University of Chicago
CMSC 23200/33250: Introduction to Computer Security, Winter 2020
Course Staff: David Cash, Alex Hoover, Rohan Kumar, Blase Ur, Valerie Zhao
Revision 1 (Tuesday, Feb. 11th at 10:03pm) to change the output format of your clusters for Problem 3 and Problem 4 to be easier to grade.
Revision 2 (Wednesday, Feb. 12th at 12:15am) to add URLs for the queueing system.
Revision 3 (Wednesday, Feb. 12th at 4:49pm) A sentence that should have been in 3.8 was accidentally in 3.9. Fixed.
Revision 4 (Thursday, Feb. 13th at 11:49pm) Just adding extra URLs as we've launched additional servers for your testing.
Revision 5 (Saturday, Feb. 15th at 2:50pm) Updated submission guidelines for 3.8.

Assignment 5: Web Tracking

Due at 11:59pm Monday, February 17

Introduction

In previous assignments, you attacked the security of a system. Here, you will attack users' privacy, further learning about how easy it is to track people online to enable targeted advertising or even more nefarious uses. This assignment lets you more deeply explore the different techniques we discussed in class used to track users on the web. In particular, you will set up a web server on a Linux virtual machine (VM) and create your own tracking infrastructure. When you are ready to test your infrastructure, you will indicate this on a machine we control, causing web traffic to visit your site. Because the visits incur non-trivial overhead on our end because of the way we are isolating browser visits between students, we have created a series of test cases that each cause a small amount of traffic focused on a single tracking-related attribute. In the final step of the project, you will indicate that you are ready for a full test, which will fire up our full fleet of browsing traffic.

Like many of our previous assignments, you will need to gain a bit of experience with a number of commonly used technologies to complete this assignment. This approach helps you explore computer security and privacy on an intellectual level while at the same time gaining practical computing skills. On the skill side, to complete this assignment will probably require you to learn at least a little bit of Linux (Ubuntu) server administration on the command line, Apache (or some other web server) configuration, MySQL (or some other database), JavaScript, and either PHP or Python to connect to the database. You will also have to think deeply about online tracking and how to leverage both overt and subtle information leaks to infer when website page visits are from the same person.

Before getting to the problems, we discuss the rules for this assignment and how you can access the assignment infrastructure for your tracking system.

Rules

Collaboration policy. Please respect the following collaboration policy: You may discuss problems with up to 3 other students in the class, *but you must write up your own responses and*

your own code. You should never see your collaborators' writing, code, flags, or the output of their code. At the beginning of your submission write-up, you must indicate the names of your (1, 2, or 3) collaborators, if any. You may switch groups between assignments but not within the same assignment.

Sources. Cite any sources you use. You may Google liberally to learn about how to configure Apache, use JavaScript, use MySQL (or another database of your choosing), write to the database (e.g., using PHP), or navigate the command line. You may also Google liberally to learn the concepts underlying fingerprinting. **You may not, however, use or consult existing libraries for browser fingerprinting. Your solutions, as well as the precise way you turn the concepts into code, should be your own.** You **must**, however, note at the end of your writeup for each task the sources you referenced. Please note any you found particularly helpful since we might include them in future versions of this writeup.

Campuswire. We encourage you to post questions on Campuswire, but do not include any of your code in the Campuswire posts. If you have a question that you believe will reveal secrets you have discovered while working on the assignment, post privately to just the instructors. If you have a question that you believe will be of general interest or clarifies the assignment, please post publicly. If you are uncertain, post privately; we will make public posts that we believe are of general interest.

Outside attacks. Your approach for this assignment should be that which is discussed below. Do not attempt to compromise our server, sniff your classmates' network traffic, or do other nefarious things. You will not receive credit for breaking into the server. In fact, you will lose credit for doing so. **You are not permitted to view or copy what your classmates are doing. All servers have visit logs that the course staff can read.**

Grading. Responses will be graded for correctness and clarity.

