USENIX Security '26 Artifact Appendix – The State of Passkeys: Studying the Adoption and Security of Passkeys on the Web

Louis Jannett [1], Andreas Mayer [2], Maximilian Westers [2], Vladislav Mladenov [1], Christian Mainka [3], and Jörg Schwenk [1]

[1]Ruhr University Bochum
[2]Heilbronn University of Applied Sciences
[3]University of Wuppertal

A Artifact Appendix

A.1 Abstract

We analyze the landscape and security of passkeys on real-world websites using two tools: (1) PASSKEYS-RADAR aggregates multiple sources to identify passkey-enabled websites (§3). We manually registered passkeys on 208 sites and analyzed supported features and passkey management (§4). (2) PASSKEYS-ATTACKER automates security tests after manual account setup and login triggering (§5). We semi-automatically tested 103 passkey-enabled sites (§6).

Our artifacts comprise three components:

- **PASSKEYS-RADAR** aggregates passkey community directories and Internet Archive data, merges them into a unified list of passkey-enabled sites, and presents the results via a web UI (`./radar`). A separate DETECTOR module independently scans 18M CrUX domains (`./detector`) using a parallel-execution framework (`./detector/taskly`, not part of our contribution); we contribute the scanning tasks (`./detector/tasks`). All generated data since 2021 is available in `./data`.
- **PASSKEYS-ATTACKER** is a web-based tool with a frontend (`./tools/frontend`), backend (`./tools/backend`), and Chrome extension (`./tools/extension`). For ethical reasons and because real-world websites are dynamic and many issues were fixed after responsible disclosure, the evaluation does not attempt to reproduce the reported vulnerabilities. Instead, we implemented an intentionally vulnerable learning platform for artifact evaluation (`./learning`) that covers all vulnerabilities from Table 2 and enables safe, controlled experimentation with PASSKEYS-ATTACKER.
- **EVALUATION RESULTS** are processed and visualized in Jupyter notebooks (`./notebooks`), reproducing the figures and numbers from the paper.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

PASSKEYS-ATTACKER simulates a WebAuthn client and authenticator that deliberately skips security checks, enabling fine-grained manipulation of WebAuthn messages throughout the protocol. Therefore, it must be used only in a dedicated testing browser (profile) and with accounts created exclusively for testing. Do not use it in production or with private passkeys (e.g., for your banking account). Use it responsibly and do not exploit vulnerabilities against accounts owned by others. Although most artifacts run in Docker, we recommend an isolated virtual machine for maximum containment.

A.2.2 How to access

- All artifacts are available on GitHub: https://github.com/RUB-NDS/state-of-passkeys-artifacts.
- Releases are automatically archived on Zenodo; the latest release is available at https://doi.org/10.5281/zenodo.17898769.
- This evaluation uses version 4, archived at https://doi.org/10.5281/zenodo.18418005.
- The actively developed PASSKEYS-ATTACKER is also maintained separately for ongoing and future work: https://github.com/RUB-NDS/passkeys.tools. This is in addition to the version included in the main artifacts repository.
- Project website: https://passkeys.tools.
- Public instances are available for PASSKEYS-RADAR (https://radar.passkeys.tools), PASSKEYS-ATTACKER (https://attacker.passkeys.tools and https://db.passkeys.tools), and the learning platform (https://learning.passkeys.tools).
- The latest Chrome extension is also available via https://attacker.passkeys.tools/extension.zip.

A.2.3 Hardware dependencies

Our artifacts run on any machine capable of running a modern GUI-based OS (e.g., `Ubuntu 24.04 LTS`) with Docker, Chrome, Python 3, and Pipenv installed.

A.2.4 Software dependencies

Our artifacts should run on any modern GUI-based OS (x86 or ARM) that is capable of running Docker, Chrome, Python 3, and Pipenv. We tested and recommend `Ubuntu 24.04 LTS` or `macOS Tahoe 26.2` (on the host or in a VM). Windows 11 is untested but should work as well.

A.2.5 Benchmarks

None.

