一种Java反编译器的通用对抗手段

如何生成反编译失败的class文件?

build.gradle.kts

```
plugins {
    java
    kotlin("jvm") version "1.3.61"
}

group = "org.vidar"
version = "1.0.0"

repositories {
    mavenCentral()
    maven("https://maven.hackery.site/")
}

dependencies {
    implementation(kotlin("stdlib-jdk8"))

    arrayOf("asm", "asm-tree", "asm-commons").forEach {
        implementation(group = "org.ow2.asm", name = it, version = "7.2")
    }

    implementation("codes.som.anthony:koffee:7.1.0")
}

configure<JavaPluginConvention> {
    sourceCompatibility = JavaVersion.VERSION_1_8
}
tasks {
    compileKotlin {
        kotlinOptions.jvmTarget = "1.8"
    }
    compileTestKotlin {
        kotlinOptions.jvmTarget = "1.8"
    }
}
```

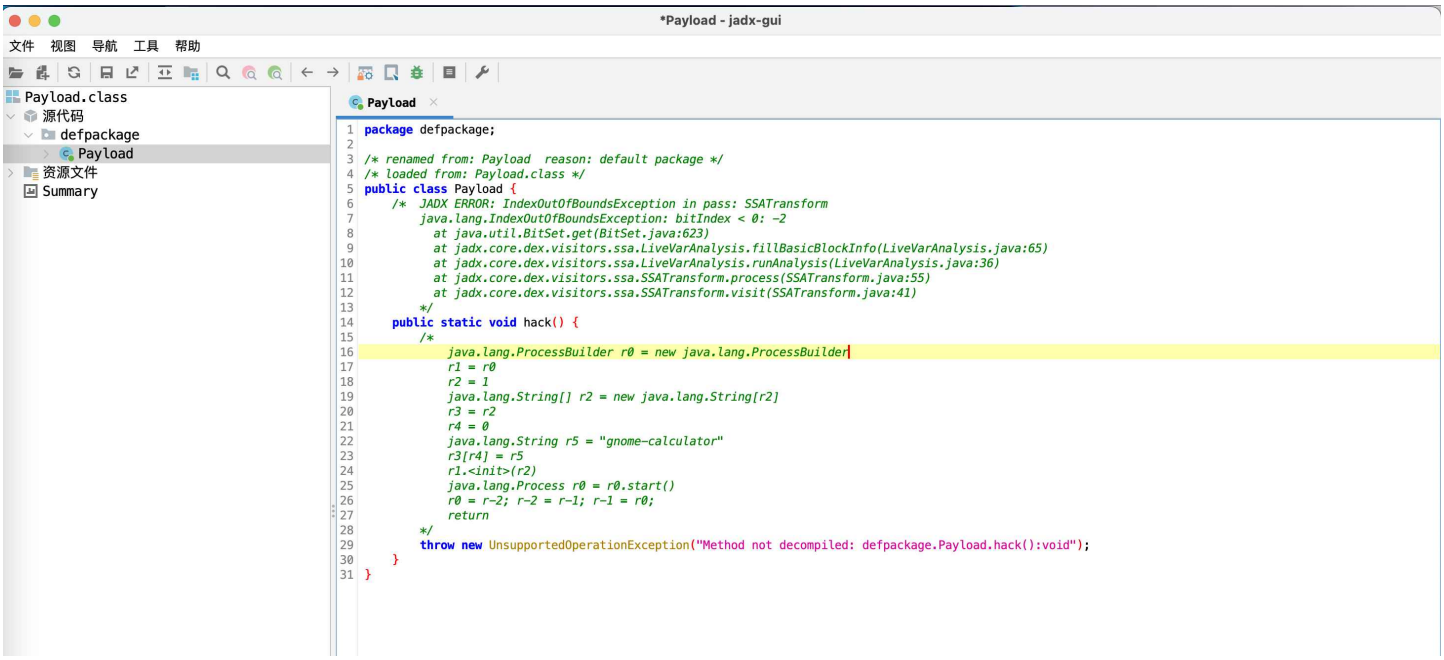
使用koffee直接操作class指令

```
import codes.som.anthony.koffee.assembleClass
import codes.som.anthony.koffee.insns.jvm.*
import codes.som.anthony.koffee.modifiers.public
import org.objectweb.asm.ClassWriter
import org.objectweb.asm.tree.ClassNode
import java.io.FileOutputStream
import java.io.PrintStream


fun saveClz(payload: ClassNode) {

    val classWriter = ClassWriter(ClassWriter.COMPUTE_MAXS)
    payload.accept(classWriter)
    val fos = FileOutputStream("Payload.class")
    fos.write(classWriter.toByteArray())
    fos.close()
    println("success")
}

fun main() {
    saveClz(assembleClass(public, "Payload") {
        method(public + static, "hack", void) {

            new(ProcessBuilder::class.java)
            dup
            iconst_1
            anewarray(String::class.java)
            dup
            iconst_0
            ldc("gnome-calculator")
            aastore
            invokespecial(ProcessBuilder::class.java,"<init>",void, Array<String
            invokevirtual(ProcessBuilder::class.java,"start",Process::class.java
            pop

            swap

            _return
        }
    })
}
```

