CS 253 Sample Final Exam – Questions

True or False –

T 1. Cross-site scripting is a type of injection attack.

F 2. SQL is the primary language targeted by cross-site scripting attacks.

T 3. In a DOM-based XSS, the malicious input is injected into the client application at runtime in the user's browser.

F 4. You should set the "HttpOnly" flag in a cookie to ensure that the cookie is sent over an encrypted channel.

F 5. Persistent XSS (also known as "Stored XSS") occurs when a malicious user convinces a victim to send a request to a server with malicious input and the server echoes the input back to client.

F 6. The server can always trust HTTP headers such as the User-Agent header to correctly identify the browser in use because these headers are set by the web browser.

F 7. In general, it's safe to trust data that comes from the user in query parameters, form fields, headers, cookies, and file uploads.

T 8. The idea of defense-in-depth is to force the attacker to find multiple exploitable vulnerabilities in order to produce a successful attack; one attack on its own is not enough.

F 9. The goal of Time-based One-time Password (TOTP) codes is to protect the user from phishing attacks.

Short Answer (max 50 words) –

1. Which character is most likely to be used in an XSS attack that escapes out of an HTML attribute? Choose from: the single quote ('), the null byte, the less than sign (<), or the greater than sign (>). single quote
2. Name the three parts of a URL that are used to determine the URL's origin. protocol, hostname, port
3. Why is it a bad idea to allow servers software like Express, Apache, or Nginx to advertise the server name and version that is in use in HTTP response headers? For instance, Express sends the following header: "`X-Powered-By: express`" in all responses, unless this is explicitly disabled.

Free Response (max 150 words) –

Same Origin Policy 1: Explain how "origins" in the web browser are conceptually analogous to "processes" in operating systems.

Same Origin Policy 2: Which of the 4 HTTP requests from this page on https://example.com are allowed?

```
<!doctype html>
<html lang='en'>
  <head>
    <meta charset='utf-8' />
    <link rel='stylesheet' href='https://other1.com/style.css' />
  </head>
  <body>
    <img src='https://other2.com/image.png' />
    <script src='https://other3.com/script.js'></script>
    <script>
      const res = await fetch('https://other4.com/cool.mp4')
      const data = await res.body.text()
      console.log(data)
    </script>
  </body>
</html>
```

You can assume that the other sites do NOT send any special HTTP headers such as Access-Control-Allow-Origin, which are also known as "CORS" headers.

Fingerprinting: You're a web developer working for a national newspaper. The product manager informs you that a decision has been made to ban users of the Brave web browser because of its ad-blocking capabilities which are negatively affecting revenue. It is your responsibility to implement this functionality. Booo! Brave uses the same User-Agent header value as the Chrome browser, so it's not possible to distinguish Brave users by merely looking at this header value.

Propose a fingerprinting method you could use to distinguish Brave users from other browser users. It's okay if your method has some false positives, as long as it recognizes all Brave users.

XSS: There are some JavaScript functions that can never safely use untrusted data as input, even if escaped with our handy `htmlEscape()` function from Assignment 1. For example:

```
<script>
  window.setInterval('ESCAPED_USER_DATA_HERE')
</script>
```

Why is this the case?

XSS 2: Describe the heuristic that Chrome's XSS Auditor uses to detect XSS attacks. What were the limitations of the XSS Auditor?

CSP: Given the following CSP and HTML page, list which resources will be blocked from loading?

```
Content-Security-Policy: default-src 'none'; script-src 'self' 'unsafe-inline';
img-src *; style-src 'self' https:;

<!doctype html>
<html lang='en'>
  <head>
    <link rel='stylesheet' href='/style.css' />
    <link rel='stylesheet' href='https://stylish.example.com/style.css' />
    <style>body { font-size: 99px; }</style>
  </head>
  <body>
    <script>alert('Sup!')</script>
    <img src='https://images.example.com/foo.jpg'>
    <img src='https://images.example.com/bar.jpg'>
    <img src='/logos/large-logo.jpg'>
    <script src='/bundle.js'></script>
    <script src='https://random.example.com/analytics.js'></script>
  </body>
</html>
```

