor.extract()
                    com.tangosol.util.extractor.ReflectionExtractor.extract()
                        Method.invoke()
                        ...
                        com.tangosol.util.extractor.ReflectionExtractor.extract()
                        Method.invoke()
                            Runtime.getRuntime.exec()
```

三、不同package下的class

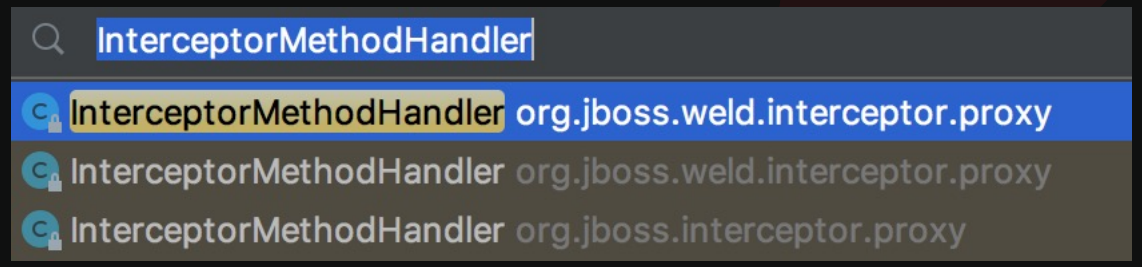
CVE-?/CVE-2018-3245

```
private static final String[] DEFAULT_BLACKLIST_CLASSES = new String[]{"org.codehaus.groovy.runtime.ConvertedClosure",
"org.codehaus.groovy.runtime.ConversionHandler", "org.codehaus.groovy.runtime.MethodClosure",
"org.springframework.transaction.support.AbstractPlatformTransactionManager", "java.rmi.server.UnicastRemoteObject",
"java.rmi.server.RemoteObjectInvocationHandler",
"com.bea.core.repackaged.springframework.transaction.support.AbstractPlatformTransactionManager"  "java.rmi.server.Remot
"com.tangosol.coherence.rest.util.extractor.MvelExtractor", "java.lang.Runtime",
"oracle.eclipselink.coherence.integrated.internal.cache.LockVersionExtractor",
"org.eclipse.persistence.internal.descriptors.MethodAttributeAccessor",
"org.eclipse.persistence.internal.descriptors.InstanceVariableAttributeAccessor", "org.apache.commons.fileupload.disk.Di
"oracle.jdbc.pool.OraclePooledConnection"};
```

三、不同package下的class

CVE-?/CVE-2021-2064

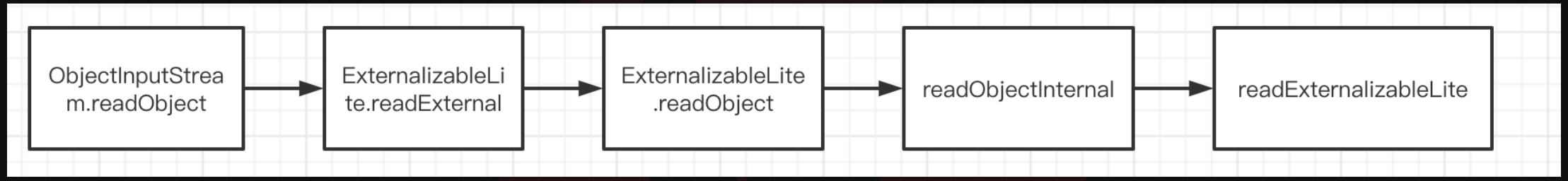
```
private static final String[] DEFAULT_BLACKLIST_PACKAGES = new String[]{"org.apache.commons.collections.functors",
"com.sun.org.apache.xalan.internal.xsltc.trax", "javassist", "java.rmi.activation", "sun.rmi.server", "org.jboss.interceptor.builder",
"org.jboss.interceptor.reader", "org.jboss.interceptor.proxy", "org.jboss.interceptor.spi.metadata", "org.jboss.interceptor.spi.model",
"com.bea.core.repackaged.springframework.aop.aspectj", "com.bea.core.repackaged.springframework.aop.aspectj.annotation",
"com.bea.core.repackaged.springframework.aop.aspectj.autoproxy", "com.bea.core.repackaged.springframework.beans.factory.support",
"org.python.core", "com.bea.core.repackaged.aspectj.weaver.tools.cache", "com.bea.core.repackaged.aspectj.weaver.tools",
"com.bea.core.repackaged.aspectj.weaver.reflect", "com.bea.core.repackaged.aspectj.weaver",
"com.oracle.wls.shaded.org.apache.xalan.xsltc.trax", "oracle.eclipselink.coherence.integrated.internal.querying",
"oracle.eclipselink.coherence.integrated.internal.cache"};
private static final String[] DEFAULT_BLACKLIST_CLASSES = new String[]{"org.codehaus.groovy.runtime.ConvertedClosure",
"org.codehaus.groovy.runtime.ConversionHandler", "org.codehaus.groovy.runtime.MethodClosure",
"org.springframework.transaction.support.AbstractPlatformTransactionManager", "java.rmi.server.UnicastRemoteObject",
"java.rmi.server.RemoteObjectInvocationHandler",
"com.bea.core.repackaged.springframework.transaction.support.AbstractPlatformTransactionManager", "java.rmi.server.RemoteObject",
"com.tangosol.coherence.rest.util.extractor.MvelExtractor", "java.lang.Runtime",
"oracle.eclipselink.coherence.integrated.internal.cache.LockVersionExtractor",
"org.eclipse.persistence.internal.descriptors.MethodAttributeAccessor",
"org.eclipse.persistence.internal.descriptors.InstanceVariableAttributeAccessor", "org.apache.commons.fileupload.disk.DiskFileItem",
"org.jboss.weld.interceptor.builder.MethodReference", "org.jboss.weld.interceptor.spi.metadata.MethodMetadata",
"oracle.idbc.pool.OraclePooledConnection"};
```



只影响12.1.3版本

四、自定义序列化过程的调用链

CVE-2020-14756使用自定义的反序列化调用链来绕过黑名单列表



使用loadClass和newInstance进行对象还原