可以看到FernFlower反编译失败

```
//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by FernFlower decompiler)
//


5 usages
public class Payload {
    public static void hack() {
        // $FF: Couldn't be decompiled
    }
}
```

jadx-gui也同样失败

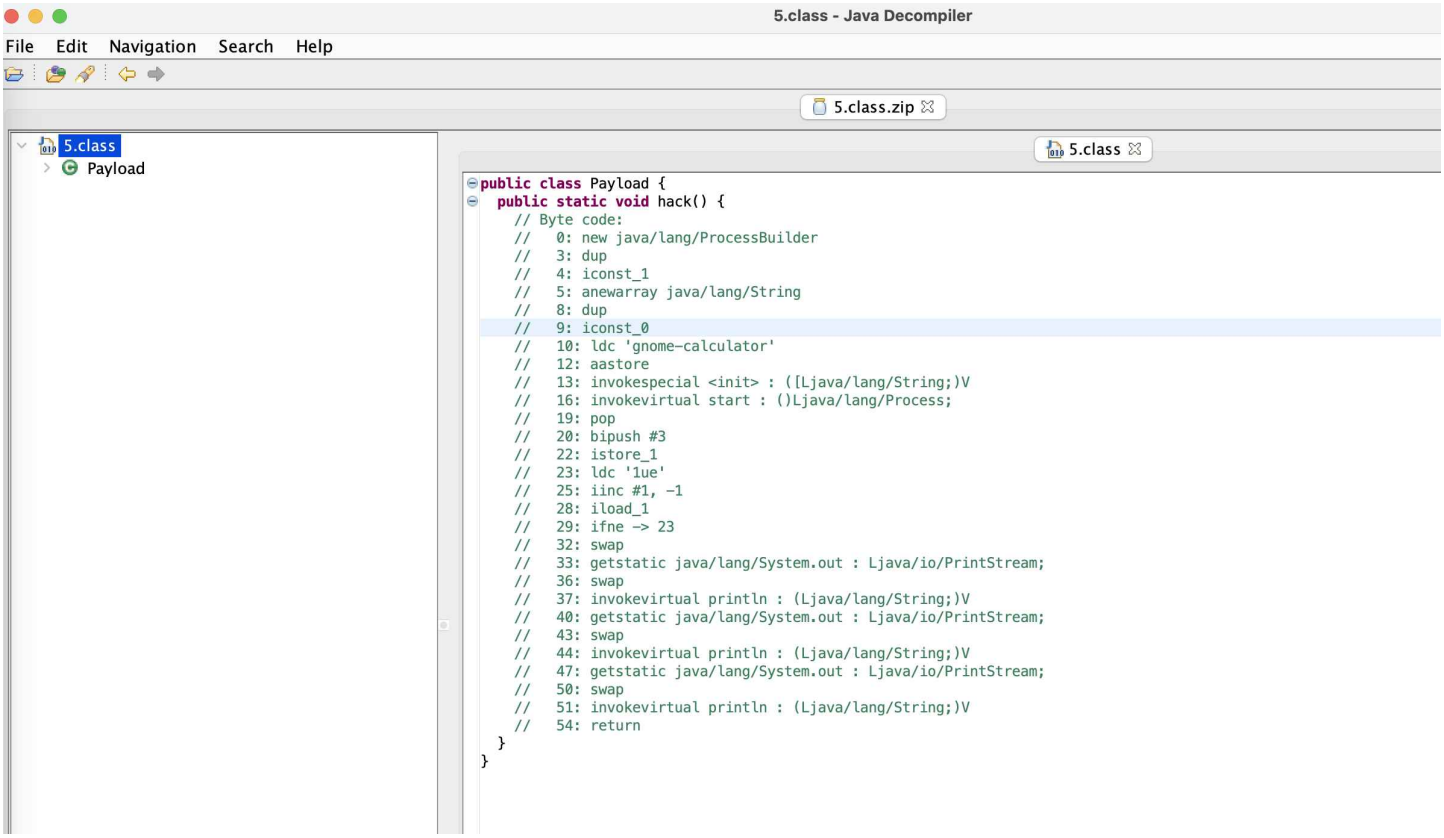


现在我们就可以肆无忌惮的给我们的class添加指令了，例如我们再添加一些逻辑指令来混淆

```
fun main() {
    saveClz(assembleClass(public, "Payload") {
        method(public + static, "hack", void) {

            new(ProcessBuilder::class.java)
            dup
            iconst_1
            anewarray(String::class.java)
            dup
```

