USENIX Security '24 Artifact Appendix: Arcanum: Detecting and Evaluating the Privacy Risks of Browser Extensions on Web Pages and Web Content

Qinge Xie, Manoj Vignesh Kasi Murali, Paul Pearce, Frank Li
Georgia Institute of Technology

A Artifact Appendix

A.1 Abstract

In this work, we develop Arcanum, a dynamic taint tracking system for modern Chrome extensions designed to monitor the flow of sensitive user content from web pages. Arcanum defines a diverse set of taint sources ranging from meta-data, to content DOM elements, location information, history data, and cookies. From these sources, Arcanum is able to track data flow to a variety of exit taint sinks, including all forms of web requests and storage APIs. A key feature of Arcanum is allowing researchers to instrument specific web page elements as tainted at runtime via JS DOM annotations. We deploy Arcanum to test all functional extensions currently in the Chrome Web Store for the automated exfiltration of user data across seven sensitive websites: Amazon, Facebook, Gmail, Instagram, LinkedIn, Outlook, and PayPal. We observe significant privacy risks across thousands of extensions, including hundreds of extensions automatically extracting user content from within web pages.

The Arcanum prototype is built on Chromium Browser version 108.0.5359.71. We open-source all Chromium patches of the Arcanum implementation, allowing users to build and adapt Arcanum from the Chromium source code. In the artifacts, we provide custom Chrome extensions for testing Arcanum's functionality, as well as representative real-world extensions that were evaluated and flagged by Arcanum.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

When evaluating extension behavior on a web page using Arcanum, we strongly recommend users to record that web page and replay it across the extensions evaluated using WprGo. By only visiting the pages once and replaying the captured response, we can minimize additional load on a site regardless of the number of extensions evaluated. We provide web page recordings for all experimental target pages in this artifact. We also note that Arcanum itself does not require replay and can be run on live web pages.

A.2.2 How to access

This artifact, including 1) all Chromium patches of the Arcanum implementation, 2) sample extensions, 3) JavaScript files for annotating specific DOM elements on web pages, 4) web page recordings, and 5) Python test case scripts for each sample extension, is hosted on GitHub and can be accessed via: `https://github.com/BEESLab/Arcanum/releases/tag/1.0`. Additionally, we provide two Docker images on Docker Hub that have the necessary dependencies for 1) building Arcanum from the Chromium source code, and 2) running Arcanum to test sample extensions. They can be pulled via "`docker pull xqgtiti/arcanum_build:latest`", and "`docker pull xqgtiti/arcanum_run:latest`", respectively.

A.2.3 Hardware dependencies

Our experiments can be conducted either directly on a single physical host machine or alternatively in a VM-based lab environment on a host machine.

A x86_64 (amd64) machine with at least 8 GiB RAM, 4 cores/8 threads CPU, and at least 100 GiB of free disk space is required. More than 16 GiB RAM is highly recommended. For reference, all our experiments were conducted on a physical machine with 512 GiB RAM, 32 cores/128 threads CPU, running Ubuntu 20.04.6 LTS (Kernel Linux 5.4.0-173-generic). The provided test cases were also evaluated on another Linux server with 8 CPUs and 16 GiB RAM.

A.2.4 Software dependencies

Git is required for fetching the Chromium source code 108.0.5359.71 (Linux) and depot tools.

To build Arcanum, we provide a Docker image (Ubuntu 20.04) that includes all necessary dependencies for building a version of Chromium patched with the Arcanum implementation. Alternatively, users can follow the official instruction to build your own Docker container.

To run Arcanum, we provide a Docker image (Ubuntu 18.04) that includes all necessary dependencies. We strongly

recommend using this pre-configured image, as our test case scripts partly rely on its settings (such as software executable paths). If you choose to build the environment manually, the following software dependencies are required:

- Go 1.19.12 Linux
- WprGo v0.0.0-20230901234838-f16ca3c78e46
- Python 3.8.0 Linux (with Selenium 4, pyvirtualdisplay)
- ChromeDriver 108.0.5359.71
- Xvfb
- Chromium Dependencies

While we have not verified compatibility with versions other than those listed above, we believe that our artifacts will work with Ubuntu 18.04, 20.04, and 22.04, Python 3.8+ (Selenium 4), and any versions of Xvfb and WprGo.

