New issue

Jump to bottom

Escape <!--, --> and <script #20

Merged mikesamuel merged 1 commit into OWASP:master from fmeum:fhenneke_escape on Jun 7, 2020

Conversation 21 Commits 1 Checks 0 Files changed 3

fmeum commented on Jun 4, 2020 • edited — Contributor

In HTML script elements, `</script` is not the only substring that switches the HTML parser state even when contained in a JS string literal. This commit adds escaping for `<!--`, `-->` and `<script`, which appear to be all substrings that cause state transitions which are not reset by the `"` closing the string literal.

The injection of `<!--`, `-->` and `<script` in script elements is less severe than that of `</script` since they do not directly allow the execution of arbitrary HTML/JS. However, they could potentially be used to selectively disable scripts by causing syntax errors.

The following examples demonstrate the effect of these injections in a JSON context. In all cases, the alert is not executed, but will be executed if the value of the `json` variable is replaced with a harmless literal such as "foobar". The first example is the most relevant one since it does not have any further requirements on the contents of the script. The second and especially the third example are much more theoretical, but still valid HTML/JS.

```
<script>
    var json = "<!--<script>";
    alert("not executed");
</script>
```

```
<script>
    var unrelated_variable = "<!--";
    // ...
    var json = "<script>";
    alert("not executed");
</script>
```

```
<script>
<!--<script>
    var json = "-->"
</script />
alert("not executed");
</script>
```

jmanico commented on Jun 4, 2020 — Member

Thank you for this thoughtful contribution!
...

fmeum commented on Jun 4, 2020 — Contributor Author

Here is another example that shows why escaping `<!--` by itself is necessary:

```
<script>
    var json = "<!--";
    // ...
    var unrelated_variable = "<script>";
    alert("not executed");
</script>
```

mikesamuel reviewed on Jun 5, 2020

View changes

src/main/java/com/google/json/JsonSanitizer.java Outdated — Show resolved

src/main/java/com/google/json/JsonSanitizer.java Outdated — Show resolved

src/test/java/com/google/json/JsonSanitizerTest.java — Show resolved

Escape <!--, --> and <script ... 53ceaac

fmeum force-pushed the fhenneke_escape branch from **58c9691** to **53ceaac** 2 years ago — Compare

fmeum commented on Jun 5, 2020

View changes

src/test/java/com/google/json/JsonSanitizerTest.java

```
...  ...   @@ -56,14 +56,12 @@ public static final void testSanitize() {
 56   56       assertSanitized("\"foo\"");
```

```
57  57      assertSanitized("\"foo\"", "'foo'");
58  58      assertSanitized(
59      -       "\"<script>foo()<\\/script>\"", "\"<script>foo()</script>\"");
60      -   assertSanitized(
61      -       "\"<script>foo()<\\/script>\"", "\"<script>foo()</script>\"");
```

fmeum on Jun 5, 2020 — Contributor, Author

Is this a duplicated test?

mikesamuel on Jun 5, 2020 — Contributor

Sure looks that way. AFAICT, there's no non-visible ASCII subtleties being tested there.

```
> hexdump -C JsonSanitizerTest.java

000007e0  22 29 3b 0a 20 20 20 20  61 73 73 65 72 74 53 61  |");.    assertSa|
000007f0  6e 69 74 69 7a 65 64 28  0a 20 20 20 20 20 20 20  |nitized(.       |
00000800  20 22 5c 22 3c 73 63 72  69 70 74 3e 66 6f 6f 28  | "\"<script>foo(|
00000810  29 3c 5c 5c 2f 73 63 72  69 70 74 3e 5c 22 22 2c  |)<\\/script>\"",|
00000820  20 22 5c 22 3c 73 63 72  69 70 74 3e 66 6f 6f 28  | "\"<script>foo(|
00000830  29 3c 2f 73 63 72 69 70  74 3e 5c 22 22 29 3b 0a  |)</script>\"");.|
00000840  20 20 20 20 61 73 73 65  72 74 53 61 6e 69 74 69  |    assertSaniti|
00000850  7a 65 64 28 0a 20 20 20  20 20 20 20 20 22 5c 22  |zed(.        "\"|
00000860  3c 73 63 72 69 70 74 3e  66 6f 6f 28 29 3c 5c 5c  |<script>foo()<\\|
00000870  2f 73 63 72 69 70 74 3e  5c 22 22 2c 20 22 5c 22  |/script>\"", "\"|
00000880  3c 73 63 72 69 70 74 3e  66 6f 6f 28 29 3c 2f 73  |<script>foo()</s|
00000890  63 72 69 70 74 3e 5c 22  22 29 3b 0a 20 20 20 20  |cript>\"");.    |
```

mikesamuel approved these changes on Jun 5, 2020

View changes

src/test/java/com/google/json/JsonSanitizerTest.java

```
...  ...   @@ -56,14 +56,12 @@ public static final void testSanitize() {
56  56      assertSanitized("\"foo\"");
57  57      assertSanitized("\"foo\"", "'foo'");
58  58      assertSanitized(
59      -       "\"<script>foo()<\\/script>\"", "\"<script>foo()</script>\"");
60      -   assertSanitized(
61      -       "\"<script>foo()<\\/script>\"", "\"<script>foo()</script>\"");
```