```
public static ExternalizableLite readExternalizableLite(DataInput in, ClassLoader loader) throws IOException {
    ExternalizableLite value;
    if (in instanceof PofInputStream) {...} else {
        String sClass = readUTF((DataInput)in);
        WrapperDataInputStream inWrapper = in instanceof WrapperDataInputStream ? (WrapperDataInputStream)in : null;

        try {
            value = (ExternalizableLite)loadClass(sClass, loader, inWrapper == null ? null : inWrapper.getClassLoader()).newInstance();
        } catch (InstantiationException var6) {
            throw new IOException("Unable to instantiate an instance of class '" + sClass + "'; this is most likely due to a missing pub
        } catch (Exception var7) {
            throw new IOException("Class initialization failed: " + var7 + "\n" + getStackTrace(var7) + "\nClass: " + sClass + "\nClassL
        }

        if (loader != null) {
            if (inWrapper == null) {
                in = new WrapperDataInputStream((DataInput)in, loader);
            } else if (loader != inWrapper.getClassLoader()) {
                inWrapper.setClassLoader(loader);
            }
        }

        value.readExternal((DataInput)in);
```

CVE-2020-14756调用链

```
CVE-2020-14756:
    AttributeHolder.readExternal()
    AttributeHolder.readExternal()
        ExternalizableHelper.readObject()
        ExternalizableHelper.readObject()
            ExternalizableHelper.readObjectInternal()
            ExternalizableHelper.readExternalizableLite()
                TopNAggregator$PartialResult.readExternal()
                TopNAggregator$PartialResult.add()
                    SortedBag.add()
                        TreeMap.put()
                        TreeMap.compare()
                            SortedBag$WrapperComparator.compare()
                                AbstractExtractor.compare()
                                    MvelExtractor.extract()
                                        MVEL.executeExpression()
```

五、在反序列化过程中调用黑名单列表的class

CVE-2021-2394绕过方法

SerializationHelper.java

```
39  @    public static AttributeAccessor readAttributeAccessor(DataInput in) throws IOException {
40           int id = ExternalizableHelper.readInt(in);
41           if (id == INSTANCE_ACCESSOR) {
42               // Usage of a different class is intentional:
43               // InstanceVariableAttributeAccessor is deserialize into InstanceVariableAttribut
44               // which behaves like PersistenceObjectAttributeAccessor if the class is woven (i
45               // and like InstanceVariableAttributeAccessor otherwise.
46               InstanceVariableAttributeAccessorExtended accessor = new InstanceVariableAttribut
47               accessor.setAttributeName((String)ExternalizableHelper.readObject(in));
48               return accessor;
49           } else if (id == METHOD_ACCESSOR) {
50               MethodAttributeAccessor accessor = new MethodAttributeAccessor();
51               accessor.setAttributeName((String)ExternalizableHelper.readObject(in));
52               accessor.setGetMethodName((String)ExternalizableHelper.readObject(in));
53               accessor.setSetMethodName((String)ExternalizableHelper.readObject(in));
```

- MethodAttributeAccessor类修复CVE-2020-14825时已加入到黑名单