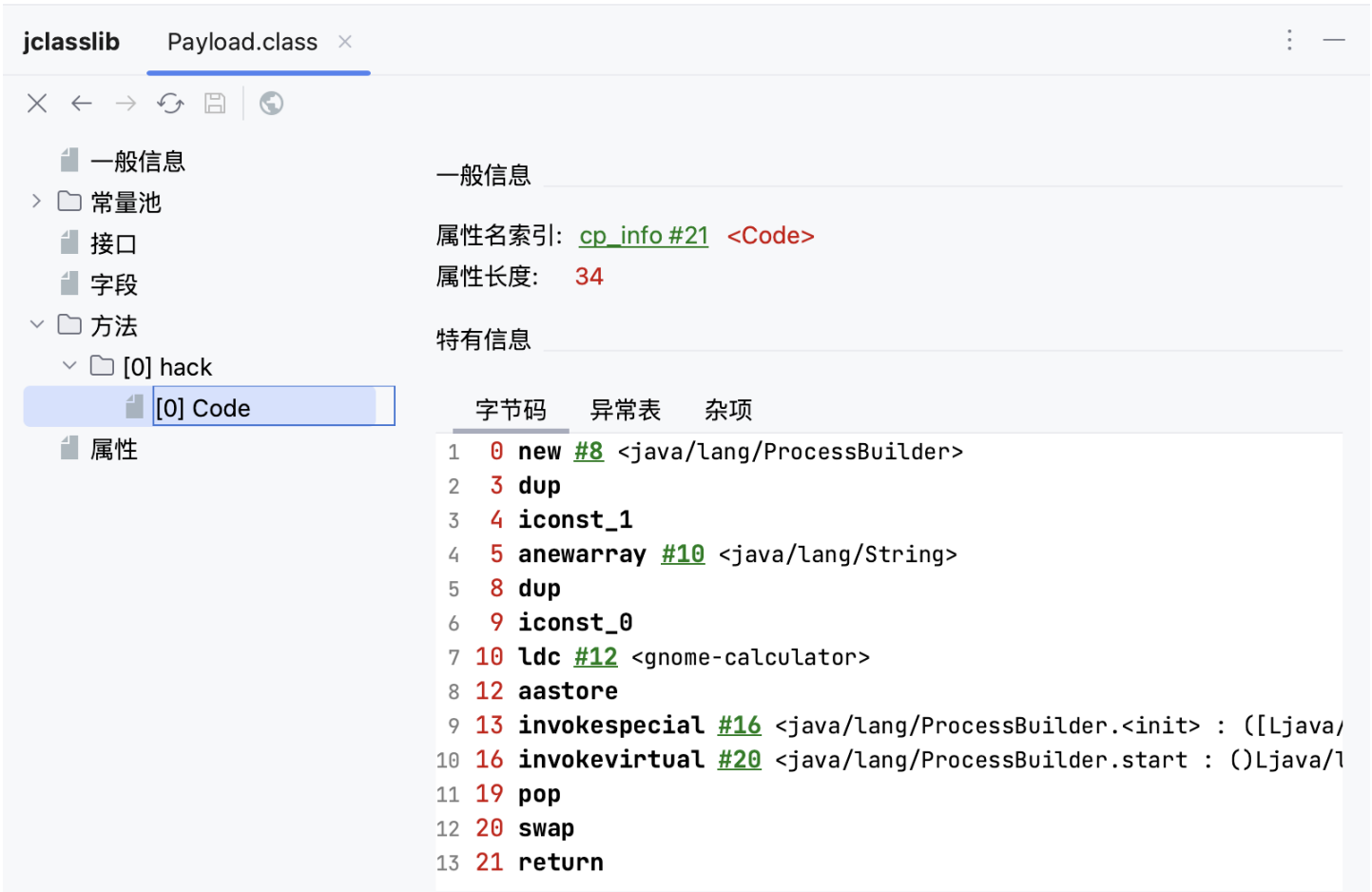
```
            iconst_0
            ldc("gnome-calculator")
            aastore
            invokespecial(ProcessBuilder::class.java,"<init>",void, Array<String
            invokevirtual(ProcessBuilder::class.java,"start",Process::class.java
            pop

            bipush(3)
            istore_1
            +L["loop_start"]
            ldc("1ue")
            iinc(1, -1)
            iload_1
            ifne(L["loop_start"])
            swap
            for (i in 0 until 3) {
                getstatic(System::class, "out", PrintStream::class)
                swap
                invokevirtual(PrintStream::class, "println", void, String::class
            }

            _return
        }
    })
}
```

这样也让jd-gui反编译失败



为什么反编译会失败?

其实你使用jclasslib直接查看jvm指令还是可以看到的（当然只是部分，我们可以再混淆一下，）



而反编译器失败的原因是，反编译器或者说jvm在运行时会对class的合法性进行校验

对于案例中的Payload.class，如果你去掉了 **`swap` 指令**，你会发现反编译依然可以成功。

而为什么在最后加了一个swap指令，反编译就会失败，或者说jvm校验class会不通过？

你可以在https://en.wikipedia.org/wiki/List_of_Java_bytecode_instructions中查看swap指令对应的意思，就是交换2个栈顶的值，那么如果前面没有一些对入栈的操作，jvm在校验class的时候很有可能会失败（至少hotspot时这样）

swap	5f	0101 1111		value2, value1 → value1, value2	swaps two top words on the stack (note that value1 and value2 must not be double or long)

如何运行/加载Payload.class?

