Java安全漫谈 - 04.RMI篇(1)

> 这是代码审计知识星球中Java安全的第四篇文章。

这一篇开始我们来说说RMI，从历史开始说起。

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。

我们直接从一个例子开始演示RMI的流程吧。

首先编写一个RMI Server：

```
1  package org.vulhub.RMI;
2
3  import java.rmi.Naming;
4  import java.rmi.Remote;
5  import java.rmi.RemoteException;
6  import java.rmi.registry.LocateRegistry;
7  import java.rmi.registry.Registry;
8  import java.rmi.server.UnicastRemoteObject;
9
10 public class RMIServer {
11     public interface IRemoteHelloWorld extends Remote {
12         public String hello() throws RemoteException;
13     }
14
15     public class RemoteHelloWorld extends UnicastRemoteObject implements
   IRemoteHelloWorld {
16         protected RemoteHelloWorld() throws RemoteException {
17             super();
18         }
19
20         public String hello() throws RemoteException {
21             System.out.println("call from");
22             return "Hello world";
23         }
24     }
25
26     private void start() throws Exception {
27         RemoteHelloWorld h = new RemoteHelloWorld();
28         LocateRegistry.createRegistry(1099);
29         Naming.rebind("rmi://127.0.0.1:1099/Hello", h);
30     }
31
32     public static void main(String[] args) throws Exception {
```

```
        new RMIServer().start();
    }
}

```

一个RMI Server分为三部分：

1. 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()`
2. 一个实现了此接口的类
3. 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。这就是我们所谓的Server了。

因为要写文章，我这里写的就比较简单，三个东西我就放在一个类里了。

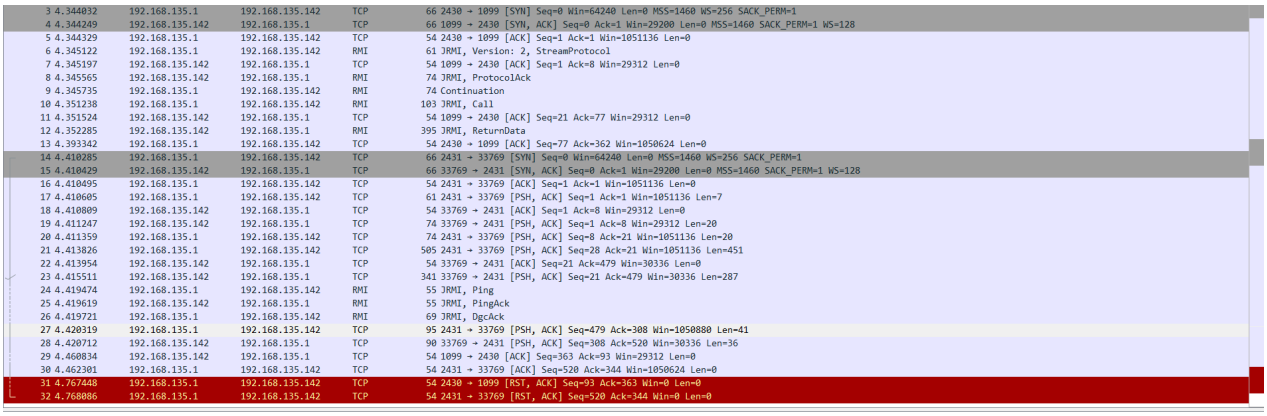
接着我们编写一个RMI Client：

```
package org.vulhub.Train;

import org.vulhub.RMI.RMIServer;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;

public class TrainMain {
    public static void main(String[] args) throws Exception {
        RMIServer.IRemoteHelloWorld hello = (RMIServer.IRemoteHelloWorld) Naming.lookup("rmi://192.168.135.142:1099/Hello");
        String ret = hello.hello();
        System.out.println( ret);
    }
}
```

