apostrophecms / **apostrophe** Public

Code | Issues 121 | Pull requests 6 | Discussions | Actions | Security and

Commit `0e57dd0`

boutell authored 2 weeks ago · 97 / 97 · Verified

```
Merge commit from fork

fix requires upgrading BOTH apostrophe and @apostrophecms/seo. A new mechanism for safely
emitting JSON nodes has been introduced to make this type of vulnerabilit
y unlikely in the future. Thanks to [K Shanmukha Srinivasulu Royal]
(https://github.com/Chittu13) for reporting the vulnerability.
```

main · sanitize-html@2.17.3 ••• apostrophe@4.29.0

1 parent 42af766 commit 0e57dd0

5 files changed +118 -12

Top

Filter files…

- .changeset
  - thin-pears-search.md
- packages
  - apostrophe
    - lib
      - safe-json-script.js
    - modules/@apostrophecms/template
      - index.js
  - seo
    - lib
      - nodes.js
    - test

unit-tests.js

Search within code

.changeset/thin-pears-search.md

```
@@ -0,0 +1,6 @@
+ ---
+ "apostrophe": minor
+ "@apostrophecms/seo": patch
+ ---
+
+ Fix an XSS vulnerability allowing arbitrary markup to be inserted via the "SEO Title" or "Meta Description" fields provided by the @apostrophecms/seo module. The fix requires upgrading BOTH apostrophe and @apostrophecms/seo. A new mechanism for safely emitting JSON nodes has been introduced to make this type of vulnerability unlikely in the future. Thanks to [K Shanmukha Srinivasulu Royal](https://github.com/Chittu13) for reporting the vulnerability.
```

packages/apostrophe/lib/safe-json-script.js

```
@@ -0,0 +1,27 @@
+ // Serialize `data` to a JSON string that is safe to embed inside an HTML
+ // `<script>` element. `JSON.stringify` on its own does NOT escape the
+ // sequences `</script>`, `<!--` or `<![CDATA[`, so untrusted data (e.g.
+ // editor-provided SEO fields) in a JSON body could otherwise break out of
+ // the surrounding script tag and inject arbitrary HTML/JS (stored XSS).
+ // Escaping `<` as its `\u003c` form keeps the JSON valid while neutralizing
+ // all of those sequences. Line and paragraph separators are also escaped
+ // since they are valid in JSON but illegal in some JavaScript parsers.
+ //
+ // This is the single source of truth for that escaping. The template
+ // `renderNodes` helper uses it to render `{ json: ... }` node bodies, so in
+ // most cases you should just build a node like:
+ //
+ //   {
+ //     name: 'script',
+ //     attrs: { type: 'application/ld+json' },
+ //     body: [ { json: data } ]
+ //   }
```

```
19 + //
20 + // and let `renderNodes` do the right thing.
21 +
22 + module.exports = function safeJsonForScript(data) {
23 +   return JSON.stringify(data, null, 2)
24 +     .replace(/</g, '\\u003c')
25 +     .replace(/\u2028/g, '\\u2028')
26 +     .replace(/\u2029/g, '\\u2029');
27 + };
```

...he/modules/@apostrophecms/template/index.js

```
@@ -35,6 +35,7 @@ const path = require('path');
35 35   const { stripIndent } = require('common-tags');
36 36   const { SemanticAttributes } = require('@opentelemetry/semantic-conventions');
37 37   const voidElements = require('void-elements');
   38 + const safeJsonForScript = require('../../../lib/safe-json-script');
38 39
39 40   module.exports = {
40 41     options: { alias: 'template' },
@@ -1140,12 +1141,23 @@ module.exports = {
1140 1141         //     attrs: { href: '/some/path', rel: 'stylesheet' }
1141 1142         //   }
1142 1143         // ]
1143      -       // Node object SHOULD have either `name`, `text`, `raw` or `comment` property.
1144      -       // A node with `name` can have `attrs` (array of element attributes)
1145      -       // and `body` (array of child nodes, recursion).
     1144 +       // Node object SHOULD have either `name`, `text`, `raw`, `json` or
     1145 +       // `comment` property. A node with `name` can have `attrs` (array of
     1146 +       // element attributes) and `body` (array of child nodes, recursion).
1146 1147         // `text` nodes are rendered as text (no HTML tags), the value is always a string.
1147 1148         // `comment` nodes are rendered as HTML comments, the value is always a string.
1148 1149         // `raw` nodes are rendered as is, no escaping, the value is always a string.
     1150 +       // `json` nodes are rendered as a JSON serialization of the value,
     1151 +       // safely escaped for inclusion inside a `<script>` element so that
```

```
+        // untrusted content cannot break out of the surrounding script tag.
+        // Use this instead of building a `raw` body from `JSON.stringify()`
+        // yourself, e.g.:
+        //
+        //   {
+        //     name: 'script',
+        //     attrs: { type: 'application/ld+json' },
+        //     body: [ { json: myData } ]
+        //   }
         renderNodes(nodes) {
           if (!Array.isArray(nodes)) {
             self.logError(
@@ -1164,6 +1176,9 @@ module.exports = {
             if (node.raw != null) {
               return node.raw;
             }
+            if (node.json != null) {
+              return safeJsonForScript(node.json);
+            }
             if (node.name != null) {
               const name = self.apos.util.escapeHtml(node.name);
               const attrs = Object.entries(node.attrs || {})
```