```
private static final String[] DEFAULT_BLACKLIST_CLASSES = new String[]{"org.codehaus.groovy.runtime.ConvertedClosure",
"org.codehaus.groovy.runtime.ConversionHandler", "org.codehaus.groovy.runtime.MethodClosure",
"org.springframework.transaction.support.AbstractPlatformTransactionManager", "java.rmi.server.UnicastRemoteObject",
"java.rmi.server.RemoteObjectInvocationHandler",
"com.bea.core.repackaged.springframework.transaction.support.AbstractPlatformTransactionManager", "java.rmi.server.Remo
"com.tangosol.coherence.rest.util.extractor.MvelExtractor", "java.lang.Runtime",
"oracle.eclipselink.coherence.integrated.internal.cache.LockVersionExtractor",
"org.eclipse.persistence.internal.descriptors.MethodAttributeAccessor",
"org.eclipse.persistence.internal.descriptors.InstanceVariableAttributeAccessor", "org.apache.commons.fileupload.disk.D
"org.jboss.weld.interceptor.builder.MethodReference", "org.jboss.weld.interceptor.spi.metadata.MethodMetadata",
"oracle.jdbc.pool.OraclePooledConnection"};
```

CVE-2021-2394调用链

```
AttributeHolder.readExternal()
AttributeHolder.readExternal()
    ExternalizableHelper.readObject()
    ExternalizableHelper.readObject()
    ExternalizableHelper.readObjectInternal()
    ExternalizableHelper.readExternalizableLite()
        TopNAggregator$PartialResult.readExternal()
        TopNAggregator$PartialResult.add()
            SortedBag.add()
                TreeMap.put()
                TreeMap.compare()
                    SortedBag$WrapperComparator.compare()
                        FilterExtractor#extract()
                            MethodAttributeAccessor#getAttributeValueFromObject()
                                Method.invoke()
```

六、不同weblogic发行版本黑名单列表不一样

WebLogic 12.1.3版本 2020年10月份补丁

忘记修复了？直接捡漏？

```
private static final String[] DEFAULT_BLACKLIST_CLASSES = new String[]{"org.codehaus.groovy.runtime.ConvertedClosure",
"org.codehaus.groovy.runtime.ConversionHandler", "org.codehaus.groovy.runtime.MethodClosure",
"org.springframework.transaction.support.AbstractPlatformTransactionManager", "java.rmi.server.UnicastRemoteObject",
"java.rmi.server.RemoteObjectInvocationHandler",
"com.bea.core.repackaged.springframework.transaction.support.AbstractPlatformTransactionManager", "java.rmi.server.RemoteObject",
"com.tangosol.coherence.rest.util.extractor.MvelExtractor", "java.lang.Runtime"};
```

WebLogic 12.2.1.3版本 2020年10月份补丁

```
private static final String[] DEFAULT_BLACKLIST_CLASSES = new String[]{"org.codehaus.groovy.runtime.ConvertedClosure",
"org.codehaus.groovy.runtime.ConversionHandler", "org.codehaus.groovy.runtime.MethodClosure",
"org.springframework.transaction.support.AbstractPlatformTransactionManager", "java.rmi.server.UnicastRemoteObject",
"java.rmi.server.RemoteObjectInvocationHandler",
"com.bea.core.repackaged.springframework.transaction.support.AbstractPlatformTransactionManager", "java.rmi.server.RemoteObject"
"com.tangosol.coherence.rest.util.extractor.MvelExtractor", "java.lang.Runtime",
"oracle.eclipselink.coherence.integrated.internal.cache.LockVersionExtractor",
"org.eclipse.persistence.internal.descriptors.MethodAttributeAccessor",
"org.eclipse.persistence.internal.descriptors.InstanceVariableAttributeAccessor"};
```

红框中的黑名单类是修复CVE-2020-14825漏洞,这个漏洞是影响WebLogic 12.1.3版本的,但是在2020年 10月份补丁中没有加入到12.1.3版本中,在2021年1月份补丁中才进行修复并额外分配了CVE-2021-2108

七、一条新的调用链

- 由于WebLogic是采用黑名单进行修复漏洞,WebLogic默认加载了很多jar包,WebLogic 10.3.6版本默认加载了874个jar包,在这些jar包中组合一条新的反序列化调用链就可以直接绕过黑名单列表。
- 新的反序列化调用链有:
- CVE-2015-4852、CVE-2016-0638、CVE-2017-3248、CVE-2020-2555、CVE-2020-14756、CVE-2021-2382、CVE-2022-21350、CVE-2023-21837

八、第三方jar没有更新到最新版本