Assignment Tech Set-Up and Overview

This assignment involves *setting up your own server* and configuring it (both through installing and configuring existing software packages and through writing your own code) to track website visitors.

Our CS department Tech Staff (Bob, Colin, Phil, and Tom) has generously created a virtual machine for each student in the class. These virtual machines run Linux, specifically Ubuntu 18.04.4. Furthermore, they have created DNS entries for each of you so that `<cnetid>-cs232.cs.uchicago.edu` resolves to your VM. For example, Blase's VM would be accessible through a web browser at `http://blase-cs232.cs.uchicago.edu`.

Instead of using a password to authenticate, you will log in using an ssh key pair. We had you create an ssh key pair and give it to us on `https://uchicago.co1.qualtrics.com/jfe/form/SV_ehcPpMC1tlN5RgF`

To log in,

- (Mac/Linux) Run "`ssh <cnetid>@<cnetid>-cs232.cs.uchicago.edu`" (so, for example, Blase would do "`ssh blase@blase-cs232.cs.uchicago.edu`"
- (Windows) Probably the easiest way is to use Putty. The tutorial at `https://www.howtoforge.com/ssh_key_based_logins_putty` seems reasonable from what we remember

from using Putty. Note that the tutorial is 4 pages long, though the next button might not be obvious. The server name is "`<cnetid>-cs232.cs.uchicago.edu`", whereas the tutorial uses `192.168.0.100`. Note that in Step 4, you won't actually be able to connect because you don't know the password. Step 5 is what you alread did to generate and send us your key. We did Step 6 for you (though check it out, because you can do the same and avoid needing to type your password to log into `linux.cs.uchicago.edu`). You can skip Step 9 and Step 10.

Choice of Languages and Tools

As in past assignments, you may implement your code in whatever programming language and environment you choose. Below, we give detailed pointers towards tutorials for doing it in the way we think is easiest. If you prefer nginx to Apache or MongoDB to MySQL, etc., feel free to do what you want. Note, though, that we can provide the most detailed assistance for our recommended approach.

What and How to Submit

You will submit a set of files:

1. A file `<your cnetid>-writeup.pdf/txt` that contains the information requested by each problem/sub-problem. This file is shared across all four problems.
2. A file `<your cnetid>-problem3.txt` (it must be a text file) with your clusters for the sub-problems in Problem 3. There should be exactly 9 lines in this file. We detail the required format below.
3. A file `<your cnetid>-problem4.txt` (it must be a text file) with your clusters for Problem 4. There should be exactly 1 line in this file. We detail the required format below.
4. All individual files that you generated in solving the problems. These are discussed for each individual problem below. Your shared writeup file should make clear (e.g., by referencing file names) which files were used to solve which problems.

You will upload these files to Canvas. You may zip/tar/gzip/7zip/etc. them or submit them individually.

Final Notes and Hints

This assignment was built this year by Blase and Rohan fresh for this class, while CS Tech Staff created and deployed the VMs and DNS entries. It is the first time it has been deployed in any setting. Please let us know if you find bugs or need instructions fleshed out by posting on Campuswire.

1 Problem 1: Setting up a web server/certificate (10 points total)

1.1 Log in

Your first job in this assignment is to set up your VM as a web server. To log into your server (assuming you've provided us your public key and given us time to set it up), follow the instructions

above.

1.2 Install Apache as your web server (3 points)

Great, now you're logged into your VM and seeing an Ubuntu command prompt. Now, you need to install a webserver (we'll use Apache) and a way to create databases (we'll use MySQL). On the command line of your VM, follow these steps, which are based on `https://phoenixnap.com/kb/how-to-install-lamp-stack-on-ubuntu`, yet adjusted for what we'll need.