正常情况下我们想要运行Payload.class中hack()函数的逻辑，应使用如下这段代码

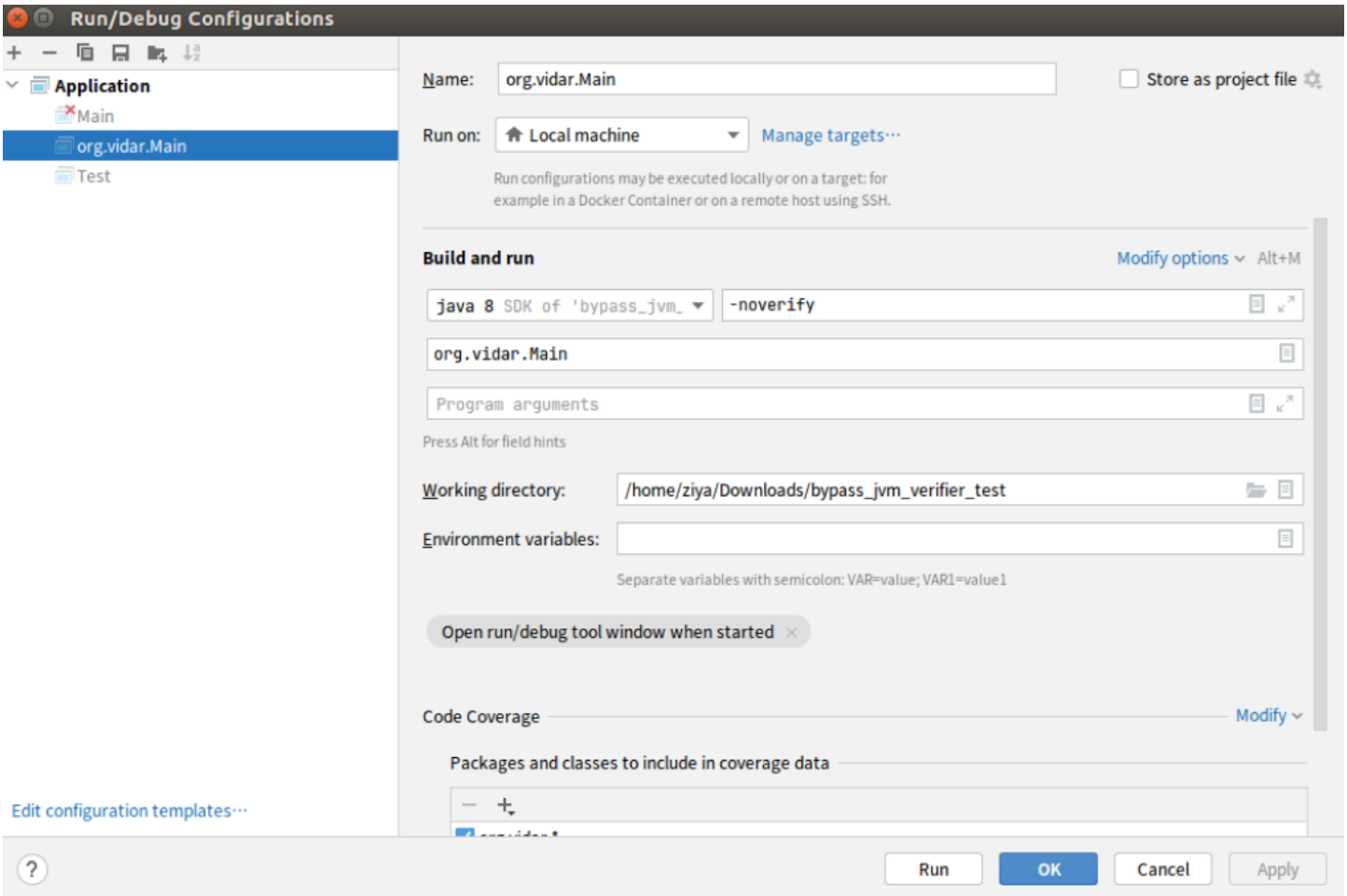
```
1  package org.vidar;
2
3  import java.io.IOException;
4  import java.lang.reflect.InvocationTargetException;
5  import java.lang.reflect.Method;
6  import java.nio.file.Files;
7  import java.nio.file.Paths;
8
9  public class Main {
10     public static void main(String[] args) throws ClassNotFoundException, NoSuch
11         Class<?> payload = new InMemoryClassLoader().findClass("Payload");
12         Method m = payload.getDeclaredMethod("hack");
13         m.setAccessible(true);
14         System.out.println(m);
15         m.invoke(null);
16     }
17
18     static class InMemoryClassLoader extends ClassLoader {
19         @Override
20         protected Class<?> findClass(String name) throws ClassNotFoundException
21             byte[] classData = new byte[0];
22             try {
23                 classData = Files.readAllBytes(Paths.get("/home/1ue/Downloads/Pa
24             } catch (IOException e) {
25                 e.printStackTrace();
26             }
27             return defineClass(name, classData, 0, classData.length);
28         }
29     }
```

```
30 }
31
```

但是直接运行会报错java.lang.VerifyError也就是class没有通过校验

但是jvm有一个**-noverify**的启动参数，其作用是禁用字节码验证

我们在IDEA中设置一下



再次运行

```
    Class<?> payload = new InMemoryClassLoader().findClass( name: "Payload");
    Method m = payload.getDeclaredMethod( name: "hack");
    m.setAccessible(true);
    System.out.println(m);
    m.invoke( obj: null);
}

1 usage
static class InMemoryClassLoader extends ClassLoader {
    1 usage
    @Override
    protected Class<?> findClass(String name) throws ClassNotFoundException {
        byte[] classData = new byte[0];
```


```
org.vidar.Main
# Failed to write core dump. Core dumps have been disabled. To enable core dumping, try                    a again
#
# An error report file with more information is saved as:
# /home/ziya/Downloads/bypass_jvm_verifier_test/hs_err_pid38748.log
#
# If you would like to submit a bug report, please visit:
#   http://bugreport.java.com/bugreport/crash.jsp
#
Current thread is 140077872101120
Dumping core ...

Process finished with exit code 134 (interrupted by signal 6: SIGABRT)
```