```
private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException {
    if (!Boolean.getBoolean(SERIALIZABLE_PROPERTY)) {
        throw new IllegalStateException("Property " + SERIALIZABLE_PROPERTY + " is not true, rejecting to deserialize a DiskFileItem.")
    } else {
        in.defaultReadObject();
        if (this.repository != null) {
            if (!this.repository.isDirectory()) {
                throw new IOException(String.format("The repository [%s] is not a directory", this.repository.getAbsolutePath()));
            }

            if (this.repository.getPath().contains("\u0000")) {
                throw new IOException(String.format("The repository [%s] contains a null character", this.repository.getPath()));
            }
        }

        OutputStream output = this.getOutputStream();
        if (this.cachedContent != null) {
            output.write(this.cachedContent);
        } else {
            FileInputStream input = new FileInputStream(this.dfosFile);
            IOUtils.copy(input, output);
            this.dfosFile.delete();
            this.dfosFile = null;
```

WebLogic默认使用了Apache Commons FileUpload 1.3.3的jar包，这个版本的jar包虽然修复了文件上传漏洞,但是仍然存在任意文件删除漏洞

WebLogic分配CVE-2021-2047漏洞编号

修复方案把DiskFileItem类加入到黑名单列表

```
private static final String[] DEFAULT_BLACKLIST_CLASSES = new String[]{"org.codehaus.groovy.runtime.ConvertedClosure",
"org.codehaus.groovy.runtime.ConversionHandler", "org.codehaus.groovy.runtime.MethodClosure",
"org.springframework.transaction.support.AbstractPlatformTransactionManager", "java.rmi.server.UnicastRemoteObject",
"java.rmi.server.RemoteObjectInvocationHandler",
"com.bea.core.repackaged.springframework.transaction.support.AbstractPlatformTransactionManager", "java.rmi.server.RemoteObject",
"com.tangosol.coherence.rest.util.extractor.MvelExtractor", "java.lang.Runtime",
"oracle.eclipselink.coherence.integrated.internal.cache.LockVersionExtractor",
"org.eclipse.persistence.internal.descriptors.MethodAttributeAccessor",
"org.eclipse.persistence.internal.descriptors.InstanceVariableAttributeAccessor", "org.apache.commons.fileupload.disk.DiskFileItem",
"org.jboss.weld.interceptor.builder.MethodReference", "org.jboss.weld.interceptor.spi.metadata.MethodMetadata",
"oracle.jdbc.pool.OraclePooledConnection"};
```

九、后反序列化漏洞

```
getReferent:76, ForeignOpaqueReference (weblogic.jndi.internal)
getObjectInstance:106, WLNamingManager (weblogic.jndi.internal)
resolveObject:1037, BasicNamingNode (weblogic.jndi.internal)
resolveObject:1009, BasicNamingNode (weblogic.jndi.internal)
lookupSharable:1578, BasicNamingNode (weblogic.jndi.internal)
lookupSharable:47, PartitionHandler (weblogic.jndi.internal)
lookup:531, ServerNamingNode (weblogic.jndi.internal)
lookup:84, RootNamingNode (weblogic.jndi.internal)
lookup:307, WLEventContextImpl (weblogic.jndi.internal)
lookup:435, WLContextImpl (weblogic.jndi.internal)
lookup:417, InitialContext (javax.naming)
resolveObject:463, NamingContextImpl (weblogic.corba.cos.naming)
resolve_any:369, NamingContextImpl (weblogic.corba.cos.naming)
_invoke:108, _NamingContextAnyImplBase (weblogic.corba.cos.naming)
invoke:240, CorbaServerRef (weblogic.corba.idl)
invoke:246, ClusterableServerRef (weblogic.rmi.cluster)
run:534, BasicServerRef$2 (weblogic.rmi.internal)
doAs:368, AuthenticatedSubject (weblogic.security.acl.internal)
runAs:163, SecurityManager (weblogic.security.service)
handleRequest:531, BasicServerRef (weblogic.rmi.internal)
run:138, WLSExecuteRequest (weblogic.rmi.internal.wls)
_runAs:352, ComponentInvocationContextManager (weblogic.invocation)
runAs:337, ComponentInvocationContextManager (weblogic.invocation)
doRunWorkUnderContext:57, LivePartitionUtility (weblogic.work)
runWorkUnderContext:41, PartitionUtility (weblogic.work)
runWorkUnderContext:644, SelfTuningWorkManagerImpl (weblogic.work)
execute:415, ExecuteThread (weblogic.work)
run:355, ExecuteThread (weblogic.work)
```

CVE-2023-21839 调用栈

```
...
context.rebind("kcon2023",foreignOpaqueRefe
rence);
context.lookup("kcon2023");
...
```

在反序列化操作完成之后出现的漏洞

WLNamingManager.class

Decompiled .class file, bytecode version: 51.0 (Java 7)

Alternative source available for the class weblogic.jndi.internal.WLNamingManager