A.3 Set-up

While newer versions should also work, we refer to specific versions that we have used to verify our artifacts.

A.3.1 Installation

Installation:

- **Install Docker**: Follow the official installation guide at https://docs.docker.com/get-started/get-docker/. Then, run `docker --version` and ensure that version `29.1.5` (or newer) is installed and working. Also run `docker compose version` and verify that version `v5.0.1` (or newer) is installed and functional.
- **Install Chrome**: Download and install Chrome from https://www.google.com/chrome. After installation, open Chrome, navigate to `chrome://version`, and confirm that version `144.0.7559.97` (or newer) is installed.
- **Install Python**: Install Python `3.14.2` via https://www.python.org/downloads/release/python-3142/ (or using a package manager of your choice). Verify the installation by running `python3 --version` and confirming it reports `3.14.2`.
- **Install Pipenv**: Install Pipenv `2026.0.3` by following https://pipenv.pypa.io/en/latest/installation.html. We recommend using your system package manager (e.g., `apt` or `brew`) where available. Verify the installation by running `pipenv --version` and confirming it reports `2026.0.3`.
- **Clone the repository**: Clone the repository using `git clone https://github.com/RUB-NDS/state-of-passkeys-artifacts.git`, or download and unpack the artifact bundle from Zenodo: https://doi.org/10.5281/zenodo.18418005.
- **Install Pipenv dependencies**: Change into the notebook directory with `cd ./notebooks` and run `pipenv install` to install all dependencies.

Configuration:

- **Configure Chrome profiles**: Open Chrome and click the profile icon. Select "Manage Chrome profiles" and create two new, clean browser profiles named "Victim" and "Attacker". This setup emulates two isolated browser instances and keeps the experiment separate from potential existing personal or work profiles.
- **Configure the Chrome extension**: In *both* profiles, open `chrome://extensions`, enable "Developer mode" (top right), click "Load unpacked", and select the `./tools/extension` directory. Then, open the extension popup via the extension icon next to the address bar. Ensure the extension is enabled and that the Frontend URL is set to the default `https://passkeys.tools`. For the operation mode, select `Profile 1` in the victim profile and `Profile 2` in the attacker profile.
- **Configure PASSKEYS-ATTACKER**: In the victim profile, open `https://passkeys.tools/settings` and select "Remote Storage". Verify that the "Remote Server URL" is set to the default `https://db.passkeys.tools`. Choose a secret (or generate one) and enable end-to-end encryption. Click "Test Connection" and confirm that the message "Connection successful!" is displayed. Finally, click "Save Storage". Repeat the same steps in the attacker profile, but make sure to use the *same* secret as in the victim profile.
- **Configure the learning platform**: In the victim profile, open `https://learning.passkeys.tools`, click "Instance", and copy the UUID shown under "Current Instance". Then, in the attacker profile, open `https://learning.passkeys.tools`, click "Instance", paste the victim UUID into "Switch Instance", and click "Switch".

A.3.2 Basic Test

Validate PASSKEYS-ATTACKER: In the victim profile, open `https://learning.passkeys.tools` and create a new account. Then, go to `https://learning.passkeys.tools/settings`, click "Add Passkey", and (optionally) enter a name. Select "Register Passkey"; the PASSKEYS-ATTACKER interceptor should open, display the WebAuthn request, and generate a valid response. Click "Send" and confirm that the new passkey appears in the account settings.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): PASSKEYS-RADAR and the DETECTOR module (§3.1) aggregated 8,523 websites from multiple sources (§4.1). After merging (§3.2), 872 unique sites remained (§4.1). We excluded 664 sites from further analysis (Table 3) because manual account registration was feasible on only 208 sites (§4.2). We then manually analyzed these 208

sites, including their modes and algorithms (Figure 6). While we tracked passkey-enabled sites over time (Figure 5), the manual analysis is based on the snapshot retrieved on April 22nd, 2025 (§4.2). These findings are supported by experiments (E1), (E2), and (E4).

