Homework 05

Due: 11:59 pm, Thursday March 3

Please hand in your homework as a single PDF file named hw05.pdf by running cs166_handin hw05 in a folder containing only that file on a department machine. While we will not take points off for turning in extraneous files, it makes our lives difficult. Handins that are not in PDF format will not be accepted.

Web Technology

Problem 1

For this problem you may want to look up details online, but make sure to cite your sources. See the collaboration policy for details.

In this problem, you’ll explore some of the details of HTTP. In order to do this question, you’re going to need to be on a Unix machine (either your own Mac or Linux computer, or ssh’d into the department), and you’re going to need to have a program listening on a network port for TCP connections. For example, to listen for connections on port 1234, you can do:

$ nc -l -p 1234 # Debian (which runs on the CS department machines)
$ nc -l 1234 # OS X or Ubuntu

This will wait for new TCP connections on that port, and will quit once the connection has closed - to handle multiple connections, you’ll need to run the command multiple times. Everything that is sent over the connection will be printed to the terminal, and you can respond (send data the other way over the connection) by sending data on nc’s stdin (e.g., typing in the terminal).

a) For each of the following user agents, connect to this port using HTTP (i.e., http://localhost:1234):

- curl (command-line utility)
- wget (command-line utility)
- normal browser (Chrome/Chromium, Firefox/Iceweasel, etc; specify which one you use)

For each, answer the following questions:

i. Submit the text that the user agent sends to the server as its request as part of your answer to this homework (if you’re using LaTeX, consider the verbatim environment).

ii. Describe what each line of the request means

b) What differences do you see between the requests performed by the user agents you tested? Why do these differences happen?
Problem 2

Give a detailed step-by-step description of what happens when you type http://cs.brown.edu/courses/cs166 into your browser's navigation bar and hit enter. Since we haven't yet covered networking, you may assume that anything more detailed than “the computer connects to the computer with the IP address 1.2.3.4” just works and doesn’t require explanation. Give a plausible example of what the HTTP request and response might look like (including headers - see here[1] if you need a reference). Be sure to include any actions involved in visiting the web page including DNS requests and whatnot. Be sure to include details on how the web server determines what its response should be (how does it know not to return a 404 Not Found error? how does it know what the contents of the page should be?). After the response is returned and interpreted by the browser, are any more web requests made? What could cause them?

Problem 3

While many sites use cookies to track users and store information about their browsing habits, there exist more sophisticated techniques that are not as easy to detect. This is called browser fingerprinting, and it allows sites to target ads and track users without ever storing any cookies on the client machine.

1. By just visiting a website, what are some possible ways your browser could reveal information about you that could be used to fingerprint you? (Hint: what might make your browser different than someone else’s?) Are any of these mitigated by using a private browser session?

2. A common cookie-less tracking technique is called a tracking pixel, where a single pixel is loaded onto your webpage from a different server. How could the loading of this image alone be used to track users?

3. An ETag is an identifier (which the browser may treat as a meaningless string) that a server uses to uniquely identify a particular version of a resource for the purposes of caching. When a resource is cached by the browser, any associated ETag is cached as well. When a future request is made for the cached resource, the browser will include the ETag, which allows the server to identify the version of the resource in the browser’s cache. It can then respond with the current version or, if the cached version is up-to-date, simply respond with response code 304 (“Not Modified”), indicating to the browser that it should simply use the version in its cache.

Explain how a server could abuse ETAGs to track users. That is, explain how a server could ETAGs to be able to tell that two HTTP requests came from the same user, even if that user had deleted all of their cookies.

Problem 4

For this problem, it may be useful to consult the Request methods section of Wikipedia’s article on HTTP[2]

Many web pages utilize two common HTTP request methods, GET and POST, for various API calls.

a) When is it more appropriate to use a GET request, and when is it more appropriate to use a POST request?

b) When sending data to a server, GET and POST requests store data in different places in the HTTP request. Explain.

c) What are some ways that sensitive data could be accidentally leaked when using a GET request?

d) There is no way, by clicking on a link or typing a URL into your browser’s navigation bar, to initiate a POST request - both of these will initiate GET requests. Given the semantics of GET vs POST (see the referenced article for details), explain why it’s a security feature to prevent either of these (clicking on a link or typing a URL in the navigation bar) from initiating POST requests.

Pre-work

These are questions to which we do not expect you to know the answer. We don’t expect you to know the details underlying the questions. As long as you state any (reasonable) assumptions you are making, and be explicit about what your understanding of the background material is, we will give you most of the credit so long as the reasoning based on that understanding is sound. Think of it like a math problem - show your work, and we won’t take off too much for minor calculation mistakes.

Web Security

Problem 5

Many web sites written in interpreted programming languages place code files throughout a directory tree which mirrors the URL structure of the website. For example, if the web root of www.foo.com is /var/www, requesting http://www.foo.com/about.php might execute the PHP script at /var/www/about.php, and requesting http://www.foo.com/users/list.php might execute the PHP script at /var/www/users/list.php. Additionally, some websites store uploaded files in directories inside the web root so that they can be requested. For example, a web server might place uploads in /var/www/uploads, so that if you uploaded bar.jpg, requesting http://www.foo.com/uploads/bar.jpg would result in being served the image stored at /var/www/uploads/bar.jpg. How might a malicious user gain unauthorized access to a web server employing both of these techniques?