Mending MOP’s Content Security Policy

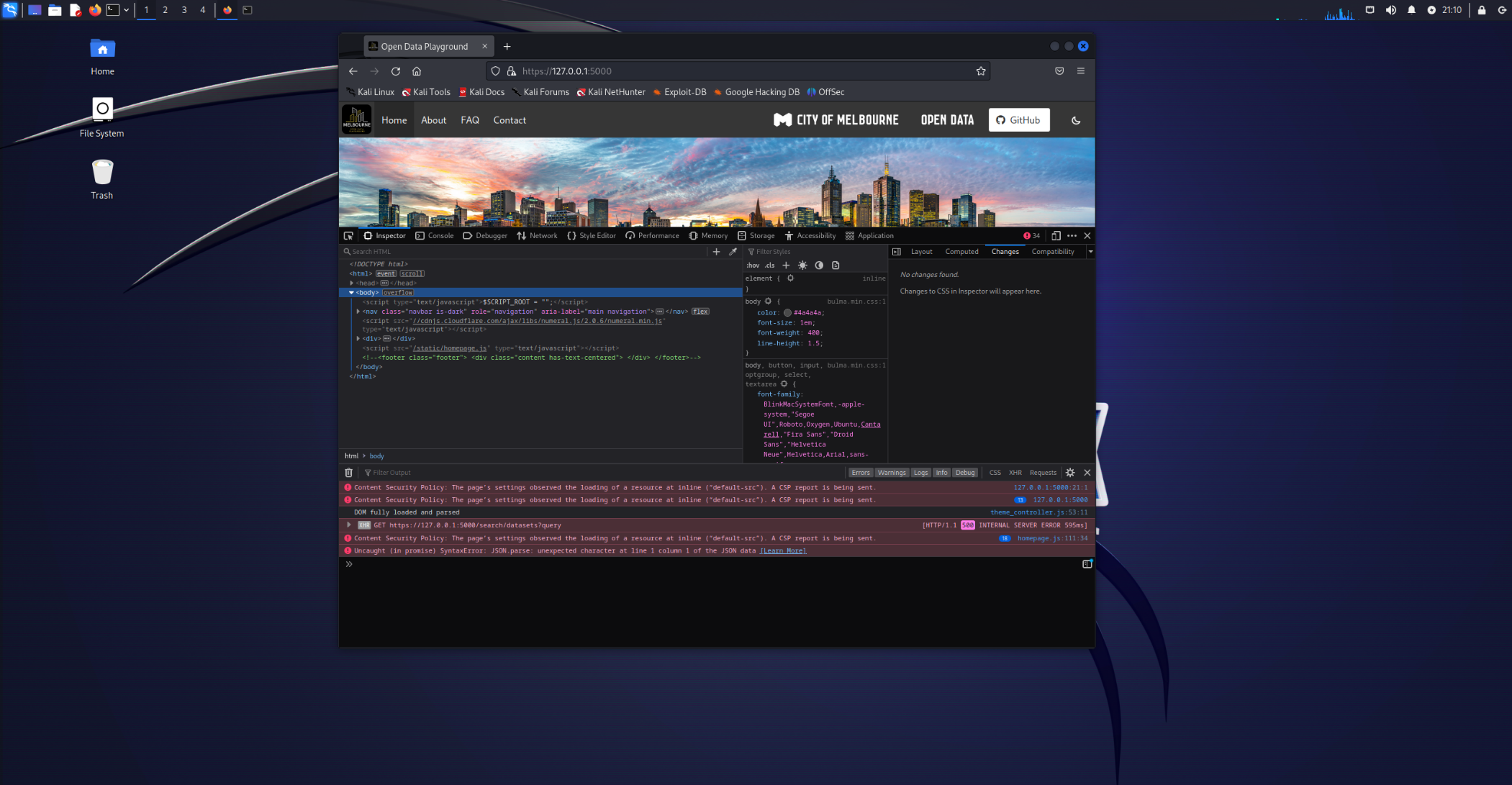
## Executive Summary

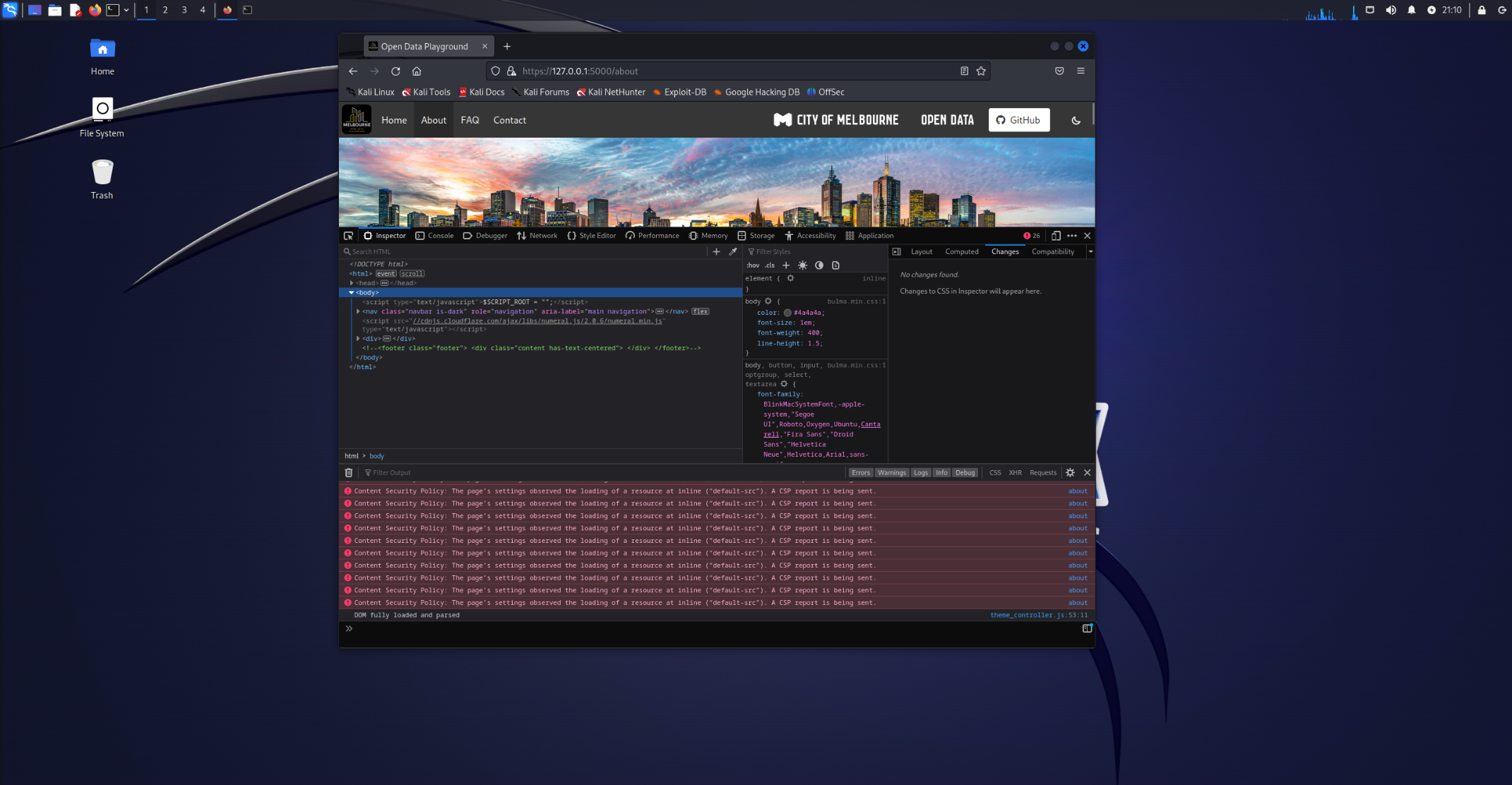
In following from my previous sprint’s attempt to fix the Content Security Policy (or CSP) of the MOP project, I have implemented a working CSP that currently allows for many external resources used for the website’s core functions. These errors would have slipped through because of the website’s notably lax CSP previously noted in the source code inspection as a well as website vulnerability-scanning tools, and leaving this issue unaddressed will leave the site open for injection attacks and defacement.

