Homework 2: Caught in the Web

Due: Thursday, March 14 @ 11:59 pm EST

Overview and instructions

This homework has 3 problems. All students must complete all problems; there is no extra component for CS1620/CS2660 students (good luck on Bob's Router!).

Note on collaboration

You are welcome (and encouraged!) to collaborate with your peers, but the solutions you write down must be **your own work** (ie, written by you). You are responsible for independently understanding all work that you submit—after discussing a problem as a group, you should ensure that you are able to produce your own answers independently to ensure that you understand the problem. For more information, please see the course Collaboration Policy.

In your submission, we ask that you include a brief *collaboration statement* describing how you collaborated with others on each problem—see the next section for details.

How to submit

You will submit your work in PDF form on Gradesope to the assignment labeled **"HW2"**. Your PDF should conform to the following requirements:

- **Do not** include any identifying information (name, CS username, Banner ID, etc.) in your PDF, since all homeworks are graded anonymously
- Each problem should start on a separate page. When you submit on Gradescope, you will be asked to mark which pages correspond to which problem
- At the start of each problem, write a brief *collaboration statement* that lists the names and CS usernames of anyone you collaborated with and what ideas you discussed together
- If you consulted any outside resources while answering any question, you should cite them with your answer

Problem 1: Cookies at Blue University

(15 pts) Blue University runs a web server at `blue.university`, where it hosts different pages for students and courses. Here's what you know about it:

1. Each student and staff member can upload web pages to a directory on the webserver to host a personal website. For example, user `alice` can upload any files to the server and make them available at `http://blue.university/people/alice/`
2. The webserver is pretty old, and allows users to view pages using both HTTP and HTTPS, ie. at `http://blue.university` and `https://blue.university`, respectively. Data sent via HTTPS is *encrypted in transit*, whereas HTTP is a plaintext protocol.[1].

In addition, the cs666 course staff have implemented an assignment submission system at `http://blue.university/courses/cs666/submissions`. Students taking cs666 provide an email and password to log in to the submission site. The website then issues authenticated users a cookie which contains a session ID that allows the user to authenticate themselves without having to type in their password again for 24 hours. You view the cookie in your browser and it looks like this:

```
Name:     sessionid
Value:    [random session ID...]
Domain:   blue.university
SameSite: Strict
```

Based on what you know about the Blue University website and the new cs666 submission system, consider the following questions. For each part, your answer should be no more than 1–2 sentences.

Part a) Bob is a cs666 staff member. Eve manages to eavesdrop on Bob's network connection while Bob visits the site at `http://blue.university/courses/cs666/submissions`. Can Eve acquire Bob's `sessionid` cookie for the submission system? Explain why/why not.

Part b) Alice can create a custom personal webpage at the path `http://blue.university/users/alice/`, but cannot change anything else on the `blue.university` webserver. Can Alice acquire Bob's `sessionid` cookie for the submission system? If so, outline how Alice might set up her webpage to steal the cookie; if not, explain why.

Part c) The Blue University administrators now consider various modifications to the site's cookie scheme:

(i) Suppose the administrators add the `Secure:TRUE` property to the cookie. First, identify what Bob has to change in the way he visits the submission site. Then, answer: does this change any of your answers to part (a) or part (b)? Why or why not.

(ii) Suppose the University adds the `HttpOnly:TRUE` property to the cookie *instead of the `Secure` property*. Does this change any of your answers to part (a) or part (b)? Explain.

Hint: For a concise description of these cookie attributes, we recommend this link: `https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies`. You can also find information in lectures 7–11.

Part d) For this part, consider Alice, the adversary from part (b), and ignore the modifications made to the cookie scheme in part (c).

(i) Suppose the administrators change the URL of the submission system to `cs666.blue.university/handin` and change the `Domain` property of the cookie to `cs666.blue.university`. Does this change your answer to part (b)? Explain.

(ii) Suppose the University decides to keep the domain of the system as `blue.university` (and keeps the `Domain:blue.university` property). Instead, they decide to put each users page on its own subdomain—for instance, Alice's user page is now located at `alice.blue.university`. Does this change your answer to part (b)? Explain.

[1] We'll learn more about HTTPS/TLS in a few lectures—for now, this is all we need to know about it.

Problem 2: Getting more from markdown

You are on the engineering team for a company that builds an web application for writing documentation: users write documentation in markdown (a lightweight text markup language), which your application renders as a fancy, publicly-viewable webpage.

Consider the following questions about ways to extend your applications. For each part, your answers should be no more than 1–2 sentences.

Part a) Your users are constantly submitting feature requests for new and more expressive ways to write markdown. In a meeting, your company's customer experience team gives a report on these feature requests and suggests two new features for your team to implement:

(i) New syntax to highlight super-important text in red, which would generate HTML like this:

```
<span style="color:red">markdown from user here</span>
```

(ii) New tags to let users add some of their own Javascript and CSS styling so they can format their own text, freeing your team from needing to write features like (i) ever again!

As someone who has a background in web security, how would you respond? Based on what you've learned about in this class, which feature(s) would you feel comfortable adding to your app from a security standpoint?
More concretely: In 1–2 sentences, pick which feature(s) are problematic explain your reasoning by stating the potential risks and the amount of work that might be required to implement them safely. Write your response as if you were talking to your coworkers in a meeting: to be most effective, it should be *concise, polite, and targeted to an audience with only a general CS background*—not everyone has the security knowledge you do!

Part b) Say your team is tasked with implementing feature (i) from part (a) and investigates how to actually make the changes. Your app relies on a popular, open-source library called `libmarkstuff` that parses the markdown and outputs HTML. It turns out that in order to implement feature (i), you would need to modify the code for `libmarkstuff`, rather than your own application's code. To resolve this, your coworker suggests making a private fork of the `libmarkstuff` repository that includes your custom feature (and any other features you might add later).
Does this add to the risk level for implementing the feature? Explain why or why not.

Problem 3: Tracking without cookies

While many sites use cookies to track users and store information about their browsing habits, there exist other techniques that are not as easy to detect and counter by the user.

Consider the following questions–for each part, your answer should be no more than 1–2 sentences.

Part a) A classic non-cookie tracking technique is to use a *tracking pixel*, where websites embed a tiny $1x1$ pixel image from an analytics service in their webpages. How could this image alone be used to track users across different websites?

Part b) In modern times, *browser fingerprinting* is a broad class of strategies for sites to gather various information about a user's browser to uniquely identify a specific user, allowing sites to target ads and track individual users without ever storing any cookies. In this problem, you'll view your own browser fingerprint! To do this:

- In an Incognito/Private Browsing window, go to `https://amiunique.org` and click **"View Fingerprint."** This page runs several tests using Javascript to gather information about your browser and the features it supports, combining them into a unique fingerprint.
- Look over the various data used to generate your fingerprint. For each feature, the report shows the percentage of users that share the same value for that feature. Scroll through the features and list **three** features with low similarity scores that you found interesting or surprising. (For more info on each feature, click the "i" button on the left.)
- In a sentence or two, consider: why do browsers have this much identifiable information? Is there anything you might be able to do to avoid having a unique fingerprint?
  (There are no correct answers here; we're just looking for something thoughtful.)