CS 253 Final Exam Fall 2021 – Solutions

Problem 1. True or False

1. True
2. False
3. False
4. False
5. True
6. True
7. False
8. False
9. True

10. True
11. True
12. True
13. True
14. True (but accept False with explanation too)
15. False
16. True
    Assuming that the request came from a browser, the Origin header is guaranteed to be un-spoofable.
17. False
18. False

19. True
20. True
21. False
22. True
23. True
24. True
25. False
    This is false for two reasons. First, it is typically the browser (or more broadly, the user agent) that uses CT, rather than the website itself. Second, CT does not act "at the HTTPS protocol level", but is a separate protocol from HTTP and TLS.

Problem 2. Short Answers

1. Protocol/scheme, hostname, port

2. The HSTS header forces the browser to use HTTPS instead of HTTP on all future requests. This prevents a passive network attacker from observing the traffic, or an active network attacker from performing an SSLstrip man-in-the-middle attack.

3. Denial of service. A network attacker could modify an HTTP response and add an HSTS header with a long expiration time. Doing this on a website which does not actually support HTTPS would break the website for this user.

4. A website can request to be added to the HSTS preload list which is built into all the major browsers.
   Alternatively, sites can use the proposed DNS “HTTPS” record as an alternative to HSTS preload.

5. Frame A could remove Frame B from its DOM and add a new frame which loads any arbitrary URL. This is functionally equivalent to simply navigating Frame B. **[NOTE: This question was too confusing, so we gave everyone full credit.]**

6. The attacker can manually set their session ID cookie to the hash of the username they want to login as. Since SHA-256 is a public algorithm, this is trivial for them to do.

7. This design is secure. The counter value is not reversible, since 2^{256} is too large of a state space (as large as the output space of SHA-256 itself!) to search.

   Also acceptable: Not secure. No delimiter between username and counter means that years “feross” and “feross1” can be confused because SHA256(“feross1” + 1) = SHA256(“feross” + 11) will produce the same hash and, therefore, the same session ID.

8. Open a new window to xyz.com/call.html with window.open() and use the window reference to send a postMessage with an arbitrary phone number.

9. Yes.

10. No, the attack fails. The default-src 'self' directive means that script content is only allowed from external scripts loaded from the same origin. Inline scripts are blocked unless 'unsafe-inline' is present. Since the XSS attack is an inline script, it is blocked.

11. No, the attack fails. The script-src 'self' 'nonce-PAk3kslfKFAoaP423' allows script content from external scripts loaded from the same origin, or any script element with the specified nonce set as an attribute, e.g., <script nonce='PAk3kslfKFAoaP423'>. Since the attack XSS is an inline script without the specified nonce present, the attack is blocked.

    To get full credit, the answer must mention the nonce, and must not say that the CSP rejects all inline scripts (as inline scripts with the correct nonce still work).

12. The 'unsafe-inline' directive allows any inline <script> to execute, with no restrictions. This means that an attacker-inserted <script> will run.

13. There are many valid approaches. For example: rate limiting login attempts based on IP address, rate limited login attempts based on number of distinct accounts particular IP is logging into, showing a CAPTCHA after a certain number of failed attempts, ban users after a certain number of login attempts, deploy multi-factor authentication.

14. Yes, an active network attacker can modify the top-level page (since it is loaded over HTTP) and replace the iframe with a fake login form which sends the password to the attacker.

15. Man-in-the-middle attack. Note here that the website's **certificate** is already publicly available as part of the TLS protocol, and combined with the stolen **private key** allows the attacker to impersonate the website and/or eavesdrop on supposedly-secure communication.

16. The browser dimensions, the list of installed fonts, the user agent of the browser in use, the specific quirks of their graphics card (canvas fingerprinting), the specific quirks of their audio hardware (web audio fingerprinting), installed browser plugins, color depth, whether the Do Not Track header is sent (ironic)

17. A CORS preflight request is sent so the browser can **check to see if a server understands** the CORS protocol and is okay with the browser issuing potentially-destructive requests.

18. No. The request to https://axess.stanford.edu will be sent to the server but **the response will not be readable** by the page because it is a **cross-origin read** which is not allowed **unless there is an Access-Control-Allow-Origin header** present on the response.

