AntiSamy User Guide

This document was prepared by Julian Cohen and generously donated to the OWASP AntiSamy project. It introduces the project to unfamiliar users and describes how to get, configure and use AntiSamy in your application.

May 11, 20100

1. Introduction

AntiSamy is an HTML, CSS and JavaScript filter for Java that sanitizes user input based on a policy file. AntiSamy is **not** an HTML, CSS and JavaScript validator. It is merely a way to make sure HTML, CSS and JavaScript input strictly follows rules defined by a policy file. The project's main site and repository are shown here:

http://www.owasp.org/index.php/Category:OWASP_AntiSamy_Project
http://code.google.com/p/owaspantisamy/

AntiSamy does a very good job of removing malicious HTML, CSS and JavaScript, but in security, no solution is guaranteed. AntiSamy's success depends on the strictness of your policy file and the predictability of browsers' behavior. AntiSamy is a sanitization framework; it is up to the user how it does its sanitization. Using AntiSamy does not guarantee filtering of all malicious code. AntiSamy simply does what is described by the policy file.

AntiSamy is simple from an external view. AntiSamy's function is based on an XML policy file. The policy file is read in to create a Policy object that is used by AntiSamy. The Policy object is then passed to AntiSamy to be used in the parsing and filtering of input. AntiSamy uses the third party libraries Xerces, Batik and NekoHTML to help facilitate parsing and filtering of input.

2. Setting Up

Download the current AntiSamy jar file from http://code.google.com/p/owaspantisamy/downloads/list or from Maven's central repo (under org.owasp.antisamy). These are the dependencies that you will need to satisfy. These are all also available in Maven's central repo, or can be downloaded from the locations specified:

xercesImpl.jar http://xerces.apache.org/mirrors.cgi#binary
batik.jar http://xmlgraphics.apache.org/batik/download.cgi
nekohtml.jar http://sourceforge.net/projects/nekohtml/

Once all these jars are in your classpath, you're ready to start using AntiSamy. A code snippet invoking AntiSamy is broken down in the next section.

```
import org.owasp.validator.html.*;  // Import AntiSamy
```

Here, all necessary AntiSamy objects are imported to be used.

```
String POLICY_FILE_LOCATION = "antisamy-1.4.1.xml";  // Path to policy file
String dirtyInput = "<div><script>alert(1);</script></div>";  // Some fake input
```

In this step, the path to the XML policy file was declared, and some fake, possibly malicious, data was stored as our user input.

```
Policy policy = Policy.getInstance(POLICY_FILE_LOCATION);  // Create Policy object
```

The Policy object was created and populated from the XML policy file in this line. Policy objects can also be read from InputStreams or File objects directly.

```
AntiSamy as = new AntiSamy();  // Create AntiSamy object
CleanResults cr = as.scan(dirtyInput, policy, AntiSamy.SAX);  // Scan dirtyInput
```

Here, an AntiSamy object was created and used to sanitize user input based on the Policy object with the SAX parser.

```
System.out.println(cr.getCleanHTML());  // Do something with your clean output!
```

The output from AntiSamy was written to the console. As is shown in the "Output", script tag was removed:

Input:

```
<div><script>alert(1);</script></div>
```

Output:

```
<div></div>
```

A large part of getting a Cross Site Scripting attack to execute is context. An attacker injecting JavaScript inside a tag would use a different payload than if they were injecting JavaScript inside an attribute. This is what is meant my "context." When AntiSamy filters your input, context plays a major role in if that filtered output is going to be malicious or not. To avoid anything exceedingly complicated, AntiSamy assumes that all filtered output will be placed in between a start and end tag.

Each line from this code snippet shows whether or not it's safe to place AntiSamy into a particular HTML context. If AntiSamy data is used in any other context than what's shown it's safe to assume that it's being used unsafely.

```
<div>[SAFE]</div>
<input type="text" value="[UNSAFE]">
<textarea>[UNSAFE]</textarea>
<script>var j = '[UNSAFE]';</script>
```

For more on this distinction, see the following OWASP AntiSamy mailing list threads [1] [2] and the OWASP XSS Prevention Cheat Sheet.

[1] https://lists.owasp.org/pipermail/owasp-antisamy/2009-October/000263.html
[2] https://lists.owasp.org/pipermail/owasp-antisamy/2011-May/000400.html
[3] https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

3. Policy Files

AntiSamy comes with several example policy files based on the function of real-world websites. You can use these as your policy files, or build upon them to create more specific policies.