Command injection: The following Node.js program implements an HTTP server which accepts a user-provided filename and returns the contents of the specified file to the user, if it exists on the server. The file should only be returned if it exists in a folder named "static" where static files intended for viewing are stored.

```
const express = require('express')
const childProcess = require('child_process')
const app = express()

app.get('/', (req, res) => {
  res.send(`
    <h1>File viewer</h1>
    <form method='GET' action='/view'>
      <input name='filename' />
      <input type='submit' value='Submit' />
    </form>
  `)
})

app.get('/view', (req, res) => {
  const { filename } = req.query
  const child = childProcess.spawnSync('cat', [filename])
  if (child.status !== 0) {
    res.send(child.stderr.toString())
  } else {
    res.send(child.stdout.toString())
  }
})

app.listen(4000, '127.0.0.1')
```

There is a glaring security issue with the design of this server. What is the issue? How could the issue be fixed?

HSTS: What is the purpose of the Strict-Transport-Security header? What attack does it protect against?

CS 253 Sample Final Exam – Solutions

True or False –

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

Short Answer (max 50 words) –

1. The single quote

2. Protocol, hostname, port

3. Attackers can use the information about which server and version is in use to mount a targeted attack against your server. Additionally, if you're not running the latest version of the server software there may be publicly-known vulnerabilities that the attacker can leverage against the specific version you are running.

Free Response (max 150 words) –

Same Origin Policy 1: Just as an operating system process isolates different programs' address spaces from each other with the goal of preventing them from interfering with each other, whether maliciously or accidentally, the web browser isolates sites from different origins with the goal of preventing them from interfering with each other. In OSes, it's the kernel which enforces process separation. On the web, it's the browser which enforces origin separation.

Same Origin Policy 2: The first 3 requests are allowed because they are "simple" requests initiated by HTML elements such as <link>, <img>, and <script>. Furthermore, the data is not read directly by JavaScript on the page. The last request to https://other4.com 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.

Fingerprinting: Since the Brave browser is blocking ads, we could include an image in the page that looks just like an ad, i.e. uses common ad image dimensions (300 x 250, 728 x 90, etc.) or is named like an ad (ad.jpg, sponsor.jpg, etc.), or has HTML class names that ads commonly use (ad, advert, sponsored, etc.). Then we include JavaScript code that checks to see if the image successfully loaded. If it didn't load, then we know the user is using Brave and can redirect them to a page telling them they are blocked from using the site. (Btw, this is a common technique that real sites use to detect when users are blocking ads.)

XSS: Even if you escape the user data included in the first string argument to setInterval, you are still in trouble. This argument is a function or string which will be executed by the program. Even if you remove control characters like the single quote (') and so on from the input, you are still letting the attacker essentially run whatever code they want, except that it can't contain whatever characters you escaped. You're letting the attacker provide a string that is executed as code. This is the definition of XSS!

XSS 2: The XSS Auditor prevents reflected XSS attacks from running by detecting if a URL contains a string this is also present in the page's HTML. This pattern usually indicates an XSS attack. One limitation – attackers can snipe whatever legitimate JS they want out of the page by simply including it in the URL. Another limitation – sometimes sites legitimately include a string in e.g. a URL query parameter that also appears within the page and the XSS Auditor breaks these pages.

CSP: The following resources are blocked by the CSP:

```
<style>body { font-size: 99px; }</style>
<script src='https://random.example.com/analytics.js'></script>
```

Command injection: The server allows the user to read any file on the server, not just the files in the "static" folder. This is a "directory traversal attack". 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.

Note that since spawnSync was used instead of the dangerous execSync, the filename itself is sanitized and the attacker can't add whatever they want to the end of the command to cause it to be executed. If execSync had been used, the attacker could provide a filename of 'hello.txt; rm -rf /' to delete the contents of the whole server.

HSTS: The HSTS header ensures that the user's browser will only make connections to the server over HTTPS, even if they visit the site over HTTP, which is unencrypted. The user might visit over HTTP if they type in the URL by hand and forget to add "https://" to the front of the URL, or if they click a link to the site which uses "http://" instead of "https://". HSTS automatically upgrades the user's request to HTTPS in these scenarios.