Question 1 ***Warmup: SOP*** **(15 min)**

The Same Origin Policy (SOP) helps browsers maintain a sandboxed model by preventing certain webpages from accessing others. Two resources (can be images, scripts, HTML, etc.) have the same origin if they have the same protocol, port, and host. As an example, the URL `http://inst.berkeley.edu/eecs` has the protocol HTTP, its port is implicitly 80, the default for HTTP, and the host is inst.berkeley.edu.

Fill in the table below indicating whether the webpages shown can be accessed by `http://amazon.com/store/item/83`.

Origin	Can Access?	Reason if not
http://store.amazon.com/item/83		
http://amazon.com/user/56		
https://amazon.com/store/item/345		
http://amazon.com:2000/store		
http://amazin.com/store		

Solution:

Origin	Can Access?	Reason if not
http://store.amazon.com/item/83	No	different host
http://amazon.com/user/56	Yes	
https://amazon.com/store/item/345	No	different protocol
http://amazon.com:2000/store	No	different port
http://amazin.com/store	No	different host

Question 2 ***SQL Injection*** **(15 min)**

(a) Explain the bug in this PHP code. How would you exploit it? Write what you would need to do to delete all of the tables in the database.

```
$query = "SELECT name FROM users WHERE uid = $UID";
// Then execute the query.
```

(Here, `$UID` represents a URL parameter named `UID` supplied in the HTTP request. The actual representation of such a value in PHP is a bit messier than we've shown here. We leave out the syntactic details so we can focus on the functionality.)

(b) How does blacklisting work as a defense? What are some difficulties with blacklisting?

(c) What is the best way to fix this bug?

Solution:

(a) The bug is that the `uid` parameter can be interpreted as a command when properly formatted. For example, to delete the `users` table, pass in the following as the `uid`:

```
0; DROP TABLE users;
```

(b) **Blacklisting** means escaping what you consider "dangerous" characters – essentially characters that can be used to change control flow or be interpreted as commands rather than as data (e.g., quotation marks and semicolons). A difficulty in blacklisting is that it is all too easy to forget to avoid one dangerous character, which leaves a vector of attack.

(c) In this case, a simple fix would be to use a **whitelist** since `uid` only needs digits. In essence, you are constraining the type of `$UID` to an integer. Such a whitelisting approach can also work for strings, but is prone to errors. See below for a better solution. The underlying issue is that data can be interpreted as a command. The solution to this general issue is to separate the *parsing* of the query from the *execution* (when the data is supplied). ***Prepared statements*** (or *parameterized queries*) offer exactly this. The SQL expression is only parsed once, with placeholders for data. In a second step, the placeholders are replaced with the user input, without changing the intent of the SQL expression. Consider the following example:

```
$query = $db->prepare('SELECT name FROM users WHERE uid = :user');
$query->execute(array(':user' => $UID));
```

The first line defines the SQL expression with a placeholder "`:user`" that is substituted with user input in the second line. Note that the substituted input is *not* parsed as SQL anymore as this already happened in the first line. Therefore an attacker cannot provide bogus SQL commands because they will only be interpreted as data that is bound to the variable `:user`.

Question 3 ***Clickjacking*** **(10 min)**

In this question we'll investigate some of the click-jacking methods that have been used to target smartphone users.

(a) In many smartphone browsers, the address bar containing the page's URL can be hidden when the user scrolls. What types of problems can this cause?

> **Solution:** If the real address bar is hidden, it's much easier for an attacker to create and place their own on the website, fooling victims into thinking they're browsing on sites they aren't. JavaScript can scroll the page, hiding the address bar as soon as the page loads, allowing an attacker complete freedom to place a fake address bar.
>
> For more info, check out https://www.usenix.org/legacy/event/upsec/tech/full_papers/niu/niu_html/niu_html.html (section 4.2.2)

(b) Smartphone users are used to notifications popping up over their browsers as texts and calls arrive. How can attackers use this to their advantage?

> **Solution:** By simulating an alert or popup on the website, an attacker can fool users into clicking malicious links. This can allow attackers to pose as phone applications such as texting apps or phone apps, which enables phishing.

(c) QR codes haven't taken off and become ubiquitous like some thought they would. Can you think of any security reasons why this might be the case? (If you aren't familiar with QR codes, ask another group!)

> **Solution:** QR codes placed in public places are perfect targets for people with malicious websites. They can post their own, pretending to be links to useful websites, and instead linking to phishing sites. Or, they can modify and paste over existing codes, which only keen observers would notice.