JAVA-RMI

- `RMI`全称是`Remote Method Invocation`，远程方法调用，从这个名字就可以看出，他的目标和RPC其实是类似的，是让某个Java虚拟机上的对象调用另一个Java虚拟机中对象上的方法，只不过RMI是Java独有的一种机制
- RMI Server：

```
package aaaa.rmi;

import java.rmi.Naming;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.server.UnicastRemoteObject;
public class RMIServer {
      public interface IRemoteHelloWorld extends Remote {
            public String hello() throws RemoteException;
      }
      public class RemoteHelloWorld extends UnicastRemoteObject implements
IRemoteHelloWorld {
            protected RemoteHelloWorld() throws RemoteException {
                  super();
            }
            public String hello() throws RemoteException {
                  System.out.println("call from");
                  return "Hello world";
            }
      }
      private void start() throws Exception {
            RemoteHelloWorld h = new RemoteHelloWorld();
            LocateRegistry.createRegistry(1099);
            Naming.rebind("rmi://127.0.0.1:1099/Hello", h);
      }
      public static void main(String[] args) throws Exception {
            new RMIServer().start();
      }
}
```

- 一个`RMI Server`分为三部分：
  - 一个继承了`java.rmi.Remote`的接口，其中定义我们要远程调用的函数，比如这里的`hello()`

    - 一个实现了此接口的类
    - 一个主类，用来创建`Registry`，并将上面的类实例化后绑定到一个地址，这就是所谓的Server了
- RMI Client：

```
package aaaa.rmi;

import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;

public class RMIClient {
      public static void main(String[] args) throws Exception {
            RMIServer.IRemoteHelloWorld hello = (RMIServer.IRemoteHelloWorld)
                        Naming.lookup("rmi://192.168.128.178:1099/Hello");
            String ret = hello.hello();
            System.out.println( ret);
     }
}
```

- 使用`Naming.lookup`在`Registry`中寻找到名字是Hello的对象，后面的使用就和在本地使用一样了
- 一个RMI过程有以下三个参与者：`RMI Registry`、`RMI Server`、`RMI Client`
- 通常我们在新建一个`RMI Registry`的时候，都会直接绑定一个对象在上面，也就是说我们示例代码中的Server其实包含了`Registry`和`Server`两部分

```
LocateRegistry.createRegistry(1099);   //创建并运行RMI Registry
Naming.bind("rmi://127.0.0.1:1099/Hello", new RemoteHelloWorld());   //行将
RemoteHelloWorld对象绑定到Hello这个名字上
```

- `Naming.bind`的第一个参数是一个URL，形如：`rmi://host:port/name`，其中，`host`和`port`就是`RMI Registry`的地址和端口，name是远程对象的名字
- 如果`RMI Registry`在本地运行，那么`host`和`port`是可以省略的，此时`host`默认是`localhost`，`port`默认是`1099`：

```
Naming.bind("Hello", new RemoteHelloWorld());
```

攻击RMI Registry

- Java对远程访问`RMI Registry`做了限制，只有来源地址是`localhost`的时候，才能调用`rebind`、`bind`、`unbind`等方法
- 不过`list`和`lookup`方法可以远程调用，`list`方法可以列出目标上所有绑定的对象：

```
String[] s = Naming.list("rmi://192.168.128.178:1099");
```

- `lookup`作用就是获得某个远程对象，那么，只要目标服务器上存在一些危险方法，通过RMI就可以对其进行调用

RMI利用codebase执行任意代码

- `Java Applet`是可以运行在浏览器中的，在使用`Applet`的时候通常需要指定一个`codebase`属性：

```
<applet code="HelloWorld.class" codebase="Applets" width="800" height="600">
</applet>
```

- 除了`Applet`，`RMI`中也存在远程加载的场景，也会涉及到`codebase`，`codebase`是一个地址，告诉Java虚拟机应该从哪个地方去搜索类，有点像`CLASSPATH`，但`CLASSPATH`是本地路径，而`codebase`通常是远程URL，比如`http`、`ftp`等
- 如果指定`codebase=http://example.com/`，然后加载`org.vulhub.example.Example`类，则Java虚拟机会下载这个文件`http://example.com/org/vulhub/example/Example.class`，并作为`Example`类的字节码
- RMI的流程中，客户端和服务端之间传递的是一些序列化后的对象，这些对象在反序列化时，就会去寻找类，如果某一端反序列化时发现一个对象，那么就会去自己的`CLASSPATH`下寻找想对应的类；如果在本地没有找到这个类，就会去远程加载codebase中的类
- 如果`codebase`被控制，就可以加载恶意类了
- 在RMI中，可以将`codebase`随着序列化数据一起传输，服务器在接收到这个数据后就会去`CLASSPATH`和指定的`codebase`寻找类，由于`codebase`被控制导致任意命令执行漏洞
- 漏洞条件：
  - 安装并配置了`SecurityManager`
  - Java版本低于7u21、6u45，或者设置了`java.rmi.server.useCodebaseOnly=false`
  - 在`java.rmi.server.useCodebaseOnly`配置为 true 的情况下，Java虚拟机将只信任预先配置好的`codebase`，不再支持从RMI请求中获取
- 看着P神的文章想跟着复现一下不知道抽什么风一直报错，放弃了，我选择嗯看
- 研究一下他的源码，大概意思就是，先起一个`RMIServer`，在编译运行的时候设置三个参数：

```
java.rmi.server.hostname=192.168.135.142 //是服务器的IP地址，远程调用时需要根据这个值来访问RMI Server
java.rmi.server.useCodebaseOnly=false //支持从RMI请求中获取Codebase
java.security.policy=client.policy //授权
```

- 再建立一个RMIClient.java，在另一个位置运行，此时`RMI Server`在本地`CLASSPATH`里找不到类，会去加载`Codebase`中的类

```
import java.rmi.Naming;
import java.util.List;
import java.util.ArrayList;
import java.io.Serializable;
public class RMIClient implements Serializable {
      public class Payload extends ArrayList<Integer> {}
      public void lookup() throws Exception {
            ICalc r = (ICalc)
                        Naming.lookup("rmi://192.168.135.142:1099/refObj");
            List<Integer> li = new Payload();
            li.add(3);
            li.add(4);
            System.out.println(r.sum(li));
      }
       public static void main(String[] args) throws Exception {
            new RMIClient().lookup();
      }
}
```

- 查看`example.com`的日志，慧收到了来自Java的请求`/RMIClient$Payload.class`，因为还没有实际放置这个类文件，所以会出现异常
- 只需要编译一个恶意类，将其class文件放置在Web服务器的`/RMIClient$Payload.class`即可