XSS Defender 用户手册

应方明 (yingfm@corp.netease.com)

概述

在这份文档中，介绍在项目中使用 XSS Defender 需要的准备工作；对 XSS Defender 配置原理和各个配置项的用途进行简单的说明，并演示了如何配置不同过滤器实例来满足特定需求；最后，介绍在了 XSS Defender 的主要功能扩展点，当默认功能无法满足特定需求时，用户可以自己对过滤器进行扩展。

快速入门

在工程中引入 XSS Defender

项目使用 Maven 管理依赖

如果项目使用 Maven 管理依赖关系，只需要编辑 pom.xml 文件，在依赖中加入 XSS Defender 即可，xssdefender-1.3.4.jar 已经提交到公司仓库服务器上 (http://mvn.hz.netease.com/artifactory/repo)

```
<project xmlns=http://maven.apache.org/POM/4.0.0
    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
          http://maven.apache.org/maven- v4_0_0.xsd">
    ….
    <dependencies>
          …
    <dependency>
            <groupId>com.netease.security</groupId>
            <artifactId>xssdefender</artifactId>
        <version>1.3.4</version>
        <type>jar</type>
       <scope>compile</scope>
```

```
    </dependency>
      …
      </dependencies>
</project>
```

直接引用 Jar 包

如果项目没有使用 Maven 来管理依赖，需要把 xssdefender-1.3.4.jar 及依赖的 jar 包复制到项目的 lib 目录中，并加入到 classpath，就可以在项目中使用 XSS Defender 提供的相关功能

应用

首先需要得到过滤器实例，主要有两种方式：

1. 使用 XSSFilter 的构造函数，创建一个新的过滤器实例

```
XSSFilter filter = new XSSFilter();

XSSFilter filter = new XSSFilter(option_file, name);
```

2. 使用包装类 XSSFilterWrapper 中提供的静态方法 getFilter 取得指定配制的过滤器对象，其中特定配制的过滤器只维持一个实例

```
XSSFilter filter = XSSFilterWrapper.getFilter(option_file, name);
```

其中，***option_file*** 为用户指定的配制文件，其中包含了各个不同过滤器实例的附加配制(具体见后面配制部分说明)，既可以指定具体的文件路径，如/filters.properties，也可以在 classpath 中查找，如 classpath:filters.properties；***name*** 为用户配制中指定的特定过滤器对象。当使用默认构造函数，或用户配制及过滤器对象不存在时，将返回一个使用基础配制的过滤器实例

得到过滤器实例后，可以对 HTML 文本进行过滤，可以得到保存过滤信息的 FilterResult 对象：

```
public FilterResult filter(String html, int type, long timeout)
            throws InterruptedException,      TimeoutException
public FilterResult filter(String html, int type)    throws InterruptedException, TimeoutException
```

如只需要得到过滤后的文本，可使用接口：

```
public String getFilteredHTML(String html, int type)
        throws InterruptedException, TimeoutException
public String getFilteredHTML(String html, int type, long timeout)
    throws InterruptedException, TimeoutException
public String getFilteredHTML(String html) throws InterruptedException, TimeoutException
```

其中，***content*** 为需要过滤的 HTML 内容，***type*** 表示 HTML 代码有应用中的作用，主要类型有：

- *XSSFilter.TYPE_NORMAL_HTML*：普通的 HTML，这个是默认类型
- *XSSFilter.TYPE_ATTRIBUTE_URL*：作为一个 url 属性填充到页面中，如 img 标签的 src 地址，a 标签的 href 链接等
- *XSSFilter.TYPE_ATTRIBUTE_STYLE*：作为标签的样式填充到页面中
- *XSSFilter.TYPE_ATTRIBUTE_OTHER*：其他属性值
- *XSSFilter.TYPE_PLAIN_TEXT*：作为纯文本信息显填充到页面中

过滤过程中可能会抛出两个异常：***InterruptedException*** 和 ***TiemoutException***，表示执行行过滤的线程被中断及过滤逻辑无法在指定的时间内完成，上层逻辑需要对异常进行处理：使用更长的超时时间等