A.2.5 Benchmarks

To benchmark the performance of Arcanum, we provide two types of sample extensions in this artifact as the dataset:
Custom Extensions. We provide sample extensions implemented by ourselves to demonstrate how extensions can be tested using Arcanum, on seven websites that were experimented with in our paper (Amazon, Facebook, Gmail, Instagram, LinkedIn, Outlook, and PayPal). For each site, we provide one Manifest Version 2 (MV2) extension and one Manifest Version 3 (MV3) extension. In addition to these extensions, we also provide custom extensions to guide users in testing the taint tracking process of Arcanum, including testing different taint sources, sinks, and propagation cases.
Real-world Extensions. We provide representative extensions from the Chrome Web Store that have been tested and flagged by Arcanum. Specifically, we include all extensions discussed as case studies of Section 4.10, as well as those listed in Table 7 of Section 4.5 (i.e., the flagged extensions with the most users in each web content category) in our paper. The extension IDs are listed in Table 1:

Extension ID	Paper Section
aamfmnhcipnbjjnbfmaoooiohikifefk	Case Study, Table 7
haphbbhhknaonfloinidkcmadhfjoghc	Case Study, Table 7
jdianbbpnakhcmfkcckaboohfgnngfcc	Case Study, Table 7
oadkgbgppkhoaaoepjbcnjejmkknaobg	Case Study, Table 7
blcdkmjcpgjojjffbdkckaiondfpoglh	Case Study
kecadfolelkekbfmmfoifpfalfedeljo	Table 7
nkecaphdplhfmmbkcfnknejeonfnifbn	Table 7
bahcihkpdjlbndandplnfmejnalndgjo	Table 7
pjmfidajplecneclhdghcgdefnmhhlca	Table 7
mdfgkcdjgpgoeclhefnjgmollcckpedk	Table 7

Table 1: Real-world extension IDs.

Web Page Recording Files. Since popular sites periodically mutate to combat automation/scraping, we provide recording files for each target page of the seven websites (listed in Table 3 of our paper) for consistent evaluation. JS scripts for annotating privacy-sensitive DOM elements on each web page are also provided. All sample extensions mentioned above are tested by Arcanum using these recording files in this artifact.

A.3 Set-up

A.3.1 Installation

This section describes how to set up a build environment for Chromium and build a version of Chromium with the patches of the Arcanum implementation. The set-up is mostly based on the official Chromium build instructions on Linux.

1. On the host machine, clone the Chromium depot tools to a specific directory (e.g., `$HOME`) and add their path to the PATH environment variable.

```
1 git clone https://chromium.googlesource.
    ↪ com/chromium/tools/depot_tools.git
2 export PATH="${HOME}/depot_tools:$PATH"
```

2. Get the Chromium source code (this may take a while depending on your network connection).

```
1 mkdir ${HOME}/chromium && cd ${HOME}/
    ↪ chromium/
2 fetch --nohooks chromium
```

3. In `src/`, check out the branch for Chromium version 108.0.5359.71. You could also refer to the official instructions on working with Chromium release branches.

```
1 cd src
2 gclient sync --with_branch_heads --
    ↪ with_tags
3 git fetch
4 git checkout tags/108.0.5359.71
5 gclient sync --with_branch_heads --
    ↪ with_tags
```

4. Prepare the Docker container for building. Pull the provided Docker image for building, then launch a Docker container from this image. Make sure to mount the host directory containing the Chromium source code and the depot_tools into the container:

```
1 docker pull xqgtiti/arcanum_build:latest
2 docker run -it --mount src=${HOME},target
    ↪ ="/mnt/build/",type=bind --name=
    ↪ build xqgtiti/arcanum_build:latest
```

5. Prepare build in the Docker container's interactive shell. Add the path of Chromium depot tools to the PATH environment variable. The command "`gn args ...`" automatically opens a file (`args.gn`) in the default text editor. Replace the contents of this file with the contents of the file "~`/build/args.gn`" from the artifact GitHub repository.

```
export PATH="/mnt/build/depot_tools:$PATH"
cd /mnt/build/chromium/src/
gn args out/Default
```

6. After updating the contents of the `args.gn` file, run the above command again to finalize the build preparations.

```
gn args out/Default
```

7. Build an unmodified Chrome and its Linux installer (this may take a while depending on the host machine's performance).

```
ninja -C out/Default chrome
ninja -C out/Default "chrome/installer/
    ↪ linux:unstable_deb"
```