客户端就简单多了，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。

虽说执行远程方法的时候代码是在远程服务器上执行的，但实际上我们还是需要知道有哪些方法，这时候接口的重要性就体现了，这也是为什么我们前面要继承`Remote`并将我们需要调用的方法写在接口`IRemoteHelloWorld`里，因为客户端也需要用到这个接口。

为了理解RMI的通信过程，我们用wireshark抓包看看：



3	4.344032	192.168.135.1	192.168.135.142	TCP	66 2430 → 1099 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
4	4.344249	192.168.135.142	192.168.135.1	TCP	66 1099 → 2430 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
5	4.344329	192.168.135.1	192.168.135.142	TCP	54 2430 → 1099 [ACK] Seq=1 Ack=1 Win=1051136 Len=0
6	4.345122	192.168.135.1	192.168.135.142	RMI	61 JRMI, Version: 2, StreamProtocol
7	4.345197	192.168.135.142	192.168.135.1	TCP	54 1099 → 2430 [ACK] Seq=1 Ack=8 Win=29312 Len=0
8	4.345565	192.168.135.142	192.168.135.1	RMI	74 JRMI, ProtocolAck
9	4.345735	192.168.135.1	192.168.135.142	RMI	74 Continuation
10	4.351238	192.168.135.1	192.168.135.142	RMI	103 JRMI, Call
11	4.351524	192.168.135.142	192.168.135.1	TCP	54 1099 → 2430 [ACK] Seq=21 Ack=77 Win=29312 Len=0
12	4.352285	192.168.135.142	192.168.135.1	RMI	395 JRMI, ReturnData
13	4.393342	192.168.135.1	192.168.135.142	TCP	54 2430 → 1099 [ACK] Seq=77 Ack=362 Win=1050624 Len=0
14	4.410285	192.168.135.1	192.168.135.142	TCP	66 2431 → 33769 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
15	4.410429	192.168.135.142	192.168.135.1	TCP	66 33769 → 2431 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
16	4.410495	192.168.135.1	192.168.135.142	TCP	54 2431 → 33769 [ACK] Seq=1 Ack=1 Win=1051136 Len=0
17	4.410605	192.168.135.1	192.168.135.142	TCP	61 2431 → 33769 [PSH, ACK] Seq=1 Ack=1 Win=1051136 Len=7
18	4.410809	192.168.135.142	192.168.135.1	TCP	54 33769 → 2431 [ACK] Seq=1 Ack=8 Win=29312 Len=0
19	4.411247	192.168.135.142	192.168.135.1	TCP	74 33769 → 2431 [PSH, ACK] Seq=1 Ack=8 Win=29312 Len=20
20	4.411359	192.168.135.1	192.168.135.142	TCP	74 2431 → 33769 [PSH, ACK] Seq=8 Ack=21 Win=1051136 Len=20
21	4.413826	192.168.135.1	192.168.135.142	TCP	505 2431 → 33769 [PSH, ACK] Seq=28 Ack=21 Win=1051136 Len=451
22	4.413954	192.168.135.142	192.168.135.1	TCP	54 33769 → 2431 [ACK] Seq=21 Ack=479 Win=30336 Len=0
23	4.415511	192.168.135.142	192.168.135.1	TCP	341 33769 → 2431 [PSH, ACK] Seq=21 Ack=479 Win=30336 Len=287
24	4.419474	192.168.135.1	192.168.135.142	RMI	55 JRMI, Ping
25	4.419619	192.168.135.142	192.168.135.1	RMI	55 JRMI, PingAck
26	4.419721	192.168.135.1	192.168.135.142	RMI	69 JRMI, DgcAck
27	4.420319	192.168.135.1	192.168.135.142	TCP	95 2431 → 33769 [PSH, ACK] Seq=479 Ack=308 Win=1050880 Len=41
28	4.420712	192.168.135.142	192.168.135.1	TCP	90 33769 → 2431 [PSH, ACK] Seq=308 Ack=520 Win=30336 Len=36
29	4.460834	192.168.135.142	192.168.135.1	TCP	54 1099 → 2430 [ACK] Seq=363 Ack=93 Win=29312 Len=0
30	4.462301	192.168.135.1	192.168.135.142	TCP	54 2431 → 33769 [ACK] Seq=520 Ack=344 Win=1050624 Len=0
31	4.767448	192.168.135.1	192.168.135.142	TCP	54 2430 → 1099 [RST, ACK] Seq=93 Ack=363 Win=0 Len=0
32	4.768086	192.168.135.1	192.168.135.142	TCP	54 2431 → 33769 [RST, ACK] Seq=520 Ack=344 Win=0 Len=0

