USENIX Security ’25 Artifact Appendix: *“I have no idea how to make it safer”*: Studying Security and Privacy Mindsets of Browser Extension Developers

Shubham Agarwal, Rafael Mrowczynski, Maria Hellenthal, Ben Stock

CISPA Helmholtz Center for Information Security, Saarbruecken, Germany
{shubham.agarwal, mrowczynski, hellenthal, stock}@cispa.de

A Artifact Appendix

A.1 Abstract

We studied the security and privacy mindset of browser extension developers by conducting a semi-structured interview with them. As presented in the main paper, we interviewed 21 extension developers and observed that they only have surface knowledge of S&P issues and threats associated with extensions. To worsen the situation, they reported having little to no idea on how to best these threats in most cases.

We open-source all the artifacts associated with the design of the interviews – the interview guide, the code book, the email invites used for recruitment, and the pre-screening survey questionnaire used to select candidates. Notably, our semi-structured interview also required the participants to work on a set of coding tasks to assess their working knowledge of security and privacy practices. Hence, we also open source the prototypical Web application and the set of browser extensions used for individual coding tasks in their functional state for transparency and reuse in the future.

A.2 Description & Requirements

The functional components of our artifact include a prototypical e-commerce application and the set of browser extensions that we created to facilitate the coding tasks. The Web application is based on the Django framework and includes a `docker-compose.yml` file that allows quick and easy deployment on any host machine, with `docker` installed on it.

Further, each of the browser extensions contains logic to perform interactions with the DOM of the visited Web page (e.g., create buttons, inject iframes, etc.) based on the corresponding coding tasks. Thus, to verify their functionality, it is necessary to load individual extensions into the browser, and hence, a headful evaluation environment is required. However, the browser extensions are platform-agnostic and are tested to work on the following browsers: *Google Chrome*, *Brave*, *Arc*, *Opera*, *Microsoft Edge* and *Mozilla Firefox*.

A.2.1 Security, privacy, and ethical concerns

There are no security and privacy concerns associated with our artifacts. The prototype Web application and the browser extensions do not collect any data or interact with any of *our* remote servers whatsoever. However, we note that the browser extensions, corresponding to *Coding Task 2* and *Coding Task 3*, send network requests with session cookies, if available, to `amazon.de` and `connect.facebook.com`, respectively, necessary for the functionality of the coding tasks. Thus, we advise our users and evaluators to load these extensions in a *clean/incognito* environment to avoid any privacy issues.

A.2.2 How to access

The artifact can be accessed and downloaded from the following link: `https://doi.org/10.5281/zenodo.15550089`.

A.2.3 Hardware dependencies

There are no hardware dependencies for our artifact other than the fact that the functionality of the browser extensions can only be observed in a headful environment (i.e., PC or laptop with display and *not* a remote server).

A.2.4 Software dependencies

The operating system environment should allow running (docker) container environment. Alternatively, as we detail later, one can run the Web applications without docker by installing Django and other necessary Python packages directly on their host machine. Further, as mentioned before, the functional evaluation of browser extensions requires them to be loaded into the browser. Thus, at least one of the following browsers must be installed on the machine: *Google Chrome*, *Brave*, *Arc*, *Opera*, *Microsoft Edge* and *Mozilla Firefox*.

A.2.5 Benchmarks

None.

A.3 Set-up

We now outline the steps to set up the evaluation environment for the Web application and the browser extensions. As mentioned above, the tests only require the Web application to be running (either through the docker container or natively) and a browser that supports extensions. Once the artifact is downloaded, please unzip the following folders for evaluation: `dev_study.zip` and `extension_coding_tasks.zip`.

A.3.1 Installation

We assume our users/evaluators already have one of the required browsers installed on their machine. If not, we will allow them to install the browser they choose for the tests. Next, to enable loading unpublished browser extensions locally, please follow the following steps for Chromium browsers:

1. Navigate to `chrome://extensions`.
2. Toggle the **Developer Mode** button on the top-right corner of the page to switch it on. This allows browser extensions to be loaded from a local machine.

Firefox does not require any manual configurations to load extensions locally.

To install docker, please follow the instructions on the official documentation based on the host environment, accessible at `https://docs.docker.com/engine/install/`. After installing the docker, please ensure that the docker daemon is already running in the background, as follows:

```
$ docker --version
```

`[Docker Alternative]`: If the host environment does not allow running docker containers, one can run the Web application natively through Python. For this, they need to install the related project dependencies first, as follows:

```
$ pwd
./extension-developer-study

$ cd dev_study/apps/app1/
$ pip install -r requirements.txt
```

A.3.2 Basic Test

To run the Web application on your machine:

- First, navigate to the application directory
  ```
  $ cd dev_study
  ```
- Next, run the docker container.
  ```
  $ docker compose up --build -d
  ```
- Alternatively, if you plan to run the Web application natively, execute the following:
  ```
  $ cd apps/app1/
  $ python manage.py runserver 9000
  ```

Now, the Web applications should be running and accessible at: `http://127.0.0.1:9000`. Default login credentials: username: `foo`, password: `bar`.

Note: While running the application in a docker-less fashion, please ensure that the `python` version is configured correctly to `>=3.10` for compatibility with dependencies. The default port number specified within the `docker-compose.yml` files is 9000. So, please ensure the port is available for use or set the desired port number in the corresponding files.

Next, to load any of the browser extensions into the browser:

For Chromium browsers:

1. Navigate to `chrome://extensions`.
2. Then, click **Load unpacked** and select one of the extension directories (e.g., extension_coding_tasks/primary/task1).

For Mozilla Firefox:

1. Navigate to `about:debugging#/runtime/this-firefox`.
2. Then, click **Load Temporary Add-on...** and select the manifest of one of the extensions (e.g., extension_coding_tasks/primary/task2/manifest.json).

