USENIX Security '26 Artifact Appendix: "Tap" Without Tapping: A Tag Discovery Forgery Attack on Android NFC

Yilin Li
School of Cyber Science and Technology, Shandong University

Jianliang Wu(✉)
Simon Fraser University

Chaoshun Zuo
Independent Researcher

Qingchuan Zhao
City University of Hong Kong

Xiaofeng Liu(✉) Xiangpu Song Chengyu Hu Shanqing Guo(✉)
School of Cyber Science and Technology, Shandong University

A Artifact Appendix

A.1 Abstract

This artifact accompanies our USENIX Security '26 paper *"Tap" Without Tapping: A Tag Discovery Forgery Attack on Android NFC*. The artifact has been awarded the *Artifacts Available*, *Artifacts Functional*, and *Results Reproduced* badges. The reproduced-results scope is limited to the released static-analysis pipeline and aggregate-measurement results.

The package provides a reviewer-runnable measurement and reproduction workflow. The main path uses shipped intermediate records to reproduce the paper-level aggregate numbers and derived CSV/Markdown outputs without re-parsing the full 76,333-app corpus. The artifact also includes controlled toy APKs and qualitative case-study evidence for functionality inspection. Real-world attacker source code, app-specific payloads, vendor-specific trigger recipes, raw APKs, and AndroZoo credentials are not included.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The main evaluation path is offline and non-destructive. It runs local scripts over shipped intermediate data and derived result files. It does not contact third-party services, target real-world applications, modify system settings, or require an Android device.

The package includes controlled demonstration APKs only. These APKs demonstrate the dispatch-forgery pattern in a toy setting and do not target third-party applications or live services. The artifact withholds real-world attacker implementations, app-specific payloads, vendor-specific exploit recipes, and reusable end-to-end attack instructions. The full boundary is documented in `WITHHELD_MATERIALS.md`.

Optional Android-side demonstrations should be run only in an isolated emulator or non-personal test device. These optional paths are not required for the main functionality and reproduction workflows covered by the awarded badges.

A.2.2 How to access

The final artifact is archived on Zenodo under the following permanent concept DOI:

`https://doi.org/10.5281/zenodo.20303896`

This concept DOI is the permanent stable reference for the artifact and resolves to the latest archived version. The artifact remains under restricted access because it contains sensitive per-app measurement and validation materials while platform-side remediation and coordinated disclosure remain relevant. Access requests can be submitted through the Zenodo record. The package includes a `README.md` with installation, evaluation, and reproduction instructions.

A.2.3 Hardware dependencies

None for the main evaluation path. The artifact runs on a standard CPU-based host. It does not require GPUs, TEEs, FPGAs, embedded boards, high-memory servers, Android devices, raw APK downloads, AndroZoo API keys, or access to third-party services.

An Android device or emulator is needed only for optional device-side demonstrations or dynamic re-collection. These optional paths are outside the byte-for-byte reproduction claim.

A.2.4 Software dependencies

The default path requires a POSIX-like environment. The author-tested and officially supported range is Python 3.9–3.12. Newer Python versions may also work, but are not part of the author-tested support range. The script `setup.sh` installs the required Python packages, including `pandas`, `loguru`, `tqdm`, `requests`, and `tabulate`. The `adb` tool is needed only for optional Android-side demonstrations.

Windows users should use WSL2 or an equivalent POSIX-compatible environment.

A.2.5 Benchmarks

The dataset is a custom AndroZoo-derived corpus. The artifact ships the SHA list and the precomputed intermediate records needed for reproduction. Raw APKs and AndroZoo credentials are not redistributed.

A.3 Set-up

A.3.1 Installation

From the artifact root, run:

```
$ ./setup.sh
```

This installs the Python dependencies needed for the reviewer-runnable path.

A.3.2 Basic Test

After installation, run:

```
$ ./check_artifact.sh
$ ./run_smoke_test.sh
```

The first script checks required files and package integrity. The second script runs a short sanity test over the packaged data and derived outputs. A successful run should finish without errors.