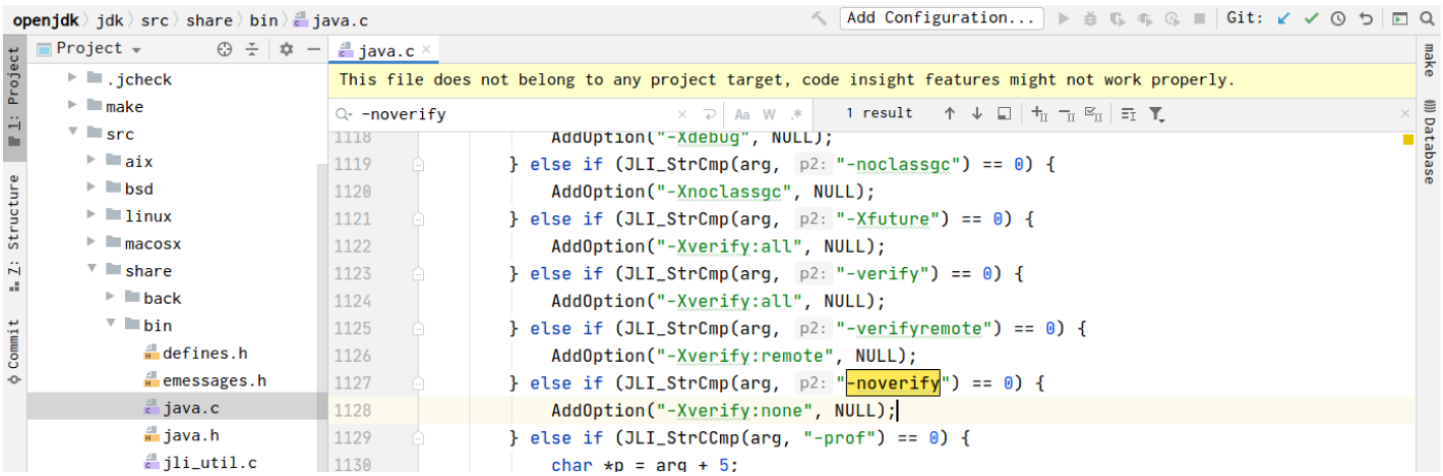
如何在java层利用?

如果是修改jvm的启动参数，这显得略有些鸡肋

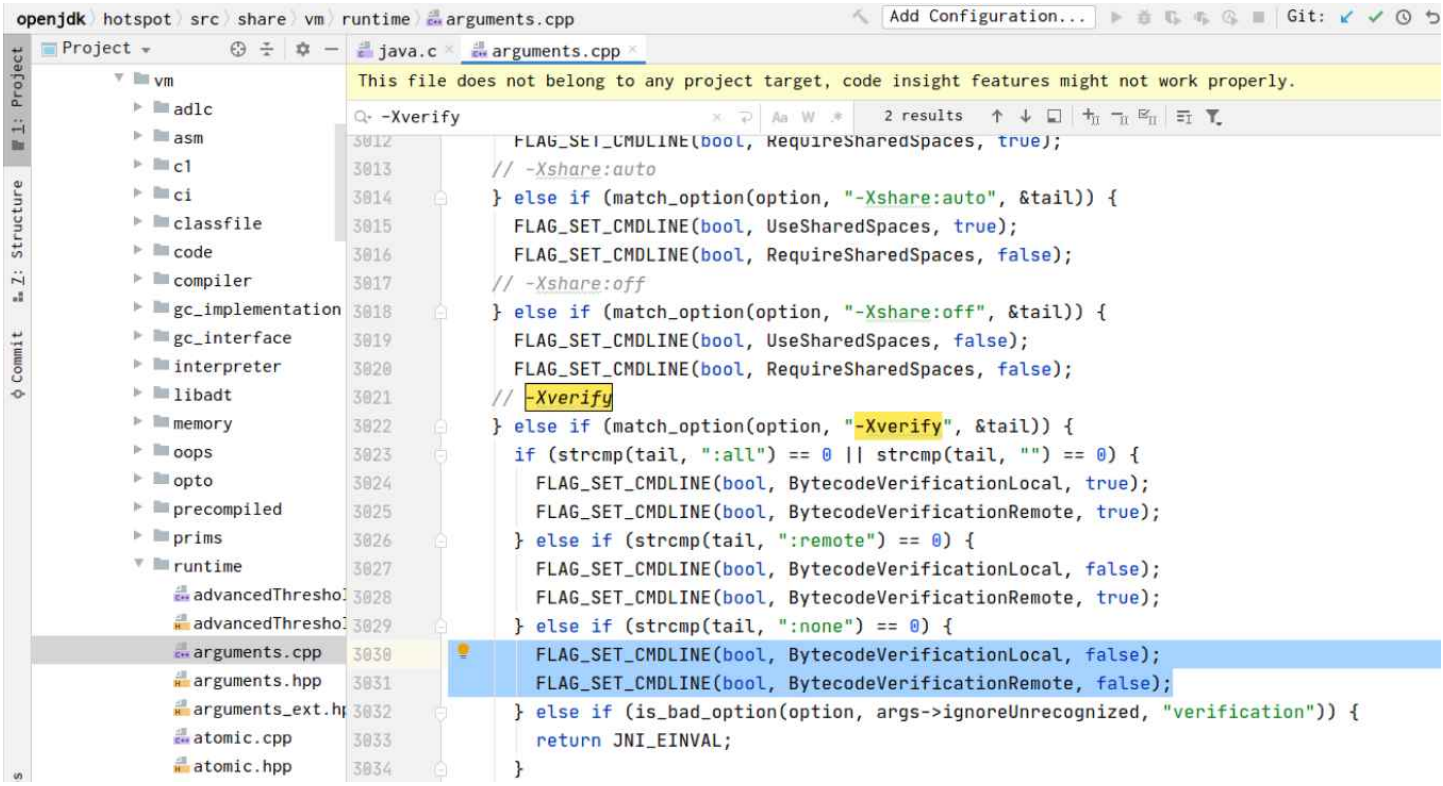
不过庆幸的是我们可以利用**Unsafe**来为jvm添加了-noverify的参数的效果