```
67    public static Object getObjectInstance(Object boundObject, Name name, Context
68        if (boundObject instanceof ClassTypeOpaqueReference) {
69            Hashtable jndiEnv = ThreadLocalMap.get();
70            if (jndiEnv != null && Boolean.parseBoolean((String)jndiEnv.get("weblo
71                boundObject = ((ClassTypeOpaqueReference)boundObject).getObjectCla
72            } else {
73                boundObject = ((OpaqueReference)boundObject).getReferent(name, ctx
74            }
75        } else if (boundObject instanceof OpaqueReference) {
76            boundObject = ((OpaqueReference)boundObject).getReferent(name, ctx);
```

ForeignOpaqueReference.class

Alternative source available for the class weblogic.jndi.internal.ForeignOpaqueReference

Decompiled .class file, bytecode version: 51.0 (Java 7)

```
47    public Object getReferent(Name name, Context ctx) throws NamingException {
48        InitialContext context;
49        String providerURL;
50        if (this.jndiEnvironment == null) {
51            context = new InitialContext();
52        } else {
53            Hashtable properties = this.decrypt();
54            providerURL = (String)this.jndiEnvironment.get("java.naming.provider.url");
55            if (providerURL != null && (this.signature == null || this.timestamp == null
56                throw new NamingException("Invalid ForeignOpaqueReference signature");
57            }
58
59            context = new InitialContext(properties);
