

Analysis of Cookie Activity in Web Search Traffic

Tord-Vincent Heggland

May 28, 2026

Abstract

This report aims to find trends in Web Search activity connected to `tracking_hints` and cookies. Playwright is used to automate the process of performing web-queries. Data is downloaded as `HAR` files and transformed into `CSV` files using automation scripts. The `CSV` files are loaded into Power BI, where the data is transformed and visualized using Power Query. It is found that privacy-focused tools tend to better respect privacy in web-queries, while Google shows significantly more tracking-related behavior and privacy leakage compared to the other Search Engines.

Contents

1	Introduction	1
2	Theory	2
2.1	EDA - Exploratory Data Analysis	2
2.2	VENV - Virtual Environment	3
2.3	HAR - HTTP Archive	3
2.4	Proxy	4
3	Method	4
3.1	Research design	4
3.2	Test environment	5
3.3	Data collection	7
3.4	Data processing	7
3.4.1	From HAR to CSV files	8
3.4.2	ETL process in Power BI	8
3.5	Limitations of the method	9
4	Results	9
5	Discussion	11
5.1	Discussion of Methodology and Theory	11
5.2	Discussion of the Result	15
6	Conclusion	16
A	Appendices	A-1
A.1	Additional visualization on Browser specific cookies.	A-1
A.2	Automation scripts	A-3
A.3	Power Query transformation	A-12

1 Introduction

Background

This project was delivered from Noroff Academy. A student could either choose to work in a group, or work independently. The topic for the project could either be chosen by professor and teacher at Noroff Academy, or the student could choose their own topic.

For this scope, the student chose to work independently with a self-chosen topic. The student has experienced that work may lose interest if there is nothing driving the curiosity behind the project. For that reason, the topic for this work is to find trends in `tracking_hints` in web-queries. The student has a strong interest in networking, system administration, and Linux-based environments through personal projects and private experimentation. That is why this topic is chosen, to make use of personal skills in an academic work. Overall, the main work environment and method used in this project are thought in Noroff Academy. This aspect of the work creates motivation to complete the project, because the topic is of personal interest to the student.

As visualized in Figure 1, which presents entries returning `tracking_hints=yes`, Google was the only Search Engine that consistently interacted with third-party domains. This finding is significant because Google also heavily used cookies on entries associated with `tracking_hints`, as further discussed throughout this paper.

Hypothesis

The student's hypothesis for this work is that DuckDuckGo and Brave will return fewer instances of `tracking_hints` and cookies. On the other hand, the student expects Google and Bing to use cookies alongside `tracking_hints`, as these Search Engines are known for tracking user activity and providing personalised advertisements in the web browser.

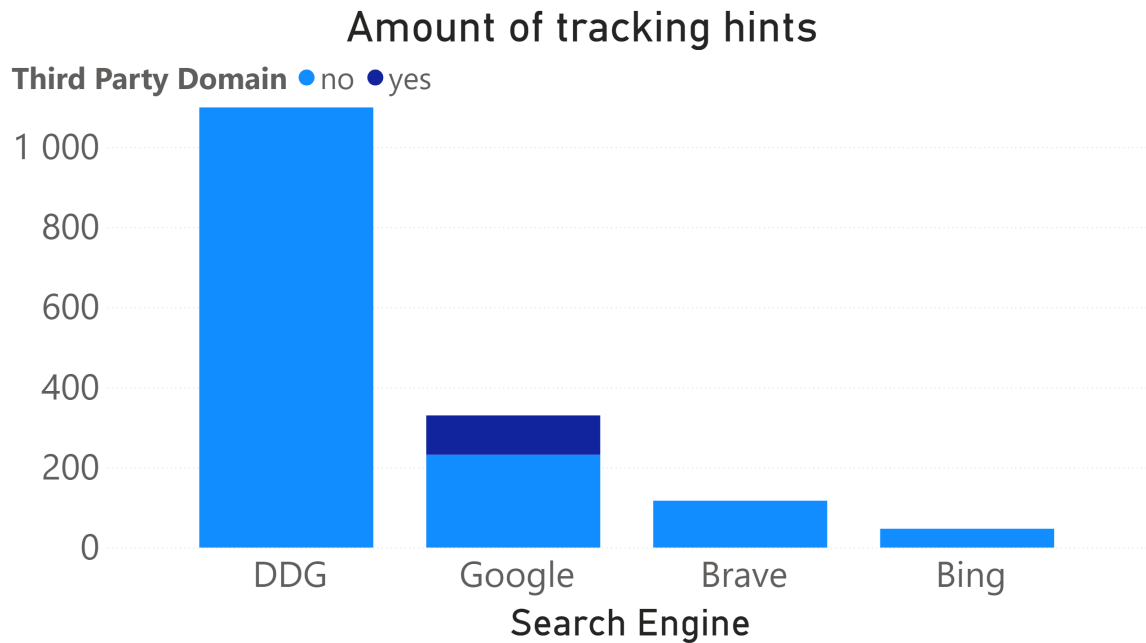


Figure 1: Third-party domains across search engines

2 Theory

This section explores the necessary theory for this work. There are two main parts of the theory: one that covers Noroff’s course materials, and one that covers topics that are necessary in order to understand the progress of this work. The topic necessary to understand for this work are EDA - Exploratory Data Analysis, VENV - Virtual Environment, and HAR - HTTP Archive will be explored in subsection 2.1, 2.2, and 2.3 respectively.

2.1 EDA - Exploratory Data Analysis

EDA - Exploratory Data Analysis, this is a crucial part of a data analysis process. It takes the process from when retrieved the raw dataset, and investigates how to present it professionally, and academically. Noroff 2026

To visualize a raw, unfiltered dataset, some decisions and restrictions need to be made before the data can be presented. Take, for example, a dataset where a lot of data is missing, and some data in the table is misleading. The EDA process is supposed to make the unfiltered data ready for visualization. During that process, several important questions need to be stated in order to narrow the research topic and the findings to one specific theme.

One example the EDA process may narrow the dataset, is to focus on one specific data variables, and investigates the trends corresponding that variables. Take for instance in

this course, all dataset is filtered for data parameters where variable `tracking_hint` is equal to yes, because this work tries to identify the trends of HTTP request and responses, when the dataset returns that the entry of interest addresses that it could be a tracking parameter.

The EDA process may investigate outliers in the dataset, visually, which do not seem to reflect the rest of the dataset. For example why some parameters may have a lot more cookie count per entry than other entries. And if the outliers actually reflect the data correctly, why may the trend be different in some cases than others.

2.2 VENV - Virtual Environment

This section covers the theory of the methodology for this work. To automate the process to retrieve raw dataset as HAR files, Virtual Environment was used to create and run scripts that simulate the process of manually retrieving raw HAR files when doing web queries. More on that in section 3

Listing 1: Creating the virtual environment using python for this work

```
1 cd "$WORKSPACE"
2 python -m venv .example-env
3 source .example-env/bin/activate
4 pip install pytest-playwright
5 playwright install chromium
6 playwright install firefox
```

Listing 1 demonstrates how to create a virtual environment using python to run specific python scripts with specific packages. In Listing 1 illustrates how to install playwright and some web browser in that virtual environment. When installing packages and program into the virtual environment, all programs are stored in following folder: `"$WORKSPACE"/.example-env/`, in the folder created when running line 2 in Listing 1. And once executed line 3, all files in folder `"$WORKSPACE"/.example-env/bin/` are executable from the command line ([12. Virtual Environments and Packages 2026](#); [Installation / Playwright Python 2026](#)).

2.3 HAR - HTTP Archive

HAR is short for HTTP Archive. Every time a web-browser is loaded, it performs many thousand requests and returns several responses. Each one is stored in a table in the web browser, where all parameters may be visualized. It could either be a POST request, or a GET request ([Network request list — Firefox Source Docs documentation 2026](#)), either way, each response or request returns as an entry in this table. The table may be downloaded for inspection. All data for the latest web searches may be found in that table.

When HAR table is downloaded, it is not downloaded as a CSV files, but instead as a .har file. And in order to make use of that dataset, it has to be transformed into something that Power BI may read. The process of this is explained in section 3.

Cookies are essential in order for a web browser to work normally. It contains login information for the current user experience. For example when a customer is logged in to a website to shop, the shop cart uses cookies, information about the customer to store the current session. So that next time a customer opens the same website, the same items are still stored in the shopping cart (MDN Web Docs 2026). This is convenient for most cases, but for some cases it can be used for tracking, and for personalize ads on your web-browser.

2.4 Proxy

A proxy acts as an intermediary between a client and the destination website. This means that all web traffic goes through a virtual tunnel before it reaches the destination. For every web query performed, the web traffic has to go through another server before it reaches its destination. This helps to hide the user's identity on the web because the destination can only see the intermediary and not the client itself. This means that when you visit a website, it cannot see your public IP address, only the public IP address of the proxy being used.