For full aggregate reproduction, run:

```
$ ./run_reproduce.sh
```

A successful execution ends with:

```
RESULT-A: PAPER-CLAIM CHECK PASS
RESULT-B: BYTE-FOR-BYTE DIFF PASS
```

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Dataset construction. The paper's analyzable set contains **76,333** apps, derived from 76,345 candidate SHA entries and 76,334 manifest-feature records. This is checked in experiment E1.

(C2): Reader/Writer NFC apps and handlers. The analysis identifies **928** apps with Reader/Writer NFC handlers and **1,089** Reader/Writer handler components. This is checked in E1.

(C3): Potentially vulnerable apps and handlers. The analysis identifies **601** apps with at least one potentially vulnerable handler and **703** unique potentially vulnerable handler components. This is checked in E1.

(C4): Dynamic-validation candidates. The dynamic-validation input contains **708** attempt-level rows. This is checked in E1.

(C5): Dynamic triggerability. The 708 attempted invocations yield 618 successful installs and **550** `INVOKE_SUCCESS` outcomes. This is checked in E1.

(C6): Controlled dispatch-forgery demonstration. The controlled example APKs demonstrate the forged NFC dispatch pattern in a toy setting. This is inspected in E3 and is not part of the byte-for-byte aggregate reproduction claim.

A.4.2 Experiments

(E1): Static-pipeline and aggregate reproduction [5 human-minutes + 1–3 compute-minutes + 1GB disk].
This is the main reproduction experiment corresponding to the *Results Reproduced* badge. It supports C1–C5.
Preparation: Run `./setup.sh` if needed, then run `./check_artifact.sh`.
Execution: Run:
`$ ./run_reproduce.sh`
Results: The script regenerates a fresh reproduction output tree from the shipped intermediates, checks the headline paper claims, and byte-compares regenerated CSV/Markdown outputs against the shipped copies. The expected final output is:
`RESULT-A: PAPER-CLAIM CHECK PASS`
`RESULT-B: BYTE-FOR-BYTE DIFF PASS`
The reproduced dynamic-success total is 550.

(E2): Smoke test [2 human-minutes + under 1 compute-minute + under 1GB disk].
This experiment demonstrates the functionality corresponding to the *Artifacts Functional* badge.
Preparation: Run `./setup.sh` if needed.
Execution: Run:
`$ ./run_smoke_test.sh`
Results: The expected result is a successful sanity test over the packaged data and derived outputs.

(E3): Controlled dispatch-forgery demonstration [10–20 human-minutes + optional device/emulator time].
This experiment supports C6. It is optional and is not part of the *Results Reproduced* claim.
Preparation: Read `poc/README.md`. Use an isolated emulator or non-personal test device if running the Android-side demonstration.
Execution: Follow `poc/README.md`. The helper entry point is:
`$ cd poc`
`$ ./fire_attacker.sh`
Results: The expected result is a controlled toy demonstration of forged NFC dispatch between the included example APKs. It does not target third-party applications or live services.

(E4): Optional APK-level static demo [5–10 human-minutes + input-dependent compute time].
This experiment helps reviewers inspect the static

pipeline on APK inputs. It is not required for paper-level aggregate reproduction.

Preparation: Use the default controlled APK for the single-APK path, or place APKs in the batch input directory.

Execution: Run:

```
$ ./run_single_apk_demo.sh
$ ./run_batch_apk_demo.sh
```

Results: The scripts show how the static pipeline emits feature, reachability, exposure, and validation-harness records. The paper-level aggregate reproduction uses the shipped intermediates rather than re-downloading or re-parsing the full corpus.

A.5 Notes on Reusability

The static side of the artifact can be reused with compatible feature and reachability inputs. The dynamic-aggregation side can consume CSV files with the same schema as the released validation-command or dynamic-result records. This lets future evaluators inspect the same pipeline structure on additional APK sets while keeping the released package safe and self-contained.

The artifact intentionally separates reusable measurement code from sensitive operational details. This supports independent inspection of the paper's static-analysis and aggregate-measurement claims without releasing real-world target-specific exploit materials during ongoing remediation.

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/.