USENIX Security '25 Artifact Appendix: Big Help or Big Brother? Auditing Tracking, Profiling, and Personalization in Generative AI Assistants

Yash Vekaria
UC Davis

Aurelio Loris Canino
UNIRC

Jonathan Levitsky
UC Davis

Alex Ciechonski
UCL

Patricia Callejo
UC3M

Anna Maria Mandalari
UCL

Zubair Shafiq
UC Davis

A Artifact Appendix

A.1 Abstract

This artifact provides a reproducible framework for capturing and analyzing the network traffic generated while auditing Generative AI (GenAI) browser assistants. Our experiments enable researchers to investigate tracking, profiling and personalization in GenAI-powered extensions. We use a MITM-based infrastructure to record network traffic in the form of '.flow' files and then parsing them into a '.csv' file for further analysis of data collection and sharing practices of these assistants. Additionally, we also utilize our semi-automated framework for understanding profiling and personalization of the studied browser assistants using our novel prompting framework.

A.2 Description & Requirements

This artifact provides framework to collect data for auditing tracking, profiling, and personalization in Generative AI (GenAI) browser extensions. Our framework comprises capturing and analyzing the network traffic behaviour when using a GenAI extension. The artifact repository consists of the following components:

- **Flows/**: Directory contains the captured network traffic via Mitmproxy in the form of `.flow` files, generated while interacting with GenAI browser extensions. This directory is located in the current working directory and contains sub-directories labeled with extension name being tested (e.g., Merlin), and add the following sample `.flow` files available in the repository:
  - Merlin-Lin-Control.flow
  - Merlin-Lin-Search.flow
  - Merlin-Lin-Browse.flow
  - Merlin-Lin-Summarize.flow
- **Output/**: Directory contains CSV files generated by processing the captured `.flow` files. A sample CSV for the extension: Merlin is available in the repository as `Merlin.csv`. You can create this directory to place this `.csv` file in.
- **parse_flows_to_csvs.py**: Python script that parses `.flow` files into summarized CSV files, extracting key details about relevant flows. This includes timestamp, request domain, payload, response type, cookies, domain's parent organization and disconnect list based tracker's category (see Section A.2.5).
- **generate_entity_domain_mapping.py**: Python script that parses individual entity-domain mapping files for each domain from DuckDuckGo's tracker-radar repository and generates a consolidated `ddg.json`.
- **disconnect.json**: Disconnect's tracking protection list used to map domains to predefined tracker categories such as Advertising, Analytics, and Social.
- **ddg.json**: DuckDuckGo's entity-domain list maps the domains to corresponding parent organizations. `ddg.json` is an aggregated single file generated by parsing individual mapping files. tracking protection list used to map domains to predefined tracker categories such as Advertising, Analytics, and Social.
- **requirements.txt**: Lists Python package dependencies required to run the analysis pipeline.
- **README.md**: Detailed instructions for setting up Mitmproxy, configuring it in Google Chrome, capturing network traffic, and executing the analysis pipeline.

A.2.1 Security, privacy, and ethical concerns

This artifact involves intercepting HTTPS traffic using a custom root certificate generated by Mitmproxy. Evaluators must install and trust this certificate in a dedicated browser profile, which introduces potential security risks:

- All HTTPS traffic from the configured browser instance may be logged in a decrypted form by Mitmproxy.
- Evaluators must be careful in choosing what type of online personal or private spaces they login to. This is because in presence of GenAI assistants, user's sensitive information can be potentially be extracted and shared with assistant's server or LLM model's server.
- It is recommended to use a fresh Chrome profile and remove the Mitmproxy certificate after experiments.

For more information on ethical considerations, we encourage to read the corresponding section in the main paper.

A.2.2 How to access

The artifact can be accessed via Zenodo at the following link: https://doi.org/10.5281/zenodo.15530229

A.2.3 Hardware dependencies

No specialized hardware is required; the artifact can be run on standard consumer machines such as a laptop or a desktop.

A.2.4 Software dependencies

The artifact is compatible with any operating system such as Windows, Linux, and macOS. It requires Python 3.8+, `mitmproxy` for traffic interception, and relevant Python packages as listed in `requirements.txt`. Detailed setup instructions are provided in the README file.

A.2.5 Benchmarks

We use two external resources in our framework:
• *Disconnect List*: allows categorizing tracking domains into one of the 11 categories such as Advertising, Analytics, and Social. It is available at https://github.com/disconnectme/disconnect-tracking-protection/blob/master/services.json.
• *DuckDuckGo's Entity-Domain Mapping*: allows mapping a domain to its parent organization or owner. It is available at https://github.com/duckduckgo/tracker-radar/tree/main/domains/US.

A.3 Set-up

The steps listed here aid in setting up the necessary environment for evaluating the artifact.