This is where Tor comes in. Tor routes your traffic through multiple servers using encryption. Therefore, when it reaches its destination, the source of the request is essentially untraceable (Tor Project 2026; Cloudflare 2026).

3 Method

This section describes the methodology used in this research. Any technical concepts and terminology references in this section are further explained in section 2, and discussed in section 5.

3.1 Research design

This research is designed using tools to simulate human interaction with simple web searches. Each search is designed to be anonymous, with no browser history or cookies stored before the search is performed. Before each search, the browser history is cleared and cookies are removed. The browser profiles used contain no login data, preventing the web queries from being connected to any real person.

Four search engines are used in this process (Brave, Bing, DuckDuckGo and Google), once in each web browser: Firefox and Chromium. This means that each web query

is performed eight times. For this work, several queries are created to widen the data collection. The web queries are as follows:

- weather oslo
- migraine symptoms
- vitamin d deficiency
- running shoes
- coffee grinder
- best laptop for students
- electric car charging
- cheap flights to london
- home insurance
- python list tutorial
- banana bread recipe
- news norway

Data collection is performed using either a Tor proxy to help hide the identity of the person performing the web searches, or a normal network connection where web traffic may be used to identify the user.

3.2 Test environment

When pressing **Ctrl + Shift + C** and click on **Network**, a log of Network traffic shows up. This window is open and the web-history is emptied before performing a web-search manually. This process gives a clean anonymous log web traffic from only one web query as known in Figure 2. Right to "No throttling" is a settings icon. Clicking on that bottom gives the options on Figure 3. Each query could be done manually, or the processes of collecting data could be automated. For this research the process of collecting first-hand raw data was automated. A tool used to automate web-queries is python using Playwright ([Installation / Playwright Python 2026](#)). Playwright is installed in a virtual environment packages using python ([12. Virtual Environments and Packages 2026](#)). All collection of data is done in Linux shell, and doing EDA is done in Microsoft PowerBI.

Once the installation done, web-browsers of choice may be installed inside the virtual environment. Firefox and chromium were installed inside the virtual environment. Now the environment for retrieving raw, real-world-event data for this analysis.

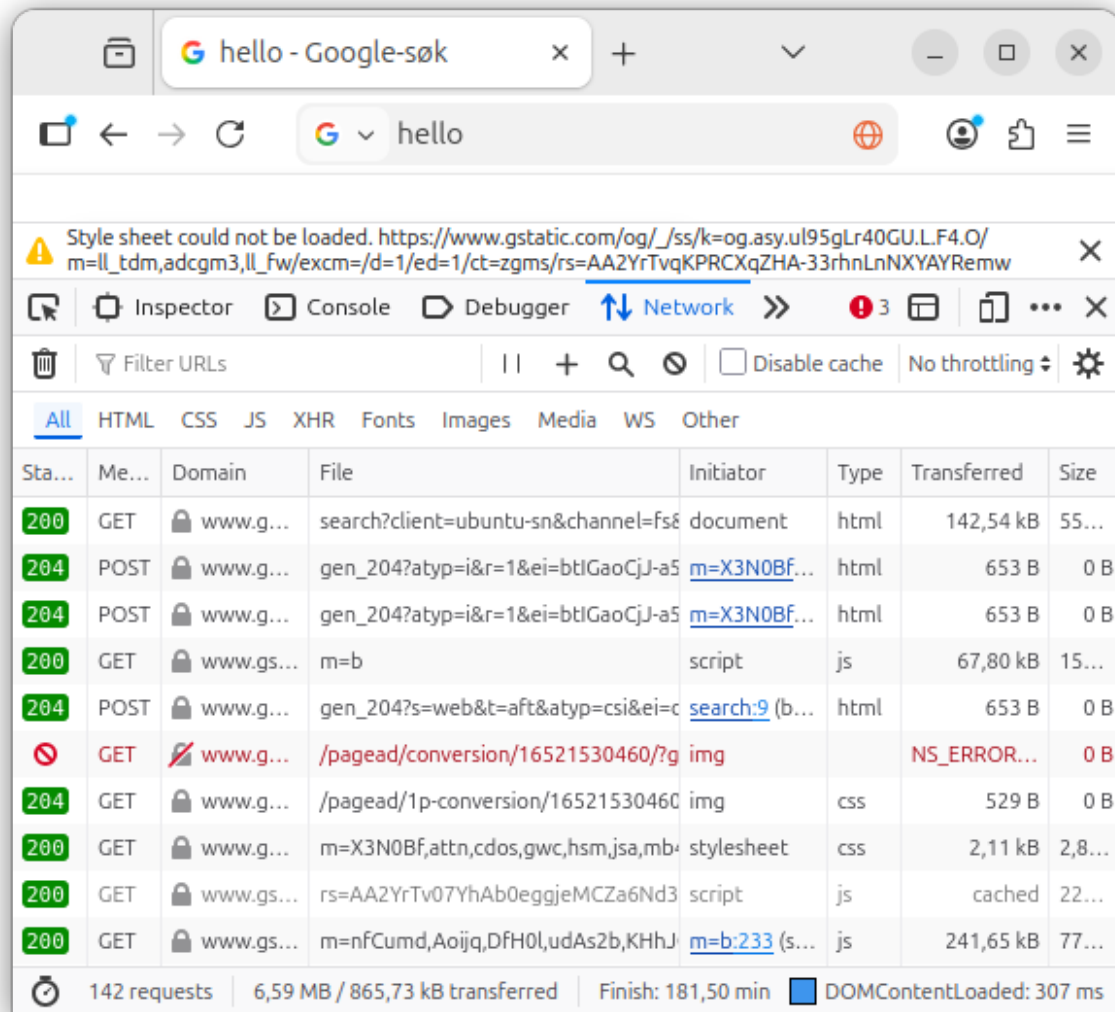


Figure 2: Network traffic by a simply web search

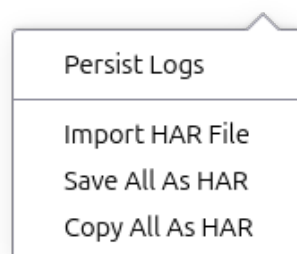


Figure 3: Download Har files.

3.3 Data collection

Three Scripts, two python files and one bash files are used to automate the data collection process. The files can be found under folder `./scripts/`. The bash file (`./scripts/many_search.sh`) uses the python file (`./scripts/capture_search_har.py`) to automate the process of retrieving data. It essentially loops for each query used, and each web-browser used, and for each proxy used and stores them into different folders.

Listing 2: Playwright data collection command

```
1 capture_search_har \  
2     --query "weather_oslo" \  
3     --browser chromium \  
4     --wait-until load \  
5     --headed \  
6     --output-dir tor_chromium \  
7     --proxy socks5://127.0.0.1:9050
```

The Listing 2 is an example of a prompt using the python file to automate the retrieving the data. It opens a web-browser, preform a web-search, and saves the output to a directory of choice, or in the default current directory. Input `--query` is the only mandatory input for this function. All the option are optional, but defaults to a default value. Input `--proxy` is optional, if not used, it uses the desktops current official IP address to preform the web-search. If the `--proxy` option is specified, the provided value will be used as the proxy endpoint. Meaning the web-search would only see, for this instance, the Tor's endpoint and its official IP address when performing the web-search (Tor Project 2026). `--headed` and `--wait-until load` are important. The first one tells Playwright to open a physical window when performing a web-search, and not just a HTTPS call, the second one tells Playwright to wait until the web-browser is fully loaded. The rest of the inputs are self-explanatory.

After retrieving all the HAR files through the automated Python workflow, the dataset is ready for processing.

3.4 Data processing

This section contains several important steps in the data processing pipeline, all leading to the exploratory data analysis (EDA) performed in Microsoft Power BI. As of now, the data set is segregated into several HAR files, which is unreadable to Microsoft Power BI. This section is a step-by-step process from raw data collection to finished visualized tables in Power BI.

3.4.1 From HAR to CSV files

In order to perform data analysis in Power BI, the data sett had to be converted from HAR files to CSV files. Once collecting data in subsection 3.3, several data entries needed to be ready before it could be extracted, transformed and loaded into tables in subsubsection 3.4.2.

A python script at (`./scripts/har_entries_to_csv.py`) reads all the `.har` files in folder `./data/` and prints two output files. The first one is `./har_entries.csv`, and the second one is `./har_summary.csv`. Four of each of those files were created, each for each proxy type and web-browser of choice. Working directory decides which proxy and web-browser that is used for that current data collection. More on that in section 5. `./har_entries.csv` contains every request from the web-search as one entry, or one row in the csv file. `./har_summary.csv` summaries its respective `./har_entries.csv` file. Which means it takes all the input from several `.har` file and summarize one `.har` file in one row. In contrast, the output in file `./har_entries.csv` does not summaries the `.har` files, it takes the raw data and presents one entry in a `.har` as one row in file `./har_entries.csv`, and does not do any data processing.