1. `sudo apt-get update`
2. `sudo apt-get install apache2`
3. `sudo rm /etc/apache2/sites-available/default-ssl.conf` (to remove a file we don't need)
4. `sudo vi /etc/apache2/sites-available/000-default.conf`
5. This brings you into a config file, where you'll notice the following line commented out: `#ServerName www.example.com`. Uncomment this line and change it to match your subdomain (e.g., `ServerName davidcash-cs232.cs.uchicago.edu`)
6. `sudo service apache2 restart` (to restart Apache)
7. Verify that this worked by opening up a browser on your own computer and going to your server (e.g., `http://davidcash-cs232.cs.uchicago.edu`). You should see the Apache2 Ubuntu Default Page.

1.3 Get a certificate (3 points)

Ok. Now let's get an HTTPS certificate for your server using Let's Encrypt. Run the following:

1. `sudo apt install python-certbot-apache`
2. `sudo letsencrypt`
3. When you are asked about redirecting to HTTPS, choose "Redirect," which is choice 2.
4. `sudo service apache2 restart` (to restart Apache)
5. Verify that this worked by opening up a browser on your own computer and going to your server (e.g., `https://davidcash-cs232.cs.uchicago.edu` ...notice the s in https). You should still see the Apache2 Ubuntu Default Page, but now you're connected over HTTPS.

1.4 Install PHP

- `sudo apt-get install php libapache2-mod-php php-mysql`

Instead of following this step, feel free to substitute whatever you want to use in place of PHP (e.g., Python). We chose PHP since you saw it briefly in the last assignment.

1.5 Install MySQL as your database and create your database schema (4 points)

Next we need a database. We suggest using MySQL. Follow the steps on `https://www.digitalocean.com/community/tutorials/how-to-install-mysql-on-ubuntu-18-04` to install MySQL on your VM. Some quick notes. When you do `sudo mysql_secure_installation` you should say yes to everything they ask and, as part of those steps, create a root password. Save it somewhere. In addition, complete Step 3 even though the tutorial calls it optional. By the end of Step 3, you'll have created a user for your MySQL instance.

Now you need to create a database and probably create a table in the database. You're going to use this database in subsequent parts of the assignment to store the data, so you should read the rest of the assignment first to guide your design of a table schema. You can initialize the database from the mysql console. As you did above, run `mysql -u <theUserYouCreated> -p` and then follow the directions in `https://www.tutorialsweb.com/sql/working-with-mysql.htm` starting at Step D. For a more comprehensive tutorial, check out `https://www.tutorialspoint.com/mysql/index.htm`. You could instead do all of the database administration directly in PHP without opening the console: `https://www.w3schools.in/mysql/php-mysql-create/`

What to submit.

We can just visit your VM to verify that you did most of the things we asked. That is, if we can't reach the website hosted on your VM, we'll assume you did not do this part. In other words, keep your VM up so that we can verify you finished this step.

In your shared write-up file, `<your cnetid>-writeup.pdf/txt`, present the schema (column headers) you chose for your database and justify this design.

2 Problem 2: Be able to write to your database (10 points)

In the remainder of the assignment, you will be writing HTML/JavaScript that tracks visitors to your website. That code will track visitors on the client side (their own computer), so you will need to send the information back to your server. Similar to how you had a sink (that we provided) in the previous assignment to collect data, here you will need to write code that takes information sent by your JavaScript code and writes it to the database you just created.

One of the first things you'll probably want do is to take ownership of the directory for your webpage: `sudo chown -R <cnet>:<cnet> /var/www/html`

Similarly, when you upload files, you may want to make sure the permissions are correct (e.g., PHP files need to have execute permissions for all users since your Apache server runs as www-data, not your user. If the permissions are wrong, fix them using *chmod*. Note that we didn't have any problems with the default permissions, so you may be able to skip this step.

We recommend thinkig about your design in the following way: Someone visits your webpage. They make GET requests to your server for HTML and JavaScript code you will have written. This HTML and JavaScript code will do the profiling described below in Problem 3. Now, you have fingerprinted the user on their own computer, but you need to get the data back to your server. Therefore, as part of this JavaScript code, make an AJAX request to your server sending back the values you have profiled. (For developing and testing this current problem, simply use placeholder values instead of profiles.) You'll need some code running on your server (e.g., PHP, Python, etc. that is somewhat similar to sink.php we provided you for the last assignment) to retrieve these values and (different from sink.php) write them to the database you created in the last problem.

Once you have established that people visiting your webpage have their (placeholder) profile sent back to your server and written to your database, you are ready to proceed.

What to submit.

Submit all code you wrote for this part.

In your shared write-up file, `<your cnetid>-writeup.pdf/txt`, briefly discuss your design and give us pointers to the file names of the code that accomplished each part of your design.

3 Problem 3: Tracking particular attributes (65 points total)

This problem is the main point of this assignment. Here, you will explore using a number of different methods, from cookies to browser fingerprinting, to track visitors to your site. For each of these sub-problems, we have written test cases. When you have updated your HTML / JavaScript to profile people in the way specified by each sub-problem and deployed it on your server, you can test it with your own browser by visiting that page. Once that seems to be working (comparing your browser configuration to what you see being written to your database), you can start request traffic from our test fleet. Go to `https://blase-cs232.cs.uchicago.edu/` or `https://insecurityclass.cs.uchicago.edu/` or `https://rohankumar.uchicago.tech/` and enter your cnet id and the desired test case into our queueing system. Our system sends traffic on the order of a handful of visits to your server.

As described below, you will be turning in your profiling code as well as a clustering of pages visits based on your profiles. Each visit will be labeled between **1** and **500** based on the file name visited. In other words, there are 500 possible files we could be visiting, and your server should be able to fulfill GET requests for any of those 500. To make this easier for you, we have distributed a short script called **duplicate.sh** that, if you name your HTML file **track.html**, will make 500 identical copies of it numbered **1.html** through **500.html**.

3.1 Basic cookie tracking (5 points)

To detect which page visits are associated with each other (from the same user), set and get cookies.

Test using the "track using cookies" test case. (Our test case for this problem causes 3 visits to your page, 2 of which are from the same person.)

3.2 Track by IP address (10 points)

To detect which page visits are associated with each other (from the same user), track the IP address of the visitor. Note that you may have to make some sort of external call to figure out their IP address; the client's IP address isn't natively available in JavaScript.

Use the "detect different IP addresses" test case.

3.3 Track visitors by based on window size (5 points)

To detect which page visits are associated with each other (from the same user), examine the size of the window in which the visitor is viewing the page.

Test using the "detect different devices (user agents) and window sizes" test case.

3.4 Track visitors by fingerprinting based on the user agent string (10 points)

To detect which page visits are associated with each other (from the same user), rely on the client's user agent string.

Test using the "detect different devices (user agents) and window sizes" test case.

3.5 Track visitors by fingerprinting based on whether cookies are enabled (5 points)

To detect which page visits are associated with each other (from the same user), check whether the client is letting you set cookies at all.

Test using the "detect whether cookies are enabled" test case.

3.6 Track visitors by fingerprinting based on whether the DNT header is enabled (5 points)

To detect which page visits are associated with each other (from the same user), check whether the client is sending the Do Not Track header.

Test using the "detect whether DNT is enabled" test case.

3.7 Track visitors by fingerprinting based on whether popups are enabled (5 points)

To detect which page visits are associated with each other (from the same user), check whether the client is blocking pop-up windows.

Test using the "detect whether popups are enabled" test case.

3.8 Track visitors by fingerprinting based on the fonts installed (15 points)

In class, we discussed how different fonts render text as different sizes on screen and that the browser can view the computed size of different elements on a webpage. Exploit this fact to detect which fonts the user has installed.

