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。

```java
public Tag get(Class<? extends Tag> handlerClass) throws JspException {
    synchronized(this) {
        if (this.current >= 0) {
            Tag handler = this.handlers[this.current--];
            return handler;
        }
    }

    try {
        if (Constants.USE_INSTANCE_MANAGER_FOR_TAGS) {
            return (Tag)this.instanceManager.newInstance(handlerClass.getName(), handlerClass.getClassLoader());
        } else {
            Tag instance = (Tag)handlerClass.getConstructor().newInstance();
            this.instanceManager.newInstance(instance);
            return instance;
        }
    } catch (Exception var5) {
        Throwable t = ExceptionUtils.unwrapInvocationTargetException(var5);
        ExceptionUtils.handleThrowable(t);
        throw new JspException(var5.getMessage(), t);
    }
}
```

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

```java
public int doStartTag() throws JspException {
    this.component = this.getBean(this.getStack(), (HttpServletRequest)this.pageContext.getRequest(), (HttpServletResponse)this.pageContext.g
    Container container = Dispatcher.getInstance().getContainer();
    container.inject(this.component);
    this.populateParams();
    boolean evalBody = this.component.start(this.pageContext.getOut());
    if (evalBody) {
        return this.component.usesBody() ? 2 : 1;
    } else {
        return 0;
    }
}
```

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.

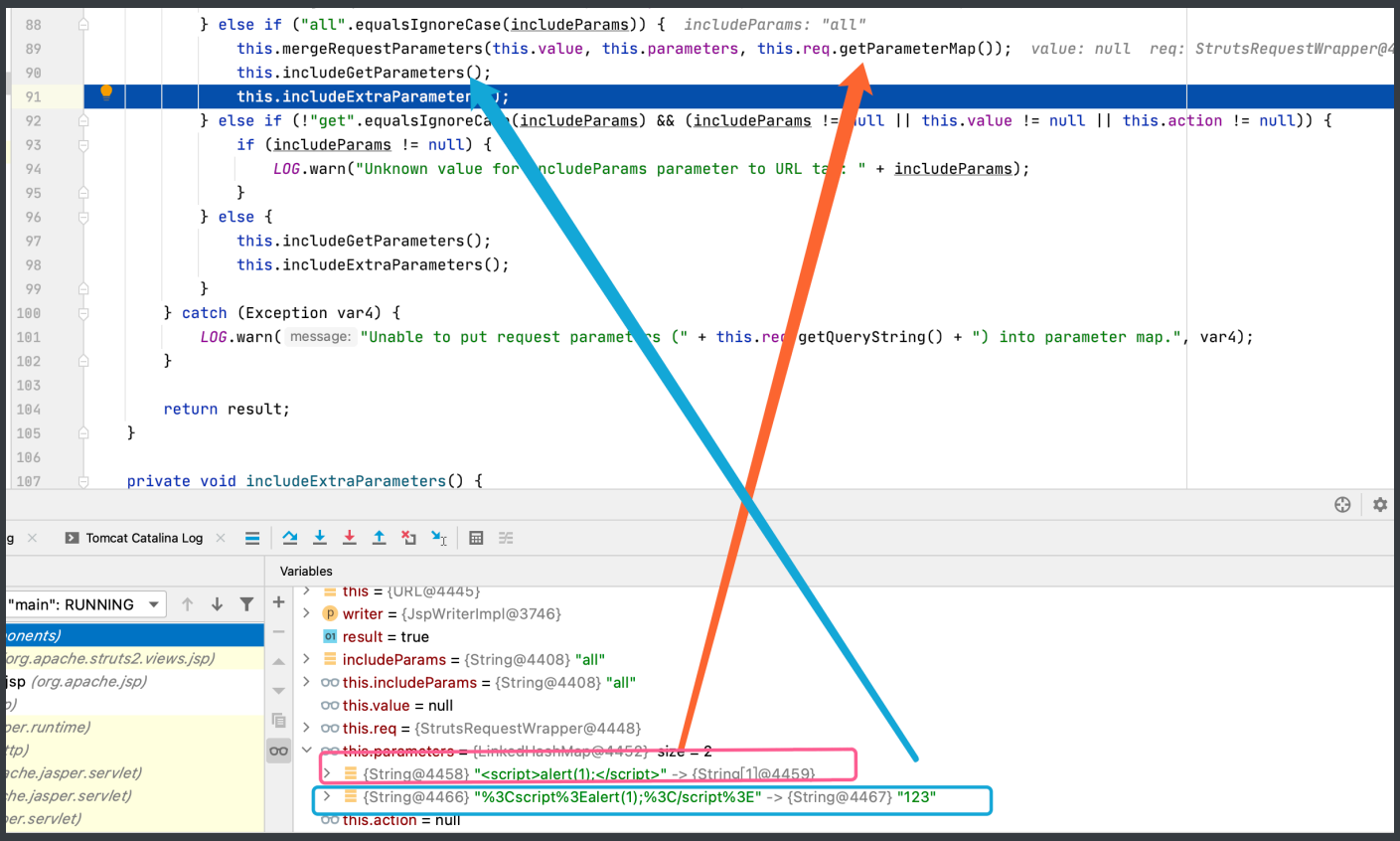
1. `this.mergeRequestParamters` 利用 `Rquest::getParameterMap` 获取未经过编码后的输入

2. `this.includeGetParamters` 利用 `Rquest::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 `Rquest::getParameterMap` 获取用户输入并为做过滤，如果置 `includeParams='get'` 只能调用 `Rquest::getQueryString` 返回用户输入经过URL编码.



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。

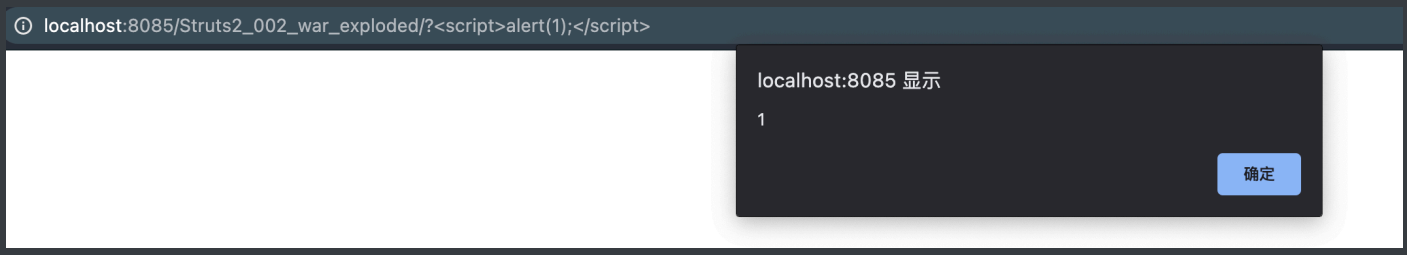
```
116   buildParametersString(params, link);  params:  size = 2
117
118   try {
119       result = encodeResult ? response.encodeURL(link.toString()) : link.toString();  encodeResult: true  response: ResponseFacade@4450
120   } catch (Exception var12) {
121       result = link.toString();  link: "/Struts2_002_war_exploded/?<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
```

`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%3E
174           } catch (IOException var7) {
175               throw new StrutsException("IOError: " + var7.getMessage(), var7);
176           }
177       }
178
```

漏洞复现

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



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/