The file `./har_summary.csv` was discarded in favour of `./har_entries.csv`. It contains the raw data, and will be used on the ETL process in Power BI in subsubsection 3.4.2

3.4.2 ETL process in Power BI

ETL stands for Extract, Transform, and Load. Some of the Extract and Transform process was done in subsubsection 3.4.1. The dataset is not ready to be loaded and merged into Power BI.

The dataset is separated into four CSV files in each folder, which represents the case of those entries. For instance, if the proxy used is Tor and the browser used is Chromium, the location of the dataset is as follows: `./tor_chromium/`, as Listing 2 indicates. The CSV files consist of equal file name, which is generated from the python script `capture_search_har`. All CSV files was loaded into each segregated folders in the student private working area at SharePoint. Listing 6 in subsection A.3 shows the total query done in Power BI for each instance. When a CSV file is loaded as source into Power BI, some autoformatting is done by Power BI itself, and Listing 6 illustrate the code Power BI generates, and some more formatting.

As explained, only folder name describes which proxy used and which browser used for each instance. To take account for this, each `./har_entries.csv` had to manually loaded to Power BI for each instance. Once one table was loaded, two new columns had to be added which specified its proxy and browser for the current entries. After this is done all for tables could be merged into one table main table. Before merging, each table was named as following: `har_entries_<proxy>_<browser>` for its respective proxy and

browser. Once merged, the new table got the name `har_entries_all` which was further used for creating tables for this work. Those tables are presented in section 4.

At last, for all observation, the whole table `har_entries_all` was filtered for the variable `tracking_hint` to be equal to `yes`. Meaning every entry that did not address any hint to tracking was immediately filtered out.

3.5 Limitations of the method

Some limitations of this work are related to the process in which the analysis is performed. When retrieving a `.har` file, the text file is unstructured and contains large amounts of data noise. The scripts do not always guarantee data consistency. The output files did not specify which proxy or browser used.

4 Results

This section introduces all the findings in this work. The main priority is the variable `tracking_hints`, as the work tries to identify relationships between trackings and cookies. Every graph in this work is filtered for `tracking_hints=yes`. Meaning the tables retrieved are larger than those visualized in this work.

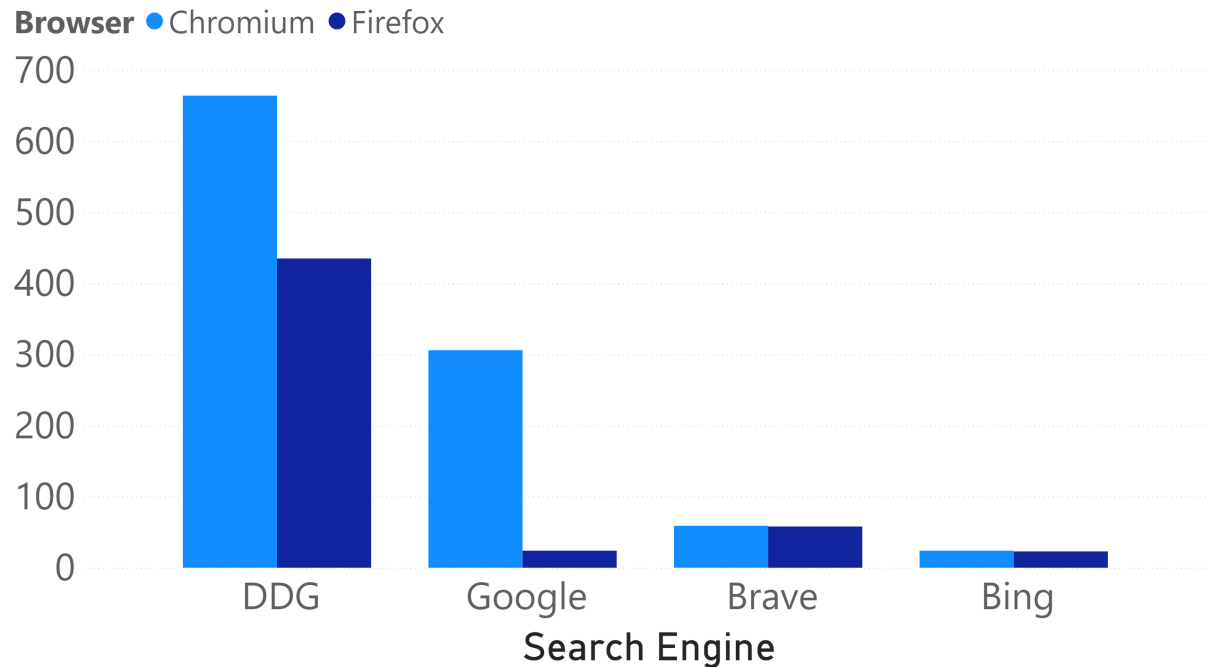


Figure 4: Tracking hints on each chromium and firefox

The first figure, Figure 4, illustrates the distribution of tracking hints across browsers and search engines. DuckDuckgo appears to be the Search Engine that caches and identify

most hints to tracking. Bing and Brave appear to conservative addressing any tracking hints. While only on DuckDuckGo and Google, the choice of web-browser seem to play a crucial role. Chromium addresses more hints to tracking in Google than Firefox does.

Furthermore, request cookies counts and response cookies counts will be presented in the Results below, which will be the main focus of the following results.

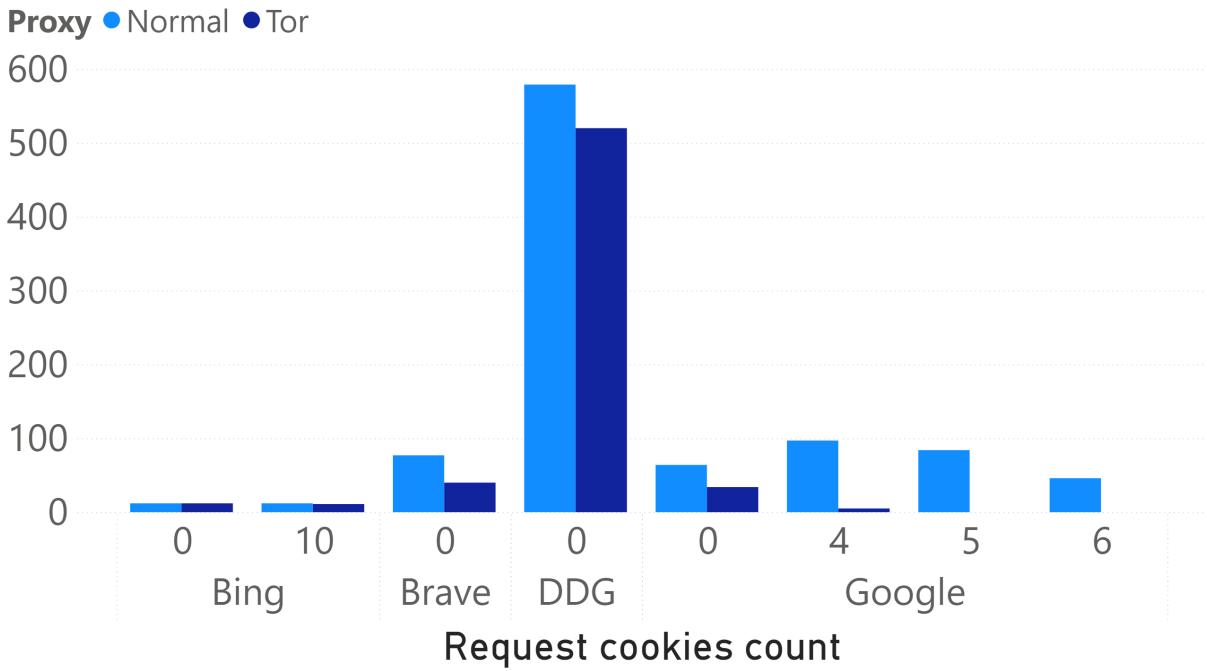


Figure 5: Proxy mode configuration for request cookies

Figures 5 and 6 illustrate the cookie counts for each tracking hint. DuckDuckGo and Brave show no cookies across all `tracking_hints`. In contrast, only Bing and Google use cookies on entries identified as tracking hints. The number above the name of the Search Engine indicates the number of cookies, and the left bar indicates how many instances of those are associated with a tracking hint. Both Bing and Google shows cookies count on requests, but only Google shows cookies count on response.

Figures 5 and 6 express how using a proxy influences tracking hits on the web. Using Tor as a proxy returns no cookie responses on tracking hints, while some cookies still leak through in request cookies associated with tracking hints.