19. No. Since https://bank.com and https://attacker.com are different origins, they are not allowed to directly access each other's DOMs across an <iframe> boundary as the attacker's code attempts to do.

20. SQL injection
    Stored XSS
    Reflected XSS

21. TLS/SSL Strip
    JavaScript sandbox escape
    Stored XSS
    Reflected XSS

Problem 3. The Great Cannon

a. Web browsers outside China

b. Websites that included the Baidu script were allowed to make requests to GitHub because the **same origin policy allows "simple" GET requests** to be sent to any origin.
   Also acceptable: the browser can't tell a man-in-the-middle attack has happened because the request uses HTTP. Same-origin policy does not apply

c. The following options are correct:

   - Every website that uses Baidu's analytics changes the script tag URL so it loads over HTTPS instead of HTTP. (Assume the script was also available over HTTPS.)
   - Baidu adds baidu.com to the HSTS (HTTP Strict Transport Security) preload list
   - Baidu switches its analytics server to only be accessible using HTTPS

d. None of the above.

e. A hash of the script being loaded

   Hashing the script's URL does NOT work, since it doesn't protect against the content of the script changing due to an attack, which is what we have here.

   Digital signatures can technically work as well, but it is more complicated to generate since you would need a key to sign the script, and then create some other secure way of telling the browser which key you used so it can verify the signature. Here, a public hash function suffices, and is thus the **BEST** solution.

f. Yes, instead of returning a malicious script (which would fail the integrity check), they can 302 redirect the requests for the analytics scripts to a page on github.com. Requests will be sent to GitHub but the integrity check will fail.

   Another answer: Yes, the analytics script is bound to send some requests to Baidu's servers. GC can redirect those requests too.

g. Whenever the third-party analytics script changes, the integrity check will fail and the website owner will need to update the integrity hash for it to start working again.

Problem 4. Cookies

a. The "Path" attribute does not protect against unauthorized reading of the cookie by other pages on the same origin. So it is possible for https://stanford.edu/~attacker to **include an <iframe> that loads https://stanford.edu/~clueless and then read out the cookie from the frame**. Since both pages are on the same origin, it's possible for the attacker page to access the DOM of the victim page and read the iframe.contentDocument.cookie property to steal the sessionId cookie.
b. **HttpOnly** would have **prevented the cookie from being accessible to client-side JavaScript**, thus protecting it from pages on the same site.
c. **Not protected**. The attacker can still **iframe the https://stanford.edu/~clueless page and reach into the DOM** to read out the private content for a logged in user.

Problem 5. Coffee Shop Wi-Fi

a. Other coffee shop patrons
   The manager of the store next door
   The coffee shop's ISP
   The website awesome-security-stuff.com
b. e.g., HTTPS, TLS, VPN
c. The website awesome-security-stuff.com
d. Yes. Facebook can inspect the Referer header on the request generated by the <img> element.

Problem 6. Delete Account

a. **CSRF**. Any website can send a **cross-origin POST** request by **submitting a form** to the /delete-account endpoint, which will delete the currently logged-in user without any user interaction or confirmation.

b. Any CSRF mitigation would work. Examples:
    - SameSite cookies
    - CSRF authenticity token
    - Require that the user submits their username or password in the request (not as cookie!), which the attacker wouldn't know.
    - Use a non-simple request (e.g., DELETE method)

Problem 7. User Agent

a. Stored XSS

b. The attacker can curl the web server with a malicious user-agent string (e.g., “<script>alert(document.cookie)</script>”). The server stores this in the userAgents array and includes it in future page loads. The next visitor arrives at the site and the server includes the unsanitized string in the HTML output.

Problem 8. CSP

3, 5, 8

Problem 9. Cross Site Script Inclusion (XSSI)

a. The attacker site can **define a displayData() function** and then **include the script** from (1) in their page. The user’s cookies will be sent with the request and the script will run in the context of attacker.com, calling the attacker’s function with the user’s account info.
```
<script>function displayData() { … }</script>
<script src="https://bank.com/userdata.js"></script>
```

b. Change the script to make a **fetch** request for a **JSON** file. Change userdata.js to be a JSON file, so that only websites which are same-origin to bank.com can read the response.

c. SameSite

d. Any answer accepted!