Na'Sir Miller
ITWS2110
10/20/23

1.1 Describe in your own words how the web works! In as much detail as you can, describe **all** the sequences of events that take place from the time a user presses Enter on the keyboard after typing in www.rpi.edu into the address bar to when the webpage is finished rendering in the browser. Specifically, tell me in great detail the **two protocols** we discussed in class in action. (8 points)

1. A user looks for a site they want to find, and types in their query.
2. The search will then contain a protocol (https or https depending on its security certification), subdomain, the domain name (usually similar to query user inputs), the port number of the server, any path(s), the actual users' query and any fragments (all elements of the URL).
3. The TCP/IP (protocols) define how data should be sent through the internet, also splitting the data packets into smaller ones to ensure requests is fulfilled.
4. The Domain Name System then converts our URL/search to an IP address to be readable by computer(s) with the port number of the server, to send the user's requested website a HTTP request.
5. HTTP requests are made to the requested site (ww.rpi.edu), where the site then processes the requests, and sends back another request. (Three way Handshake)
6. In this new request, the content will be 200–the user has access to www.rpi.edu, otherwise a 400, 404, or another response will occur. If the latter occurs, the user may need to just re-query the website, because all the packets may have been loss (although somewhat unlikely).

1.2 Explain what is meant by a Uniform Interface in a REST API. (5 points)

The Uniform Interface is the interaction interface between the client (requesting party) and server (requested party) to exchange data. Furthermore, due to this sort of uniformity/should be consistency, the client should be able to navigate the entire API and consume the HTTP responders (that are being sent back to them). Additionally, this REST API constraint tries to bring a central, and consistent API architecture across the web to make data exchange safe, efficient, and effective—although all the opposite may happen.

1.3 Explain how your browser chooses which CSS rule to apply to a tag in the case where there are multiple rules that could apply. (3 points)

If there are two or more CSS rules regarding the same element(s), the highest specificity selector will have whatever said value is applied. Oftentimes, the recency of a selection of a property determines the value. So if I say,

```
div {
margin: 20px;
}
```

And then later state

```
div {
margin: 60px;
}
```

The more recent definition will be used when displaying the CSS, since it is the most recent occurrence. Additionally, any use of classes and IDs also creates additional priority; id's and classes take precedence over rules since they are often the most recently interpreted. Moreover, the aforementioned effects are calculated in the specificity algorithm to decide (consistently) what rules should apply to a often used element.

1.4 What command would you use to change the ownership of a file or directory on a Unix machine (such as your Azure VM)? Show me a complete command invocation to make a directory named `/var/www/html` be owned by a user named `callab5` and a group also named `callab5`. (4 points)

You would use chmod. For example,

```
sudo chown -R callab5:callab5 /var/www/html
```

3. Choose one of the attacks you learned about in the Google Gruyere activity and walk me through how and why that attack works, and what you can do in order to mitigate such an attack in your term project, explaining the mitigation and why it works to prevent the attack. You may use code snippets or pseudo-code to help explain things.

Cross-Site Scripting (XSS) is an attack/vulnerability where a malicious user may inject javaScript or some other web-renderable code to the website for it to be ran. What this can do, is truly dependent on the malicious user's desire. For example, if the user injects some console.log(data) they may be able to see some of RPM's user data. Where this can be especially worrisome is if the malicious user gets access to our users locations or payment info–if we even decide to do add such. In order to prevent a simple yet destructive attack like this, we will minimize "entry points." We will try to minimize where users can possibly put code to be read by the server. Additionally, we will have users put relative locations and not exact locations, or better yet have them not put a location period. And instead use a google maps api to find meetup locations to prevent attackers from vulnerable info. Moreover, for payment info, we would not store payment data and instead fully rely on Stripe and whatever security measures they suggest to

prevent randoms access to payment data. Moreover, we must develop a backend script that escapes user search queries and fragments to prevent URL injection, which is something we will look into as we integrate our frontend to the backend.