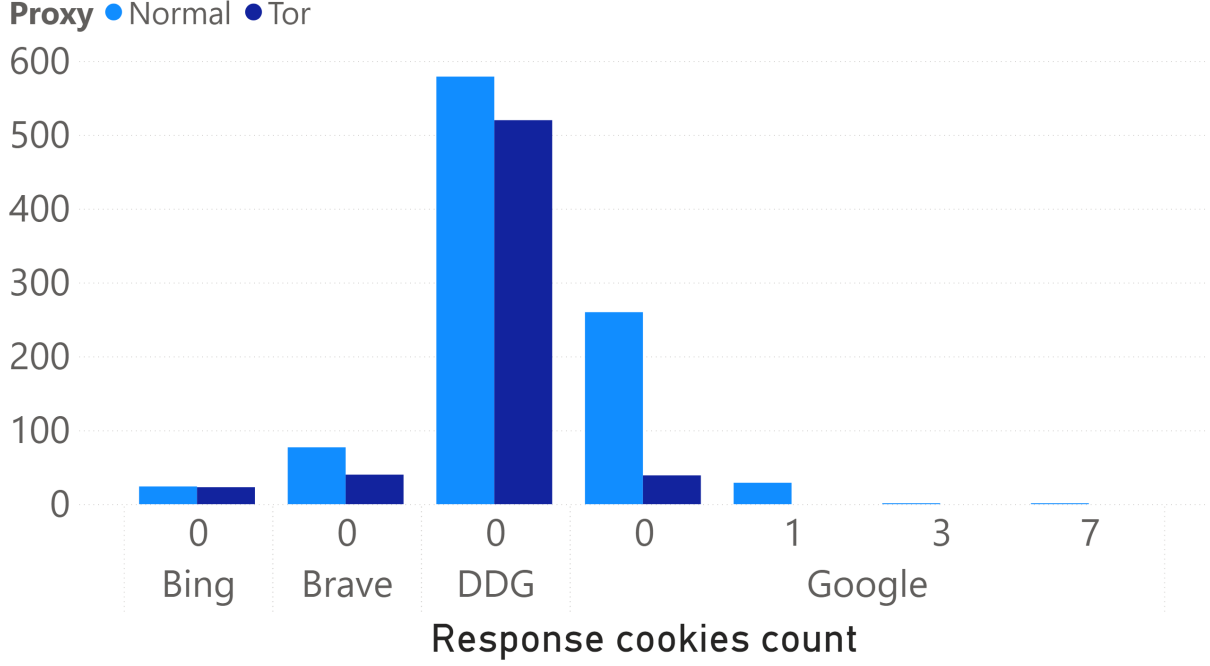


Figure 6: Proxy mode configuration for response cookies

In subsection A.1 additional visualization on cookies comparison on request and responses are added.

5 Discussion

Throughout this paper, several statements and choices have been made without been accounted for in earlier sections. This section seeks to explain and connect all the loose threads. Why is HAR files used, and fully explain the order of methodology, and every part of it. Lastly, results will be explained in this section, in subsection 5.2.

5.1 Discussion of Methodology and Theory

As explained, this work aims to connect the trends between `tracking_hint` and cookies in web queries. There are many ways to approach this. Each web queries had to be done anonymously. Any cookies stored in a web-browser leaks personalized data for the current user when performing a web query. To make process of collection data, EDA, as anonymously as possible, a new browser profile was created for this cause. However, for each web-query, the cookies in this new browser profile had to be deleted. To make this as anonymous as possible, all DNS blocking and personalized configuration for student's private network was disabled for a limited period of time. The student's personal computer was used in this EDA process. The only data that might point to the student is such

as the official IP addresses. Such IP addresses are dynamically assigned from the IPS provider, so that is not a crucial leakage of data.

The only data parameters visualized in Power BI is those of importance for supporting the student's theories. Such parameters as **Request Cookies**, **Response Cookies** and **tracking_hint**.

Dataset parameters

For this course only entries for when **tracking_hints=yes** are inspected. Any entry for when **tracking_hints=no**, is filtered out. This is due to main problem on this work, to find trends on web queries when the entry hints to tracking. This means all other entries are of no interest for this work.

The parameter **tracking_hint** indicates any entry on web requests or responses that could be tracking or not. Might not be tracking, but the entry thinks that it may be tracking. This means the number of instances of **tracking_hints** does not reflect how much tracking that is actually queried on that one web query. In contrast, if a Search Engines many **tracking_hints**, it may reflect that Search Engine used may be paranoid toward any hints for tracking. Likely when a motorcyclist may be paranoid on any car on the road. This is a good thing, because it allows the user to inspect those parameter that hints to tracking, and inspect what source the entry has. However, what is worth noticing is when tracking hint shows cookies requests and responses on the same entries. Cookies are data of information on the current session of the user using the web browser. Meaning if an entry returns tracking hint = yes and no cookies, then the server on the other and where the request is retrieved from has no data on you. In contrast, if the entry return both cookies and **tracking_hints** = yes, then this means that the server on the other end that might track your activity on the web has personal data on you from the web queries.

Meaning DuckDuckGo has a high occurrence rate of **tracking_hints** that could have indicated to that the Search Engine did not filter out any entries that may point tracking. However, that is not the case, this high occurrence rate actually addresses all cases that could be tracking, that other Search Engines did not detect. So a high occurrences rate on **tracking_hints** does not reflect your data actually being tracked. It is only when the entries where tracking hint is equal to yes, and the current entry returns cookie count, that your data could be tracked. From figures in section 4, only entries from Search Engines such as Google and Bing return cookies on entries where **tracking_hints** is equal to yes. When preform data collection, the student by habit rejected cookies on Google Search Engines, and still the Search Engine returned cookies, while DuckDuckGo and Brave did not return any cookies, on neither requests nor responses.

Tor proxy

Tor proxy was used to help hide the user's identity on the web. As explained in subsection 2.4, a proxy routes traffic through an intermediary server before accessing the web. Tor proxy tunnels traffic through several relay servers, making direct identification of the user more difficult. This is because the destination web-server does not see the user's original IP address, but instead observes the IP address of the Tor exit relay.

The results show that fewer entries on cookies for `tracking_hints` when using a Tor proxy. This is implicit due to that web servers are able to identify Tor exit relay by its IP address, so any tracking do not point to the user. So any malware, or unethical hackers are not able to track your activity on web. However, normal websites are able to identify any Tor exit relay. For example, finn.no block any incoming requests from a Tor exit relay. This is due to that Tor is not a trusted exit relay. Trusted exit relay are normally connected to home routers to any normal household using, for example, Telenor as ISP. That is because anyone accessing the web through Telenor's ISP is automatically assumed to be a Norwegian human being accessing the web. What's interesting, even though when a Tor proxy is used, only Google and Bing Search Engines returns cookies on requests, and only Google returns cookies on response as well. Meaning Google uses cookies still when user is using proxy.

Working environment

The working environment for this project was segregated into two steps. One for data collection, and one for data analysis.

Working environment - data collection

The process of data collection used in this project is out of the scope for the learning materials at Noroff Academy. However, the EDA process plays a crucial role for this process. What parameters to show, how to retrieve data efficiently, etc. The count of different queries is twelve, meaning it would take too much time and resources to collect this data manually. Codex was used to create the automated scripts to automate the process of retrieving data, see sections A.2. However, the student has investigated the dataset, and anyone can investigate them themselves. After inspecting the automation scripts, it can be confirmed that the only thing it does is open a web browser with a query using the Search Engine of choice, and download the result into a raw HAR file. It is essentially doing the same as explained in Methodology, visualized in Figure 2 and Figure 3, repeatedly throughout the data collection process. A larger dataset may reduce the impact of isolated outliers or irregular observations that do not reflect the overall trend.

Working environment - data analysis

For this part, Power Query is used to perform data analysis of large dataset. With twelve different queries done once in each browser of choice and Search Engine of choice, the dataset would be too enormous for use in an ordinary Excel sheet. Power BI is created to handle such large data set. First performing data table convention, and then merging them into one large dataset. When one HAR file could contain entries between 2 and 500, approximately, it is reasonable that such large dataset is too big for Excel to handle. Excel only handles about 200 entries before it starts to lag on a normal student's computer. Power BI is created to handle such large dataset, and that is the reason for performing data analysis in Power BI. The automation scripts do not filter for which proxy used and which browser used. This means a student could prompt the automation scripts to use a specific proxy and browser, but the output HAR files do not indicate any proxy nor browser of choice. That is why the Listing 3 was created to retrieve raw data in a controlled environment, sorting the results into different folders that segregates any proxy and browser of choice. Each HAR files are stored into a folder `./data/` inside the case folder because each case created 48 HAR files. By sorting all HAR files into the folder `./data/` inside each case folder, running the python-scripts became much cleaner when visualizing the content for each case folder and later uploading the data into separate tables in Power BI.

Listing 6 is an example for how one table was read into Power BI, it was done once for each four cases that make up for using two browsers and two proxies. The methodology is identical for each four cases. Listing 6 performs data cleaning and data conversion. Power BI automatically recognized data type when a new data source is loaded into Power BI. So the only thing needed to be done was to make sure that all convention is finished, and then duplicate the query, and change data source according to which table to read into Power BI. For each query, two columns were added. Each to hardcode the Proxy of case, and the Browser of case. This is important, because Listing 7 merges all queries into one large table that is used to create all the figures in section 4. If the dataset did not have any hardcoded variable for Proxy and Browser, it would not be possible to filter for those variables when creating those figures.