mikesamuel on Jun 5, 2020 — Contributor

Sure looks that way. AFAICT, there's no non-visible ASCII subtleties being tested there.

```
> hexdump -C JsonSanitizerTest.java

000007e0  22 29 3b 0a 20 20 20 20  61 73 73 65 72 74 53 61  |");.    assertSa|
000007f0  6e 69 74 69 7a 65 64 28  0a 20 20 20 20 20 20 20  |nitized(.       |
00000800  20 22 5c 22 3c 73 63 72  69 70 74 3e 66 6f 6f 28  | "\"<script>foo(|
00000810  29 3c 5c 5c 2f 73 63 72  69 70 74 3e 5c 22 22 2c  |)<\\/script>\"",|
00000820  20 22 5c 22 3c 73 63 72  69 70 74 3e 66 6f 6f 28  | "\"<script>foo(|
00000830  29 3c 2f 73 63 72 69 70  74 3e 5c 22 22 29 3b 0a  |)</script>\"");.|
00000840  20 20 20 20 61 73 73 65  72 74 53 61 6e 69 74 69  |    assertSaniti|
00000850  7a 65 64 28 0a 20 20 20  20 20 20 20 20 22 5c 22  |zed(.        "\"|
00000860  3c 73 63 72 69 70 74 3e  66 6f 6f 28 29 3c 5c 5c  |<script>foo()<\\|
00000870  2f 73 63 72 69 70 74 3e  5c 22 22 2c 20 22 5c 22  |/script>\"", "\"|
00000880  3c 73 63 72 69 70 74 3e  66 6f 6f 28 29 3c 2f 73  |<script>foo()</s|
00000890  63 72 69 70 74 3e 5c 22  22 29 3b 0a 20 20 20 20  |cript>\"");.    |
```

src/main/java/com/google/json/JsonSanitizer.java — Show resolved

src/main/java/com/google/json/JsonSanitizer.java — Show resolved

mikesamuel self-assigned this on Jun 5, 2020

mikesamuel commented on Jun 5, 2020 — Contributor

I'll merge this tomorrow morning (US ET) and push a version to maven central.

When I put out an advisory, does it need to touch on anything besides:

- this affects embedders who are embedding sanitized JSON in HTML `<script>` elements after a `<!--` sequence
  - either as part of a *SingleLineHTMLOpenComment*
  - or embedded in JavaScript as in

    ```
    ["<!--", x<!--y, /<!--/] // <!--
    ```

- users should update to version **TBD**

fmeum commented on Jun 5, 2020 • edited — Contributor, Author

> I'll merge this tomorrow morning (US ET) and push a version to maven central.
>
> When I put out an advisory, does it need to touch on anything besides:

- this affects embedders who are embedding sanitized JSON in HTML `<script>` elements after a `<!--` sequence
  - either as part of a *SingleLineHTMLOpenComment*
  - or embedded in JavaScript as in

    ```
    ["<!--", x<!--y, /<!--/] // <!--
    ```

- users should update to version **TBD**

I think that this potentially affects **every** embedder, not just those that already have a `<!--` in their script (see the first example above): By injecting `<!--<script>`, a script element can be forced to continue through the first end tag until a second one is encountered.

If there is no second `</script>` tag, the script will simply not execute. With a stored injection, this could be a DoS attack, with a reflected injection, this could still be used to selectively disable parts of scripts.

If there is a second `</script>` tag on the page, the result will likely be a syntax error right on the first `</script>` since it is pretty unlikely that `</script>` itself forms valid JS. But this is not completely impossible, e.g.

```
alert(1)
</script />
alert(2)
```

is a valid JavaScript expression which compares `undefined` to a regular expression and the resulting truth value `false` to `undefined`, in the process showing two alerts instead of just one. While it is extremely unlikely that this can be used for XSS in practice, it might allow for creative nonce-based CSP bypasses given another injection point right after the `</script>` tag.

Edit: The following is an example of a CSP bypass when the attacker can additionally inject in the script end tag:

```
<script nonce="secretnonce">
    // ...
    var json = "injection_point_1_valid_JSON"
    // ...
    // The following command can be anything as long as it doesn't end with a semicolon
    var foo = 36
</script injection_point_2>
```

In this case, the attacker can inject `<!--<script>` in injection point 1 and `/;alert(1)//` in injection point 2 to execute arbitrary JS, which I believe would not be possible without the JSON injection. This also applies if injection point 2 comes right after the script end tag on the same line. I currently don't see a way for this to work if `</script>` is followed by a line break, but there may be some part of the JS syntax I have missed.

All in all, the full scope seems pretty difficult to assess, but most embedders could at least suffer from a broken page as a result of sucu injection. Everything beyond that is probably more of a theoretical issue.