8. Build a version of Chrome patched with the Arcanum implementation (located in "∼/`patches/`" in the artifact GitHub repository).

```
cd /mnt/build/chromium/src/
git apply ~/patches/chromium.patch
cd /mnt/build/chromium/src/v8/
git apply ~/patches/v8.patch

cd /mnt/build/chromium/src/
gn args out/Arcanum
cp out/Default/gn.args out/Arcanum/
gn args out/Arcanum
ninja -C out/Arcanum chrome
```

9. Build a Linux installer for Arcanum, you can then find the `.deb` file in "`../out/Arcanum/`".

```
ninja -C out/Arcanum "chrome/installer/
    ↪ linux:unstable_deb"
cd /mnt/build/chromium/src/out/Arcanum/
ls chromium-browser-unstable_108
    ↪ .0.5359.71-1_amd64.deb
```

A.3.2 Basic Test

Pull the provided Docker image for running Arcanum, and then launch a Docker container from this image. Note that the "`--privileged`" flag is required. You can also mount any directory that is convenient for transferring files from the host machine.

```
docker pull xqgtiti/arcanum_run:latest
docker run -it --privileged --name=run
    ↪ xqgtiti/arcanum_run:latest
```

Copy the Arcanum installer file (i.e., the `.deb` file) to "`/root/Arcanum/`" in the Docker container and decompress it. Note that we use this path in the test case code as the Arcanum executable path. Please modify the variable in the code if you place the installer elsewhere.

```
cd /root/Arcanum/
ar x chromium-browser-unstable_108
    ↪ .0.5359.71-1_amd64.deb
tar -vxf control.tar && tar -vxf data.tar
```

Run the basic test case `Basic_Test.py` from the artifact GitHub repository in the interactive shell of the container, using the pre-configured Python 3.8. The basic test case uses Selenium to launch Arcanum (a modified Chromium) with an empty extension pre-installed and navigates to a web page.

```
python3.8 ~/Test_Cases/Basic_Test.py
```

If Arcanum runs normally, you should see "`Basic Test: Success.`" in the output.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): The Functionality of Arcanum.

- Arcanum operates on the modern browser architecture, as proven by the implementation patches for Chromium 108.
- Arcanum supports both MV2 and MV3 extensions. This is proven by experiment E1, where we test custom extensions for both MV2 and MV3.
- Arcanum is able to track user sensitive data from within web pages and support taint propagation across a comprehensive set of browser, web, and JavaScript APIs used by extensions. This is proven by both experiments E1 and E2. In E1, we test custom extensions that trigger different types of taint sources (including `chrome.history`, `document.location`, etc.), taint sinks (including `Fetch`, `XMLHttpRequest`, etc.), and various propagation processes (including binary data buffer propagation, Chrome message passing APIs, etc.), as described in Section 3 of our paper. In E2, we test real-world extensions covering more propagation cases, such as the storage taint sink, literal creation for compound JS types, JS Promise, etc.
- Arcanum produces detailed taint propagation logs. This is proven by both experiments E1 and E2, where the test case results are determined by whether the expected content appears in the taint logs.

(C2): Arcanum Can Track Specific Web Page Content.
Arcanum allows researchers to instrument specific web page elements as tainted at runtime via JS DOM annotations. This is proven by both experiments E1 and E2, where we annotate specific sensitive DOM elements on seven target pages and test the extensions with these annotations.

(C3): Our Experiments are Reproducible.
In deploying Arcanum, we are able to discover automated exfiltration of user data by real-world extensions

across seven popular websites. This is proven by experiment E2 described in Section 4.5 and Section 4.10 of our paper. Specifically, we provide all extensions discussed as case studies in Section 4.10 and those whose IDs are listed in Table 7 of Section 4.5. Our test cases evaluate whether Arcanum can successfully flag each sample extension, identifying whether the extension collects specific tainted page content on a target page and propagates the information to a taint sink.

