CSP und Inline Ressourcen in Html

Gliederung

1. Einleitung
2. Content Security Policy (CSP)
3. Cross-Site Scripting (XSS)
4. CSS Filter
5. Data Filter (Experimentell)
6. JS Filter
7. Programmvorschau
8. Verwendete Technologien
9. Fazit

Content Security Policy (CSP)

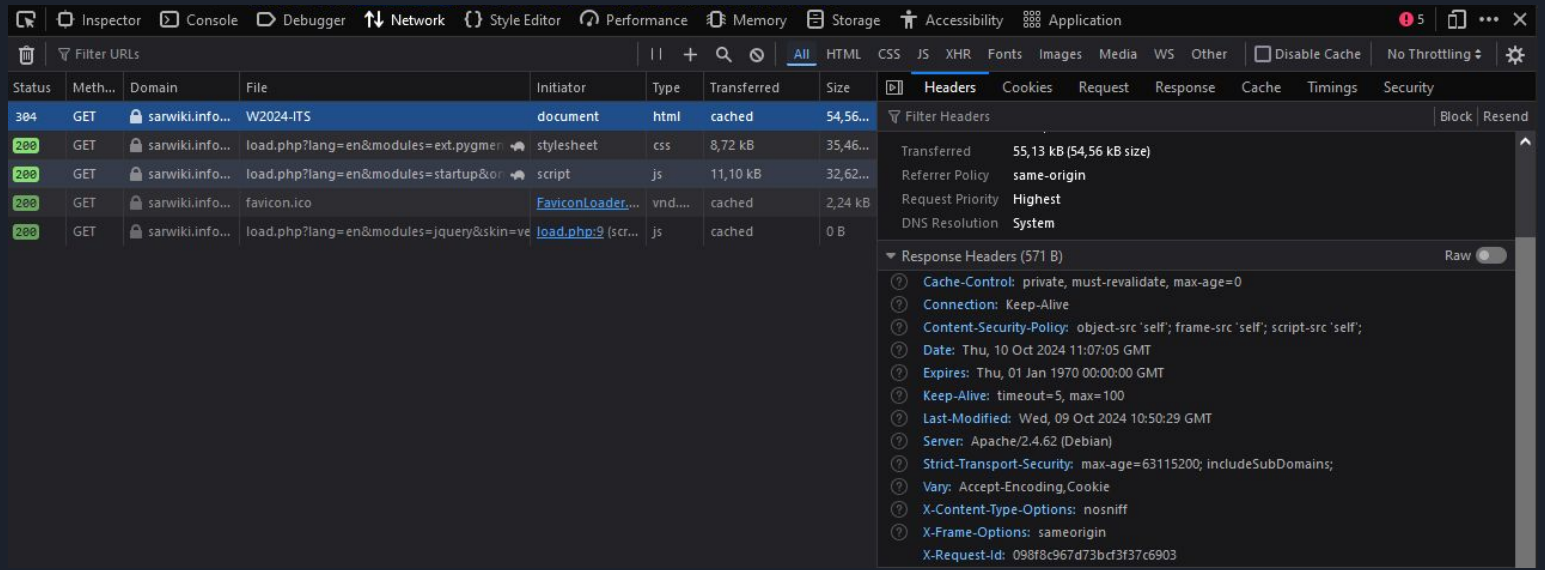
Web-Sicherheitsrichtlinie (2004)

Kontrolliert Ressourcen (z.B. Skripte, Stylesheets, Bilder)

Reduziert das Risiko von Angriffen wie Cross-Site-Scripting (XSS)

Content Security Policy (CSP)

HTTP-Header (Server), Meta-Tag



Content Security Policy (CSP)

Content-Security-Policy: object-src 'self'; frame-src 'self'; script-src 'self';

Content Security Policy im HTML-Tag definiert

```
<meta http-equiv="Content-Security-Policy"
content="default-src 'self'">
```

HTTP Response Header > Meta Tag

Content Security Policy (CSP)