The extension should be loaded without errors/warnings. The functional test environment is now set up.

A.4 Evaluation workflow

A.4.1 Major Claims

The following are the major claims that need to be evaluated for the functionality of our artifact:

(C1): The prototypical Web application is functional and contains necessary pages required for the coding tasks.

(C2): The browser extensions corresponding to each coding task is functional and injects necessary DOM elements into the e-commerce application above.

Note: The browser extensions are only functional to the point that it injects DOM elements into the appropriate page (in E1), and includes partial logic to perform operations (in E2 & E3). The study participants implemented the rest of the S&P-related logic, as listed in Table 1 of the main paper.

A.4.2 Experiments

(E1): *[Coding Task 1] [<5 human-minutes]:*

- Load the `task1` extension from `extension_coding_task/primary/task1/` into the browser.
- After running the Web application locally, navigate to `http://127.0.0.1:9000` and login with default credentials.
- Navigate to `http://127.0.0.1:9000/products` or click on the *Product* tab on top. Then, click on the *Order Now* button for any of the listed items to navigate to the *Order* page.
- The `task1` extension should inject two additional buttons – *Save Address for Later* and *Autofill Address Data*, before the *Continue to Payment* button (as shown in Figure 1). Disabling the extension will remove the two injected buttons from the UI.

(E2): *[Coding Task 2] [<5 human-minutes]:*

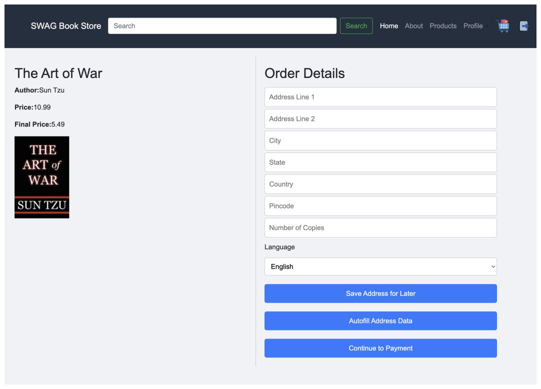

Figure 1: Expected view of the *order* page when `task1` extension is loaded.

- Load the `task2` extension from `extension_coding_task/primary/task2/` into the browser.
- Navigate to http://127.0.0.1:9000/products or click on the *Product* tab.
- The browser extension injects an additional button – `Check on Amazon` below the `Order Now` for each listed product, as shown in Figure 2. This button and the associated logic are supposed to allow checking the price of the listed product on *Amazon*.
- However, when clicking on the injected button, the *iframe* does not work since the interview participants are supposed to fix the issue. Disabling the corresponding extension will remove the button from the UI.

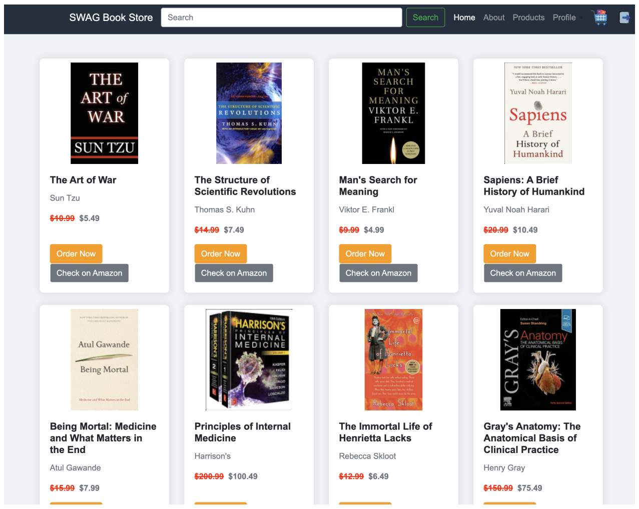

Figure 2: Expected view of the *product* page when `task2` extension is loaded.

(E3): [Optional]: *[Coding Task 3] [<5 human-minutes]:*

- Load the `task3` extension from `extension_coding_task/primary/task3/` into the browser.
- Navigate to http://127.0.0.1:9000/products or click on the *Product* tab.
- Here, since the application sends the CSP header, which blocks all third-party scripts from being executed. Thus, the page UI has no visual cues to verify the extension's functionality.
- However, one can open the *Developer Tools* and verify the functionality by looking for the corresponding error in the console – *inject.js:10 Refused to load the script 'https://connect.facebook.net/en_US/sdk.js' because it violates the following Content Security Policy directive: "script-src 'self'". Note that 'script-src-elem' was not explicitly set, so 'script-src' is used as a fallback*.
- Alternatively, one can also inspect the outgoing network logs to https://connect.facebook.net/en_US/sdk.js.

A.5 Notes on Reusability

The Web Application & Browser Extensions: The prototypical e-commerce application could very well be extended with more functionalities and pages without actually capturing any data based on the requirements and research focus. Similarly, the open-sourced extension boilerplate could be used as is or modified accordingly to interact with the e-commerce application developed here. However, we strictly advise against using our extension source code for real-world extension development or publishing, as the manifest of these extensions includes redundant host and API permissions.

Interview Guide: Our interview guide elaborates on the structure, topical areas, and related questions we asked our participants in our semi-structured interviews. Future studies may build upon this guide for similar studies on browser extensions, or, in general that may include coding tasks.

Email Invites: The content of the email invites that we sent to our participants only partially disclosed our background. That is, we only identified ourselves as *Computer Scientists* and not *Security/Privacy experts* to eliminate any potential bias during recruitment or in later phases of our study, however, without compromising the ethical boundaries of recruitment. Researchers may reuse our email content after appropriate changes to recruit developers under similar constraints.

A.6 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/.