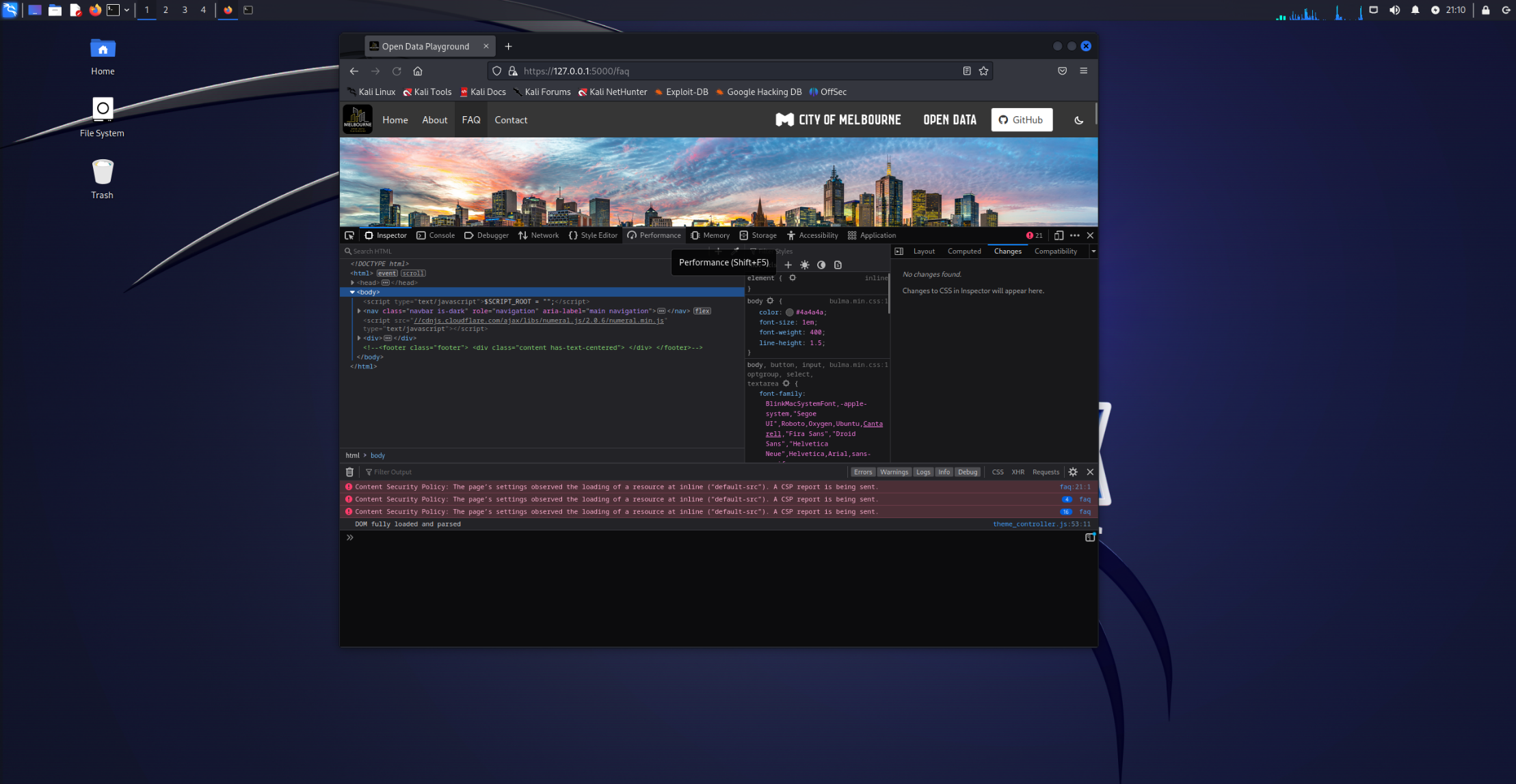
The objective of this report will cover overall what the Content Security Policy I have developed will allow. After going through the implementation, there are recommendations for the development team in fixing other use-cases to make the website secure.