所以我们需要知道-noverify到底做了什么

- 如图，-noverify的启动选项在java.c文件中定义，相当于添加了 `-Xverify:none` 的option



- 对于 `-X` 的option，hotspot对应的处理逻辑在 `arguements.cpp` 中，对于none来说，就相当于把 `BytecodeVerificationLocal` 和 `BytecodeVerificationRemote` 这2个值设置为 **false**



- **所以我们现在的目的就是把这2个标志所对应的地址的值设置为0**

在x86的linux的jdk中，JVM公开了一些全局字段，以允许应用程序检查其当前状态。

```
$ cd /usr/lib/jvm/default/jre/lib/amd64/server/
server/ $ nm -D libjvm.so | grep gHotSpot
0000000000d222e0 B gHotSpotVMIntConstantEntryArrayStride
0000000000d222f0 B gHotSpotVMIntConstantEntryNameOffset
0000000000d222e8 B gHotSpotVMIntConstantEntryValueOffset
0000000000ce4568 D gHotSpotVMIntConstants
0000000000d222c8 B gHotSpotVMLongConstantEntryArrayStride
0000000000d222d8 B gHotSpotVMLongConstantEntryNameOffset
0000000000d222d0 B gHotSpotVMLongConstantEntryValueOffset
0000000000ce4560 D gHotSpotVMLongConstants
0000000000d22338 B gHotSpotVMStructEntryAddressOffset
0000000000d22330 B gHotSpotVMStructEntryArrayStride
0000000000d22358 B gHotSpotVMStructEntryFieldNameOffset
0000000000d22348 B gHotSpotVMStructEntryIsStaticOffset
0000000000d22340 B gHotSpotVMStructEntryOffsetOffset
0000000000d22360 B gHotSpotVMStructEntryTypeNameOffset
0000000000d22350 B gHotSpotVMStructEntryTypeStringOffset
0000000000ce4578 D gHotSpotVMStructs
0000000000d222f8 B gHotSpotVMTypeEntryArrayStride
0000000000d22310 B gHotSpotVMTypeEntryIsIntegerTypeOffset
0000000000d22318 B gHotSpotVMTypeEntryIsOopTypeOffset
0000000000d22308 B gHotSpotVMTypeEntryIsUnsignedOffset
0000000000d22300 B gHotSpotVMTypeEntrySizeOffset
0000000000d22320 B gHotSpotVMTypeEntrySuperclassNameOffset
```

```
0000000000d22328 B gHotSpotVMTypeEntryTypeNameOffset
0000000000ce4570 D gHotSpotVMTypes
```

与此同时，我们可以使用`ClassLoader`的`findNative`方法来定位java native的原生句柄，也就是可以用它来查找Java进程中的任意本机符号，包括gHotSpotXXX这种

例如下面这段来测试一下gHotSpotVMStructs（它对外暴露了JVM内部的大量信息，像原始的堆的地址，线程、栈的地址等。）

```
public class TestVMStructs {
    public static void main(String[] args) {
        Long vmStructs = BytecodeVerifierNoper.findNative("gHotSpotVMStructs", n
        Unsafe unsafe = BytecodeVerifierNoper.getUnsafe();
        System.out.printf(Long.toHexString(vmStructs)+
                ", value: "+ Long.toHexString(unsafe.getLong(vmStructs)));
    }
}
```

多运行几次，发现最后三位总是不变，也就是说HotSpot结构在内存中的实际地址为基地址+libjvm.so中的地址

```
7fcb1258dfc0, value: 7fcb1257d440
7f6245867fc0, value: 7f6245857440
7f041ca03fc0, value: 7f041c9f3440
7f0c4ca66fc0, value: 7f0c4ca56440
7f219f4f8fc0, value: 7f219f4e8440
```

如果你研究过**sa-jdi.jar**(提供调试接口)的原理，你就会发现逻辑其实和它有些相似

所以我们现在的思路：

- 遍历gHotSpotVMStructs来获取所有的VMStruct
- 遍历所有的VMStruct来获取所有的VMType
- 最后从所有的JVMType中获取所有的JVMFlag，如果是BytecodeVerificationLocal或BytecodeVerificationRemote，我们就把它置为false（0）