Erlaube alles, aber nur aus derselben Quelle

```
default-src 'self';
```

Erlaube nur Skripte aus derselben Quelle

```
script-src 'self';
```

Definiert gültige Quellen für Plugins, z.B. <object>, <embed> oder <applet>

```
object-src 'self';
```

Content Security Policy (CSP)

Server Side Konfiguration:

Apache:

.htaccess : Header set Content-Security-Policy "default-src 'self';"

Nginx:

server {} : add_header Content-Security-Policy "default-src 'self';";

Content Security Policy (CSP)

Herausforderungen:

Inhalte von Drittanbietern wie z.B. Werbung, Analysen

Gute Planung, damit keine “falschen” Inhalte blockiert werden

Cross-Site Scripting (XSS)


@AARONSANIMALS

```
<h1>Was braucht man, um gute Laune zu haben:</h1>
<ul>
    <li>Frieden</li>
    <li>Freude</li>
    <li>Eierkuchen</li>
    <li><script>alert("Sie wurden gehackt! Bitte überweisen Sie das Geld auf die IBAN: 12345678, Kontoinhaber: Bösartige Mustermann.")</script></li>
</ul>
```

XSS in HTML

Types of Cross-Site Scripting

Reflected XSS
Non-Persistent or Type I

- Benutzereingaben werden sofort zurückgegeben
- Keine sichere Verarbeitung der Eingaben
- Daten nicht dauerhaft gespeichert
- Oft nur im Browser sichtbar

Stored XSS
Persistent or Type II

- Benutzer Input auf dem Server gespeichert (z.B. in DB, Forum, Kommentarfeld)
- Angriffs-Payload kann im Browser dauerhaft gespeichert sein

DOM Based XSS
Type-0

- Angriff durch Manipulation des DOM im Browser
- HTTP-Antwort bleibt unverändert, nur DOM wird geändert

XSS Typen Zusammengeführt

Server XSS

- Unsichere Benutzerdaten in HTTP-Antwort vom Server
- Datenquelle: Anfrage oder gespeicherter Ort
- Reflected und Stored Server XSS möglich
- Verwundbarkeit im Server-Code, Browser führt eingebetteten Skript aus

Client XSS

- Unsichere Benutzerdaten aktualisieren DOM
- Unsicherer JavaScript-Aufruf: führt validen JS-Code ein
- Datenquelle: DOM, Server (AJAX, Seiten Ladevorgang)
- Reflected und Stored Client XSS.

Where untrusted data is used

Data Persistence	XSS	Server	Client
	Stored	Stored Server XSS	Stored Client XSS
	Reflected	Reflected Server XSS	Reflected Client XSS

- DOM-Based XSS is a subset of Client XSS (where the data source is from the client only)
- Stored vs. Reflected only affects the likelihood of successful attack, not nature of vulnerability or defense

Bild Quelle: https://owasp.org/www-community/Types_of_Cross-Site_Scripting

Konsequenzen und wie Sie feststellen können, ob Sie gefährdet sind

- Von Belästigung bis zu Kontoübernahme
- Session-Cookie-Diebstahl, Dateizugriff, Trojaner-Installation
- Umleitung, Content-Manipulation

- Schwer zu identifizieren und zu beheben
- Code-Review: Eingaben im HTTP-Request, die ins HTML gelangen könnten
- Tools wie Nessus und Nikto
- Eine gefundene Schwachstelle = hohe Wahrscheinlichkeit weiterer Probleme

Wie Sie sich schützen können

Recommended XSS Defenses

Server

- die Unterstützung von HTTP TRACE auf allen Webservern abschalten
- Content Security Policy
- Context-sensitive server side output encoding
- Input-Validierung möglich, aber schwieriger als Encoding

Client

- Sichere JavaScript-APIs verwenden
- JavaScript Escape
- .textContent statt .innerHTML

Attack Examples

Ohne CSP:

http://localhost:3000/

Mit CSP:

http://localhost:8080/

CSS Filter

Beautiful Soup

```
# Open and read the HTML file
with open(html_file, 'r', encoding='utf-8') as file:
    html_content = file.read()

# Parse the HTML content with BeautifulSoup
soup = BeautifulSoup(html_content, features: 'html.parser')
```