Power BI allows for only to select variables of interest when creating figures. This feature saves time used for data analysis, because the student could just ignore any parameter of no interest for the case of this work. And only chose parameters that is important for this work when creating figures. For example, parameter such as `Proxy`, `Browser`, `Search Engine` and `tracking_hint` are essential for this work. In contrast, parameters such as `time_ms` and `startedDateTime` are not essential for the results.

Due to the size and complexity of the dataset, the scope of the analysis had to be narrowed to selected variables and trends relevant to the research topic. For example,

some parameters would be interesting to analyse how it trends with `tracking_hints`. This work has already created many figures, and creating too many figures would only be overwhelming and not supporting the overall findings of this work. So next time, it would be interesting to look into parameter `is_third_party_domain`, and perform visualization to find out how it trends with other parameters. For this work, the parameter `is_third_party_domain` is only used in section 1 as an introduction to what the trends may indicate.

5.2 Discussion of the Result

Now that the dataset, working environment, and parameters have been discussed, the findings and results of this work can be further analyzed. This includes discussing how the findings may affect users and what can be learned from the statistics presented in this work.

All discussion in this section points to the figures in section 4. DuckDuckGo would seem to identify a lot of instances for any `tracking_hints`, while other Search Engines do not indicate as many instances as DuckDuckGo on `tracking_hints`. However, as discussed in subsection 5.1, that does not mean that DuckDuckGo leaks user information. It means that DuckDuckGo is the most aggressive Search Engine towards addressing possible hints to tracking. Keep in mind that all these results came only from performing single web-queries, and not doing web-browser for a longer period of time. And all instances were done with a clean slate of cookies. Meaning web-browser and Search Engines had no previous data on you. That is due to Playwright performs the web-query with no previous data when a new query instances is started. This is more efficient than manually deleting cookies between each query, both for each Browser and Search Engine. In total 192 web-queries were performed for this work. That is why the automation scripts came in handy.

One aspect that can be learned from the results is how data security relates to web tracking and cookies. Google tries to track activity on web even when you submit do not share any cookies, which the student did by habit. What's common is that using tools that promote security actually work. Take for instance Figure 10 in subsection A.1 shows that using Firefox actually reduces cookies on entries with `tracking_hints`, and overall Firefox caches fewer `tracking_hints` than Chromium. And using a Proxy such as Tor does reduce cookies on `tracking_hints`. Overall, only Brave and DuckDuckGo do not return any cookies on `tracking_hints`, while Google and Bing do.

6 Conclusion

This section concludes the overall work throughout this paper.

From the discussion in section 5 regarding the findings in section 4, it can be concluded that Google appears to be a less privacy-focused Search Engine when it comes to tracking-related behavior, with Bing following closely behind. Brave and DuckDuckGo were the only Search Engines that consistently respected privacy in terms of tracking-related behavior. DuckDuckGo also identified significantly more instances of `tracking_hints` than the other Search Engines. However, despite these detections, DuckDuckGo did not return cookies related to those entries, indicating that identifying potential tracking-related behavior does not necessarily imply privacy leakage through cookies. Using Firefox rather than Chromium and a Tor proxy also reduces instances of cookies related to `tracking_hints`.

The student's hypothesis was partly correct. The student expected that using privacy-focused tools would reduce tracking-related instances, which proved to be correct. However, it was surprising that DuckDuckGo returned a high count of `tracking_hints`, while no cookies were associated with those instances. For future work, it would be relevant to further investigate parameter `is_third_party_domain`, as mentioned in section 5

References

12. *Virtual Environments and Packages* (2026). Python documentation. URL: <https://docs.python.org/3/tutorial/venv.html> (visited on 05/15/2026).
- Cloudflare (2026). *What is a reverse proxy? Proxy servers explained*. URL: <https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/> (visited on 05/22/2026).
- Installation | Playwright Python* (2026). URL: <https://playwright.dev/python/docs/intro> (visited on 05/15/2026).
- MDN Web Docs (2026). *Using HTTP cookies*. URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Cookies> (visited on 05/22/2026).
- Network request list — Firefox Source Docs documentation* (2026). URL: https://firefox-source-docs.mozilla.org/devtools-user/network_monitor/request_list/index.html?utm_source=chatgpt.com (visited on 05/15/2026).
- Noroff (2026). *Modules 2, 4 and 7: Lessons and Tasks*. Internal course material used in the Data Analytics programme.
- Tor Project, Inc (2026). *About Tor Browser*. Support. URL: <https://support.torproject.org/tor-browser/getting-started/about-tor-browser/> (visited on 05/15/2026).

A Appendices

A.1 Additional visualization on Browser specific cookies.

