

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

AutoFail: Breaking Web Boundaries using Android's Autofill Framework

Riccardo Lamarca, Philipp Beer, and Marco Squarcina, *TU Wien*

https://www.usenix.org/conference/usenixsecurity26/presentation/lamarca

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium is sponsored by



جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology





USENIX Security '26 Artifact Appendix: AutoFail: Breaking Web Boundaries using Android's Autofill Framework

Riccardo Lamarca
TU Wien

Philipp Beer
TU Wien

Marco Squarcina
TU Wien

A Artifact Appendix

A.1 Abstract

Our artifact includes: ADAPT, an automated differential testing framework for detecting autofill vulnerabilities in browsers and password managers; proof-of-concept applications demonstrating the Cross-Context Account Oracle attack and its mitigation; scripts and datasets for the real-world analysis of embeddability headers and iframe configurations.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The web-crawler is set to spawn the browser without the sandbox enabled, which can expose the machine of the evaluator to security risks. We suggest running the crawler in a virtual machine or a disposable environment.

The proof-of-concept app of the Cross-Context Account Oracle attack is solely intended to demonstrate the attack and it does not collect any sensitive information.

A.2.2 How to access

The artifacts are hosted at https://github.com/SecPriv/autofail and archived at https://doi.org/10.5281/zenodo.20441978.

A.2.3 Hardware dependencies

System requirements: We recommend running ADAPT and the analysis tools on a machine with at least 16GB RAM. The APKs of the browsers and password managers and the Frida server require a device with arm64 architecture. We recommend either a physical Pixel 6a or an emulator. We provide the code to set up and run the emulator on macOS in the `emulator` directory.

A.2.4 Software dependencies

Host machine: We require a host machine running Linux or macOS. Note that macOS is required to use the code that we provide to set up the emulator.

Android image: We require Android 16 (API level 36) with root access (`adb` must be able to run as root) .

Google Cloud account: To execute E4, a Google Cloud account with access to BigQuery is required.

A.2.5 Benchmarks

Tranco Top 1M list: The crawler uses a pre-processed version of the Tranco top 1M websites list, included in the repository at `RealWorldAnalysis/IframeConfigurationAnalysis/crawler/data/`.

BigQuery HTTP Archive dataset: The embeddability analysis queries the public HTTP Archive dataset available on Google BigQuery. We provide the result of the query in `RealWorldAnalysis/HeaderConfigurationsAnalysis/bq_result_splitted/`

Pre-crawled iframe data: We provide pre-crawled iframe configuration data for 3,522 websites in `RealWorldAnalysis/IframeConfigurationAnalysis/crawler/data/bucketed_1M_out/` to enable immediate verification of results (E7).

A.3 Set-up

A.3.1 Installation

Clone the repository by running: `git clone https://github.com/SecPriv/autofail`.

Host machine dependencies: Run the `setup.sh` script to install the dependencies on the host machine.

Emulator: Run the `emulator/launch_emulator.sh` script to install and run the emulator. As an alternative, you can use a physical Android device.

Android device one time set-up: Run `emulator/install_apks.sh` to install the required APKs. Follow the one-time set-up in `ADAPT/README.md` to install the root CA certificate and configure domain resolution for `a.com` and `b.com`.

Run the server: Run `./ADAPT/run_server.sh` to perform the network set-up of the device, run Frida and start the server.

A.3.2 Basic Test

Run the `basic_test.sh` script that verifies that the host machine sees the Android device, that the device can reach the server and that the Frida server process is running.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): The provided test suite (Table 2) reproduces the browser and password manager vulnerabilities summarized in Table 3.

(C2): Our proof-of-concept exploits demonstrate that the applications listed in Table 6 are vulnerable to the Cross-Context Account Oracle.

(C3): The proof-of-concept application implementing the Cross-Context Account Oracle mitigation (Section 7.2) is functional and effective.

(C4): The raw dataset of requests used in Section 8.1 (**Embeddability Analysis**) can be successfully retrieved.

(C5): Executing the analysis scripts on the Section 8.1 dataset reproduces the finding that 485,470 sites are embeddable, as well as the top 5 combinations of XFO and CSP headers reported in Table 7.

(C6): The web-crawler described in Section 8.2 (**Iframe Analysis**) can be executed to successfully collect the required iframe embedding data.