CSS Filter

Style im <Head> deklariert

```
<head>
    <style>
        body {background-color: powderblue;}
        h1   {color: blue;}
        p    {color: red;}
    </style>
</head>
```

Tag inline style

```
<h1 style="color: green">This is a green heading</h1>
```

CSS Filter

```
def extract_style_tags(soup):  1 usage   Eric
    css_tags = []
    list_style_tags = soup.find_all('style')
    for tag in list_style_tags:
        css_tags.append(tag.extract())
    return css_tags
```

```
def add_link_to_html_header(soup, stylesheet_name):  1 usage   Eric

    new_link = soup.new_tag("link", rel="stylesheet", href=stylesheet_name)
    head_tag = soup.find('head')
    if head_tag:
        head_tag.append(new_link)
```

CSS Filter

Inline Problem:

```
body {background-color: powderblue;}
h1   {color: blue;}
p    {color: red;}

h1 { color: green }
```

CSS Specificity

Css Regeln werden als Gewicht dargestellt

Kollision = Regel mit mehr Gewicht gewinnt

Unentschieden = Als letztes definiert gewinnt

Die “important” Regel gewinnt immer = !important

CSS Specificity

Examples of Specificity

Selector	Code	Specificity
Inline Styles	`<div style="color: red;">`	(1, 0, 0, 0)
ID Selector	`#header { color: blue; }`	(0, 1, 0, 0)
Class Selector	`.container { color: green; }`	(0, 0, 1, 0)
Element Selector	`div { color: black; }`	(0, 0, 0, 1)

CSS Filter (Lösung)

```
for tag in soup.findAll(is_style_attr):

    if not tag.has_attr('id'):
        unique_id = str(uuid.uuid4())[:8]  # Generate a short unique id
        tag['id'] = f"element-{unique_id}"
```

```
body {background-color: powderblue;}
h1   {color: blue;}
p    {color: red;}

#element-d9aab026 { color: green }
```

CSS Filter

Mögliche Probleme:

- Hierarchie könnte in seltenen Fällen Probleme haben
- Id’s müssen verlängert werden

Data Filter (Experimentell)

Data Filter

Filter von <img> mit Base64 Enkodierung

```
images = []
for img_tag in soup.find_all('img'):
    img_details = {'src': img_tag.get('src'),
                   'attributes': {attr: img_tag.get(attr) for attr in img_tag.attrs if attr != 'src'},
                   'tag': img_tag}  # Keep a reference to the original tag
    if img_details['src']:
        images.append(img_details)
```

Data Filter

Einbinden eines <img> Tags

1. URL, die auf eine Bilddatei verweist

   (z.B. .png, .jpg)

2. Base64-kodierte Daten-URL direkt im src-Attribut

   (z.B. ...)

Data Filter

```
<img alt="wolf-emblem.svg" width="100" height="100"
src="...">
```

```
base64_regex = re.compile(r"data:(?P<mime>[\w+/]+);base64,(?P<data>[A-Za-z0-9+/=]+)")
match = base64_regex.match(image['src'])

if match:
    mime_type = match.group('mime')  # Extract MIME type
    base64_data = match.group('data')  # Extract Base64-encoded data
```

Data Filter

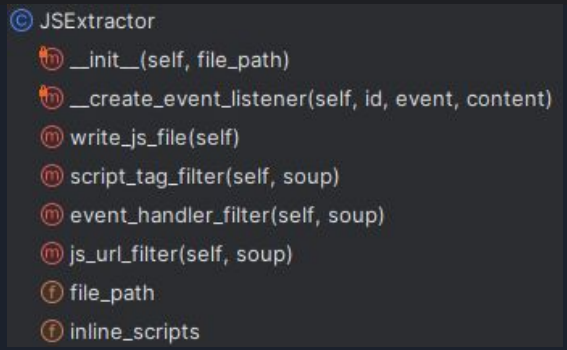
Mögliche Probleme mit Mime Types:

```
if '+' in mime_type:
    extension = mime_type.split('+')[0].split('/')[-1]
else:
    # Determine the appropriate file extension from the MIME type
    extension = mime_type.split('/')[-1]
```