packages/seo/lib/nodes.js

```
@@ -241,21 +241,18 @@ function getMetaHead(data, options) {
             '@graph': schemas
           };

-          const jsonLdString = JSON.stringify(jsonLdData, null, 2);
-
           nodes.push({
             comment: ' JSON-LD Structured Data '
           });

-          const scriptNode = {
+          // A `json` body is escaped by renderNodes so that untrusted content
+          // in SEO fields cannot break out of the surrounding script element
+          // (stored XSS).
+          nodes.push({
```

```
251  252            name: 'script',
252  253            attrs: { type: 'application/ld+json' },
253       -          body: [ {
254       -            raw: jsonLdString
255       -          } ]
256       -        };
257       -
258       -        nodes.push(scriptNode);
     254  +          body: [ { json: jsonLdData } ]
     255  +        });
259  256        }
260  257      } catch (err) {
261  258        if (process.env.APOS_SEO_DEBUG) {
```

packages/seo/test/unit-tests.js

```
@@ -1657,6 +1657,67 @@ describe('@apostrophecms/seo', function () {
1657  1657      });
1658  1658    });
1659  1659
     1660  +  describe('XSS prevention in JSON-LD', function () {
     1661  +    it('should emit JSON-LD as a json node so </script> cannot break out', function () {
     1662  +      const { getMetaHead } = require('../lib/nodes');
     1663  +      const safeJsonForScript = require('apostrophe/lib/safe-json-script');
     1664  +
     1665  +      const payload = '"></title><script>alert(1)</script>';
     1666  +
     1667  +      const data = {
     1668  +        page: {
     1669  +          title: 'XSS Test',
     1670  +          seoTitle: payload,
     1671  +          seoDescription: payload,
     1672  +          seoJsonLdType: 'WebPage',
     1673  +          _url: 'https://example.com/xss-test'
     1674  +        },
     1675  +        global: {
     1676  +          seoSiteName: 'Test Site',
     1677  +          seoSiteCanonicalUrl: 'https://example.com'
     1678  +        },
```

```
1679 +         req: {}
1680 +       };
1681 +
1682 +       const nodes = getMetaHead(data, {});
1683 +
1684 +       const jsonLdNode = nodes.find(n =>
1685 +         n.name === 'script' &&
1686 +         n.attrs &&
1687 +         n.attrs.type === 'application/ld+json'
1688 +       );
1689 +
1690 +       assert(jsonLdNode, 'JSON-LD script node should exist');
1691 +       assert(
1692 +         Array.isArray(jsonLdNode.body) && jsonLdNode.body[0],
1693 +         'JSON-LD script should have a body'
1694 +       );
1695 +
1696 +       // The body must be a `json` node, not a pre-serialized `raw` string,
1697 +       // so that renderNodes performs the safe-for-script escaping on our
1698 +       // behalf. Raw-serializing JSON inline is the exact footgun we are
1699 +       // trying to remove.
1700 +       assert(
1701 +         jsonLdNode.body[0].json != null,
1702 +         'JSON-LD body should be a json node, not a raw string'
1703 +       );
1704 +
1705 +       // Belt and suspenders: verify that when that json value is actually
1706 +       // rendered through the safe encoder, the literal `</script` sequence
1707 +       // (which would terminate the surrounding <script> element) is gone.
1708 +       const rendered = safeJsonForScript(jsonLdNode.body[0].json);
1709 +       assert(
1710 +         !/<\/script/i.test(rendered),
1711 +         'rendered JSON-LD must not contain an unescaped </script> sequence'
1712 +       );
1713 +       // Sanity check: the payload is still there, just escaped.
1714 +       assert(
1715 +         rendered.includes('\\u003c/script'),
1716 +         'payload should survive in escaped form'
1717 +       );
1718 +     });
```

```
        1719  +   });
        1720  +
1660    1721      describe('Schema Validation', function () {
1661    1722
1662    1723        it('should validate Article schema requirements', function () {
```

Comments 0

Please sign in to comment.