这就是完整的通信过程，我们可以发现，整个过程进行了两次TCP握手，也就是我们实际建立了两次TCP连接。

第一次建立TCP连接是连接远端`192.168.135.142`的1099端口，这也是我们在代码里看到的端口，二者进行沟通后，我向远端发送了一个“Call”消息，远端回复了一个“ReturnData”消息，然后我新建了一个TCP连接，连到远端的33769端口。

那么为什么我会连接33769端口呢?

细细阅读数据包我们会发现，在“ReturnData”这个包中，返回了目标的IP地址`192.168.135.142`，其后跟的一个字节`\x00\x00\x83\xE9`，刚好就是整数`33769`的网络序列：

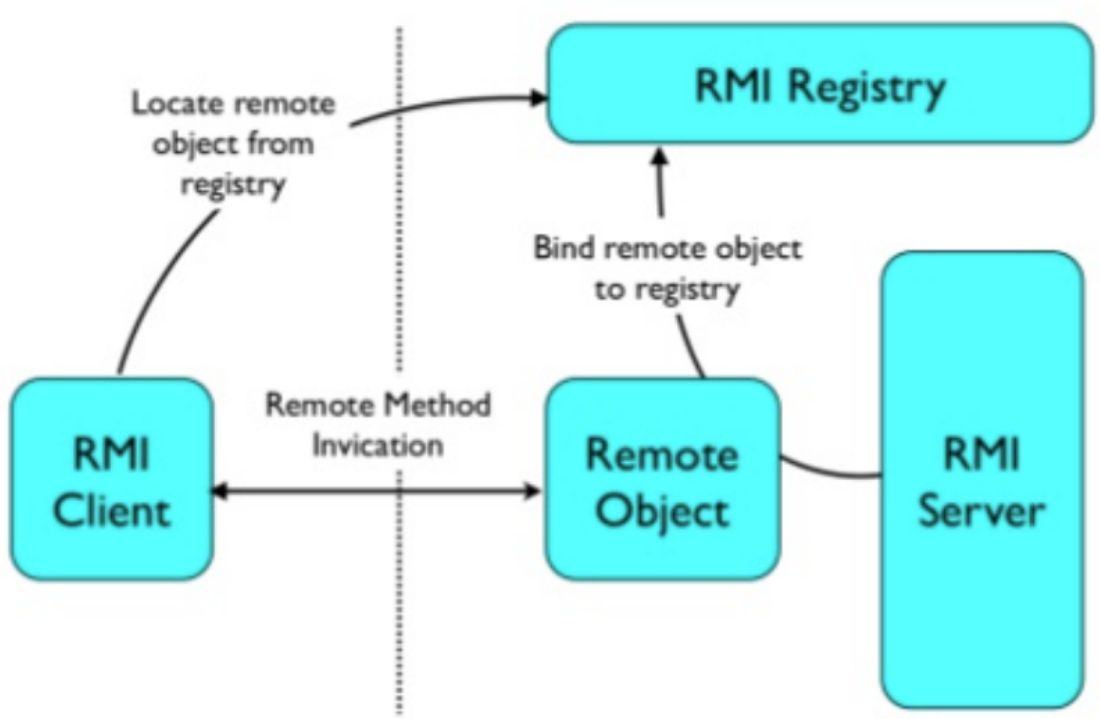
```
0030   .. .. .. .. .. .. .. ac ed 00 05 77 0f 01 18 35   ......Q....w...5
0040   cf d9 00 00 01 6c 39 4f ec 84 80 08 73 7d 00 00   .....l9O....s}..
0050   00 02 00 0f 6a 61 76 61 2e 72 6d 69 2e 52 65 6d   ....java.rmi.Rem
0060   6f 74 65 00 2a 6f 72 67 2e 76 75 6c 68 75 62 2e   ote.*org.vulhub.
0070   52 4d 49 2e 52 4d 49 53 65 72 76 65 72 24 49 52   RMI.RMIServer$IR
0080   65 6d 6f 74 65 48 65 6c 6c 6f 57 6f 72 6c 64 70   emoteHelloWorldp
0090   78 72 00 17 6a 61 76 61 2e 6c 61 6e 67 2e 72 65   xr..java.lang.re
00a0   66 6c 65 63 74 2e 50 72 6f 78 79 e1 27 da 20 cc   flect.Proxy.'. .
00b0   10 43 cb 02 00 01 4c 00 01 68 74 00 25 4c 6a 61   .C....L..ht.%Lja
00c0   76 61 2f 6c 61 6e 67 2f 72 65 66 6c 65 63 74 2f   va/lang/reflect/
00d0   49 6e 76 6f 63 61 74 69 6f 6e 48 61 6e 64 6c 65   InvocationHandle
00e0   72 3b 70 78 70 73 72 00 2d 6a 61 76 61 2e 72 6d   r;pxpsr.-java.rm