👍 1

mikesamuel merged commit `f3512ba` into `OWASP:master` on Jun 7, 2020

---

mikesamuel commented on Jun 7, 2020 — Contributor

Fix included in :1.2.1 which should be indexed by maven shortly.

@FabianHenneke did you request a CVE for this or would you like me to?

Here's a second draft of the announcement.

Users of *com.mikesamuel:json-sanitizer* should upgrade to version 1.2.1 or later.

A bug in *com.mikesamuel:json-sanitizer:1.2.0* and prior allows an attacker who controls the content of a JSON string that is later embedded in an HTML `<script>` element to confuse the HTML parser as to where the `<script>` element ends. If the attacker also controls other content, e.g. a string of non-JavaScript content adjacent to the `<script>` element, this can lead to RCE.

See #20 (comment) for details.

If you have questions, please ask at the json-sanitizer-support Google Group.

---

fmeum commented on Jun 8, 2020 — Contributor, Author

> **@FabianHenneke** did you request a CVE for this or would you like me to?

@mikesamuel I haven't. Should this go directly through MITRE or to another CNA? I would generally prefer you to do the filing in order to keep the CVE metadata consistent for this project, but can take care of it if you are busy.

> If the attacker also controls other content, e.g. a string of non-JavaScript content adjacent to the `<script>` element, this can lead to RCE.

How about the following, which is currently my best shot at describing the scope:

> This can be used to prevent the execution of `<script>` elements. If the attacker additionally controls content in or right after the `</script>` end tag, this could lead to the execution of arbitrary JavaScript and a bypass of nonce-based CSPs.

To me, RCE in the context of a Java project sounds a bit more dramatic than "execution of arbitrary JavaScript", which is why I went for the latter. Feel free to tweak this further though.

fmeum deleted the `fhenneke_escape` branch 2 years ago

---

jmanico commented on Jun 8, 2020 — Member

Why CVE? The injection occurs only if there is an attack payload as a JSON value. The sanitizer is meant to disallow illegal JSON structure not sanitize individual values per my understanding.

I assume that data submitted server-side in untrusted JSON still needs to be used safely (ie: query parameterization, etc).

I assume that JSON data embedded in HTML markup still needs to be encoded to embed it safely. (JSON serializer and encoder like https://github.com/yahoo/serialize-javascript)

May I ask what security use case is broken and what are we fixing that needs a CVE?

Aloha,
--
Jim Manico
@manicode
...

mikesamuel commented on Jun 8, 2020 — Contributor

> I assume that JSON data embedded in HTML markup still needs to be encoded to embed it safely.
> https://github.com/OWASP/json-sanitizer#output says
>
> > The output will not contain the substrings (case-insensitively) `"<script"`, `"</script"` and `"<!--"` and can thus be embedded inside an HTML script element without further encoding.

mikesamuel commented on Jun 8, 2020 — Contributor

> **@FabianHenneke** did you request a CVE for this or would you like me to?
>
> **@mikesamuel** I haven't. Should this go directly through MITRE or to another CNA? I would generally prefer you to do the filing in order to keep the CVE metadata consistent for this project, but can take care of it if you are busy.

I am currently unaffiliated with any other CNA, so I was going to go through MITRE.
I'll take care of it.

> If the attacker also controls other content, e.g. a string of non-JavaScript content adjacent to the `<script>` element, this can lead to RCE.
>
> How about the following, which is currently my best shot at describing the scope:
>
> > This can be used to prevent the execution of `<script>` elements. If the attacker additionally controls content in or right after the `</script>` end tag, this could lead to the execution of arbitrary JavaScript and a bypass of nonce-based CSPs.
>
> To me, RCE in the context of a Java project sounds a bit more dramatic than "execution of arbitrary JavaScript", which is why I went for the latter. Feel free to tweak this further though.

Good point. Will do.

jmanico commented on Jun 8, 2020 — Member

Understood, of course. I only use this tool server-side before hitting a JSON parser.

Yea, a CVE is in order.
...

mikesamuel commented on Jun 8, 2020 — Contributor

Ok. CVE request is before the CVE gnomes.

👍 1

fmeum commented on Jun 9, 2020 — Contributor, Author

@mikesamuel I missed another way this could be abused: If there is a second injection point that allows `</script>`, this can also lead to XSS. See http://portswigger-labs.net/xss/script.php?x=%3C!--%3Cscript%3E&y=%3C/script%3E%3Cimg%20src=1%20onerror=alert(1)%3E for an example.

mikesamuel commented on Jun 9, 2020 — Contributor

CVE-2020-13973

👍 2

sebmarkbage mentioned this pull request on Apr 14

React 18: bootstrapScriptContent escapes HTML so quotes can't be used facebook/react#23063

Closed

Reviewers
mikesamuel ✓

Assignees
mikesamuel

Labels
None yet

Projects
None yet

Milestone

No milestone

Development

Successfully merging this pull request may close these issues.

None yet

3 participants