60        }
61
62        Object retVal;
63        try {
64            providerURL = this.jndiEnvironment != null ? (String)this.jndiEnvironment.ge
65            if (providerURL == null && this.remoteJNDIName != null && !JNDIUtils.isValid
66                throw new NamingException("JNDI name is invalid - " + this.remoteJNDINam
67            }
68
69            retVal = context.lookup(this.remoteJNDIName);
```

使用lookup方法查找weblogic资源的时候,weblogic服务端会判断当前对象是否实现了OpaqueReference接口,如果实现了该接口会调用当前对象的getReferent方法。

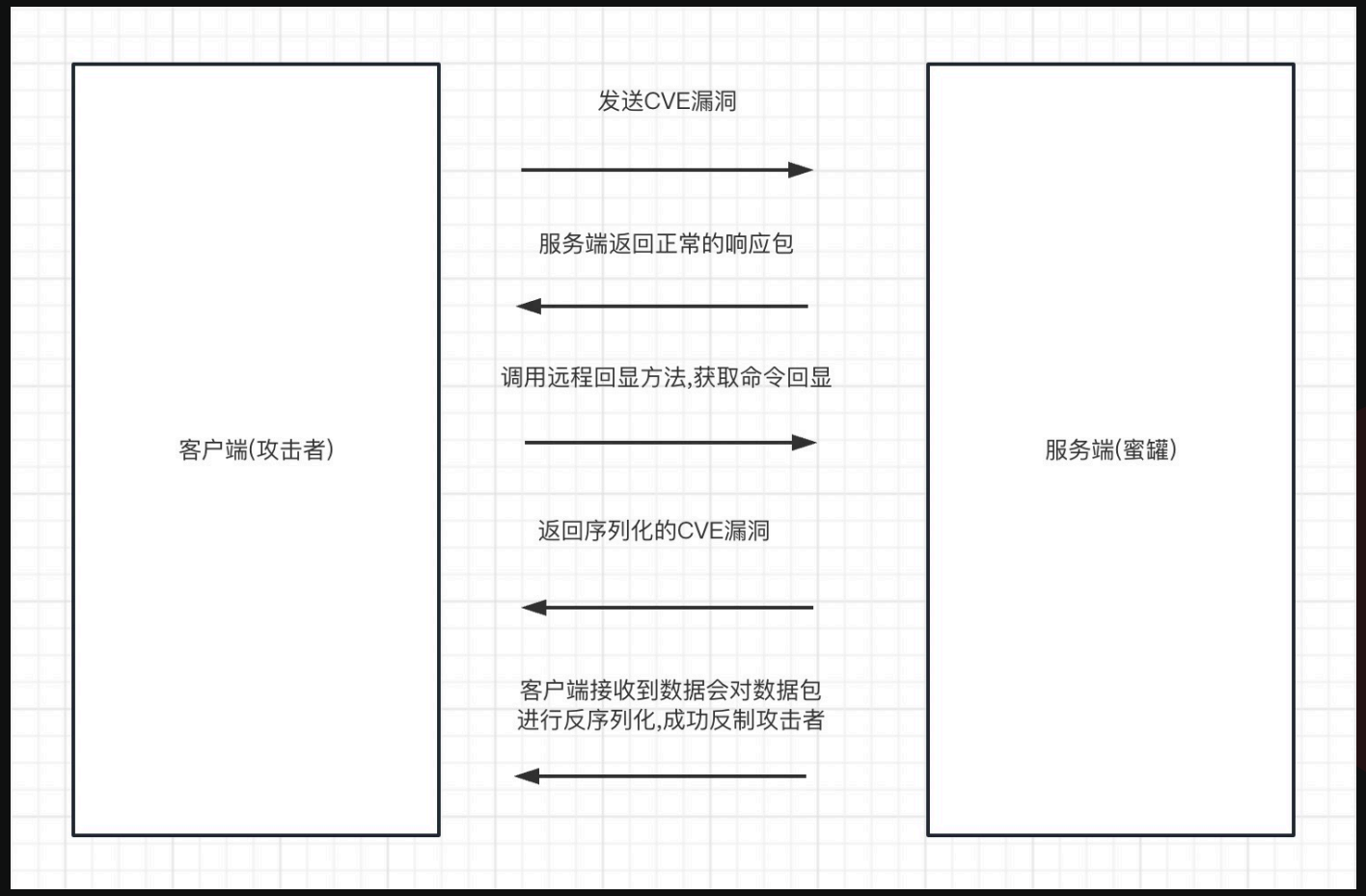
反制WebLogic攻击者

weblogic t3/iiop协议漏洞回显原理:

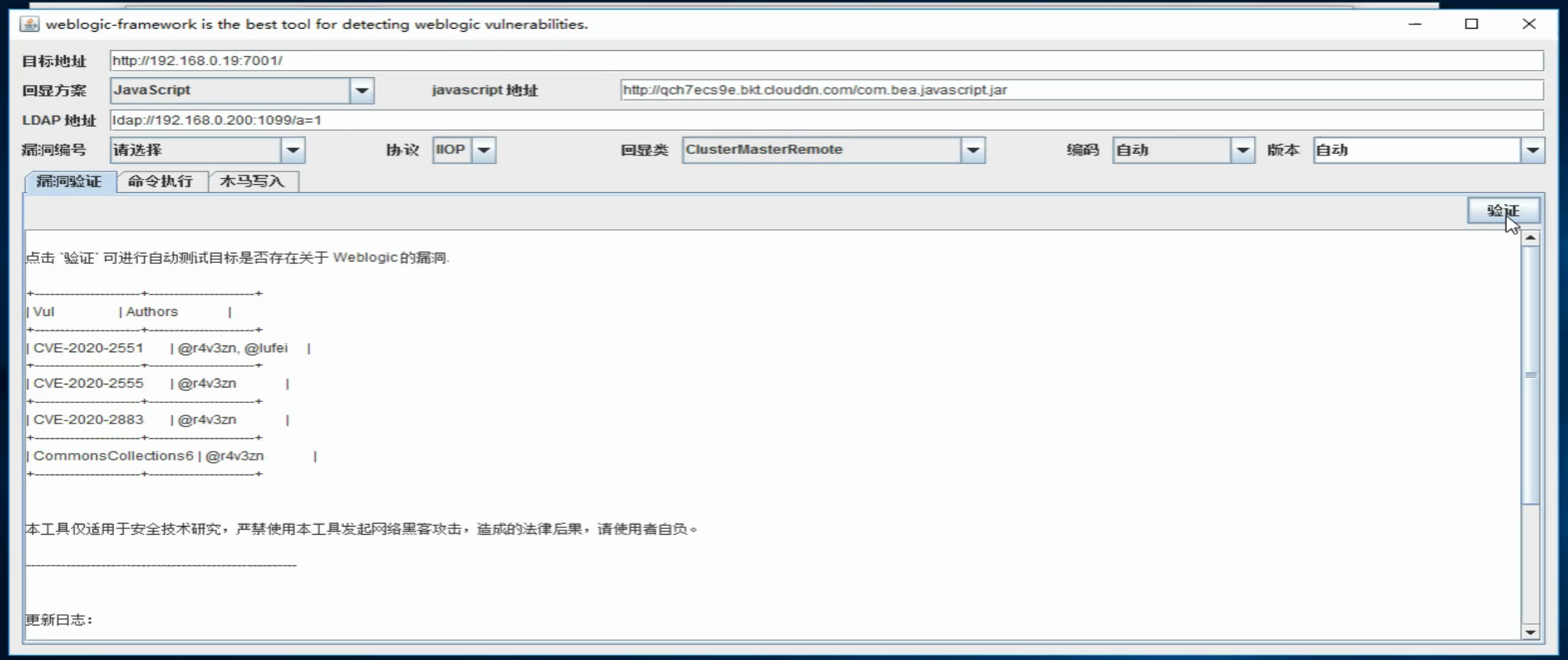
1. 寻找一个继承java.rmi.Remote的接口,并且该接口有一个方法的返回值为string类型。
2. 编写一个回显类去实现这个接口。
3. 在返回值为string类型的方法里写上要执行恶意的payload,在main方法中把当前类绑定到目标服务器中。