(C2): Due to practical restrictions, we could only fully analyze passkey implementations on 103 websites (§6.1) using PASSKEYS-ATTACKER (§5.2, §5.3), which automates 28 detection methods for 15 attack types (Table 2). For ethical reasons, and because real-world websites are dynamic and many issues were fixed after responsible disclosure, we demonstrate PASSKEYS-ATTACKER only in a controlled lab setting. To this end, we implemented an intentionally vulnerable learning platform for artifact evaluation that covers all vulnerabilities (Table 2). We provide step-by-step guidance to identify and exploit one selected vulnerability, namely *Swap cred*^{id}. These findings are supported by experiments (E3) and (E5).

A.4.2 Experiments

Experiments (E1)–(E3) prove the functionality of our artifacts. (E4) and (E5) focus on the reproducibility of our artifacts.

(E1): Identify passkey-enabled websites with PASSKEYS-RADAR *(1 human-hour + 0.1 compute-hour)*

Preparation: Deploy an empty PASSKEYS-RADAR instance by following `./radar/README.md` ("Production Deployment" and "Docker Run"). Open `http://localhost:8090` and confirm the UI loads (the initial "no data" error is expected).

Execution: In "Admin" → "Operations" → "Select & Fetch", click "Select All Directories" and then "Fetch Selected". Authenticate with "admin" and the password from `./radar/.env`. After all fetches complete, verify in "Task History" that all 12 directories were fetched. Then run the pipeline: "Operations" → "Combine" → "Merge" → "Select Unmerged" → "Merge Selected".

Results: In "Admin" → "Data Browser" → "Merged", open the newest file via "View". It should contain a deduplicated list of passkey-enabled sites plus the directories in which each site appears. In "Explore", select the next day (after the merge run) and click "Load Data" to view the merged list as an interactive table. Note that (E1) covers directory sources only; well-known discovery is handled in (E2). Compare "Overview" → "Domains by Directory" on your local instance (`http://localhost:8090`) with our public instance (`https://radar.passkeys.tools`, last updated August 27th 2025). You should observe slightly higher counts in your run, reflecting websites that have recently adopted passkeys. After artifact evaluation, we will refresh the public instance with an updated dataset.

(E2): Identify well-known documents with the DETECTOR module *(0.5 human-hour + 0–1 compute-hour)*

Preparation: Run `cd ./detector` and follow `./detector/README.md` ("Build" and "Run"). Verify that `cli.py` runs successfully and prints the help output.

Execution: Run the "Scan a single origin" command from `./detector/README.md` to scan `kayak.com` for well-known files. The scan should finish within a few seconds to about a minute. Optionally, use the provided command to scan the top 1k CrUX origins; this may take ≥1 hour depending on your Internet connection.

Results: After completion, `./detector/output/wellknown/<scan_id>/<task_id>/result.json` should be created and contain results for the scanned origin. For `kayak.com`, both documents (`/.well-known/passkey_endpoints` and `/.well-known/webauthn`) should be present and non-empty. We also include our recurring scans of well-known documents for 18M CrUX domains in `./data/wellknown`.

(E3): Identify the *Swap cred*^{id} vulnerability with PASSKEYS-ATTACKER *(1 human-hour + 0 compute-hour)*

Preparation: Set up the victim and attacker profiles as in §A.3.1. On `https://learning.passkeys.tools`, open "Instance" and click "Reset" to start fresh.

Execution: *In victim profile:* Create a victim account → account settings → select "Duplicate Cred ID" security verifier → "Add Passkey" → "Register Passkey". Keep the default "Credential ID" and "Key" in the interceptor ("Actions" → `profile1 | ES256`), then click "Send" to complete registration.

In attacker profile: Create an attacker account → account settings → select "Duplicate Cred ID" security verifier → "Add Passkey" → "Register Passkey". Select `profile1 | ES256` as "Credential ID" in the interceptor, but keep the default `profile2 | ES256` "Key", then click "Send" to complete registration. This binds the attacker key to the victim credential ID, creating a duplicate ID.

