Problem 1

Collaboration statement: I didn't consult with any peers, but I did have to do external research through OWASP, specifically:
https://owasp.org/www-community/controls/SecureCookieAttribute
https://owasp.org/www-community/HttpOnly
https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

a) Kaki can steal the cookie; she can eavesdrop on Jake's network connection, so when he is logged in, she can intercept the POST transmission and read the value field (the sessionid cookie).

b) Ben can steal the cookie. He can modify his page **http://crewmate.academy/staff/bsilverm/** to include an XSS atack, and this can be javascript code which sends the cookie to an endpoint that Ben controls

c) (i) Jake has to visit the site using **https** and not **http**. This does not change part (b), but for part (a), Kaki can no longer steal the cookie as it is necessarily encrypted in transmission.
(ii) This does not change the answer to part (a) as Kaki still visits the site using **http**. This does change the answer to part (b), as the Javascript used in Ben's XSS attack can no longer access the cookie, so he can no longer steal it.

d) (i) This prevents Ben's attack, as the scope of the cookie is only within **cs666.crewmate.academy/handin**, and thus Ben's page wouldn't be given access to it.
(ii) This doesn't prevent the attack, as the **bsilverman.crewmate.academy** domain is a subdomain of **crewmate.academy**, so when Ben's staff page, you would get access to any cookies with domain **crewmate.academy**. So, Ben's XSS attack still has access to the cookie and still works.
(iii) If the **Domain** attribute is not set, the cookie will only be sent to the origin server (not the subdomains). That means that the cookie won't be sent to Ben's page, which is at a subdomain of the origin site, and Ben's attack won't work since it won't have access to the cookie.

Problem 2

Collaboration Statement: I didn't consult with any peers, but I did have to do external research through OWASP, specifically:
https://owasp.org/www-community/controls/Content_Security_Policy
a) This prevention technique works because Javascript, which is needed to read and copy the cookie, is prevented from reading the cookie when an attacker runs it, since the script will belong to a different domain. The reason this works is because of the same-origin policy.

b) This prevention technique does work because of the same-origin policy, and specifically because cross origin reads are disabled by default (which is a browser security feature). The cookie is hidden and its origin is **https://crewmate.academy**. A CSRF attack necessarily comes from a different origin, so there's no way for such an attack to read the cookie.

c) The attacker can use a phishing attack to submit a malicious request to the form. This will get denied (as we assumed the prevention technique works), but it will get sent back to the user in order to allow them to resubmit the form. If the malicious request includes a clickjacking attack, we can use the **target** attribute of the form to display this attack in an iframe of our choice. Specifically, we can create a malicious overlay in the iframe of the original form (using the **html** target attribute), which makes it seem like the user is clicking something different than what they actually are (i.e. "Click here for a free iPhone" on top of the button to submit the form), which can make the user submit the form with contents of the attacker's choice. Since the user is the one submitting it, it'll bypass the CSRF prevention technique, whereas the attacker can't submit this form because of the same-origin policy. In summary, this works by phishing the user, submitting a form that will get rejected, and using this form (when it is sent back) to create an clickjacking attack that tricks the user into submitting the form.

Problem 3

Collaboration Statement: I didn't consult with any peers, but I did have to do external research through OWASP, specifically:
https://cheatsheetseries.owasp.org/cheatsheets/Vulnerability_Disclosure_Cheat_Sheet.html
(I couldn't find definitions of the disclosure types in the course resources)

a) Responsible disclosure is the idea of reporting a vulnerability privately to the company, but then giving more details publicly after a patch has been made available, or after a set amount of time (presumably long enough for a company to submit a patch). One advantage of this is that it gives time for an organization to fix the vulnerability, but also makes the vulnerability public knowledge afterwards so that others can learn from it. Full disclosure reports vulnerabilities publicly, which is advantageous because it pressures organizations to fix it, but has the downside of allowing the program to be exploited by anyone who reads the vulnerability report. Private disclosure reports the vulnerability directly (and privately) to the organization. This has the advantage of hiding the bug from anyone who could exploit it, but has the downside of allowing the company to ignore the bug report if they want to, since it's private and there's no pressure to fix it.

b) I would follow responsible disclosure guidelines of publicizing the vulnerability. If the company is dragging it's feet on publishing a patch, it's clear that the patch won't come without additional pressure. Releasing the vulnerability puts pressure on the company to fix the issue, and also makes it more likely that this company or other companies will take responsible disclosure reports more seriously in the future. The potential data breach is definitely unfortunate, but ultimately if the company doesn't patch the vulnerability, there's a risk of someone else discovering it anyways, and it is necessary for the company to release a fix.

c) I wouldn't accept these terms, as I believe they are coercive. I think security researchers have a responsibility to report their findings to the public, not just to the company. It makes sense that companies don't want to publicize their failings, but without the ability to disclose bugs, it's impossible for anyone but the researchers to verify that companies have fixed bugs. Thus, I wouldn't accept such terms as a security researcher.

d) I prefer the US policy on vulnerability disclosure, as I feel that it is more thorough. One such example is that it asks "What is the likelihood that adversaries will reverse engineer a patch, discover the vulnerability and use it against unpatched systems?", which considers the wider consequences of fixing a patch on one program while similar programs exist. Similarly, the US guidance asks "If the vulnerability is disclosed, how likely is it that the vendor or another entity will develop and release a patch or update that effectively mitigates it?", which is a valuable question to ask since it's entirely possible that vulnerabilities might not have effective fixes. However, I do appreciate that the UK guidance requires two separate bodies to approve of any resolution to retain a vulnerability, since I feel that governments should only hide vulnerabilities in the most important circumstances where it exceedingly clear that the benefits outweigh the costs of not fixing the bug.

Problem 4

Collaboration statement: I didn't consult any resources outside of the TA's, lectures, and resources linked in the assignment/lectures.

a) This could be an unreasonable form of security because it requires a lot of upkeep from the course staff. Every time they create new redirect links, they need to add a new site to the safelist, as well as removing it from the safelist if the site eventually becomes unsafe.

b) Not all browsers support the **Referer** header, so this might error more often than intended if the student opens a link in a browser that doesn't support this.

c) We can assume that staff members will never submit malicious links. We can modify the **/redirect** endpoint to be more secure by adding three additional endpoints when the link is created: **/timestamp**, **/key**, and **/hmac**. The timestamp should be the time at which the link is created, and the key should be a random key. The course staff can use a hash-based message authentication code (whose key is not publicly known), and should store **hmac_function(redirect ++ timestamp ++ key)** in /hmac. When a link is cliecked, the website should only redirect the user if the timestamp, key, and redirect match the entry in hmac when concatenated and encrypted. Additionally, the website should only redirect the user after a certain amount of time (i.e. 1 hour) after the timestamp, which it can do by checking the time of clicking the link and the time in the timestamp.

This avoids the issue in part (a) since we don't need to maintain a database, and the website can verify a link's authenticity simply by concatenating and hashing some of the arguments. This avoids the issue in part (b) by not using the **Referer** header, and only using endpoints in the url, which are supported in all browsers.

This satisfies **P1**, because when staff members create a link, the link includes data (the timestamp, key, and hmac) which can only be created using the key, and can only be verified using the key. This satisfies **P2** because such links can only be created using the key. This satisfies **P3** because links automatically expire after a fixed amount of time after the time stamp, so links are always eventually "revoked". This satisfies **P4** because the user isn't prompted with anything; either the site redirects the user (because the endpoints of the URL are valid), or returns an error/refuses to redirect the user.