4. 使用cve漏洞去加载这个恶意类。
5. 攻击者使用lookup方法获取回显类对象,调用方法获取回显结果。

攻击者调用lookup方法获取回显数据,此时服务端给客户端(攻击者)返回的是序列化数据,客户端(攻击者)此时会做反序列化处理,把序列化数据转成对象。由于攻击者需要发送cve漏洞,攻击者的classloader会加载cve漏洞的jar包。服务端只需要给客户端(攻击者)返回对应的cve漏洞数据包即可反制客户端(攻击者)。

客户端和服务端交互图

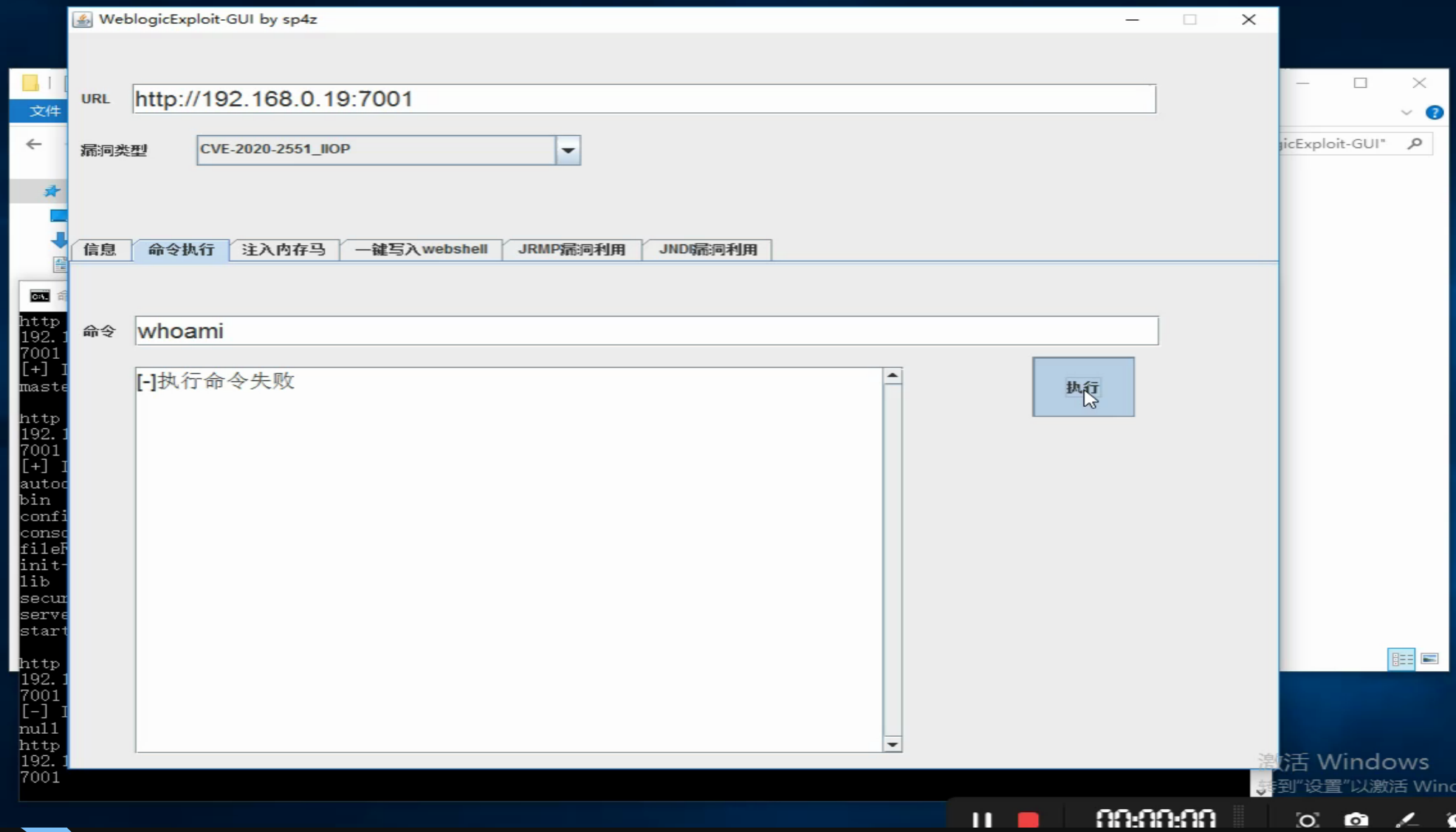

客户端(攻击者)
发送CVE漏洞
服务端返回正常的响应包
调用远程回显方法,获取命令回显
返回序列化的CVE漏洞
客户端接收到数据会对数据包
进行反序列化,成功反制攻击者
服务端(蜜罐)

反制视频(1)



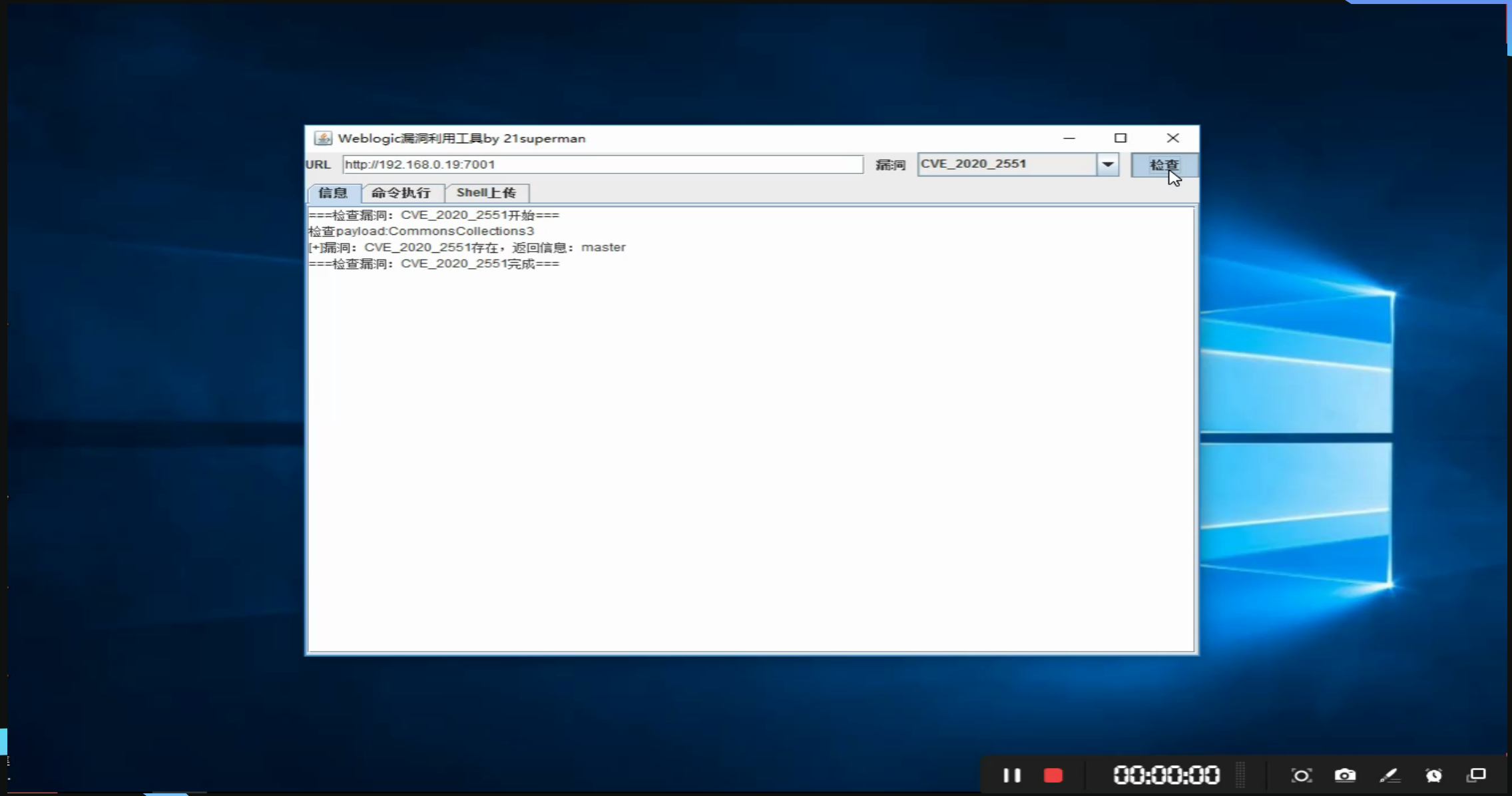
00:00:00

反制视频(2)


WeblogicExploit-GUI by sp4z
URL
http://192.168.0.19:7001
漏洞类型
CVE-2020-2551_IIOP
信息
命令执行
注入内存马
一键写入webshell
JRMP漏洞利用
JNDI漏洞利用
命令
whoami
[-]执行命令失败
执行
激活 Windows

反制视频(3)

不仅仅是这三个工具可以被反制



KCon

归源·智变

Returning to the source · Intelligent transformation

感谢您的观看！

THANK YOU FOR YOUR WATCHING