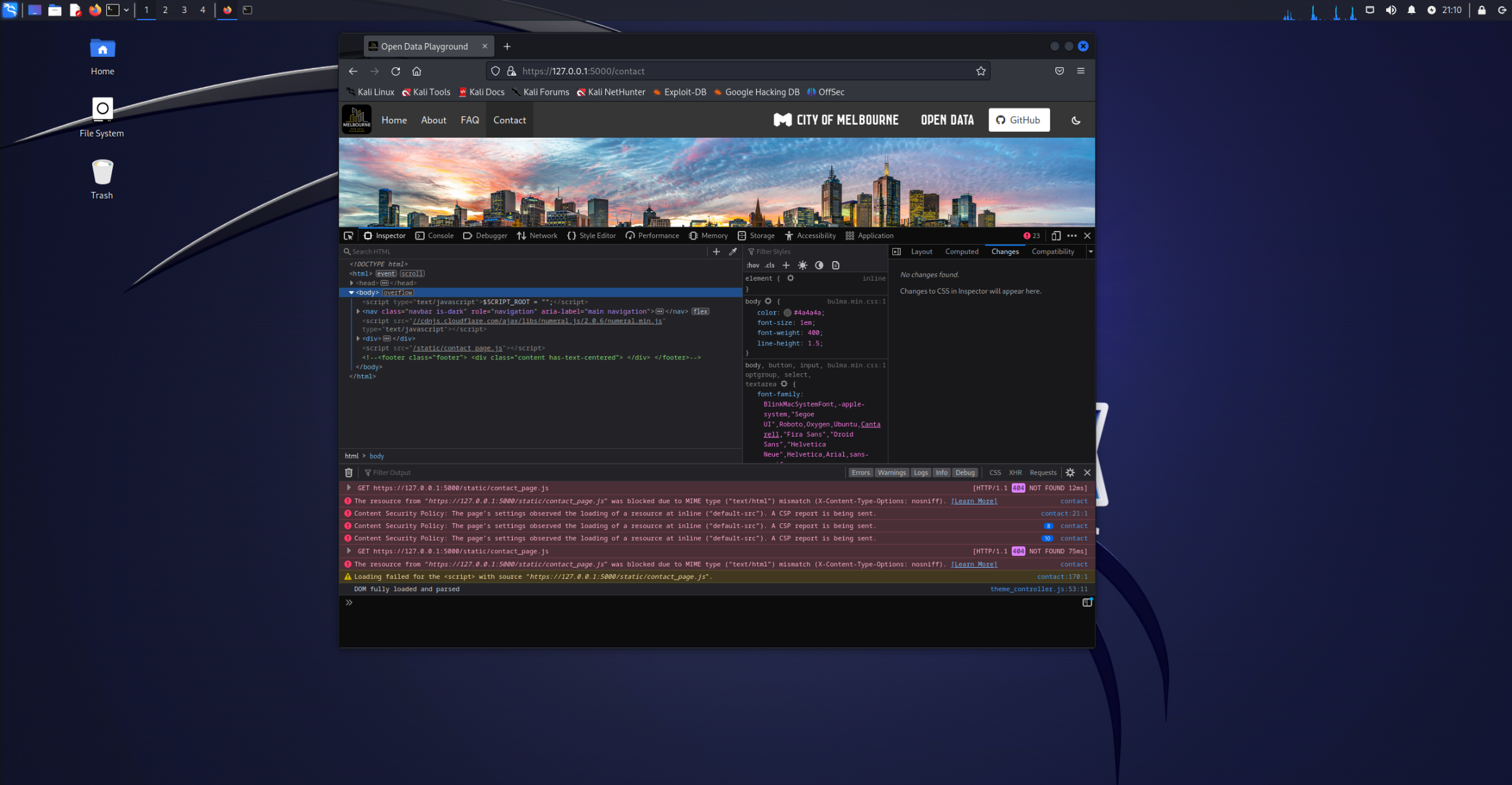
## The current state of the website’s Content Security Policy

When we look at the website either at <https://master-mop-busaytgm.ts.gateway.dev/> or locally hosting after downloading the development files, opening the inspector will reveal how many security warnings are made. For this task, we only worry about the Content Security Policy warnings that appear. Below are the error warnings from the current CSP implementation (default-src: ‘\*’ – meaning all external resources are allowed), where only inline and eval warnings are displayed.





