Q1 *CSRF Trivia* (4 points)

Mark the following statements as True or False and justify your solution. Please feel free to discuss with students around you.

Q1.1 (1 point) CSRF requires the attacker to steal the victim's session cookie in order to succeed.

○ True ● False

> **Solution:** CSRF relies on the browser automatically attaching the victim's existing valid session cookie to the forged request.

Q1.2 (1 point) JavaScript is **NOT** required to execute a CSRF attack on the attacker's malicious page.

● True ○ False

> **Solution:** CSRF attacks can also be executed on URLs with malicious actions. For example, a GET request to **`https://bank.com/transfer?amount=100&recipient=mallory`** with a valid session token might send $100 to Mallory.

Q1.3 (1 point) CSRF tokens defend against forgery by uniquely identifying which user is making the request, allowing the server to authenticate the user based on the token alone

○ True ● False

> **Solution:** The CSRF token does not authenticate the user. Instead, it verifies that the request originated from a page the server itself rendered

Q1.4 (1 point) Does setting **`HttpOnly=true`** prevent CSRF attacks?

○ True ● False

> **Solution:** Setting **`HttpOnly=true`** only prevents JavaScript from reading the cookie, but it can still be attached to requests.

Q2 *Passwords Trivia* **(3 points)**

Mark the following statements as True or False and justify your solution. Please feel free to discuss with students around you.

Q2.1 (1 point) Picking one very strong password to use on multiple sites is the safest option.

○ True ● False

Solution: If one site is breached, the reused password can be used to compromise other accounts

Q2.2 (1 point) Rate limiters protects against denial-of-service because it denies the attacker from tampering with the login for an exponential amount of time

○ True ● False

Solution: Rate limiters could actually cause a denial-of-service attack if the attacker guesses the password incorrectly on purpose

Q2.3 (1 point) Rate limiters protects against brute-force guessing attacks.

● True ○ False

Solution: Rate limiter helps mitigate brute-force attacks by slowing down and/or blocking repeated failed attempts.

Q3 *Boogle* **(4 points)**

Boogle is a social networking website that's looking into expanding into other domains. Namely, they recently started a map service to try their hand at fusing that with social media. The URL for the main website is https://www.boogle.com, and they want to host the map service at https://maps.boogle.com.

Q3.1 (1 point) For each of the following webpages, determine whether the webpage has the same origin as http://boogle.com/index.html, and provide a brief justification.

i. https://boogle.com/index.html
ii. http://maps.boogle.com
iii. http://boogle.com/home.html
iv. http://maps.boogle.com:8080

Solution:

i. False. https://boogle.com/index.html and http://boogle.com/index.html do not have the same origin, since their protocols (https) and (http) are different.
ii. False. http://maps.boogle.com and http://boogle.com/index.html do not have the same origin, since their domains (maps.boogle.com) and (boogle.com) are different. The same-origin policy performs string matching on the protocol, domain, and port.
iii. True. The paths are not checked in the same-origin policy.
iv. False. http://maps.boogle.com:8080 and http://boogle.com/index.html do not have the same origin, because their ports (8080) and (80) are different. Note that if the port is not specified, the port defaults to 80 for http and 443 for https.

Q3.2 (1 point) Describe how to make a cookie that will be sent to only Boogle's map website and its subdomains.

Solution: Set the domain parameter of the cookie to .maps.boogle.com

Q3.3 (1 point) Boogle wants to be able to host websites for users on their servers. They decide to host each user's website at https://[username].boogle.com. Why might this not be a good idea?

Solution: A malicious user could set cookies that would be sent to other users' sites as well as the entire .boogle.com domain. Also, any cookies meant for boogle.com will go to the malicious user.

Q3.4 (1 point) Propose an alternate scheme so that Boogle can still host other users websites with less risk, and explain why this scheme is better.

Note: It's okay if the users' sites interfere with each other, as long as they can't affect official Boogle websites.

