

漏洞原理

`<s:url>` 或 `<s:a>` 接收用户输入调用 `Request.getParameterMap` 直接获取没做过滤处理回显至当前界面造成XSS

漏洞版本

Struts 2.0.0 ~ Struts 2.1.8.1

漏洞详解

URLTag将url标签转换为对象

当处理 `<s:url>` 标签时调用 `class org.apache.struts2.views.jsp.URLTag` 将标签相关属性通过 `set` 方法置于URLTag对象中并返回。

```

65 public Tag get(Class<? extends Tag> handlerClass) throws JspException {
66     synchronized(this) {
67         if (this.current >= 0) {
68             Tag handler = this.handlers[this.current--];
69             return handler;
70         }
71     }
72
73     try {
74         if (Constants.USE_INSTANCE_MANAGER_FOR_TAGS) {
75             return (Tag)this.instanceManager.newInstance(handlerClass.getName(), handlerClass.getClassLoader());
76         } else {
77             Tag instance = (Tag)handlerClass.getConstructor().newInstance();
78             this.instanceManager.newInstance(instance);
79             return instance;
80         }
81     } catch (Exception var5) {
82         Throwable t = ExceptionUtils.unwrapInvocationTargetException(var5);
83         ExceptionUtils.handleThrowable(t);
84         throw new JspException(var5.getMessage(), t);
85     }
}

```

doStartTag构建标签组件并解析相关属性

初始化 URLTag 对象后，开始进行标签解析执行 ComponentTagSupport::doStartTag() 方法.该方法做了以下事情，

- 1.引入当前数据栈与上下文载入 getBean 方法中，会进一步调用 URLTag::getBean 方法返回 org.apache.struts2.components.URL 对象
2. container.inject(this.component); 将当前标签构建注入到新的容器中方便管理
- 3.通过 this.populateParams(); 进一步调用URLTag::populateParams()方法,完善 URLTag::getBean 返回的 URL 对象
- 4.调用URL::start方法解析属性

```

30 public int doStartTag() throws JspException {
31     this.component = this.getBean(this.getStack(), (HttpServletRequest)this.pageContext.getRequest(), (HttpServletResponse)this.pageContext.getResponse());
32     Container container = Dispatcher.getInstance().getContainer();
33     container.inject(this.component);
34     this.populateParams();
35     boolean evalBody = this.component.start(this.pageContext.getOut());
36     if (evalBody) {
37         return this.component.usesBody() ? 2 : 1;
38     } else {
39         return 0;
40     }
41 }

```

跟进URL::start方法，主要是通过开发人员设置的includeParams值存入相关输入到this.parameter字段中。当设置 includeParams="all" 进入相应的逻辑处理。

1. this.mergeRequestParameters 利用 Request::getParameterMap 获取未经过编码后的输入

2. this.includeGetParameters 利用 Request::getQueryString 获取URL编码后的用户输入

(前两者使用差异参考：<https://stackoverflow.com/questions/29299314/jsp-getquerystring-and-getparametermap-returning-different-parameters>)

3. this.includeExtraParameters 没做特殊处理一般为null

综上最直观造成xss是由 Request::getParameterMap 获取用户输入并为做过滤，如果置 includeParams='get' 只能调用 Request::getQueryString 返回用户输入经过URL编码。

```
88 } else if ("all".equalsIgnoreCase(includeParams)) { includeParams: "all"
89     this.mergeRequestParameters(this.value, this.parameters, this.req.getParameterMap()); value: null req: StrutsRequestWrapper@4
90     this.includeGetParameters();
91     this.includeExtraParameters();
92 } else if (!"get".equalsIgnoreCase(includeParams) && (includeParams != null || this.value != null || this.action != null)) {
93     if (includeParams != null) {
94         LOG.warn("Unknown value for includeParams parameter to URL test: " + includeParams);
95     }
96 } else {
97     this.includeGetParameters();
98     this.includeExtraParameters();
99 }
100 } catch (Exception var4) {
101     LOG.warn("message: \"Unable to put request parameters (\" + this.req.getQueryString() + \") into parameter map.\", var4);
102 }
103
104 return result;
105 }
106
107 private void includeExtraParameters() {
```

Variables

- this = {URL@4445}
- writer = {JspWriterImpl@3746}
- result = true
- includeParams = {String@4408} "all"
- this.includeParams = {String@4408} "all"
- this.value = null
- this.req = {StrutsRequestWrapper@4448}
- this.parameters = {LinkedHashMap@4452} size = 2
 - {String@4458} "<script>alert(1);</script>" -> {String@1@4459}
 - {String@4466} "%3Cscript%3Ealert(1);%3C/script%3E" -> {String@4467} "123"
- this.action = null

doEndTag处理输出内容并写入当前jsp页面中

doEndTag 更直观是对最后的输出做处理

```
24 public int doEndTag() throws JspException {
25     this.component.end(this.pageContext.getOut(), this.getBody()); component
26     this.component = null;
27     return 6;
28 }
```

跟进 `UrlHelper::buildUrl` 方法，对最后输出result进行拼接，依次拼接当前访问的WEB路径、this.paramters中的两个值。

```
116 buildParametersString(params, link); params: size = 2
117
118 try {
119     result = encodeResult ? response.encodeURL(link.toString()) : link.toString(); encodeResult: true response: ResponseFacade@4450
120 } catch (Exception var12) {
121     result = link.toString(); link: "/Struts2_002_war_exploded/?<script>alert(1);</script>=123&amp;%3Cscript%3Ealert(1);%3C/script%3E=123"
122 }
123
124 return result; result: "/Struts2_002_war_exploded/?<script>alert(1);</script>=123&amp;%3Cscript%3Ealert(1);%3C/script%3E=123"
125
126
```

`URL::end` 最后收尾没有调用模版写入，而是直接调用write写入当前jsp页面中。

```
167 id = this.getId();
168 if (id != null) {
169     this.getStack().getContext().put(id, result);
170     this.req.setAttribute(id, result); req: StrutsRequestWrapper@4448 id: null
171 } else {
172     try {
173         writer.write(result); result: "/Struts2_002_war_exploded/?<script>alert(1);</script>=123&amp;%3Cscript%3Ealert(1);%3C/script%3E=123"
174     } catch (IOException var7) {
175         throw new StrutsException("IOError: " + var7.getMessage(), var7);
176     }
177 }
178
179 return super.end(writer, body);
```

漏洞复现

http://localhost:8085/Struts2_002_war_exploded/?
%3Cscript%3Ealert(1);%3C/script%3E

localhost:8085/Struts2_002_war_exploded/?<script>alert(1);</script>

localhost:8085 显示

1

确定

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html data-blockbyte-bs-uid="82081">
  <head>...</head>
  <body data-ext-version="3.1">
    <h2>S2-001 Demo</h2>
    <p>...</p>
    "
    /Struts2_002_war_exploded/?"
    <script>alert(1);</script>
    "=&%3Cscript%3Ealert(1);%3C/script%3E=
    " == $0
    <iframe id="blockbyte-bs-sidebar" class="nottranslate" aria-hidden="true" data-pos="left">...</iframe>
    <div id="blockbyte-bs-indicator" class="blockbyte-bs-fullHeight" style="width: 2px; height: 100%; top: 0
  </body>
```

漏洞修复

比较懒,只是单纯的替换了script字符串很容易绕过.

参考: <https://dean2021.github.io/posts/s2-002/>