



Assignment 2


 Grade weight: **7/10** of the final grade

 Due: **7 June** 2023 (23:59, Nijmegen time)

Objectives:

- **Develop a web crawler** using one of [Selenium](#), [Puppeteer](#), or [Playwright](#) that satisfies the requirements listed below
- Run two crawls of the [top 500 websites](#)
 - a. **Crawl-accept**: Accept all cookies, if a consent dialog is present
 - b. **Crawl-noop**: Don't accept or reject any cookies
- Analyze the crawl data (see below)
- Prepare and present a report based on your analyses

Crawler requirements:

- Should accept as **command line parameters/options**: 1) **consent mode** (--accept or --noop) and 2) **target websites**, as a single URL/domain (-u), or a list of URLs/domains in a file (-i). For example:
 - `crawl.sh -u https://example.com`
 - `crawl.sh -i tranco-top-500-safe.csv --accept`
 - `npm run crawl -- -u https://example.com`
 - `python crawl.py -i tranco-top-500-safe.csv --noop`where tranco-top-500-safe.csv contains the list of URLs to crawl, each on a separate line based on [Tranco top sites list](#).
- Should **intercept and save HTTP request and response headers** and metadata into a JSON file for each visit, including the following details:
 - request URL
 - UNIX timestamp of the request (in milliseconds or more granular)
 - request and response header names and values (only the first 512 characters)
 -  Do not store the response content/body
- Should **take screenshots** of the viewport after page load
 - Crawl-accept: take two screenshots; before and after accepting cookies
 - `example.com_accept_pre_consent`
 - `example.com_accept_post_consent`
 - Crawl-noop: take one screenshot, after page load
 - `example.com_noop.png`

- Should be able to **accept cookies** (or data processing) on the GDPR consent dialogs (more info below).
 - Should log when clicking the accept button throws an error
- Should **handle and log errors** including the following cases (and tests):
 - domains that don't exist (test: <https://non-existent-domain-123.com/>)
 - page load errors, including timeouts (test: <https://google.com:81>)
- Should **detect canvas fingerprinting** attempts (bonus) by intercepting function calls used in canvas fingerprinting.
- Should **save the following information into a JSON** file for each page visit:
 - website_domain: domain listed in [the website list](#) (prepend https:// to convert domains into URLs)
 - post_pageload_url: URL after the page load
 - pageload_start_ts: Unix timestamp, marking the beginning of the page load
 - pageload_end_ts: Unix timestamp, marking the end of the page load
 - requests: a list of requests, containing the URL and the header names and values
 - Other information necessary for the analyses below

Analyses & the Report:

Prepare a report that contains the following analyses. You do not need to comment on the figures and tables, but you should include captions for each table and figure, which should be self-explanatory.

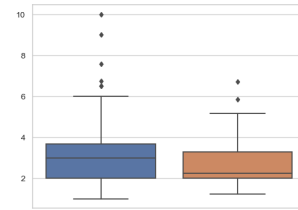
1. Add a table on the number of failures encountered during each crawl, similar to the following:

Error type	Crawl-accept	Crawl-noop
Page load timeout	5	6
DNS error	2	2
Consent click error	3	NA

2. Compare data from the two crawls using a series of box plots, where X axis is the crawl name (*Crawl-accept*, *Crawl-noop*), and Y axis is one of the metrics given in (a) to (f). For instance, the box plot for *Number of requests* will be based on *the distribution of number of requests per website* in each crawl. You will draw a separate box plot (similar to the figure below) for each of the following metrics:

- a. Page load time

- b. Number of requests
- c. Number of distinct third parties
- d. Number of distinct tracker domains
- e. Number of distinct tracker entities/companies



3. Using the metrics from 2a-e, compare data from two crawls in a table such as the following:

	Crawl-accept			Crawl-noop		
Metric	Min	Median	Max	Min	Median	Max
Page load time (s)	2.3	30.4	120	2.0	20.3	70.6
Number of requests	2	100	300
...						

4. Add a table of ten most prevalent third-party domains (based on the number of distinct websites where the third party is present), indicating whether they are classifier as tracker or not (see the Tips below).

Third-party domain	Crawl-accept	Crawl-noop	isTracker?
google-analytics.com	300	150	Yes
someotherdomain.net	250	120	No
...			

