Measuring the Prevalence of Online Tracking

Due date: February 23 11:59 PM

Overview:

In this project, you will conduct measurements to analyze the impact of using ad blocking extensions on randomly selected 100 websites with advertisements (list and random website selection code provided with the project). Specifically, you will compare the third-party[1] HTTP(S) requests, third-party cookies, and third-party JavaScript API calls in the "ad blocking mode" (ad blocking extension enabled) and the "vanilla mode" (ad blocking extension disabled).

You will use an open-source tool called DuckDuckGo Tracker Radar Collector (https://github.com/duckduckgo/tracker-radar-collector) to automatically visit websites and record HTTP(S) requests, cookies, and JavaScript API calls. You will use uBlock Origin extension for ad blocking. Instructions to add uBlock Origin are provided below.

Data collection with DuckDuckGo Tracker Radar Collector:

Vanilla mode:

1. Clone this project locally:
    a. `git clone https://github.com/duckduckgo/tracker-radar-collector.git`
2. Go to cloned repo:
    a. `cd tracker-radar-collector`
3. Install all dependencies:
    a. `npm i`
4. Download the python program `(random_websites.py)` and the websites list `(website_list.txt)` that will extract 100 websites selected at random from the `website_list.txt`
    a. Replace the placeholder WASHUID with your own and run the program (the purpose of this step is to tailor the random-number generation to each student)
5. Copy the `random_websites.txt` into the cloned repository.
6. Replace the crawler.js file provided by Tracker Radar with the crawler.js provided with the project. It will run the browser in a window in the default context, instead of the default headless mode in incognito mode.

[1] Note that an HTTP(S) request or a cookie is called third-party if they do not belong to the visited website. For example, if you visit facebook.com then requests to any domain but facebook.com will be considered third-party. Some examples of third-party requests for facebook.com would be google.com and microsoft.com. cdn.facebook.com will not be a third party to facebook.com. If you are using python, you can use https://pypi.org/project/tldextract/ to extract domain and suffix from a request.

7. Replace `collector/RequestCollector.js` with the `RequestCollector.js` provided with the project.
8. Run the command line tool:
    a.
```
npm run crawl -- --input-list ./random_websites.txt
--data-collectors 'requests,apis,cookies' --output ./vanilla_data/
--force-overwrite --verbose
```

Ad block mode:

1. Replace the crawler.js file provided by Tracker Radar with the crawler-adblock.js provided with the project. **Note that you need to copy the crawler-adblock.js code into the crawler.js file.** It includes the code to integrate browser extension with the crawler.
2. Download uBlock by visiting the following link:
    a. https://github.com/gorhill/uBlock/releases/download/1.61.0/uBlock0_1.61.0.chromium.zip
3. Unzip the uBlock folder.
4. Provide the path to unzipped folder in crawler.js by changing the following line:
    a. `const ext = 'PATH_TO_UNZIPPED_UBLOCK';`
5. Run the command line tool *(This is different from the command above. We need to specify a different output folder for this)*:
    a.
```
npm run crawl -- --input-list ./random_websites.txt
--data-collectors 'requests,apis,cookies' --output ./adblock_data/
--force-overwrite --verbose
```

You can also use the `--crawlers <number>` to override the default number of concurrent crawlers, which is picked based on the number of CPU cores. We recommend that you do not change the default. If you run into issues related to the number of crawlers, e.g., memory etc, you can decrease the number of crawlers to 2 or 3, or even 1.

Data Collection:

The above commands will crawl the websites specified in the `random_websites.txt`. This will generate a JSON (https://www.geeksforgeeks.org/read-json-file-using-python/) file for each of the crawled websites in folders named “vanilla_data” and “adblock_data”. Each JSON file will have the following objects:

1. “requests”, which contains all the requests sent and received by the website, along with their headers.
    a. HTTP cookies are present in the “set-cookie” response header.
2. “apis”, which contains a large set of JavaScript APIs called by scripts on the website. JavaScript API calls are grouped by the script that calls them.
3. “savedCalls” for some websites, contains the cookies set by different JavaScript files.
4. “cookies” contains the cookies set by JavaScript files.