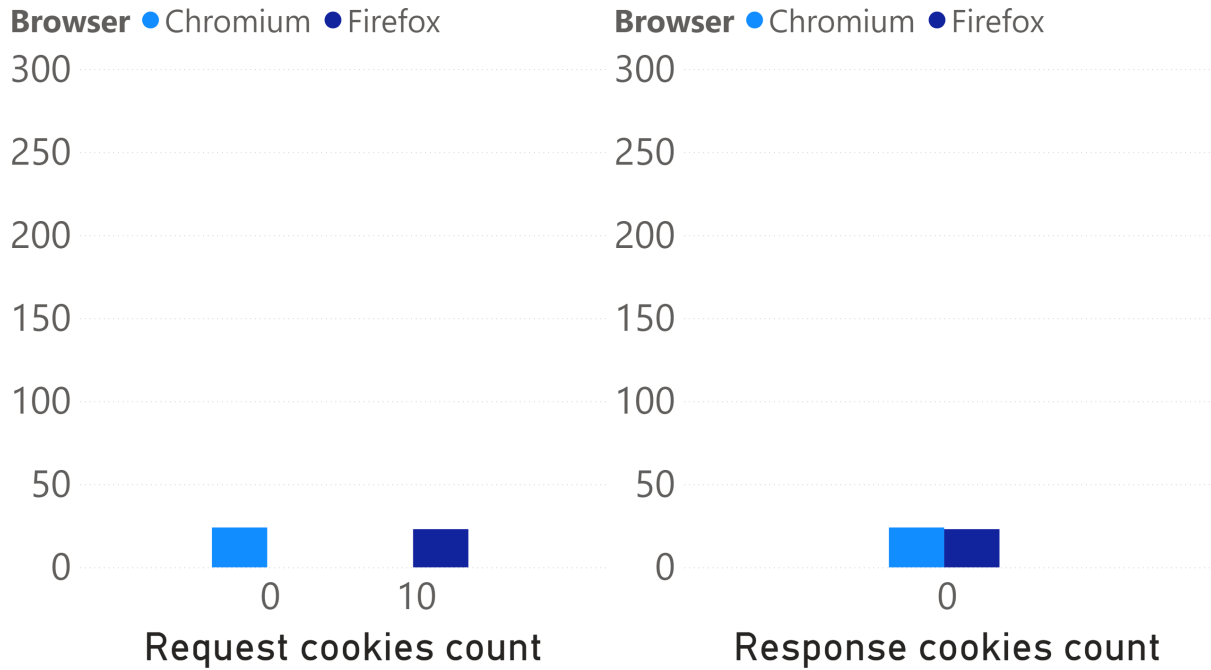


Figure 7: Cookies comparison in Bing

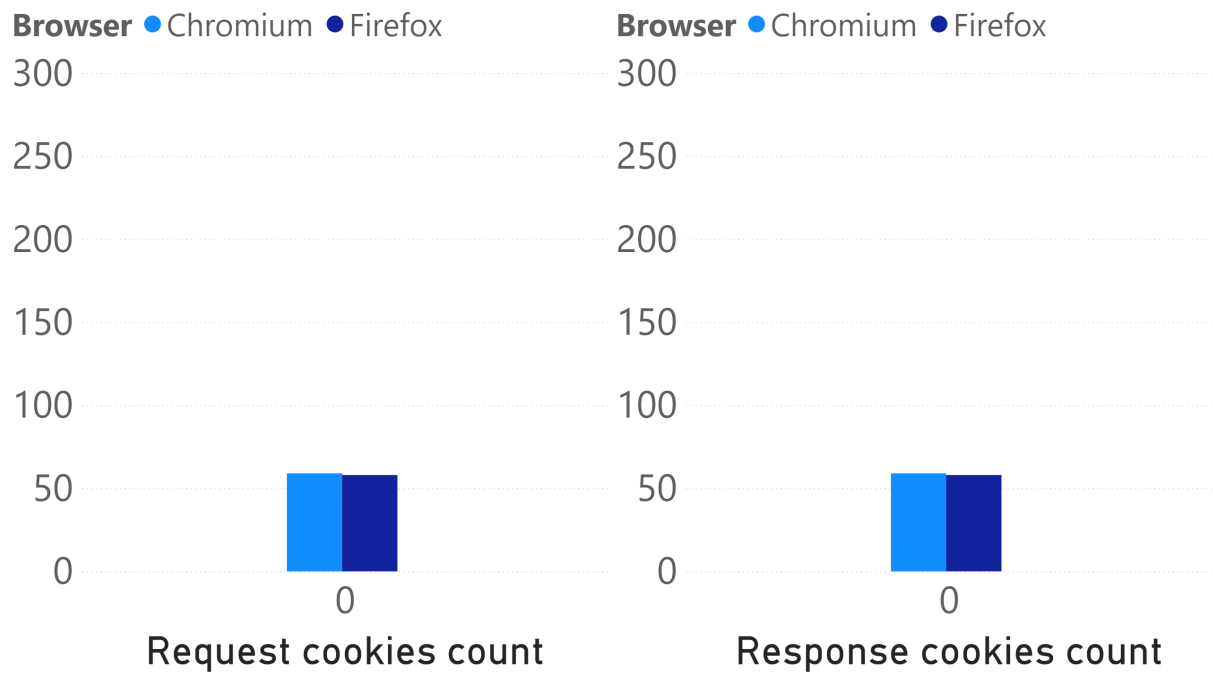


Figure 8: Cookies comparison in Brave

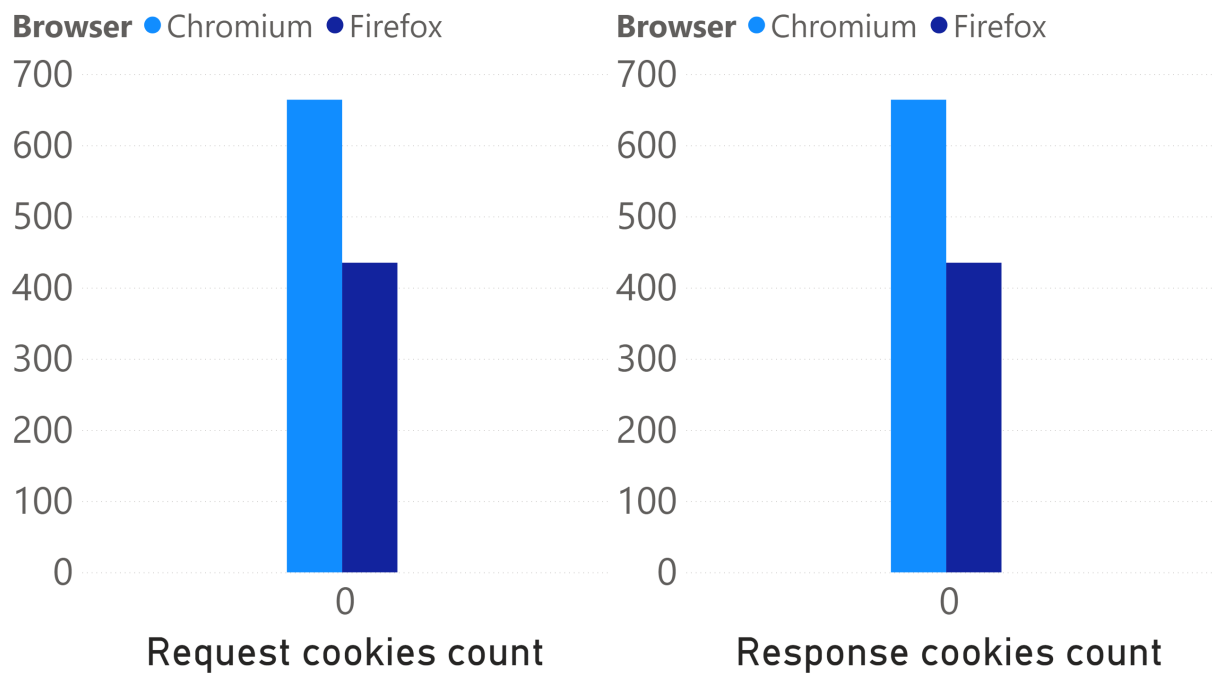


Figure 9: Cookies comparison in DuckDuckGo

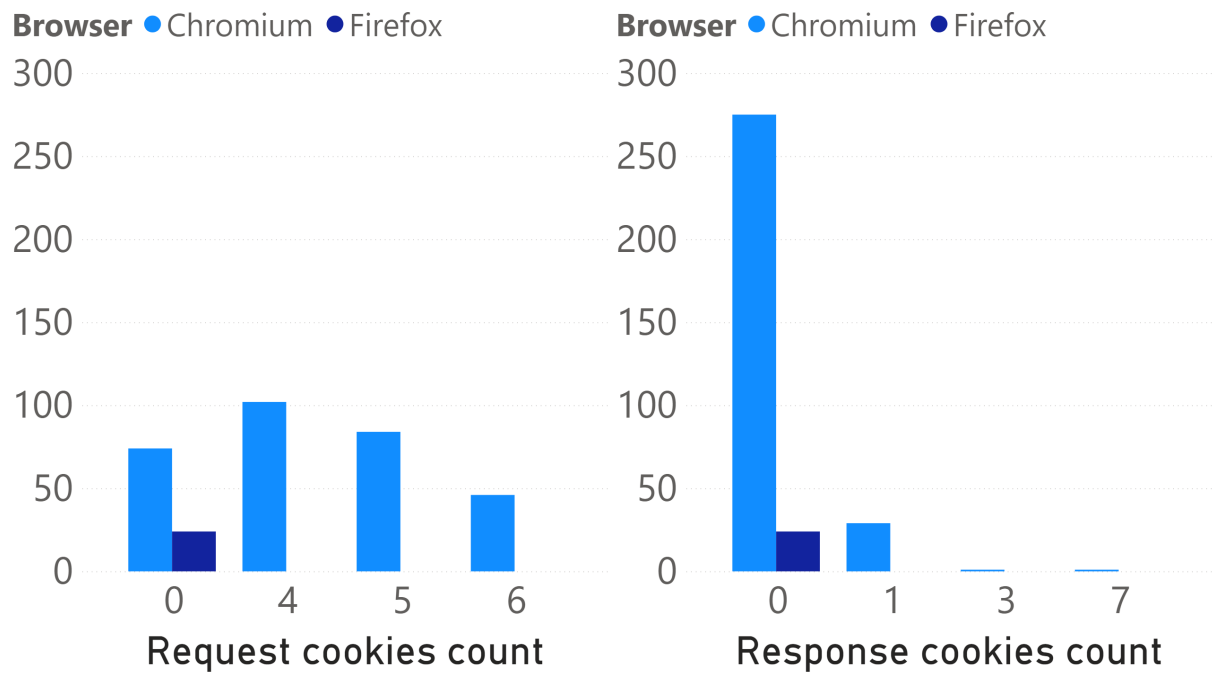


Figure 10: Cookies comparison in Google

A.2 Automation scripts

Listing 3: Bash automation script

```
language
1  #!/usr/bin/env bash
2  set -euo pipefail
3
4  QUERIES=(
5      "weather oslo"
6      "migraine symptoms"
7      "vitamin d deficiency"
8      "running shoes"
9      "coffee grinder"
10     "best laptop for students"
11     "electric car charging"
12     "cheap flights to london"
13     "home insurance"
14     "python list tutorial"
15     "banana bread recipe"
16     "news norway"
17 )
18
19 for query in "${QUERIES[@]}; do
20     echo "Running query: $query"
21
22     capture_search_bar \
23         --query "$query" \
24         --browser chromium \
25         --wait-until load \
26         --headed \
```

```

27         --output-dir normal_chromium \
28         --timeout-ms 60000
29
30     capture_search_har \
31         --query "$query" \
32         --browser chromium \
33         --wait-until load \
34         --headed \
35         --output-dir tor_chromium \
36         --timeout-ms 60000 \
37         --proxy socks5://127.0.0.1:9050
38
39     capture_search_har \
40         --query "$query" \
41         --browser firefox \
42         --wait-until load \
43         --headed \
44         --output-dir tor_firefox \
45         --timeout-ms 60000 \
46         --proxy socks5://127.0.0.1:9050
47
48     capture_search_har \
49         --query "$query" \
50         --browser firefox \
51         --wait-until load \
52         --headed \
53         --output-dir normal_firefox \
54         --timeout-ms 60000
55 done

```

Listing 4: Python capture HAR script

```

language
1  #!/usr/bin/env python3
2  """
3  Capture HAR files for search engine result pages using Playwright.
4
5  This script starts a fresh browser context per search engine, navigates to the
6  configured search URL, and writes one HAR file per engine.
7
8  It can use Tor if you pass --proxy socks5://HOST:PORT.
9  """
10
11  from __future__ import annotations
12
13  import argparse
14  from datetime import datetime
15  from pathlib import Path
16  from urllib.parse import quote_plus
17
18  from playwright.sync_api import sync_playwright
19
20
21  SEARCH_ENGINES = {
22      "google": "https://www.google.com/search?q={query}",
23      "duckduckgo": "https://duckduckgo.com/?q={query}&ia=web",
24      "bing": "https://www.bing.com/search?q={query}",
25      "brave": "https://search.brave.com/search?q={query}",
26  }

