1.1 Which API endpoint is vulnerable to CSRF? Why?
The “/api/transfer” API is vulnerable as it only checks whether the browser is authenticated. It doesn’t care where the form comes from.

1.2 Try crafting an exploit that uses the Fetch API rather than an HTML Form to automatically make a request to the server. Why does the fetch request fail and the HTML Form does not? (Hint: Same-Origin Policy)

The fetch API fails because it violates the Same-Origin Policy. It’s worth noting that even though the form submitting method also doesn’t have the same origin, it is not subjected to SOP regulation.

2.1 Is this a DOM-based, Reflected, or Stored-XSS? Why?
Reflected because the parameters are sent to the server and reflected back to the user browser to render.

2.2 First, try getting the following payload to run on the website: alert(document.domain). What is the value output by this alert? What is the meaning of this particular value and why is it dangerous for the application?

It says “bruin-market.ece117.com”. This means that our attack is at the same origin of the subject and that indicates we can almost do whatever we want through this vulnerability.

3.1 What is the CSS Attribute Selector for the gift code on the /gift/view page? Is it possible to select an attribute by value?

The CSS Attribute Selector for the gift code has id “code”, specifically, we are targeting the “value” attribute of the “code” element in the gift-view.html.

3.2 Try injecting an XSS payload, similar to Part 1. What is the error reported in the browser console?
I tried “<img src="X" onerror="alert(gift_id)">”. The error message is:
Refused to execute inline event handler because it violates the following Content Security Policy directive: "script-src 'none'". 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.

This is because the rendering endpoint “/gift/view/<id>” is defined with Content-Security-Policy (csp=True), while the “/item” does not.

4.1 What are the pros and cons of using SameSite cookie flags as opposed to different mitigations for CSRF? Select at least one other defense to compare and contrast (e.g. CSRF cookies).

Pros: easy to implement and is effective as it works for all requests from sites with different origin.

Cons: It can't distinguish the activities of the requests. Normal requests may also be block if using SameSite cookie flags for Session.

The CSRF cookies method creates tokens on the server side per user session/request. Every time the user logs in/make request, the server will transmit the CSRF token to the user. This reduces the exposure time if the token value is stolen, resulting in more safety. One downside is it may interfere with the "back" operation of the user as the user's CSRF token might have already expired. This is more efficient and flexible than SameSite cookie flags. Therefore, it is preferred.

4.2 Why might CSRF be a major security concern for companies deploying web applications? Describe the threat model for how a CSRF vulnerability could be distributed by hackers using Bruin Market as an example.

Because attackers can use the CSRF vulnerability gives attacker the ability to act like the account owner on the server side. In the example of Bruin Market, the attacker makes a phishing url such that when other people who happens to have login to their Bruin Market will automatically submit a request to transfer their money to the attacker's account. This can go without the user and company's notice, leading to tremendous economic damage.

5.1 What is one other possible defense besides the iFrame sandbox to mitigate XSS? For this response, you can consider solutions which prevent running JavaScript but explain pros and cons of such solutions.

we can enable Content-Security-Policy for the item.html page and have script-src 'none'. This prevents the execution of any JavaScript code from running in the page. The pro is it's easy to implement and have a very strong protection. The con is it can greatly reduce the capability of webpage as JavaScript can an important tool for the website to have advanced functionality.

5.2 Why might XSS be a major security concern for companies deploying web applications? Describe the threat model for how a XSS vulnerability could be used by hackers with Bruin Market as an example. One example you might want to learn about is Samy's Worm.

It's because the XSS attack can spread really fast without the company server's notice. It is quite often that when the company realizes their website has XSS vulnerability, thousands of user information has already been stolen. In the case of the Bruin Market, the attacker is pretending to be a normal user and sells items in his market. But the item description is actually a JavaScript code that other normal users' browser would render and run when opened. Through this way, the attacker can steal sensitive information from regular users such as cookies. This could lead to identity theft and other forms of damage.

6.1 Despite not being able to execute JavaScript, this webpage is still insecure. What is incomplete about the /gift/view page's Content-Security Policy (CSP)?

The CSP is incomplete as it can still load style from outside sources. This leads to the XS-leak attack that can steal valuable information of the user.

6.2 A difference between CSS Injection and other vulnerabilities in this lab is that it is purely a privacy-based attack. Describe a threat model for why CSS Injection is a major security concern for companies which have web applications with Personally Identifiable Information (PII).

When companies have web applications with Personally Identifiable Information such as SSN, they usually need to render that information on the user's browser. If the attacker is able to inject the malicious CSS style file to the webpage, he may be able to use the CSS attribute selector to locate the user's SSN and steal it.