CS 253 Final Exam Fall 2019 – Solutions

True or false (1 point each) –

1. True
2. False
3. False
4. True
5. True
6. False
7. False
8. True
9. True
10. False
11. True
12. False
13. True
14. True

Short Answer (2 points each) –

1. Protocol, hostname, port

2. The single quote

3. The attacker can manually set their session ID cookie to the number 1 to steal the first user's session. If this session is no longer valid, then they can try 2, then 3, and so on until they find a session which is still valid.

4. The attacker can login to the website and see what session ID they are assigned by the server. By manually setting their session ID cookie to a number 1 less than the ID they received, they will steal the session of the user who logged in just before them.

5. Attackers can use error messages to extract information from the system, including sensitive information that may be included in the error message such as sensitive user data, the full paths of various files on the server, as well as which frameworks and libraries are in use.

6. No, the attack fails. The script-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.

7. Yes, the attack succeeds. The CSP allows script content from https://javascript-cdn.com and the XSS attack is a script hosted on that domain.

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

9. The built-in list of CAs act as a "root of trust" which allow the browser to verify that a certificate sent by a server has been signed by a trusted CA. The browser trusts the CA to only sign certificates after verifying that the public key is actually owned by the entity in control of the domain name.

10. 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.

11. Authentication is about verifying the user is who they say they are. Authorization is about deciding if a user has permission to access a resource.

Free Response (3 points each) –

1. Same Origin Policy: 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.

2. More Same Origin Policy: 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.

3. CORS Preflight: A CORS preflight request is sent so the browser can check to see if a server understands the CORS protocol and it is okay with the browser issuing potentially-destructive requests using specific HTTP methods or HTTP headers. In the example, the browser is checking to see if a DELETE request is allowed.

4. Cookies: 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> loads https://stanford.edu/~victim 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.

5. More Cookies: There are many ways to solve this issue. The easiest way is to add the cookie attribute "SameSite=lax" or "SameSite=strict" to the session cookie so that the browser will not include the cookie on requests initiated by other sites. Note that changing the HTTP method from GET to POST does not solve the issue since any site can initiate a form submission to bank.com which will send a POST request. Other approaches: use a CSRF token, change the request in some way so that it's not a "simple" request and other sites won't be able to send it without a preflight request to the bank (which the bank can deny).

6. XSS: The 'source' query parameter is not sanitized before being inserted into the HTML response of the page, which creates an opportunity for reflected XSS. The following URL would trigger it:

https://insecure.example.com/?source=<script>alert(document.cookie)</script> The issue could be fixed by HTML escaping the `req.query.source` variable before using it in the HTML output.

7. More XSS: There are two vulnerabilities in this server code, but only one needs to be described for full credit.

The first issue is that the JavaScript string escape character (\) itself is not escaped. So the attacker can escape out of the string into the <script> context:

http://insecure.example.com/login?username=\';alert(document.cookie)//&password=123

This generates the following HTML:

```
<h1>Welcome logged in user!</h1>
<script>
  let username = '\\';alert(document.cookie)//'
  alert('Hi there, ' + username)
</script>
```

The second issue is that the attacker can break completely out of the <script> context and write whatever HTML they want by prematurely closing the <script> tag.

http://insecure.example.com/login?username=</script><script>alert(document.cookie)</script>&password=123

This generates the following HTML:

```
<h1>Welcome logged in user!</h1>
<script>
  let username = '</script><script>alert(document.cookie)</script>'
  alert('Hi there, ' + username)
</script>
```

8. CSP: The following resources are blocked by the CSP:

```
<link rel='stylesheet' href='https://stylish.example.com/style.css' />
<script>alert('We have only the BEST memes!')</script>
```

9. HSTS Preload: TLS prevents man-in-middle attacks. DNS hijacking is a form of man-in-the-middle attack. Since the DNS hijacker does not know the legitimate secret key, it cannot successfully negotiate a TLS session with victim users. The HSTS Preload List ensures that the user's browser will only make connections to the server over HTTPS, even if they visit the site over unencrypted HTTP. The user's very first connection to the site is protected by TLS which ensures they'll only ever connect to the legitimate site.

10. Command injection: There are actually two issues with this server. Either one is an acceptable answer.

The first is a command injection vulnerability – the server does not sanitize the user input before using it in a shell command, which means the attacker can add whatever they want to the end of the command and it will be executed. For example, the attacker can provide a filename of "hello.txt; rm -rf /" to delete the contents of the whole server.

Even if the first issue is fixed by switching execSync to spawnSync which sanitizes user arguments, a second issue remains – a directory traversal attack. The server allows the user to read any file on the server, not just the files in the "static" folder. For example, the attacker can provide the filename '../../../etc/passwd' to read the Unix password file, which exists outside of the static folder. The server should ensure that any provided filenames do not include '..' which allows traversing upwards in the directory hierarchy.

11. Fingerprinting: 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)

12. Logic bug: The first issue – Should use app.post() app.delete() instead of app.get() since GET requests are not supposed to have any side effects and the browser may automatically prefetch them to improve performance, which might unintentionally delete the user's account. The second issue – there is a missing return statement after the first call to res.send() which allows fallthrough to the rest of the function. In other words, the account is always deleted even if the password is wrong.

13. Winter break: Any answer accepted!