A.3.1 Installation

❶ **MITM Proxy Installation:** First, install `mitmproxy` by following the official installation guide at (https://mitmproxy.org/).

❷ **Instantiate Chrome Profile:** Create a new Google Chrome profile and configure it to route its traffic through the `mitmproxy` server linked to (`localhost:8080`). First, visit chrome://version/ in your Google Chrome browser to figure out path to Chrome executable and/or user data directory.

For Windows 11, the required command is:

```
.\chrome.exe –proxy-server="localhost:8080" –user-data-dir="C:\Users\<YourUsername>\AppData\Local\Google\Chrome\User Data\Profile<ProfileNumber>"
```

For macOS, the required command is (you may need to escape spaces in path with "\"):

```
/Applications/Google\Chrome.app/Contents/MacOS/Google Chrome –proxy-server="localhost:8080" –user-data-dir="/Users/<YourUsername>/Library/Application Support/Google/Chrome/<ProfileNumber>"
```

❷ **Install MITM Certificate:** Now, in the instatiated browser, install the *mitmproxy* root certificate by visiting `mitm.it` in the configured browser and following the prompts to add it to the system's trusted authorities. This step is required only once and ensures that HTTPS traffic can be intercepted and analyzed.

❸ **Validation:** Once the browser is configured with `mitmproxy`, validate it by launching the configured Chrome profile, and confirming that intercepted traffic is saved in a `.flow` file. This can be done by running the following commands:

In the first terminal window, start `mitmproxy`'s web interface and capture flows:

For Windows:

```
mitmweb.exe -w <flowFileName>.flow
```

For macOS:

```
mitmweb -w <flowFileName>.flow
```

In the second terminal window, open the chrome profile by running the same commands as listed in ❷. We typically manually perform experiments with each GenAI browser extension – one at a time in the opened browser instance. Once the experiments are over, we stop commands in both the terminals.

❸ **[Optional] Inspect Flow Files in Web:** The generated flow files can be manually opened or inspected as follows:

For Windows:

```
mitmweb.exe -r <flowFileName>.flow
```

For macOS:

```
mitmweb -r <flowFileName>.flow
```

❹ **Install Python Dependencies:** For the analysis phase, set up a Python virtual environment and install the required Python dependencies:
Create a virtual environment:

```
python3 -m venv <envName>
```

Activate the created virtual environment:

```
source <envName>/bin/activate
```

Install requirements:

```
pip install -r requirements.txt
```

❺ **Download Benchmarks:** For disconnect list, download `services.json` from https://github.com/disconnectme/disconnect-tracking-protection/blob/master/services.json, place it inside the current working directory, and rename it to `disconnect.json`. For generating DuckDuckGo's mapping, run the following command anywhere and then place the resultantly generated `disconnect.json` in the current working directory.

```
python3 generate_entity_domain_mapping.py
```

A.3.2 Basic Test

A basic functionality check can be performed to verify the artifact is operational. For the sake of this test, we will consider the browser extension Merlin. Execute the following steps:

First, following Step ❸ in Section A.3.1, start `mitmweb` in one terminal and instantiate a fresh or unused chrome profile in the second parallel terminal window. Once the Chrome browser is open, install the GenAI browser extension (in this case Merlin by visiting: https://chromewebstore.google.com/detail/merlin-ask-ai-to-research/camppjleccjaphfdbohjdohecfnoikec). Next, login to the extension and pin it to your browser. Now, you can open any new tab and close all other tabs. Perform three interactions: *search*, *browse*, and *summarize*. For *search*, ask a question in Google search and let Merlin auto-generate the response in side space on the right. In case, it does not but provides an option to generate, click on that option in the side space. Alternatively, extension icon can be clicked to ask a question. For *browser*, visit any webpage and ask a question about the page to the Merlin by clicking on the extension icon. Lastly, for summarize, the same webpage can be summarized using one of the extension-provided options. Now close the browser window, and terminate the processes in both the terminal windows.

The successful capture of network traffic can be confirmed by verifying the creation of the `.flow` file in the working directory. The analysis script can be now run on the captured `.flow` file to generate a summary of the network traffic that supports the results of our paper. One of the random observations from our paper can be verified using the output `.csv` file stored in the `Output` folder: "Merlin can be seen contacting `google-analytics.com` and sharing user's query prompt to this domain in the payload." The analysis script can be run as follows:

```
python3 parse_flows_to_csvs.py
```

Alternatively,

```
python3 parse_flows_to_csvs.py <extensionName> OR
python3 parse_flows_to_csvs.py Merlin
```