配制说明

在 XSS Defender 中，配制包含两个部分：1). 由 Jar 包本身提供的基础配制信息， 在 classpath 中的 base-config.properties 文件中；2). 需要用户提供的自定义配制，可以根据需求在配制中对基础配制进行补充、修改、替换、合并等操作，得到不同的针对特定需求的过滤器实例

基础配制(base-config.properties)

Base-config.properties 文件中定义了默认情况下的过滤器配制信息，其中配制主要有：

- `tagHandler`：`String` 类型，指定对嫌疑标签和属性进行处理的实现类
- `alarm`：`String` 类型，指定的报警接口的实现类

- *tagsWhitelist*: Set 类型，如果这个配制不为空，将使用白名单方式对标签进行过滤，只保留这里定义的标签，置空则表示不使用标签白名单过滤方式
- *removeNodeTags*: Set 类型，如匹配到，则去除标签及标签里面的内容
- *removeTagOnlyTags*: Set 类型，如匹配到，则去除标签本身，但保留标签内的内容
- *nodeAttrBlacklist*: Map 类型，定义具体标签需要过滤的属性，格式为 map|tag1:attr1,attr2;tag2:attr1,attr2..的形式
- *nodeAttrWhitelist*: 定义具体标签中需要保留的属性，格式同标签属性黑名单
- *keywordsCheckedAttributes*: Set 类型，需要在值中检查关键词的属性
- *scriptKeywords*: Set 类型，属性值中需要检查的关键词，注：只检查去除空格、大小写等干扰后的属性值是否会以这些词开始
- *allowedStyleProps*: Set 类型，style 中允许出现的 CSS 样式，如需要支持所有的样式，则只能设一个值：all
- *urlValidators*: Map 类型，定义了具体标签对应的正则表达式，用于验证 url 是否有效，default 支持所有标签，格式：
  map|default:reg1,reg2..;tag1:reg1,reg2… 如这个设置为空，则只检查是否以 javascript 等关键开始
- *forbidAttributes*: Set 类型，定义了全局范围需要过滤的属性，与具体标签无关，这个配制支持正则表达(/regex/)和具体的标签名
- *customTagCheckers*: Map 类型，用户自定义的针对具体标签的检查类，格式：map|tag1:class1,class2…;tag2:class1…

其中，每个配制的值由部分组成：type|value，type 表示数据结构，现在使用的有 Map、Set、String 这三类，具体配制，请参照 base-config.properties 文件

```
tagHandler=string|com.netease.security.xssdefender.filter.NodeFilter
tagsWhitelist=set|
alarm=string|
removeNodeTags=set|head,script,style,object,applet,noscript,frameset,noframes
removeTagOnlyTags=set|form,meta,body,html,label,select,optgroup,option,textarea
,title,script,xmp,applet,embed,head,frameset,iframe,noframes,noscript,object,st
yle,input,base,basefont,isindex,link,frame,param,xml,xss
```

```
nodeAttrBlacklist=map|
nodeAttrWhitelist=map|img:src,alt,width,height;a:href,target,class;
keywordsCheckedAttributes=set|background,
scriptKeywords=set|javascript,vbscript,script,actionscript
allowedStyleProps=set|font,font-size,font-weight,font-style,text-decoration,wid
th,height,border,margin,padding
urlValidators=map|default:^(https?://|/|#|mailto\\s*:).*
forbidAttributes=set|/on[a-zA-Z]+/,allowScriptAccess,allowNetworking,disabled
```

附加配制

附加配制使用户能够在默认配制的基础上对各个过滤器的配制进行调整，使得这些过滤来满足不同的实际应用，配制的格式为：

```
name.option.type=value
```

其中，name 为过滤器实例的名字，option 同基础配制中的属性名一致，type 表示附加配制与基础配制如何起作用，现有的类型有：

- add: 将附加配制和基础配制的值合并起来
- subtract: 从基础配制中减掉附加配制定义的值
- replace: 用附加配制中定义的值替换基础配制中同名的设置