代码如下：

```
package org.vidar;

import org.vidar.entity.Fld;
```

```java
import org.vidar.entity.JVMFlag;
import org.vidar.entity.JVMStruct;
import org.vidar.entity.JVMType;
import sun.misc.Unsafe;

import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class BytecodeVerifierNoper {
    private static Unsafe unsafe = getUnsafe();
    private static Method findNativeMethod = getFindNativeMethod();

    public static void nop() {
        Map<String, JVMStruct> structs = getStructs();
        System.out.println("structs size:" + structs.size());
        Map<String, JVMType> types = getTypes(structs);
        System.out.println("types size:" + types.size());
        List<JVMFlag> flags = getFlags(types);
        for (JVMFlag flag : flags) {
            if (flag.getName().equals("BytecodeVerificationLocal")
                    || flag.getName().equals("BytecodeVerificationRemote"))  {
                unsafe.putByte(flag.getAddress(), (byte) 0);
            }
        }
    }

    public static List<JVMFlag> getFlags(Map<String, JVMType> types) {
        List<JVMFlag> jvmFlags = new ArrayList<>();

        JVMType flagType = types.get("Flag");
        if (flagType == null) {
            flagType = types.get("JVMFlag");
            if (flagType == null) {
                throw new RuntimeException("Could not resolve type 'Flag'");
            }
        }

        Fld flagsField = flagType.getFields().get("flags");
        if (flagsField == null) {
            throw new RuntimeException("Could not resolve field 'Flag.flags'");
        }
        long flags = unsafe.getAddress(flagsField.getOffset());
```

```
51
52          Fld numFlagsField = flagType.getFields().get("numFlags");
53          if (numFlagsField == null) {
54              throw new RuntimeException("Could not resolve field 'Flag.numFlags'"
55          }
56          int numFlags = unsafe.getInt(numFlagsField.getOffset());
57
58          Fld nameField = flagType.getFields().get("_name");
59          if (nameField == null) {
60              throw new RuntimeException("Could not resolve field 'Flag._name'");
61          }
62
63          Fld addrField = flagType.getFields().get("_addr");
64          if (addrField == null) {
65              throw new RuntimeException("Could not resolve field 'Flag._addr'");
66          }
67
68          for (int i = 0; i < numFlags; i++) {
69              long flagAddress = flags + (i * flagType.getSize());
70              long flagNameAddress = unsafe.getAddress(flagAddress + nameField.get
71              long flagValueAddress = unsafe.getAddress(flagAddress + addrField.ge
72
73              String flagName = getString(flagNameAddress);
74              if (flagName != null) {
75                  JVMFlag flag = new JVMFlag(flagName, flagValueAddress);
76                  jvmFlags.add(flag);
77              }
78          }
79
80          return jvmFlags;
81      }
82
83
84
85      public static Map<String, JVMType> getTypes(Map<String, JVMStruct> structs)
86          Map<String, JVMType> types = new HashMap<>();
87
88          long entry = symbol("gHotSpotVMTypes");
89          long arrayStride = symbol("gHotSpotVMTypeEntryArrayStride");
90
91          while (true) {
92              String typeName = derefReadString(entry + offsetTypeSymbol("TypeName
93              if (typeName == null) {
94                  break;
95              }
96
97              String superClassName = derefReadString(entry + offsetTypeSymbol("Su
```

```
 98
 99            int size = unsafe.getInt(entry + offsetTypeSymbol("Size"));
100            boolean oop = unsafe.getInt(entry + offsetTypeSymbol("IsOopType")) !
101            boolean intType = unsafe.getInt(entry + offsetTypeSymbol("IsIntegerT
102            boolean unsigned = unsafe.getInt(entry + offsetTypeSymbol("IsUnsigne
103
104            Map<String, Fld> structFields = null;
105            JVMStruct struct = structs.get(typeName);
106            if (struct != null) {
107                structFields = struct.getFields();
108            }
109
110            JVMType jvmType = new JVMType(typeName, superClassName, size, oop, i
111            if (structFields != null) {
112                jvmType.getFields().putAll(structFields);
113            }
114
115            types.put(typeName, jvmType);
116
117            entry += arrayStride;
118        }
119
120        return types;
121    }
122
123
124    public static Map<String, JVMStruct> getStructs() {
125        Map<String, JVMStruct> structs = new HashMap<>();
126
127        long currentEntry = symbol("gHotSpotVMStructs");
128        long arrayStride = symbol("gHotSpotVMStructEntryArrayStride");
129
130        while (true) {
131            String typeName = derefReadString(currentEntry + offsetStructSymbol(
132            String fieldName = derefReadString(currentEntry + offsetStructSymbol
133            if (typeName == null || fieldName == null) {
134                break;
135            }
136
137            String typeString = derefReadString(currentEntry + offsetStructSymbo
138            boolean staticField = unsafe.getInt(currentEntry + offsetStructSymbo
139
140            long offsetOffset = staticField ? offsetStructSymbol("Address") : of
141            long offset = unsafe.getLong(currentEntry + offsetOffset);
142
143            JVMStruct struct = structs.computeIfAbsent(typeName, JVMStruct::new)
144            struct.setField(fieldName, new Fld(fieldName, typeString, offset, st
```