JS Filter

JS Filter: Struktur

```
1 usage  Artem Suttar
class JSExtractor:
    Artem Suttar
    def __init__(self, file_path):
        # Path where the extracted JavaScript will be saved
        self.file_path = file_path
        # List to store the contents of extracted <script> tags and generated functions
        self.inline_scripts = []
```



JS Filter: inline <script> tags

```
# Extract and remove inline <script> tags from the HTML
1 usage  Artem Suttar
def script_tag_filter(self, soup):
    # Find all <script> tags in the parsed HTML
    scripts = soup.find_all("script")

    # Iterate over each <script> tag, extract it, and store its content
    for script in scripts:
        script = script.extract()  # Remove the script tag from the DOM
        self.inline_scripts.append(script.string)  # Append the script content to the inline_scripts list

    # Return the modified HTML without the <script> tags
    return soup
```

JS Filter: inline event handlers

Event handler	Event handler event type
onabort	abort
onauxclick	auxclick
onbeforeinput	beforeinput
onbeforematch	beforematch
onbeforetoggle	beforetoggle
oncancel	cancel
oncanplay	canplay
oncanplaythrough	canplaythrough
onchange	change
onclick	click
onclose	close
oncontextlost	contextlost
oncontextmenu	contextmenu
oncontextrestored	contextrestored
oncopy	copy

Quelle:
https://html.spec.whatwg.org/multipage/webappapis.html#event-handlers-on-elements,-document-objects,-and-window-objects

JS Filter: inline JavaScript in URLs

Syntax

JavaScript URLs start with the `javascript:` scheme and are followed by JavaScript code. The code will be parsed as a script.

```
URL
javascript:<script>
```

Description

`javascript:` URLs can be used anywhere a URL is a navigation target. This includes, but is not limited to:

- The `href` attribute of an `<a>` or `<area>` element.
- The `action` attribute of a `<form>` element.
- The `src` attribute of an `<iframe>` element.
- The `window.location` JavaScript property.
- The browser address bar itself.

Quelle: https://developer.mozilla.org/en-US/docs/Web/URI/Schemes/javascript

JS Filter: inline JavaScript in URLs

```
elements = {
    "href": "click",
    "action": "submit",
    # src should not be used with combination of JS, we don't really hande this case
    "src": "load"
}
value = value.replace("javascript:", "", 1)
value = "event.preventDefault(); " + value
```

JS Filter: Dynamische Inline-Skripterstellung

```
document.body.innerHTML += '<script>alert("Dynamic inline script")</script>';
```

JS Filter: Dynamische Inline-Skripterstellung: Idee?

Abstract syntax tree (AST)

Bild Quelle: https://en.wikipedia.org/wiki/Abstract_syntax_tree#/media/File:Abstract_syntax_tree_for_Euclidean_algorithm.svg

JS Filter: Techniken des Funktionsvergleichs

Generierte Funktion

Generierte Funktion, basierend auf einer generierten Funktion

Programmvorschau

Verwendete Technologien

Sprache: Python

- Populär
- Einfach
- Deckt die erforderliche Anforderung

Versionsverwaltung: Git

https://github.com/SuttArt/CSP_extract_inline_resources

Docker als Testumgebung

- Einheitliche Testumgebung, unabhängig vom Host-System
- Einfache Reproduzierbarkeit der Testszenarien
- Isolierte Umgebung
- Leichtes Setup
- ideal für Teamarbeit

Fazit

Rückblick

- Keine Anmerkungen

Ausblick auf die Zukunft

- Testabdeckung
- Minifier
- CLI - Argumente
- AST für JS Funktionen
- Binärdateien für mehrere Betriebssysteme ?

Quellen

- https://portswigger.net/web-security/cross-site-scripting/cheat-sheet
- https://owasp.org/www-community/attacks/xss/
- https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html
- https://owasp.org/www-community/Types_of_Cross-Site_Scripting
- https://cheatsheetseries.owasp.org/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.html
- https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html
- https://www.rfc-editor.org/rfc/rfc2397