XSS Cheat Sheet

CQ/GRANITE ENGINEERING

Philosophy

- Allow all input - Encode all output

Do not filter or encode input that gets stored but always protect the user on output.

- Encode at the very end

Encode the output-statement itself not intermediate values, so it is always obvious that an output statement is not dangerous, and you know you are encoding for the right context.

- Don’t think too much

Encode the content no matter where it is coming from. Your code might be copied or included, and the ACLs on the property might change.

- Never do it yourself

Never write the encoding/filtering methods yourself. XSS encoding is very difficult and error prone. If something is missing in the library, please file a bug.

- Prefer a validator to an encoder

Some situations, such as href and src attributes, MUST use a validator

HTL automatically filters and escapes all variables being output to the presentation layer to prevent cross-site-scripting (XSS) vulnerabilities.

https://docs.adobe.com/docs/en/htl/docs/expression-language.html#Display%20Context

How to get the XSSAPI Service?

Java component

```
import org.apache.sling.xss.XSSAPI;
@Reference
private XSSAPI xssAPI;
```

Java

```
import org.apache.sling.xss.XSSAPI;

public class MyClass {
   private void myFunction(ResourceResolver resourceResolver) {
      XSSAPI xssAPI = resourceResolver.adaptTo(XSSAPI.Class);
   }
}
```

JSP

```
<%@ include file="/libs/foundation/global.jsp" %>
<title><%= xssAPI.encodeForHTML(title); %></title>
```

XSSAPI: Methods

Validators (excerpt)

```
// Get a valid dimension (e.g. an image width parameter)
public String getValidDimension(String dimension, String defaultValue);

// Get a valid URL (Needs request-/resourceresolver specific API, see below)
public String getValidHref(String url);

// Get a valid integer from a string
public Integer getValidInteger(String integer, int defaultValue);

// Get a valid long from a string
public Long getValidLong(String long, long defaultValue);

// Validate a Javascript token.
// The value must be either a single identifier, a literal number, or a literal string.
public String getValidJSToken(String token, String defaultValue);
```

Encoders (excerpt)

```
// Encode string to use inside an HTML tag
public String encodeForHTML(String source);

// Encode string to use inside an HTML attribute
public String encodeForHTMLAttr(String source);

// Encode string to use inside an XML tag
public String encodeForXML(String source);

// Encode string to use inside an XML attribute
public String encodeForXMLAttr(String source);

// Encode string to use as a JavaScript string
public String encodeForJSString(String source);

// Encode string to use as a CSS string
public String encodeForCSSString(String source);
```

Filters

```
// Filter a string using the AntiSamy library to allow certain tags
public String filterHTML(String source);
```

Filters potentially user-contributed HTML to meet the AntiSamy policy rules currently in effect for HTML output (see the XSSFilter service for details).

JCR based URL mapping

```
// Use one of these to get an XSSAPI suitable for validating URLs
public XSSAPI getRequestSpecificAPI(SlingHttpServletRequest request);
public XSSAPI getResourceResolverSpecificAPI(ResourceResolver resolver);
```

Taglib

Taglib

```
<cq:text property="jcr:title" tagName="h2" escapeXml="true">
```

HTL

HTL

HTL automatically filters and escapes all variables being output to the presentation layer to prevent cross-site-scripting (XSS) vulnerabilities, by detecting the correct escaping context depending on the current HTML node and / or attribute.

For more details check the available display contexts from :
https://github.com/Adobe-Marketing-Cloud/sightly-spec/blob/master/SPECIFICATION.md#121-display-context.

Exceptions

1. Output generated in <script> and <style> tags require an explicit context, since by default HTL will not add one and will instead output empty
strings.
2. The style and the HTML Event attributes [1] also require an explicit context.

[1] - *https://www.w3.org/TR/html5/webappapis.html#event-handlers-on-elements,-document-objects,-and-window-objects*

Examples

Example API usages for the most common contexts

```
<%
   String title = request.getParameter("title");
   String alertText = request.getParameter("alertText");
   String link = request.getParameter("link");
   String fontSize = request.getParameter("fontSize");
   String className = request.getParameter("className");
   XSSAPI myXssAPI = xssAPI.getRequestSpecificAPI(request);
%>
<%@ include file="/libs/foundation/global.jsp" %>
<html>
   <head><title><%= xssAPI.encodeForHTML(title); %></title></head>
   <body>
      <p><%= xssAPI.filterHTML("Text with legitimate <b>HTML</b> Tags"); %>
       </p>
      <font size="<%= xssAPI.getValidInteger(fontSize); %>">
         <a href="<%= myXssAPI.getValidHref(link) %>" >click me</a>
      </font>
      <span class="<%= xssAPI.encodeForHTMLAttr(className); %>">
         <cq:text property="jcr:description" tagName="p" escapeXml="true">
      </span>
      <script>alert('<%= xssAPI.encodeForJSString(alertText); %>');
      </script>
   </body>
</html>
```

Some exploit strings for testing

HTML attributes

```
"><script>alert(23);</script>
```

Node namest

```
"><img src=bogus onError=alert(23)>
```

JSON Attributes

```
"};alert(23);a={"a":
```

HTML tags

```
</script><script>alert(23);</script>
```

See also: OWASP XSS Filter Evasion Cheat Sheet

Adobe