00f0   69 2e 73 65 72 76 65 72 2e 52 65 6d 6f 74 65 4f   i.server.RemoteO
0100   62 6a 65 63 74 49 6e 76 6f 63 61 74 69 6f 6e 48   bjectInvocationH
0110   61 6e 64 6c 65 72 00 00 00 00 00 00 00 02 02 00   andler..........
0120   00 70 78 72 00 1c 6a 61 76 61 2e 72 6d 69 2e 73   .pxr..java.rmi.s
0130   65 72 76 65 72 2e 52 65 6d 6f 74 65 4f 62 6a 65   erver.RemoteObje
0140   63 74 d3 61 b4 91 0c 61 33 1e 03 00 00 70 78 70   ct.a...a3....pxp
0150   77 38 00 0a 55 6e 69 63 61 73 74 52 65 66 00 0f   w8..UnicastRef..
0160   31 39 32 2e 31 36 38 2e 31 33 35 2e 31 34 32 00   192.168.135.142.
0170   00 83 e9 1b 78 c2 0b 23 a0 69 c0 18 35 cf d9 00   ....x..#.i..5...
0180   00 01 6c 39 4f ec 84 80 01 01 78                  ..l9O.....x
```

```
In [6]: pack('>I', 33769)
Out[6]: b'\x00\x00\x83\xe9'
```

其实这段数据流中从`\xAC\xED`开始往后就是Java序列化数据了，IP和端口只是这个对象的一部分罢了。

所以捋一捋这整个过程，首先客户端连接Registry，并在其中寻找Name是Hello的对象，这个对应数据流中的Call消息；然后Registry返回一个序列化的数据，这个就是找到的Name=Hello的对象，这个对应数据流中的ReturnData消息；客户端反序列化该对象，发现该对象是一个远程对象，地址在`192.168.135.142:33769`，于是再与这个地址建立TCP连接；在这个新的连接中，才执行真正远程方法调用，也就是`hello()`。

我们借用下图来说明这些元素间的关系：



RMI Registry就像一个网关，他自己是不会执行远程方法的，但RMI Server可以在上面注册一个Name到对象的绑定关系；RMI Client通过Name向RMI Registry查询，得到这个绑定关系，然后再连接RMI Server；最后，远程方法实际上在RMI Server上调用。