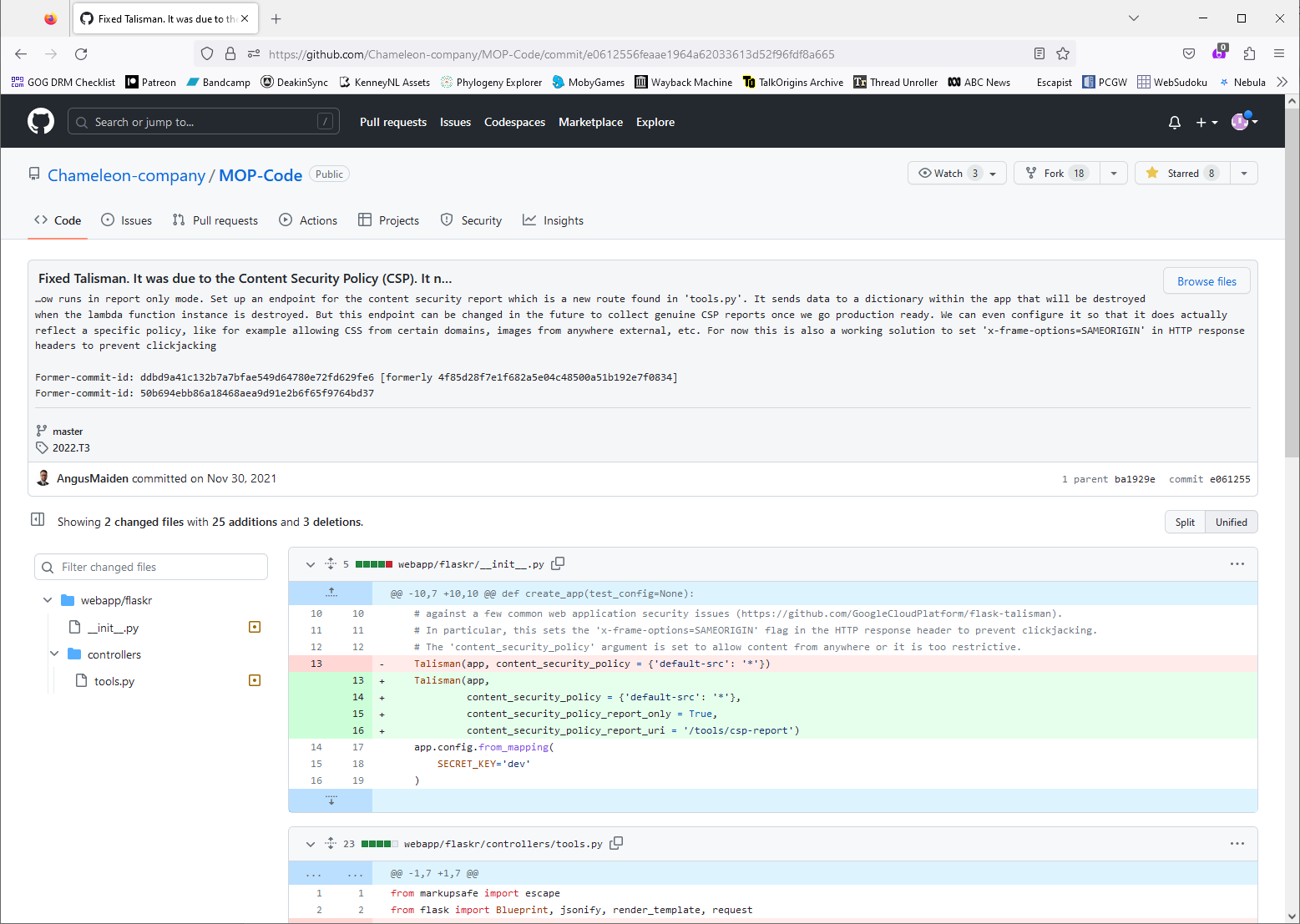




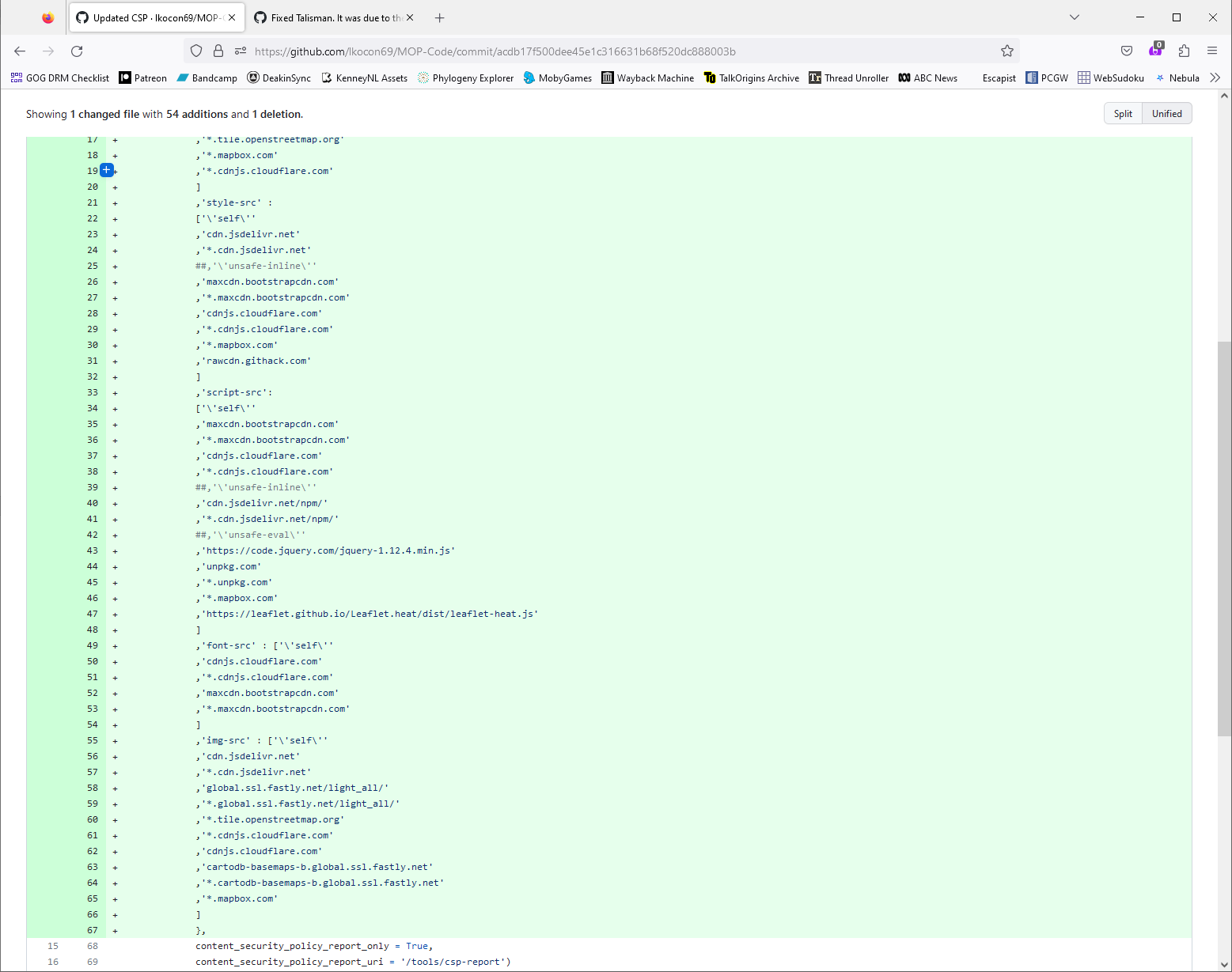
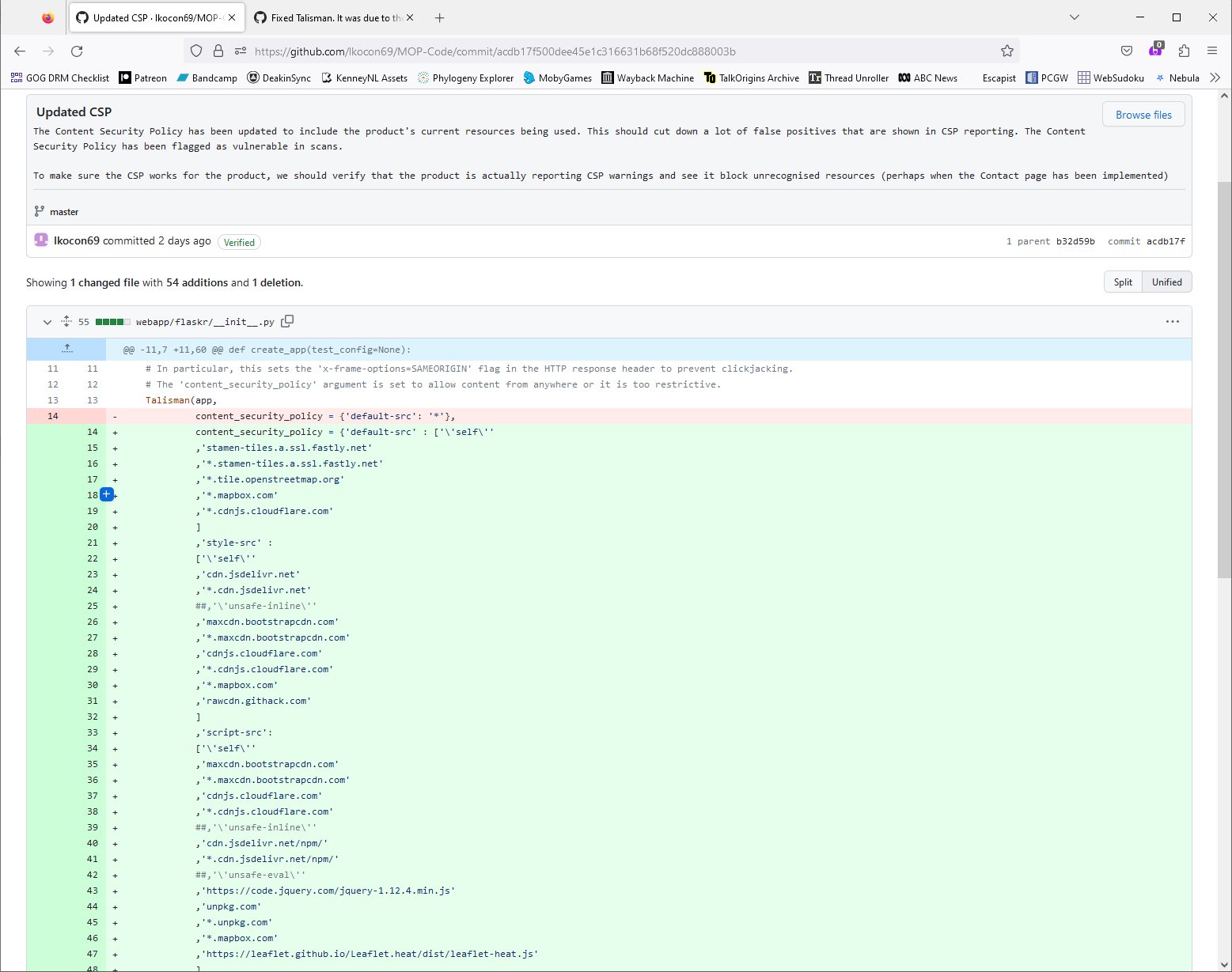
This allows every external resource to run on the project website, where all the resources used in styling and running the project and externally-hosted XSS attack can be brought onto the project. While handy for the development team’s needs of having a website that is stable when submitting new use-cases or features, this will not secure the project website.

## How we mended the lax CSP

The security team altogether has already noted the CSP as an issue, including myself when inspecting the source code directly. The CSP would not block any content because of its default configuration allowing content from anywhere. The original submission that included Talisman (where it was included AND fixed on the same day in 2021) noted how it is allowing content from anywhere otherwise it made the website non-functional. This is not good enough - not when it has been introduced specifically for security.



So ultimately the fixes made are as follows: all the resources used at all the website’s first four main pages (index, about, faq and contact) and per each use-case submitted at the time of repair (from “New Business Location” to “Evaluate Business Locations Using Pedestrian Traffic Day and Night”) were all added into the CSP. Each page was inspected in the browser and each error or CSP violation detected was recorded and included in the new CSP.



## Implementation flaws

However, an honest in-the-moment criticism of this method is how the project has dispersed the website’s security onto these other websites. Cross-site scripting attacks can be employed if an attacking script is contained on each of the wildcarded sites, and functionally bypasses the CSP whitelist while undermining its mission to quash cross-site scripting attacks. But there is something else.

A separate security issue remains in the CSP that the whitelisting scheme cannot defend: cross-site scripting attacks can still occur at the inline event (where most of the website currently works). A collective of Google security engineers had written a paper on the Content Security Policy system being unable to effectively secure websites from cross-site scripting attacks (link to the paper here: <https://dl.acm.org/doi/10.1145/2976749.2978363>). The paper introduced the “strict-dynamic” tag that can offer security mainly at inline levels, which will be of massive benefit to the project. This discovery was found at the end of the sprint, so it is not included in the submission to the current build.

Following the discovery of the paper on CSP and making a better scheme for cross-site scripting attacks, this is a task to bring forward into the next sprint.