The sample CSV can be easily generated by running the last command above to test the script: parse_flows_to_csvs.py.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): *We visit personal as well as private spaces online in presence of GenAI browser assistant and observe the assistant to collect varying granularity of data such as page's textual content, partial content, form fields, and whole DOM depending on the extension as depicted in Table 2. This is demonstrated by experiment (E1) and discussed in Section 5.3 of the paper.*

(C2): *An audit of sharing of collected information with first- as well as third-parties is measured for different user-, chat-, and browser-specific identifiers as shown in Table 3 and Figure 3. Sider and Merlin share chat identifiers with `google-analytics.com` while TinaMind shares it with `analytics.google.com`. Merlin also shares user's raw query prompts with Google Analytics. MaxAI and Harpa also send data to Mixpanel. This can be manually verified with the help of experiment (E2) and is discussed in Section 5.3 of the paper.*

(C3): *We show that some GenAI assistants profile and personalize more than others based on five user attributes: location, age, gender, income, and interests. More specifically, two extensions (Monica and Sider) profile and personalize in-context as well as out-of-context based on all five user attributes. In contrast, Perplexity and TinaMind showed no strong evidence of profiling or personalization, while Harpa exhibited only in-context personalization behavior. The results in Table 4 can be reproduced by performing experiment (E2) as discussed in Section 5.4 of the paper.*

A.4.2 Experiments

Our auditing framework is semi-automated – i.e., the infrastructure to collect data is automated but the experiments are manual.

(E1): *Auditing User Tracking (Data Collection) in GenAI Browser Assistants [30 human-minutes for experimentation + 15 human-minutes for manual analysis per extension]:*

Preparation: To reproduce results in Table 2, read Section 5.3 and refer to prompts in Table 6. Since the experiments were performed and analyzed manually, the evaluators can limit the evaluation to at most 20 scenarios, comprising of public as well as private spaces described in Table 2. Create fresh accounts for each of the 10 private spaces (wherever possible). Lastly, ensure that the crawling infrastructure is set-up and validated as explained in Section A.3. Also, select any one extension of choice to test and then ensure that you have created an extension account for that assistant. For ChatGPT for Google, you also need to link it with your `chatgpt.com` account.

Execution: To test the selected extension for each public or private space, perform the following steps. For each space, follow Section A.3.2 to start `mitmweb` in one terminal window and initiate a new browser profile in the second terminal. Within the browser instance, first install the extension, pin it, and perform the login to extension. Now, visit the space being tested and open some sensitive or copyrighted content to understand collection practices of the assistant. If it's a private space, first authenticate into the website and then navigate to an appropriate content page to test for. Refer to Table 6 row corresponding to the selected space and ask those prompts in the sidebar chat of the assistant one after the other. Once the testing complete, close the browser instance and stop the data collection. The generated .flow file can now be opened and analyzed within the browser as follows:

```
mitmweb.exe -r <flowFileName>.flow (Windows)
mitmweb -r <flowFileName>.flow (macOS)
```

Results: Manually analyze the flow file of each or a sample of evaluated spaces to validate the observed data sharing with respect to the results displayed in the Table 2 for the tested extension.

(E2): *Auditing User Tracking (Data Sharing), Profiling, and Personalization in GenAI Browser Assistants [30 human-minutes for experimentation + 15 human-minutes for manual analysis per extension]:*

Preparation: In this experiment, we perform three scenarios for each extension – search, browse, and summarize. Similar to E1, the experimentation and analysis was performed manually. Read Section 5.4 to understand how the experiments need to be performed for each scenario. Create fresh accounts for each scenario and each extension (wherever possible). If reusing the same account (e.g., Google account), ensure all data is deleted between subsequent tests and that each test is carried out in a fresh browser profile. Lastly, ensure that the crawling infrastructure is set-up and validated as explained in Section A.3. To avoid spending significant time, select any one extension of choice to test for.

Execution: Similar to Experiment E1, for each scenario per extension, use a distinct browser profile and start `mitmweb` and instance of the browser from two separate terminal windows, followed by installing the extension and performing the login. Finally, perform the experiment for each scenario by referring to Section 5.4 of our paper. Section 5.4 describes the steps to test different scenarios using profiling and personalization prompts listed in Section 8.1 (Appendix of the main paper). The webpages to visit during the browse and summarize scenarios are listed in Table 5 in Section 8.3. After running experiments for all scenarios of an extension, generate a summarized .csv file by running: parse_flows_to_csvs.py <extensionName>.

Results: The generated `.csv` file of each scenario per extension will be used to compare the data sharing practices of the assistant with the first- and third-parties. This is achieved by manually comparing the observed practices in payload, request URL, and cookies against Table 3. Likewise, third-party sharing flows represented in Figure 3 can also be easily checked. Next, profiling and personalization of the selected assistant across the three scenarios can be validated against Table 4. This is done by analyzing the responses returned by the assistant while performing the experiment itself.

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2025/.