具体可参考为博客提供的附加配制：filters.properties 文件，

```
blog.removeNodeTags.subtract=set|style,embed
blog.removeTagOnlyTags.subtract=set|embed,style
blog.nodeAttrBlacklist.replace=map|
blog.nodeAttrWhitelist.replace=map|
blog.allowedStyleProps.replace=set|all
blog.customTagCheckers.add=map|embed:com.netease.security.xssdefender.ext.blog.
EmbedTagChecker;style:com.netease.security.xssdefender.ext.blog.StyleTagChecker
;
comment.tagsWhitelist.replace=set|br,span,img,strong,em,u,p,a,blockquote,div
comment.allowedStyleProps.replace=set|font-weight,font-style,text-decoration
comment.nodeAttrWhitelist.replace=map|img:src;a:href,target,class,commentUserId
;span:style,href,target,class;blockquote:class;strong:;u:;p:;em:;
```

```
comment.urlValidators.add=map|img:^http://b\.bst\.126\.net/style/common/.*,http
://b.bst.126.net/common/portrait/.*;a:^https?://[\\w-_]+\.163\.com/?.*;
```

补充：配制中，如果表示的是集合，各个元素之间以,分隔；如果表示的是个 Map，每个元素之间以;分隔，其中每个元素中 key:value 以:分隔（以第一个:为准，后面再出现:将被忽略），最后的;最好不要省略

功能扩展

配制管理

对应基础配制和附加配制的设计，XSS Defender 中用于存储配制信息的类有三个：

- Configuration：完整的配制信息，其中属性与配制项同名，类型根据不同配制为 Set 或 Map
- BaseConfigObject：对应基础配制信息，其中属性与配制项同名，类型为 String
- AdditionalConfigObject：对应附加配制信息，其中属性有两种：与配制同名的 String 类型；配制名+Op 结尾表示配制类型的 int 型属性

现在支持的配制源为 Java property 文件，如果用户需要其他配制源，可以实现 ConfigurationLoader 接口

```
/**
 * 配制载入接口
 *
 * @author superekcah
 */
public interface ConfigurationLoader {
    /**
     * 载入基础配制信息
     *
     * @param file
     * @return
     */
    public BaseConfigObject loadBaseConfiguration(String file);
```

```
    /**
     * 读取附加配制信息
     *
     * @param file
     * @param name
     * @return
     */
    public AdditionalConfigObject loadAdditionalConfiguration(String file, String
name);


}
```

并注册到 ConfigurationManager 中

```
ConfigurationManager.registerNewConfigLoader(String suffix, ConfigurationLoader    loader);
```

其中，suffix 是文件的扩展 (file 最后一个.之后的内容，ConfigurationManager 用这个来查找匹配的配制载入方法，实际上，file 不一定是个真实的文件，用户可以根据需求采取不同的实现)。

扩展 HTML 文本处理的功能

如果用户需要在语法检查之前对 HTML 内容作一些文本上的操作，如编码转换等，可以实现 LiteralProcessor 接口

```
/**
 * 对HTML进行基于文本的处理
 *
 * @author superekcah
 */
public interface LiteralProcessor {


    /**
     * 得到这个处理过程的名字
     *
     * @return
     */
    public String getName();
```

```
    /**
     * 对传入的HTML内容进行处理，并返回处理后的结果
     *
     * @param meta
     * @return
     */
    public void doProcess(HTMLMeta meta);
}
```

并调用 LiterralProcessController 的 addProcessor(LiteralProcessor processor)方法插入到文本处理队列中

扩展基于语法分析功能

如果已经实现的 HTML 语法分析不能满足具体的应用需求，用户可以对语法分析功能进行扩展，主要方式有以下三种：

1. 扩展 CustomTagChecker 基类，重写 modify(StartTag tag)方式，实现针对重要标签的处理，然后通过配制中的 customTagCheckers 在运行动态载入。如果只有少数标签需要额外处理，推荐使用这种方式，注意：用户自定义标签检查会在最后做，因此，这个标签可能被前面标签白名单、去除标签及内容、去除标签但保留内容等步骤过滤掉而不会执行自定义标签处理过程，如果要求第一步执行自定义标签处理，参考下面的扩展方法。
2. 实现接口 TagHandler 或扩展类 NodeFilter，其中 handleAllTag 方法会被所有的标签调用，在里面可以实现用户定义的处理过程，具体参照代码或 Java Doc
3. 如扩展文本处理过程一样，用户可以实现 SemanticProcessor 接口编写自己的语法分析逻辑，并通过调用 SemanticProcessController.addProcessor(SemanticProcessor processor)方法插入到语法分析流程中