When starting out with AntiSamy, you'll probably want to start off with a base policy file. However, sometimes it may be difficult to decide which policy file to start with. The following table breaks down the various default policy files that ship with AntiSamy:

Policy	Description
antisamy-tinymce.xml	Based on the HTML WYSIWYG editor, relatively safe. This policy file only allows text formatting, and may be a good choice if users are submitting HTML to be used in a blog post.
antisamy-anythinggoes.xml	A very dangerous policy file, this will allow all HTML, CSS and JavaScript. You shouldn't use this in production.
antisamy-ebay.xml	Based on the content filtering for the popular electronic auction website, relatively safe. This policy file gives the user a little bit of freedom, and may be a good choice if users are submitting HTML for a large portion of a page.
antisamy-myspace.xml	Based on the content filtering for the popular social networking site, relatively dangerous. This policy file gives the user a lot of freedom, and may be a good choice if users are submitting HTML for an entire page.
antisamy-slashdot.xml	Based on the comment filtering on the popular news site, but not quite as strict. This policy file only allows strict text formatting, and may be a good choice if users are submitting HTML in a comment thread.
antisamy.xml	Default policy file, fairly permissive. This policy file allows most HTML elements, and may be useful if users are submitting full HTML pages.

Configuring Your AntiSamy Policy

If a base policy file doesn't meet your needs, you can tweak your policy file to your needs. Each section of the policy file is broken down in the next few sections to give the developer background on tweaking the output.

Policy File Directives [<directives></directives>]

The following table shows the directives, their default values, and their impact on the AntiSamy filtering process.

Directive	Type	Default	Description
useXHTML	boolean	*false*	When this feature is on, AntiSamy will output the sanitized data in XHTML format as opposed to just regular HTML.
omitXMLDeclaration	boolean	*true*	When "omitXMLDeclaration" is turned on, AntiSamy will automatically prepend the XML header to output. Enabling this feature will tell AntiSamy not to do that.
omitDoctypeDeclaration	boolean	*true*	When this feature is enabled, AntiSamy will automatically prepend the HTML doctype declaration. By default, AntiSamy will not prepend this header.
formatOutput	boolean	*true*	When enabled, AntiSamy will automatically format the output according to some basic rules and indentation. Kind of like "pretty print."
maxInputSize	int	*100000*	This directive specifies the maximum size (in bytes) of user input before it's validated.
embedStyleSheets	boolean	*false*	When the developer chooses to allow CSS, this directive

			will specify whether or not remote stylesheets found referenced in the user's input will be pulled down and embedded into the current user input.
maxStyleSheetImports	int	*1*	This feature allows developers to specify how many remote stylesheets can be downloaded from any one input.
connectionTimeout	int	*1000*	When "embedStyleSheets" is enabled, this timeout value (in milliseconds) will be used when fetching the offsite resource in question. This should be used to prevent validation threads from blocking when connecting to 3rd party systems that may purposefully act really, really slowly.
preserveComments	boolean	*false*	When enabled, AntiSamy will keep HTML comments supplied in the input.
nofollowAnchors	boolean	*false*	When enabled, AntiSamy will append rel="nofollow" attributes to all anchor (<a>) tags supplied in the input. This is useful for telling search engines not to associate your site with sites that are under the control of your users.
validateParamAsEmbed	boolean	*false*	When enabled, AntiSamy will treat attributes of <embed> tags in the policy the same as any <param> tags nested inside the the <embed>. This allows users to, according to policy, pass in data in either of those two methods with equal security. This is needed for sites that allow users to supply videos, etc.
preserveSpace	boolean	*false*	When enabled, this feature is intended to preserve spaces as specified in the input without normalization. Right now it only works as according to http://xerces.apache.org/xerces-j/apiDocs/org/apache/xml/serialize/OutputFormat.html#setPreserveSpace%28boolean%29.
onUnknownTag	String	*remove*	When this directive is set to "encode," it will encode the tag using HTML entities, instead of removing the tag, which is the default behavior.

Common Regular Expressions [<common-regexps> </common-regexps>]

You can declare regular expressions here that can be used in the rest of the Policy File. Example:

```
<regexp name="htmlId" value="[a-zA-Z0-9\:\-_\.]+"/>
```
Line 51 in antisamy-1.4.1.xml

This regular expression is used to determine whether text in an `id` attribute is valid or not.