```

```

27
28
29 def parse_args() -> argparse.Namespace:
30     parser = argparse.ArgumentParser(
31         description="Capture search result HAR files with Playwright."
32     )
33     parser.add_argument(
34         "--query",
35         required=True,
36         help="Search query to use, for example: 'migraine symptoms'",
37     )
38     parser.add_argument(
39         "--engines",
40         nargs="+",
41         default=list(SEARCH_ENGINES),
42         choices=sorted(SEARCH_ENGINES),
43         help="Search engines to capture. Default: all",
44     )
45     parser.add_argument(
46         "--output-dir",
47         type=Path,
48         default=Path("data"),
49         help="Directory where HAR files are written. Default: ../data/har_capture",
50     )
51     parser.add_argument(
52         "--proxy",
53         default="",
54         help="Optional proxy, for example: socks5://127.0.0.1:9050",
55     )
56     parser.add_argument(
57         "--browser",
58         choices=["firefox", "chromium"],
59         default="firefox",
60         help="Browser engine to use. Default: firefox",
61     )
62     parser.add_argument(
63         "--timeout-ms",
64         type=int,
65         default=45000,
66         help="Navigation timeout in milliseconds. Default: 45000",
67     )
68     parser.add_argument(
69         "--wait-until",
70         choices=["load", "domcontentloaded", "networkidle"],
71         default="networkidle",
72         help="Navigation wait condition. Default: networkidle",
73     )
74     parser.add_argument(
75         "--headed",
76         action="store_true",
77         help="Show the browser window instead of running headless.",
78     )
79     return parser.parse_args()
80
81
82 def safe_filename_part(value: str) -> str:
83     keep = []
84     for char in value.lower():
85         if char.isalnum():

```



```

86         keep.append(char)
87     elif char in {" ", "-", "_"}:
88         keep.append("_")
89     cleaned = "".join(keep).strip("_")
90     return cleaned[:80] or "query"
91
92
93 def main() -> None:
94     args = parse_args()
95     args.output_dir.mkdir(parents=True, exist_ok=True)
96
97     encoded_query = quote_plus(args.query)
98     query_part = safe_filename_part(args.query)
99     timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
100
101     with sync_playwright() as playwright:
102         browser_launcher = getattr(playwright, args.browser)
103         launch_options = {"headless": not args.headed}
104
105         if args.proxy:
106             launch_options["proxy"] = {"server": args.proxy}
107
108         browser = browser_launcher.launch(**launch_options)
109
110         try:
111             for engine in args.engines:
112                 search_url = SEARCH_ENGINES[engine].format(query=encoded_query)
113                 har_path = args.output_dir / f"{timestamp}_{engine}_{query_part}.har"
114
115                 context = browser.new_context(
116                     record_har_path=str(har_path),
117                     record_har_content="embed",
118                 )
119                 page = context.new_page()
120                 page.set_default_timeout(args.timeout_ms)
121                 page.goto(search_url, wait_until=args.wait_until, timeout=args.timeout_ms)
122                 context.close()
123
124                 print(f"{engine}: {har_path}")
125         finally:
126             browser.close()
127
128
129 if __name__ == "__main__":
130     main()

```

Listing 5: Python HAR to CSV script

```

language
1 #!/usr/bin/env python3
2 """
3 Convert HAR files to readable CSV files.
4
5 Output 1: har_entries.csv
6     One row per entry in log.entries. This is the most direct way to inspect
7     the HAR structure: each { ... } inside entries[] becomes one CSV row.
8
9 Output 2: har_summary.csv

```

```

10     One row per HAR file with simple totals.
11
12 The script does not remove cookie values or URLs. Treat the output as sensitive.
13 """
14
15 from __future__ import annotations
16
17 import argparse
18 import csv
19 import json
20 from pathlib import Path
21 from urllib.parse import parse_qs, urlparse
22
23
24 ENTRY_FIELDS = [
25     "har_filename",
26     "search_engine",
27     "entry_index",
28     "startedDateTime",
29     "time_ms",
30     "method",
31     "url",
32     "domain",
33     "path",
34     "query_text",
35     "status",
36     "statusText",
37     "request_cookie_count",
38     "request_cookie_names",
39     "request_cookie_values",
40     "response_cookie_count",
41     "response_cookie_names",
42     "response_cookie_values",
43     "query_param_count",
44     "query_param_names",
45     "query_param_values",
46     "request_header_count",
47     "response_header_count",
48     "post_data_present",
49     "request_body_size",
50     "response_body_size",
51     "response_content_size",
52     "transferred_bytes_approx",
53     "is_third_party_domain",
54     "tracking_hint",
55 ]
56
57
58 SUMMARY_FIELDS = [
59     "har_filename",
60     "search_engine",
61     "query_text",
62     "requests_total",
63     "unique_domains",
64     "third_party_requests",
65     "request_cookies_total",
66     "response_cookies_total",
67     "query_params_total",
68     "post_requests_total",

```

```

69     "tracking_hint_requests",
70     "transferred_kb_approx",
71     "page_load_ms",
72     "status_2xx",
73     "status_3xx",
74     "status_4xx",
75     "status_5xx",
76 ]
77
78
79 TRACKING_WORDS = [
80     "ads",
81     "adservice",
82     "analytics",
83     "collect",
84     "conversion",
85     "doubleclick",
86     "event",
87     "gen_204",
88     "googleadservices",
89     "improving",
90     "log",
91     "metrics",
92     "pagead",
93     "telemetry",
94     "track",
95 ]
96
97
98 def detect_search_engine(har_path: Path) -> str:
99     name = har_path.name.lower()
100     if "duckduckgo" in name:
101         return "DuckDuckGo"
102     if "google" in name:
103         return "Google"
104     return "Unknown"
105
106
107 def read_har(path: Path) -> dict:
108     with path.open(encoding="utf-8", errors="replace") as file:
109         return json.load(file)
110
111
112 def entries_from_har(har_data: dict) -> list[dict]:
113     return har_data.get("log", {}).get("entries", []) or []
114
115
116 def pages_from_har(har_data: dict) -> list[dict]:
117     return har_data.get("log", {}).get("pages", []) or []
118
119
120 def cookie_names(cookies: list[dict]) -> str:
121     return "|".join(cookie.get("name", "") for cookie in cookies)
122
123
124 def cookie_values(cookies: list[dict]) -> str:
125     return "|".join(cookie.get("value", "") for cookie in cookies)
126
127

```

```

128 def query_names(query_items: list[dict]) -> str:
129     return "|".join(item.get("name", "") for item in query_items)
130
131
132 def query_values(query_items: list[dict]) -> str:
133     return "|".join(item.get("value", "") for item in query_items)
134
135
136 def positive_number(value: object) -> int:
137     if isinstance(value, (int, float)) and value > 0:
138         return int(value)
139     return 0
140
141
142 def approximate_transferred_bytes(entry: dict) -> int:
143     request = entry.get("request", {}) or {}
144     response = entry.get("response", {}) or {}
145     content = response.get("content", {}) or {}
146
147     return (
148         positive_number(request.get("headersSize"))
149         + positive_number(request.get("bodySize"))
150         + positive_number(response.get("headersSize"))
151         + positive_number(response.get("bodySize"))
152         + positive_number(content.get("size"))
153     )
154
155
156 def extract_query_text_from_url(url: str) -> str:
157     parsed = urlparse(url)
158     query = parse_qs(parsed.query, keep_blank_values=True)
159     values = query.get("q", [])
160     return values[0] if values else ""
161
162
163 def has_tracking_hint(domain: str, path: str, url: str) -> str:
164     text = f"{domain} {path} {url}".lower()
165     return "yes" if any(word in text for word in TRACKING_WORDS) else "no"
166
167
168 def max_page_load_ms(entries: list[dict], pages: list[dict]) -> float:
169     max_time = 0.0
170
171     for page in pages:
172         on_load = (page.get("pageTimings", {}) or {}).get("onLoad", -1)
173         if isinstance(on_load, (int, float)) and on_load > max_time:
174             max_time = float(on_load)
175
176     for entry in entries:
177         entry_time = entry.get("time", -1)
178         if isinstance(entry_time, (int, float)) and entry_time > max_time:
179             max_time = float(entry_time)
180
181     return max_time
182
183
184 def main_domain_for_engine(search_engine: str) -> str:
185     if search_engine == "Google":
186         return "google."