A.4.2 Experiments

(E1): [Test Custom Extensions] [1 human-hour]

Preparation: Use the same Docker container from the Basic Test that has the Arcanum executable file placed in "`/root/Arcanum/`". Download all custom extensions (in "`~/Sample_Extensions/Custom/`") from the artifact GitHub repository and copy the extensions to "`/root/extensions/custom/`" in the container.

```
mkdir -p /root/extensions/custom/
cp -r ~/Sample_Extensions/Custom/* /root
    ↪ /extensions/custom/
```

Download all recordings and JS scripts for DOM element annotations from the artifact GitHub repository and place them in the "`/root/`" directory in the container:

```
mkdir -p /root/recordings/
cp -r ~/recordings/* /root/recordings/
mkdir -p /root/annotations/
cp -r ~/annotations/* /root/annotations/
```

Execution: We have prepared a test case for each custom extension. Run these test cases in the container shell using the pre-configured Python 3.8:

```
python3.8 ~/Test_Cases/Custom_Test.py
```

Each test case launches Arcanum with the corresponding web recording and DOM element annotations (or without annotations when testing non-web content taint sources), and checks whether we successfully obtain the expected content in the taint logs, demonstrating the correct taint tracking of Arcanum. You can test all custom extensions together or test a specific extension by simply commenting out other test cases in `Custom_Test.py`, such as:

```
if __name__ == "__main__":
    Amazon_Extension_MV2_Test()
    # Amazon_Extension_MV3_Test()
    # Facebook_Extension_MV2_Test()
    ...
    # Source_document_password_Test()
    # Source_document_location_Test()
    ...
```

Results: For each extension being tested, you should see "Custom Extension ${Name}: Success." in the test case output, demonstrating the correct taint tracking of Arcanum. You can refer to the test case code to see the expected content in the taint logs for each extension.

(E2): [Test Real-world Extensions] [1 human-hour]

Preparation: Use the same Docker container from the Basic Test that has the Arcanum executable file placed in "`/root/Arcanum/`". Download all real-world extensions (in "`~/Sample_Extensions/Realworld/`") from the artifact GitHub repository and copy the extensions to "`/root/extensions/realworld/`" in the container.

```
mkdir -p /root/extensions/realworld/
cp -r ~/Sample_Extensions/Realworld/*
    ↪ /root/extensions/realworld/
```

Execution: We have prepared a test case for each real-world extension. Run these test cases in the container shell using the pre-configured Python 3.8:

```
python3.8 ~/Test_Cases/Realworld_Test.py
```

Each test case launches Arcanum with the corresponding web recording and DOM element annotations, and checks whether we successfully obtain the expected content in the taint logs, aligning with the experiment results described in Sections 4.5 and 4.10. You can test all real-world extensions together or test a specific extension by simply commenting out other test cases in `Realworld_Test.py`, such as:

```
if __name__ == "__main__":
    amfmnhcipnbjjnbfmaoooiohikifefk()
    # haphbbhhknaonfloinidkcmadhfjoghc()
    # dianbbpnakhcmfkcckaboohfgnngfcc()
    ...
```

Results: For each extension being tested, you should see "Real-world Extension ${ID}: Success." in the test case output. You can refer to the test case code to see the expected content in the taint logs for each extension.
We also release all taint logs (i.e., analysis results generated by Arcanum) for each real-world extension obtained from the experiments conducted in our paper. Please check these logs located in `~/Taint_Logs/` in the artifact GitHub repository.

A.5 Notes on Reusability

- Arcanum's taint source logs, propagation logs, and the storage sink logs are located in "`/ram/analysis/v8logs/`". All other taint sink logs are in the specified user data directory of Chromium.
- When testing Arcanum with Docker, ensure to allocate sufficient CPU resources (4 logical processors or more),

especially when running multiple containers in parallel (e.g., using "`--cpus=4 --cpuset-cpus=0-3`"). Use "`--cpuset-cpus`" to specify CPUs in scenarios where preemption may occur.

- As described in Section 3.4 in the paper, we introduce a forced delay in Arcanum to ensure that a target web page will fully load before an extension injects its content script. We configure this delay as a Chrome switch "`--custom-script-idle-timeout-ms`" and "`--custom-delay-for-animation-ms`". Users can set a specific delay when recording and replaying different web pages according to their page loading times. Please refer to the provided test cases for examples of its usage. The test cases were evaluated on a Linux server with 8 CPUs and 16 GiB of RAM. If you are testing with fewer CPU resources, please consider increasing the value of the two switches mentioned above in the test case scripts.
- Please see the README file in our GitHub repository for future updates.

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/usenixsec2024/.