(C7): Running the provided evaluation scripts on the crawled iframe dataset reproduces the statistical findings detailed in Section 8.2 (**Iframe Analysis**), Table 8, and Table 9.

A.4.2 Experiments

(E1): [Test suite] [2 human-hours + 10 compute-minutes + 16GB disk]

Preparation: Navigate to the `ADAPT` directory. Connect the Android device. Install the specific browser and password manager versions detailed in Table 1. Following the **One time setup** instructions in `README.md`, install the root certificate authority required to sign the test suite's TLS certificates. For each password manager, manually create test credentials for both the `a.com` and `b.com` domains. Prior to testing a specific password manager, configure it as the device's preferred autofill service (e.g., via `Settings → Passwords, passkeys & accounts → Preferred service`). For Chrome and Brave, explicitly enable third-party external autofill within the respective browser settings.

Execution: Execute the `run_server.sh` script to initialize the test server. Proceed through the test cases by interacting with the server as outlined in the **Running the server** section of `README.md`.

Results: Observe the autofill behavior during the execution phase and verify that the results successfully reproduce the issues reported in Table 3 and detailed in Section 5.

(E2): [Cross-Context Account Oracle Attack] [10 human-minutes + 1 compute-minute + 16GB disk]

Preparation: Navigate to the `CrossContextAccountOracle` directory. Connect the Android test device (e.g., via ADB) and install the password managers specified in Table 6. For each password manager, manually create test credentials for the `a.com` domain. Prior to testing a specific password manager, configure it as the device's active autofill service (e.g., via `Settings → Passwords & passkeys & accounts → Preferred service`). Before executing the attack, open the password manager's main application interface and leave it running in the background.

Execution: Execute the `deploy-and-run.sh` script to compile, deploy, and launch the proof-of-concept attack application on the connected device.

Results: Observe the output displayed in the attack application's activity. Verify that the oracle successfully triggers and logs a result for the `a.com` domain, and confirm that no result is logged for the `b.com` domain.

(E3): [Cross-Context Account Oracle Mitigation] [10 human-minutes + 1 compute-minute + 16GB disk]

Preparation: Navigate to the `mitigation` directory. Connect the Android test device and execute the `install-apps.sh` script to install the required applications. Navigate to the device settings (e.g., `Settings → Passwords, passkeys & accounts → Preferred service`) and select `Secure Autofill PoC` as the active autofill service.

Execution: Execute the `run-poc.sh` script to launch the test environment. On the connected device, interact with the presented login form to trigger the autofill suggestion, and navigate the resulting security warning prompt.

Results: Observe the application's autofill behavior. Verify that the credentials are intentionally withheld initially, and are only listed as available *after* the user explicitly acknowledges the warning prompt.

(E4): [Headers dataset] [10 human-minutes + 10 compute-minutes + 1GB disk]

Preparation: Navigate to the `RealWorldAnalysis/HeaderConfigurationsAnalysis` directory. Ensure you have an active Google Cloud account with access to BigQuery.

Execution: Execute the query defined in `query.sql` via the BigQuery console (or CLI) and download the resulting dataset locally (e.g., as `results.json`). Next, execute the comparison script by passing the downloaded results file as an argument (e.g., `./compare_results.sh results.json`).

Results: Verify that the script outputs `SUCCESS`.

(E5): [Headers analysis] [1 human-minute + 1 compute-minute + 1GB disk]

Preparation: Navigate to the `RealWorldAnalysis/HeaderConfigurationsAnalysis` directory.
Execution: Execute the analysis script by running `verify_analysis.sh`.
Results: Verify that the script outputs `TRUE`.

(E6): [Crawler functional] [10 human-minutes + 20 compute-minutes + 1GB disk]
Preparation: Navigate to the `RealWorldAnalysis/IframeConfigurationAnalysis` directory.
Execution: Execute the crawler on a provided sample of websites by running `run_crawler_sample.sh`.
Results: Verify that the crawler completes its execution and confirm that the `evaluator_sample_output` directory is successfully generated and populated with the resulting data.

(E7): [Iframe Analysis] [10 human-minutes + 10 compute-minutes + 1GB disk]
Preparation: Navigate to the `RealWorldAnalysis/IframeConfigurationAnalysis` directory.
Execution: Execute the analysis script by running `verify_analysis.sh`.
Results: Verify that the script outputs `SUCCESS`.

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/usenixsec2026/.