Common Attributes [<common-attributes> </common-attributes>]

Here you can declare attributes here that are common to many different tags. Example:

```
<attribute name="id" description="The 'id' of any HTML attribute should not contain
anything besides letters and numbers">
                  <regexp-list>
                        <regexp name="htmlId"/>
                  </regexp-list>
            </attribute>
```
Lines 145-149 in antisamy-1.4.1.xml

This is where the `id` attribute is mapped to the `htmlId` regular expression.

Global Tag Attributes [<global-tag-attributes> </global-tag-attributes>]

You can declare attributes here that are global to all different tags. Example:

```
<attribute name="id"/>
Line 496 in antisamy-1.4.1.xml
```

The `id` attribute is global to many different tags.

Tags to Encode [<tags-to-encode> </tags-to-encode>]

You can declare tags here that will not be removed, filtered, validated or truncated, but encoded using HTML entities. Example:

```
<tag>g</tag>
Line 505 in antisamy-1.4.1.xml
```

The `g` tag doesn't actually do anything, but it isn't malicious either so we can encode it, rather than remove it.

Tag Rules [<tag-rules> </tag-rules>]

You can define parsing rules here that will be used for each tag individually. The following section shows what happens to tags according to what actions AntiSamy has decided to perform on it. This will help understand how to build your own tag-rules rules.

Default: Behavior when the tag is not listed in <tag-rules>

Input:

```
<div><anewtag id="newtag" anewattrib="attrib">Text goes
<b>here</b>.</anewtag></div>
```

Output:

```
<div>Text goes <b>here</b>.</div>
```

remove: Behavior when the tag-rule action is set to "remove" for given tag. Tag is deleted with all of its child text.

Input:

```
<div><anewtag id="newtag" anewattrib="attrib">Text goes
<b>here</b>.</anewtag></div>
```

Output:

```
<div></div>
```

filter: Behavior when the tag-rule action is set to "filter" for given tag. Delete tag, but keep its child text.

Input:

```
<div><anewtag id="newtag" anewattrib="attrib">Text goes
<b>here</b>.</anewtag></div>
```

Output:

```
<div>Text goes <b>here</b>.</div>
```

validate: Behavior when the tag-rule action is set to "validate" for given tag. Verify that its attributes and children elements follow rules defined in policy file.

Input:

```
<div><anewtag id="newtag" anewattrib="attrib">Text goes
<b>here</b>.</anewtag></div>
```

Output:

```
<div><anewtag id="newtag">Text goes <b>here</b>.</anewtag></div>
```

truncate: Behavior when the tag-rule action is set to "validate" for given tag . Keep the element, but remove all attributes.

Input:

```
<div><anewtag id="newtag" anewattrib="attrib">Text goes
<b>here</b>.</anewtag></div>
```

Output:

```
<div><anewtag>Text goes <b>here</b>.</anewtag></div>
```

Example:

```
<tag name="html" action="validate"/>
Line 513 in antisamy-1.4.1.xml
```

The `html` tag is included in the clean output after it has been validated that it has been properly used.

CSS Rules [<css-rules> </css-rules>]

You can define parsing rules here that will be used for each CSS property individually. Only CSS defined in this section is allowed. Example:

```
<property name="width" default="auto" description="">
      <category-list>
            <category value="visual"/>
      </category-list>
      <literal-list>
            <literal value="auto"/>
            <literal value="inherit"/>
      </literal-list>
      <regexp-list>
            <regexp name="length"/>
            <regexp name="percentage"/>
      </regexp-list>
</property>
Lines 2096-2108 in antisamy-1.4.1.xml
```

The CSS `width` property is allowed only when it matches these rules when used. The value of the width element must be a length, percentage or one of the two literal values "auto" or "inherit".

4. References

The following references may be useful for further reading.

OWASP AntiSamy Project Page
http://www.owasp.org/index.php/AntiSamy
AntiSamy Policy File Directives Documentation (old)
http://www.owasp.org/index.php?title=AntiSamy_Directives&oldid=71778
AntiSamy OWASP Project Assessment Checklist
http://spreadsheets.google.com/ccc?key=pAX6n7m2zaTW-JtGBqixbTw
AntiSamy Mailing List Archives
https://lists.owasp.org/pipermail/owasp-antisamy/
XSS (Cross Site Scripting) Prevention Cheat Sheet
http://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet