PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
Ministry of Higher Education and Scientific Research
University of Skikda — Department of Computer Science
Master's in Computer Security — Semester 3 (M2) — UEF322
Web and Mobile Application Security
Updated on 2026-10-25 at 18:57:55
© 2026 Mr Laïdi FOUGHALI — www.al-moualime.com/wmas — CC BY-NC-SA 4.0 License

---

Lab 2 — Observing the SOP and CORS in Practice

Background

Chapter 2 presents the Same-Origin Policy as the foundation of Web isolation, but this rule stays abstract until you have seen it **fail on purpose**. This lab makes two central course ideas tangible: (1) an origin is strictly defined by the protocol/host/port triplet – two servers on the same machine but on different ports are indeed two distinct origins – and (2) CORS is not an exception to the SOP but a mechanism of **explicit exemption**, decided server-side, never client-side. Understanding this lab means understanding why a developer can never "disable CORS from the browser": the restriction belongs to the browser, the authorization belongs to the server.

Objective

Visually observe a Same-Origin Policy (SOP) violation and its controlled resolution via CORS (see chapter 2 of the course handbook).

Prerequisites

Have studied chapter 2: the concept of origin (protocol + host + port), the Same-Origin Policy, CORS, `postMessage`.

Tools and installation

- A browser with developer tools (*Console* and *Network* tabs) – Chrome, Firefox, or Edge all work, already installed.
- Python 3 (the `http.server` module used in step 1 is built in, no installation needed): check with `python -version` or `python3 -version`.
- Flask, for step 3 only: `pip install flask`.
- Any text editor (VS Code, Notepad++, nano, etc.).

Procedure

Step 1 — Create two distinct origins

Create two folders, `site-a` and `site-b`. In each, place a minimal `index.html` file (a simple heading is enough). Open two terminals:

```
cd site-a && python -m http.server 8000
cd site-b && python -m http.server 8001
```

These two servers run on the same machine (`localhost`) but on **different ports**: strictly speaking, under the SOP, these are two different origins.

Step 2 — Trigger and observe the block

Open `http://localhost:8000` in the browser, open the console (F12), and run:

```
fetch('http://localhost:8001/').then(r => r.text()).then(console.log)
```

Observe the error message shown in the console. Note precisely the error text: it should mention *CORS* and *Access-Control-Allow-Origin*.

Step 3 — Explicitly allow the request

Replace the server on port 8001 with a small script that adds the CORS header. In Python with Flask:

```
from flask import Flask, make_response
app = Flask(__name__)

@app.route('/')
def index():
    resp = make_response('Hello from port 8001')
    resp.headers['Access-Control-Allow-Origin'] = 'http://localhost:8000'
    return resp

app.run(port=8001)
```

Rerun the exact same `fetch` command from step 2: the request now succeeds and the text is displayed in the console.

Step 4 — Check origin verification in postMessage

Look, in the source code of a page using a third-party iframe (for example an embedded video player or a payment widget on a demo site), for the `window.addEventListener('message', ...)` listener. Check in the code whether the function starts by testing `event.origin` before processing the message content. If this check is missing, explain the corresponding risk.

Comprehension questions

1. `http://localhost:8000` and `http://localhost:8001` share the same protocol and the same host. Why are they still two different origins?
2. In step 2, does the `fetch` request actually reach the server, or is it blocked before it is even sent? Justify by observing the *Network* tab in DevTools.
3. How does the `Access-Control-Allow-Origin` header added in step 3 illustrate the principle that CORS is a *server* decision, never a simple browser option?

Deliverable

- A screenshot of the CORS error obtained in step 2.
- A screenshot of the successful request in step 3.
- A sentence, in your own words, explaining why the browser blocks the first request but not the second.
- Your observation from step 4 (origin check present or not, and the associated risk).