5. For each crawl, add a scatter plot of $Y = \text{number of distinct tracker domains}$ vs. $X = \text{website's Tranco rank}$.
6. Add a table of top ten tracker entities (companies) and their prevalence (based on the number of distinct websites where the entity is present). Similar to the table in 4, but should only contain tracker entities). The **Tips** section below explains how to match domains to entities.
7. Find three cookies with the longest lifespans for each crawl. Add a separate table for each crawl having the following columns. Only include the first 5 characters of the Value attribute:

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite
CONSENT	YES+cb	.google.com	/	Sun, 10 Jan 2038 07:59:59 GMT	38	false	true	None
__Secure...	__VPgZc	.google.com	/	Wed, 28 Feb 2024 11:43:40 GMT	51	false	true	None
__Secure...	HggHr	.google.com	/	Wed, 28 Feb 2024 11:43:40 GMT	85	true	true	None

8. Find three requests with the most cookies for each crawl. Add a separate table for each crawl having the following columns:

Request hostname	Website	Number of cookies	First-party request
www.bol.com	bol.com	25	Yes
...			

9. For each crawl, identify ten most common cross-domain [HTTP redirection](#) pairs that involve a tracker domain either in the source or the target (or both).



Source domain	Target domain	Number of distinct websites
google-analytics.com	doubleclick.net	100
...		

10. Capture the images drawn by the canvas fingerprinting scripts by intercepting the return value of the [canvas.toDataURL](#) method. Identify fingerprinting scripts following the detection methodology described in [this paper](#) (Section 6.1). Copy paste the images into the final report as a table:

Website	Fingerprinting script URL	Canvas Image
example.com	http://fingerprint.com/fp.js	
...		

Tips:

- In order to detect tracking-related requests or tracker domains, use [Disconnect's blocklist](#) and (optionally) [Mozilla's trackingprotection-tools library](#). ⚠ Do not use the blocklists from the previous assignment.
- In order to map tracker domains to tracker entities (companies) use [domain_map.json](#) from DuckDuckGo. You can use the `displayName` property as the entity/company name.
- You can use code or data from the [priv-accept](#) project to identify and click "Accept" buttons on consent dialogs in the Crawl-accept crawl.

- Websites without consent dialogs should still be included in the Crawl-accept results
- You don't need to take full page screenshots, the viewport is enough.
- You are free to use publicly available Python or NodeJS packages, except the following fully equipped crawlers and similar packages (feel free to ask on Brightspace when in doubt):
 - a. <https://github.com/duckduckgo/tracker-radar-collector>
 - b. <https://github.com/openwpm/OpenWPM>
- You can use [SeleniumWire](#) or any language binding for Selenium, Puppeteer, or Playwright such as [Playwright for Python](#).
- Add titles for X and Y axes in the plots (e.g. Y-axis title: *Page load time(s)*). Add captions for figures and tables in the report.
- Wait for ten seconds after page load, and after clicking the consent dialog (if any).
 - a. Crawl-accept: Page-load -> Wait 10s -> Click accept (if any) -> Wait 10s -> Close the browser
 - b. Crawl-noop: Page-load -> Wait 10s -> Close the browser
- Use a Github/Gitlab repository for development and issue tracking. The repository can be private.
- You can run your crawls on your own computer(s), but do not run many browsers in parallel because this may get you blocklisted. You can run crawls sequentially or prefer cloud-based platforms that give free credits. You can, for example, sign up for [GitHub Student Developer Pack](#) to get \$200 free credit on cloud providers such as DigitalOcean.
- Comment your code when what you are doing is not very obvious
- Unless specified, “domain” means eTLD+1.
- Use meaningful variable and function names
 - a.  good: request_domain, response_headers, get_country_by_ip_address
 - b.  not good: foo, bar, tmp, a, do_stuff, get_data
- Your code should **not** make any calls to online APIs.

Submission

- Upload a zip file containing the data, source code and the report. Name the zip file after your group name. The zip file contents should be organized as follows (📁=folder):
 - a. report.pdf
 - b. 📁 crawler_src
 - README.md, containing the following:
 - name of the browser automation library used (Selenium/Puppeteer/Playwright)
 - instructions on how to install (when applicable) and run your crawler
 - *[Crawler source code. Can be multiple files and/or folders.]*
 - c. 📁 analysis
 - *[Analysis source code. Can be multiple files and/or folders.]*
 - d. 📁 crawl_data
 - example.com_accept.json (*JSON for example.com. Crawl-accept*)

- example.com_accept_pre_consent.png (*screenshot before clicking Accept on the consent dialog, if any. Crawl-accept*)
 - example.com_accept_post_consent.png (*screenshot after clicking Accept on the consent dialog, if any. Crawl-accept*)
 - example.com_noop.json (*JSON for example.com. Crawl-noop*)
 - example.com_noop.png (*screenshot for example.com. Crawl-noop*)
 - [same files for the other websites....]
- e. requirements.txt or package.json: Python or node packages required to run your script, if any.

Help

- Ask your questions on this Brightspace discussion list:
<https://brightspace.ru.nl/d2l/le/333333/discussions/topics/88994/View>

🍀 Good luck! 🍀