To make this task more tractable, assume that these are the only five fonts that are possible (so consider only whether or not the user has these five fonts installed):

- Abyssinica SIL
- DejaVu Sans
- GFS Baskerville
- Liberation Sans
- Roboto

Test using the "detect different fonts installed" test case. **For this sub-problem only, all 5 test case visits are different. In your write-up, you should provide us a mapping of the visit number (e.g., 100) and which of the above fonts are NOT installed (e.g., DejaVu Sans).**

3.9 Track visitors using multiple features (5 points)

To detect which page visits are associated with each other (from the same user), use a combination of all of the features above.

Test using the "multiple features at once" test case.

What to submit.

First, submit all code you wrote for this part. This might be as simple as uploading substantially updated versions of the files from Problem 2, though with different file names.

Second, report the page-visit clusters you find in each subproblem in a file called `<your cnetid>-problem3.txt`, structured as follows. For each sub-problem of Problem 3, report each of your page-visit clusters **based only on the specific metric investigated in this sub-problem**. Each sub-problem should be on its own line, and the lines (sub-problems) should be in the same order as in this document. As a result, there should be exactly 9 lines in this file. If you do not finish a sub-problem, leave its line blank. Page numbers within a cluster should be delimited by a comma, and clusters on a line should be delimited by a single space. Within each cluster, page numbers should be sorted in ascending order. Clusters on a line should be sorted in ascending order based on the first page in the cluster (the one in that cluster with the smallest page number). Leave out .html from all page visits.

For example, if you received visits to pages 53.html, 20.html, 400.html, and 32.html for Problem 3.1 (these are not the real page visits) and you determine **by this metric alone**, 53 and 32 are associated and 20 and 400 are associated, then the first line of your file would be "20,400 32,53". Note that there is exactly one space in this line. Subsequent sub-problems should have similar lines. Note that different clusters may have different numbers of pages in them, and different problems may have different number of clusters and pages. **Note that this includes subproblem 3.8 (fonts). For subproblem 3.8, in your write-up, you should specify which font(s) are missing on which page-number visit (as described later).**

Third, in your shared write-up file, `<your cnetid>-writeup.pdf/txt`, respond to the following points for each sub-problem. **Clearly delineate (using bold section headers or comparable) your response to the points below for each sub-problem:**

1. Describe your approach to putting this sub-problem's method of tracking into action. Write around one brief sentence.

2. For each sub-problem we want you to reflect on why this type of tracking remains possible in browsers. In two or three sentences, describe how this type of tracking could be stopped (e.g., what features/calls would need to be eliminated from browsers), if at all, as well as what desirable benefits to websites (if applicable to that method) would be lost as a result of eliminating that functionality. If it were up to you, would you eliminate those features to prevent this type of tracking? Why/why not?

3. *For subproblem 3.8:* Write which font(s) are **not** installed on each different visit you receive in this subproblem. For example, if the visit to 500.html does not have Roboto and the visit to 300.html does not have DejaVu Sans, write "300: DejaVu Sans, 500: Roboto".

4 Problem 4: Tracking particular attributes (15 points total)

Finally, unleash the full fleet of visits by adding yourself to the queue on `https://blase-cs232.cs.uchicago.edu/` or `https://insecurityclass.cs.uchicago.edu/` or `https://rohankumar.uchicago.tech/` . Using the full set of features from the previous problem, cluster the page visits that appear to have come from the same person.

What to submit.

You may be able to simply reuse the code from the last problem. If you did update your code, submit the updated code with a different file name.

As for Problem 3, submit a file called `<your cnetid>-problem4.txt` that contains the clusters you found. This file should have exactly one line. Clusters should be delimited by a single space, and pages within a cluster should be sorted and delimited by a comma. Follow the sorting conventions from Problem 3.

Finally, in your shared write-up file, `<your cnetid>-writeup.pdf/txt`, write at most a paragraph discussing how stable the different features you were tracking are over time. That is, which features might change for a given user a month from now? How might these changes manifest, and could you perhaps still associate the pre-change and post-change visits?