3)

a)

i) **S**poofing - pretending to be someone (or something) you aren’t.

**T**ampering - modifying data without permission

**R**epudiation - covering tracks by (for example) deleting logs

**I**nformation Disclosure - revealing information without permission

**D**enial of Service - prevent a system from providing a service

**E**levation of Privilege - perform actions above the privilege level you are meant to have

ii)

**Threat 1:** An attacker on the same local network as the user sniffs the password being sent between (5) and (6), then logs in as the user.

**Threat 2:** A malicious script included on example.com is able to obtain the token being transferred between (1) and (2). The attacker can then use this token to get the user’s password from xyz.com.

**Threat 3:** A network attacker on the same local network as the user MITMs the communication between example.com and xyz.com. It returns an iframe containing a script to send the token to the attacker. Again, the attacker can use this token to get the user’s password from xyz.com.

**Threat 4:** A network attacker uses DNS poisoning to MITM the communication between the browser and example.com. The user then triggers the bookmarklet, and xyz.com returns the token. The attacker can read this token with some JS on the page, and then uses the token to get the user’s password.

iii)

**Threat 1:** the bookmarklet should communicate with xyz.com over HTTPS.

**Threat 2:** example.com should use Content-Security-Policy, and in particular subresource integrity, to ensure that example.com doesn’t include malicious scripts.

**Threat 3:** Use HTTPS (this is the same as threat 1, perhaps this is not the best answer)

**Theat 4:** DNSSEC on xyz.com’s end could work (I think)

b) this is from the coursework

4a) Not sure what “low-level capabilities” means. I will have a guess though:

i) A web attacker can:

* Try and get the user to click a link which causes a CSRF request
* Try and get the user to visit their own malicious webpage, which can then use JS to trigger a CSRF request
* Try and insert malicious JS into some other site, which can then perform CSRF requests.

ii) An active network attacker could:

* Using ARP poisoning, MITM the connection between the user and the site, if HTTPS is not being used

iii) “Related domain attacker” not mentioned in the current course

iv) “Related domain attacker” not mentioned in the current course

b) i)

* Generate a CSRF token for the user, and include it in the form returned to the user for them to fill in
* Check that this CSRF token matches when the user submits the form
* If it doesn’t, don’t perform the token
* Generate the token in a cryptographically secure manner, from the user’s session token.

This measure increases security, since the main vector for an attacker to perform a CSRF attack would be to get the user to unknowingly click a link elsewhere, triggering the action the attacker wants to be performed using that user’s account. This plan would be foiled, since it would not be possible for the attacker to include a valid CSRF token in the request.

ii)

* Use Strict-Transport-Security