```

```

187     if search_engine == "DuckDuckGo":
188         return "duckduckgo.com"
189     return ""
190
191
192 def make_entry_rows(har_path: Path) -> list[dict]:
193     har_data = read_har(har_path)
194     entries = entries_from_har(har_data)
195     search_engine = detect_search_engine(har_path)
196     main_domain = main_domain_for_engine(search_engine)
197     rows = []
198
199     for index, entry in enumerate(entries, start=1):
200         request = entry.get("request", {}) or {}
201         response = entry.get("response", {}) or {}
202         content = response.get("content", {}) or {}
203         url = request.get("url", "")
204         parsed = urlparse(url)
205         request_cookies = request.get("cookies", []) or []
206         response_cookies = response.get("cookies", []) or []
207         query_items = request.get("queryString", []) or []
208         domain = parsed.netloc.lower()
209         path = parsed.path
210         query_text = extract_query_text_from_url(url)
211         third_party = "no"
212
213         if main_domain and domain and main_domain not in domain:
214             third_party = "yes"
215
216         rows.append(
217             {
218                 "har_filename": har_path.name,
219                 "search_engine": search_engine,
220                 "entry_index": index,
221                 "startedDateTime": entry.get("startedDateTime", ""),
222                 "time_ms": entry.get("time", ""),
223                 "method": request.get("method", ""),
224                 "url": url,
225                 "domain": domain,
226                 "path": path,
227                 "query_text": query_text,
228                 "status": response.get("status", ""),
229                 "statusText": response.get("statusText", ""),
230                 "request_cookie_count": len(request_cookies),
231                 "request_cookie_names": cookie_names(request_cookies),
232                 "request_cookie_values": cookie_values(request_cookies),
233                 "response_cookie_count": len(response_cookies),
234                 "response_cookie_names": cookie_names(response_cookies),
235                 "response_cookie_values": cookie_values(response_cookies),
236                 "query_param_count": len(query_items),
237                 "query_param_names": query_names(query_items),
238                 "query_param_values": query_values(query_items),
239                 "request_header_count": len(request.get("headers", []) or []),
240                 "response_header_count": len(response.get("headers", []) or []),
241                 "post_data_present": "yes" if request.get("postData") else "no",
242                 "request_body_size": request.get("bodySize", ""),
243                 "response_body_size": response.get("bodySize", ""),
244                 "response_content_size": content.get("size", ""),
245                 "transferred_bytes_approx": approximate_transferred_bytes(entry),

```

```

246         "is_third_party_domain": third_party,
247         "tracking_hint": has_tracking_hint(domain, path, url),
248     }
249 )
250
251 return rows
252
253
254 def make_summary_row(har_path: Path, entry_rows: list[dict]) -> dict:
255     har_data = read_har(har_path)
256     entries = entries_from_har(har_data)
257     pages = pages_from_har(har_data)
258     domains = {row["domain"] for row in entry_rows if row["domain"]}
259     status_counts = {2: 0, 3: 0, 4: 0, 5: 0}
260     query_text = ""
261
262     for row in entry_rows:
263         if row["query_text"] and not query_text:
264             query_text = row["query_text"]
265
266         status = row["status"]
267         if isinstance(status, int):
268             group = status // 100
269             if group in status_counts:
270                 status_counts[group] += 1
271
272     transferred_bytes = sum(int(row["transferred_bytes_approx"]) for row in entry_rows)
273
274     return {
275         "har_filename": har_path.name,
276         "search_engine": detect_search_engine(har_path),
277         "query_text": query_text,
278         "requests_total": len(entry_rows),
279         "unique_domains": len(domains),
280         "third_party_requests": sum(
281             1 for row in entry_rows if row["is_third_party_domain"] == "yes"
282         ),
283         "request_cookies_total": sum(int(row["request_cookie_count"]) for row in
            entry_rows),
284         "response_cookies_total": sum(
285             int(row["response_cookie_count"]) for row in entry_rows
286         ),
287         "query_params_total": sum(int(row["query_param_count"]) for row in entry_rows),
288         "post_requests_total": sum(1 for row in entry_rows if row["method"] == "POST"),
289         "tracking_hint_requests": sum(1 for row in entry_rows if row["tracking_hint"] ==
            "yes"),
290         "transferred_kb_approx": round(transferred_bytes / 1024, 2),
291         "page_load_ms": round(max_page_load_ms(entries, pages), 2),
292         "status_2xx": status_counts[2],
293         "status_3xx": status_counts[3],
294         "status_4xx": status_counts[4],
295         "status_5xx": status_counts[5],
296     }
297
298
299 def write_csv(path: Path, fieldnames: list[str], rows: list[dict]) -> None:
300     with path.open("w", newline="", encoding="utf-8") as file:
301         writer = csv.DictWriter(file, fieldnames=fieldnames)
302         writer.writeheader()

```

```

303         writer.writerows(rows)
304
305
306 def parse_args() -> argparse.Namespace:
307     parser = argparse.ArgumentParser(description="Convert HAR files to readable CSV files
308         .")
309     parser.add_argument(
310         "--input-dir",
311         type=Path,
312         default=Path("data"),
313         help="Folder with .har files. Default: data",
314     )
315     parser.add_argument(
316         "--entries-output",
317         type=Path,
318         default=Path("har_entries.csv"),
319         help="CSV with one row per log.entries item. Default: har_entries.csv",
320     )
321     parser.add_argument(
322         "--summary-output",
323         type=Path,
324         default=Path("har_summary.csv"),
325         help="CSV with one row per HAR file. Default: har_summary.csv",
326     )
327     return parser.parse_args()
328
329 def main() -> None:
330     args = parse_args()
331     har_files = sorted(args.input_dir.glob("*.har"))
332
333     if not har_files:
334         raise SystemExit(f"No HAR files found in {args.input_dir}")
335
336     all_entry_rows = []
337     summary_rows = []
338
339     for har_path in har_files:
340         entry_rows = make_entry_rows(har_path)
341         all_entry_rows.extend(entry_rows)
342         summary_rows.append(make_summary_row(har_path, entry_rows))
343
344     write_csv(args.entries_output, ENTRY_FIELDS, all_entry_rows)
345     write_csv(args.summary_output, SUMMARY_FIELDS, summary_rows)
346
347     print(f"Wrote {len(all_entry_rows)} entry rows to {args.entries_output}")
348     print(f"Wrote {len(summary_rows)} summary rows to {args.summary_output}")
349
350
351 if __name__ == "__main__":
352     main()

```

A.3 Power Query transformation

Listing 6: Power Query ETL script

language

```

1  let
2      Kilde = Csv.Document(
3          Web.Contents(
4              "https://example.sharepoint.com/.../tor_chromium/har_entries.csv"
5          ),
6          [
7              Delimiter = ",",
8              Columns = 30,
9              QuoteStyle = QuoteStyle.None
10         ]
11     ),
12
13     #"Promoted Headers" =
14         Table.PromoteHeaders(
15             Kilde,
16             [PromoteAllScalars = true]
17         ),
18
19     #"Changed Column Types" =
20         Table.TransformColumnTypes(
21             #"Promoted Headers",
22             {
23                 {"har_filename", type text},
24                 {"search_engine", type text},
25                 {"entry_index", Int64.Type},
26                 {"startedDateTime", type datetime},
27                 {"time_ms", type text},
28                 {"method", type text},
29                 {"url", type text},
30                 {"domain", type text},
31                 {"path", type text},
32                 {"query_text", type text},
33                 {"status", Int64.Type},
34                 {"statusText", type text},
35                 {"request_cookie_count", Int64.Type},
36                 {"response_cookie_count", Int64.Type},
37                 {"query_param_count", Int64.Type},
38                 {"request_header_count", Int64.Type},
39                 {"response_header_count", Int64.Type},
40                 {"tracking_hint", type text}
41             },
42             "en"
43         ),
44
45     #"Added Search Engine Column" =
46         Table.AddColumn(
47             #"Changed Column Types",
48             "SearchEngine",
49             each
50                 if Text.Contains([har_filename], "bing")
51                 then "Bing"
52                 else if Text.Contains([har_filename], "google")
53                 then "Google"
54                 else if Text.Contains([har_filename], "duckduckgo")
55                 then "DuckDuckGo"
56                 else if Text.Contains([har_filename], "brave")
57                 then "Brave"
58                 else "Unknown"
59         ),

```



```

60
61     #"Added Proxy Column" =
62         Table.TransformColumnTypes(
63             Table.AddColumn(
64                 #"Added Search Engine Column",
65                 "Proxy",
66                 each "Tor"
67             ),
68             {"Proxy", type text}}
69     ),
70
71     #"Added Browser Column" =
72         Table.TransformColumnTypes(
73             Table.AddColumn(
74                 #"Added Proxy Column",
75                 "Browser",
76                 each "Chromium"
77             ),
78             {"Browser", type text}}
79     )
80
81 in
82     #"Added Browser Column"

```

Listing 7: Power Query merge script

```

language
1  let
2      Kilde = Table.Combine({
3          har_summary_normal_chromium,
4          har_summary_normal_firefox,
5          har_summary_tor_chromium,
6          har_summary_tor_firefox
7      })
8  in
9      Kilde

```