Solution: Boogle should create a new domain exclusively for user hosted content, like https://[username].boogleusercontent.com. This way, user sites cannot set cookies that will affect all boogle domains due to the cookie setting policy. This is known as a cookie tossing attack, and is one of the reasons why github hosts user sites on github.io instead of github.com (see https://blog.github.com/2013-04-09-yummy-cookies-across-domains/).

Q4 *Cross-Site Request Forgery (CSRF)* **(5 points)**

In a CSRF attack, a malicious user is able to take action on behalf of the victim. Consider the following example. Mallory posts the following in a comment on a chat forum:

```
<img src="http://patsy-bank.com/transfer?amt=1000\&to=mallory"/>
```

Of course, Patsy-Bank won't let just anyone request a transaction on behalf of any given account name. Users first need to authenticate with a password. However, once a user has authenticated, Patsy-Bank associates their session ID with an authenticated session state.

Q4.1 (1 point) Explain what could happen when Alice visits the chat forum and views Mallory's comment.

> **Solution:** The `img` tag embedded in the form causes the browser to make a request to
>
> `http://patsy-bank.com/transfer?amt=1000\&to=mallory`
>
> with Patsy-Bank's cookie. If Alice was previously logged in (and didn't log out), Patsy-Bank might assume Alice is authorizing a transfer of 1000 USD to Mallory.

Q4.2 (1 point) Patsy-Bank decides to check that the `Referer` header contains patsy-bank.com. Will the attack above work? Why or why not?

> **Solution:** In most cases, it will solve the problem since the `Referer` header will contain the blog's URL instead of patsy-bank.com.
>
> However, not all browsers send the `Referer` header, and even when they do, not all requests include it.

Q4.3 (1 point) Describe one way Mallory can modify her attack to always get around this check.

> **Solution:** She can have the link go to a URL under Mallory's control which contains patsy-bank.com such as
>
> `(patsy-bank.com).attacker.com`
>
> or
>
> `attacker.com/attack?dummy=patsy-bank.com`.
>
> Then this page can redirect to the original malicious link. Now the `Referer` header will pass the check.
>
> Another solution, is if the Patsy-Bank has a so-called "open redirect"
>
> `http://patsy-bank.com/redirect?to=`*url*,
>
> the referrer for the redirected request will be
>
> `http://patsy-bank.com/redirect?to=....`
>
> An attacker can abuse this functionality by causing a victim's browser to fetch a URL like http://patsy-bank.com/redirect?to=http://patsy-bank.com/transfer..., and from patsy-bank.com's perspective, it will see a subsequent request for http://patsy-bank.com/transfer... that indeed has a `Referer` from patsy-bank.com.

Q4.4 (1 point) Recall that the `Referer` header provides the full URL. HTTP additionally offers an `Origin` header which acts the same as the `Referer` but only includes the website domain, not the entire URL. Why might the `Origin` header be preferred?

Solution: Leaking the entire URL can be a violation of privacy against users. As an example, consider Alice transferred money by visiting

`http://patsy-bank.com/transfer?amt=1000\&to=bob`

and subsequently went to a website under an attacker's control - now the attacker has learned the exact amount of money Alice sent and to who. The `Origin` header would only leak that Alice was at the patsy-bank.com.

As a sidenote not directly related to the question, the `Origin` is a very useful way to solve the CSRF problem since it makes it much easier for multiple, trusted sites to make some action. For example, Patsy-Bank might trust

`http://www.trustedcreditcardcompany.com`

to directly transfer money from a user's account. This is a use-case that the CSRF token-based solution doesn't support cleanly.

Q4.5 (1 point) Almost all browsers support an additional cookie field `SameSite`. When `SameSite=strict`, the browser will only send the cookie if the requested domain **and** origin domain correspond to the cookie's domain. Which CSRF attacks will this stop? Which ones won't it stop? Give one big drawback of setting `SameSite=strict`.

Solution: It stops almost all CSRF attacks, except those involving open redirects from the website in question or if the website itself has an XSS vulnerability.

However, setting `SameSite=strict` can greatly limit functionality since any external links that require a user to be logged in won't work. For instance, consider a friend sends you a Facebook link via email,clicking on that link will require you to sign in again since your session cookie wasn't sent with the request.