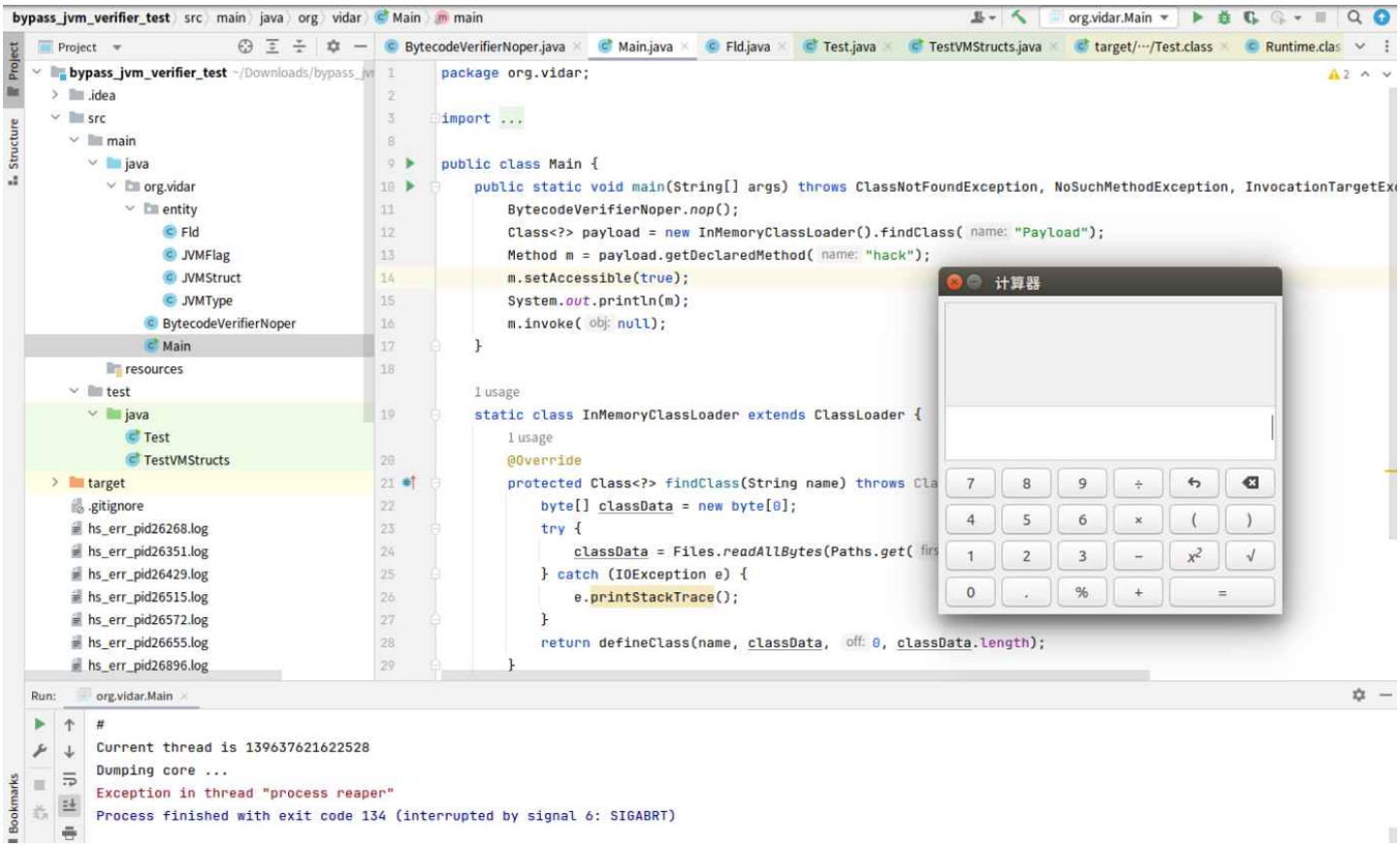
```
145
146             currentEntry += arrayStride;
147         }
148
149         return structs;
150     }
151
152     public static long symbol(String name) {
153         return unsafe.getLong(findNative(name,null));
154     }
155
156     public static long offsetStructSymbol(String name) {
157         return symbol("gHotSpotVMStructEntry" + name + "Offset");
158     }
159
160     public static long offsetTypeSymbol(String name) {
161         return symbol("gHotSpotVMTypeEntry" + name + "Offset");
162     }
163
164     public static String derefReadString(long addr) {
165         return getString(unsafe.getLong(addr));
166     }
167
168     public static String getString(long addr) {
169         if (addr == 0L) {
170             return null;
171         }
172         StringBuilder stringBuilder = new StringBuilder();
173         int offset = 0;
174
175         while (true) {
176             byte b = unsafe.getByte(addr + offset);
177             char ch = (char) b;
178             if (ch == '\u0000') {
179                 break;
180             }
181             stringBuilder.append(ch);
182             offset++;
183         }
184         return stringBuilder.toString();
185     }
186
187     public static Long findNative(String name,ClassLoader classLoader) {
188         try {
189             return (Long) findNativeMethod.invoke(null,classLoader,name);
190         } catch (IllegalAccessException e) {
191             throw new RuntimeException(e);
```

```
192         } catch (InvocationTargetException e) {
193             throw new RuntimeException(e);
194         }
195     }
196 
197 
198     private static Method getFindNativeMethod() {
199         try {
200             Method findNative = ClassLoader.class.getDeclaredMethod("findNative"
201             findNative.setAccessible(true);
202             return findNative;
203         } catch (NoSuchMethodException e) {
204             throw new RuntimeException(e);
205         }
206     }
207 
208 
209     public static Unsafe getUnsafe() {
210         try {
211             Constructor constructor = Unsafe.class.getDeclaredConstructor();
212             constructor.setAccessible(true);
213             return (Unsafe) constructor.newInstance();
214         } catch (NoSuchMethodException e) {
215             throw new RuntimeException(e);
216         } catch (InstantiationException e) {
217             throw new RuntimeException(e);
218         } catch (IllegalAccessException e) {
219             throw new RuntimeException(e);
220         } catch (InvocationTargetException e) {
221             throw new RuntimeException(e);
222         }
223     }
224 }
225 
```

具体代码可见
https://github.com/luelueking/Bypass_JVM_Verifier/tree/main/bypass_jvm_verifier_test/src/main/java/org/vidar

效果

- 在运行/加载非法class之前，我们先使用BytecodeVerifierNoper的nop函数，来使jvm不会对class进行检查
- 然后运行、加载我们的class



> 操作环境：ubuntu16，x86 jdk8

- 下面该到玩jvm指令的时间🐶了

参考

https://som.codes/blog/2019-12-30/jvm-hackery-noverify/

https://zhuanlan.zhihu.com/p/451838451

https://juejin.cn/post/6992108216695930917