Results: *Victim lockout:* Switch to the victim profile and log out. On the login page, select the "Discoverable" demo verifier and click "Login with Passkey". Use `victim` as "User Handle" and `profile1 | ES256` for "Credential ID" and "Key", then click "Send". Verify the error message and that the victim cannot log in.

Recovery: In the attacker profile, delete the attacker passkey from account settings. Repeat the victim login attempt and confirm the victim can log in again.

(E4): Landscape evaluation results *(0.5 human-hour + 0.1 compute-hour)*

Preparation: Run `cd ./notebooks` and execute `pipenv run jupyter notebook`. This should open the Jupyter UI in your browser.

Execution and Results: *Adoption of passkeys (Figure 5)*: Open `radar.ipynb` in the Jupyter UI and click "Restart the kernel and run all cells". The notebook reads `./data/merged` to generate Figure 5 and additional charts not included in the paper.

Table 1: Mapping of `sheet.ipynb` cells to paper sections

Cell	Paper Reference
5	§6.2.7
6	Table 3, §4.2 (“Intro”)
7	§4.2 (“Confirming Passkey Registrations”)
8	§4.2 (“Removing Passkeys”)
10	§4.2 (“Removing Passkeys”)
11	Figure 6a
12	§4.2 (“Passkey Authentication Modes”)
13	Figure 6b
14	§4.2 (“Passkey Algorithms”)
15	§4.2 (“Passkey Scopes”)
18	§4.2 (“User Identifier”)
21	§4.2 (“Key Attestation”)
22	§4.2 (“Authenticator Selection”, “User Verify”)
24	§6.2.*, Table 2
27	§6.2 (“Attack Severity”)
28	Figure 8b
29	Figure 8a

Fetching all sources: Our landscape and security analyses use the snapshot generated on April 22nd, 2025. Verify the aggregated 8,523 websites with: `jq '[.directories[], .wellknown[]] | map(length) | add' 2025-04-22-12-36-53-combined.json`.

Merging all sources: Verify that merging yields 872 unique sites with: `jq 'length' 2025-04-22-12-36-53-merged.json`.

Manual evaluation sheet: We manually visited all 872 sites to attempt account creation and passkey registration. The results are provided as `sheet.xlsx` and `sheet.csv`, containing all domains from `2025-04-22-12-36-53-merged.json` sorted by Tranco rank. Column “Availability” records whether a site was analyzable and, if not, the failure reason. For example, `sheet.xlsx` contains 10 entries labeled “Payment method required”, matching Table 3.

(E5): Security evaluation results *(0.5 human-hour + 0.1 compute-hour)*

Preparation: Same as (E4).

Execution and Results: In the Jupyter UI, open `sheet.ipynb` and click “Restart the kernel and run all cells”. The notebook reads `sheet.csv` and reproduces most numbers and figures used in the paper. Table 1 maps notebook cells to the corresponding parts of the paper.

Example 1: Cell 24 should output “Test Create Context Nonsense: 5”, corresponding to Table 2 (*Context* → *Nonsense* → Reg. (Create)), i.e., 5 vulnerable sites.

Example 2: Cell 15 should output “Upscoped: 80”, corresponding to “80 sites (38%) explicitly scope their passkeys to the top-level domain” (§4.2).

A.5 Notes on Reusability

Both PASSKEYS-RADAR and PASSKEYS-ATTACKER are designed as extensible platforms for ongoing passkey research. PASSKEYS-RADAR can serve as a longitudinal observatory for tracking passkey adoption trends. Researchers can add new community directories, integrate additional data sources, or adapt the DETECTOR module to scan for emerging well-known document formats as the WebAuthn ecosystem evolves. PASSKEYS-ATTACKER provides a modular framework for security analysis: its interceptor architecture allows researchers to implement new attack vectors. A promising direction for future work is fully automating account creation and passkey registration workflows, which would enable large-scale security assessments by integrating PASSKEYS-ATTACKER. Similarly, developers can embed PASSKEYS-ATTACKER into their own unit and integration tests to continuously verify that their WebAuthn implementations correctly enforce security checks. Contributions to both tools are welcome via GitHub.

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/usenixsec2026/.