Tasks:

Task 1) Follow the above instructions to visit 100 websites in the "vanilla mode" and "ad blocking mode". **[3 points]**

Task 2) Plot the distribution of the number of third-party requests made by 100 websites in the "vanilla mode". List the top-10 most popular third-party domains. Briefly explain the functionality of these popular third-party domains. Compare the distribution of third-party HTTP(S) requests made by the websites in the "vanilla mode" and "ad blocking mode". Discuss the differences between the two distributions. **[4 points]**

Task 3) Plot the distribution of the number of third-party cookies (both HTTP and JS) stored by 100 websites in the "vanilla mode". List the top-10 most popular third-party domains storing cookies on websites. Briefly explain the functionality of these popular third-party domains. Compare the distribution of third-party cookies stored by the websites in the "vanilla mode" and "ad blocking mode". Discuss the differences between the two distributions. For the top-10 domains, manually analyze the cookies values and see what they contain. **[4 points]**

Task 4) Plot the distribution of the number of cookie syncing domains[2] on 100 websites in the "vanilla mode". List the top-10 most popular domains involved (either as *sender* or *receiver*) in cookie distribution. Briefly explain the cookies being synced and functionality of these popular domains. **[6 points]**

Task 5) Implement fingerprinting heuristics[3] from the *Online Tracking: A 1-million-site Measurement and Analysis* paper. Run these heuristics on the crawled data. List URLs of scripts that match the heuristics. List and discuss JavaScript API methods called by those scripts that are not discussed in the 1 million site measurement paper. Note that the million-site paper discusses Canvas, Canvas Font, WebRTC, AudioContext, and Battery API. **[6 points]**

Submission Guidelines:
Submit all code files (e.g. python scripts) with a README.txt (include details on running code files), data files (e.g. JSON files), and report (in PDF) in a single zip file.

Contact TA or me via email or Slack if you have any questions or meet him during office hours.

---

[2] The heuristic to detect cookie syncing is described at the end of this document.
[3] Important note: The original heuristics assume access to values passed and returned to individual APIs, however, this data is not provided by DuckDuckGo (it needs to be instrumented to get that data). Thus you need to simplify the heuristics to only look for the presence of the APIs. More detail on fingerprinting heuristic can be found in: *Online Tracking: A 1-million-site Measurement and Analysis (http://randomwalker.info/publications/OpenWPM_1_million_site_tracking_measurement.pdf)*

1. **You cannot share your code/data with other students in the class.**
2. **You can take help but you are not allowed to directly copy any code/data from the Internet (including ChatGPT).**
3. **Late submissions are not allowed unless you have prior permission to do so.**

Heuristic to Detect Cookie Syncing: Two parties are considered to have cookie synced if an *identifier cookie (described below)* appears in specific locations within the request, referer, and location[4] URLs extracted from HTTP request and response pairs. To determine the domains (sender, who send the cookie to be synced; receiver, who receives synced cookie) involved in cookie syncing of a synced cookie , use the following heuristic:

1. If the name or value of a cookie appears in the request URL: the requested domain is the recipient of a synced cookie.
2. If the name or value of a cookie appears in the referer URL: the referring domain is the sender of the cookie, and the requested domain is the receiver.
3. If the name or value of a cookie appears in the location URL: the original requesting domain is the sender of the cookie, and the redirected location domain is the receiver.

Identifier cookies can be determined as follows: Browsers store cookies in a structured key-value format, allowing sites to provide both name string and value string. Many sites further structure the value string of a single cookie to include a set of named parameters. Cookie values should be parsed assuming the following format:

(name1=)value1|...|(nameN=)valueN

Where | represents any character except a-zA-Z0-9_-=. A (cookie-name, parameter-name, parameter-value) tuple is determined to be an *identifier cookie* if it meets the following criteria:

1. The cookie has an expiration date over 90 days in the future,
2. $8 \leq$ length(parameter-value) ≤ 100, and
3. parameter-value does not change during the measurement

More detail on cookie syncing heuristic can be found in Section 4 (Detecting ID cookies) and Appendix (Section 13.3) of Online Tracking: A 1-million-site Measurement and Analysis (http://randomwalker.info/publications/OpenWPM_1_million_site_tracking_measurement.pdf)

[4] referer and location are properties in the HTTP header. More information on HTTP headers: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers