USENIX Security ’25 Artifact Appendix: Ariadne: Navigating through the Labyrinth of Data-Driven Customization Inconsistencies in Android

Parjanya Vyas
University of Waterloo
parjanya.vyas@uwaterloo.ca

Haseeb Ur Rehman Faheem
University of Waterloo
hfaheem@uwaterloo.ca

Yousra Aafer
University of Waterloo
yousra.aafer@uwaterloo.ca

N. Asokan
University of Waterloo
asokan@acm.org

A Artifact Appendix

A.1 Abstract

Vendor customization of the Android framework is known to introduce security concerns. One type of customization is data-driven, involving changes to access-controlled framework variables, which we call data holders. Analyzing the security of data-driven customization has not been explored in prior work because it faces several challenges as it requires modeling implicit access control (AC) relations among Java objects and their corresponding operation semantics. Existing Android AC inconsistency detection approaches struggle to discover data-driven AC inconsistencies.

We propose a novel approach, Ariadne, to address these challenges by (1) constructing an abstract representation, the AC dependency graph, to model AC relationships among framework data holders, and (2) using it to detect missing AC enforcement in data holders and their corresponding APIs. Using two AOSP and 11 custom ROMs, we show that Ariadne detects 30 unique data-driven AC inconsistencies which cannot be detected by existing approaches. Therefore Ariadne can offer more comprehensive protection by effectively complementing existing AC inconsistency detection approaches.

This artifact provides a proof-of-concept Java-based implementation of Ariadne. It takes a decompiled Android ROM as input and constructs an AC dependency graph. It then infers the expected AC enforcement for each API using a customized flooding algorithm. Each inference is assigned a *relevance weight* that reflects the tool’s confidence in the inference. Finally, it outputs a list of APIs where the inferred AC is stronger than the existing enforcement, reporting only those exceeding a configurable relevance cutoff.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

No risk present for the evaluators, since the tool simply performs static analysis on the included dataset of ROMs.

A.2.2 How to access

The latest version of the artifact can be accessed at https://doi.org/10.5281/zenodo.15612788.

A.2.3 Hardware dependencies

None.

A.2.4 Software dependencies

The only dependency our tool has is Java 17.

A.2.5 Benchmarks

We provide all the decompiled ROMs used for evaluation in a directory called ‘dataset’.

A.2.6 Interpreting the output

Each analysis of the individual ROMs performed by Ariadne creates a file called ‘apis.csv’ in the ‘Output’ directory. This file contains two columns:

- **API:** The name of the API.
- **Recommended Access Controls:** A list of AC checks recommended by Ariadne for the API.

Each access control recommendation in the second column includes the following fields:

fromApi The source API from which the access control requirement is originated.

acType The type of access control, e.g., `Permission`, `UID`, `AppOps`, etc.

level The normalized sensitivity level, such as `NONE`, `NORMAL`, `DANGEROUS`, `SYS_OR_SIG`.

operator The conditional operator used in the check (e.g., `EQ`, `GT`, `LT`).

values The hardcoded value compared against in the check (e.g., permission string, UID).

access type The type of access guarded, such as `GET`, `SET`, `MODIFY`, etc.

weight The relevance confidence score assigned to the recommendation.

Example: Consider the CSV entry representing `getExtensions` API with a recommended access control shown in Row 1 of Table 1. This entry indicates that `getExtensions` should be guarded by a UID-based system-level access control, inferred from the related `hasExtension` API, with a high confidence score of 0.85. The absence of this control is flagged as a data-driven inconsistency by Ariadne.

A.2.7 System used for experiments

All experiments were conducted on a MacBook Pro equipped with an Apple M3 Pro chip, 18 GB of RAM, and a 500 GB SSD, running macOS Sequoia.

A.3 Set-up

A.3.1 Installation

We provide the permanent artifact at the Zenodo repository mentioned above. For convenience, we also publish a docker image with all dependencies and setup.
To use the docker image:

1. Ensure docker is installed and available to use on your machine (https://docs.docker.com/get-docker/)

2. Run the following command to pull and run the docker image for Ariadne:
```
docker run -rm -pull=always -it
parjanyavyas/ariadne:latest
```

To build and run Ariadne setup from the Zenodo repository:

1. Install Java 17 and ensure it is available to use.

2. Download the artifact ZIP file and extract it.

3. Create two new sub-folders in the extracted folder called ‘Input’ and ‘Output’.

A.3.2 Basic Test

Use the following command to run a basic test:
```
./gradlew startAnalysis
-Dpath="./Sample" -DoutPath="./Output"
-Dorg.gradle.java.home="<JAVA_HOME>"
-Duser.dir="."
```
After successful completion, verify that the “Output” directory contains the file ‘apis.csv’ with the content from Table 1.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Ariadne reports accuracy of 96.6% and 97.5% using AOSP Pixel 5 (v12) and AOSP Pixel 6 Pro (v13) respectively. Refer to Section 7.3 whose results are reported in column 6 of Table 6.

(C2): Ariadne has been used to uncover new inconsistencies in the included custom Android ROMs. Refer to Section 7.4 whose results are reported in column 5, Row 3-13 of Table 6.

(C3): Ariadne identifies APIs with new AC inconsistencies, and suggests appropriate AC checks, in the included custom Android ROMs.

A.4.2 Experiments

(E1): [30 human-minutes + 24-240 compute-hours (depending on underlying system used for the experiment) + 10GB disk]: Analyze AOSP ROMs to evaluate the accuracy of Ariadne by counting total number of APIs and inconsistencies in the two AOSP ROMs. For each of the two AOSP ROMs included in the dataset, perform the following:
Preparation: Create ‘Input’ and ‘Output’ folders if they do not already exist. Copy the decompiled AOSP ROM from dataset to ‘Input’ folder. Clear the ‘Output’ folder (if not already cleared).
Execution: Run the command as described in the README:
```
./gradlew startAnalysis
-Dpath="./Input" -DoutPath="./Output"
-Dorg.gradle.java.home="<JAVA_HOME>"
-Duser.dir="."
```
Results: The execution should produce an api csv file for the given ROM. This file can be interpreted as specified in Section A.2.6.

API Name	Access Control Annotation
android. service. securespaces. SecureSpacesService. getExtensions() Ljava/ util/ List;	ACAnnotation {acWithSrc = AccessControlSource{ fromApi = ’android. service. securespaces. SecureSpacesService. hasExtension(Ljava/ lang/ String;)Z’, ac = ProgrammaticAccessControl{ acType=Uid, level=SYS_OR_SIG, operator=GT, values=[10000]}}, accessType=GET, weight=0.8573749999999999}
android. service. securespaces. SecureSpacesService. getUserRestrictions() Ljava/ util/ List;	ACAnnotation {acWithSrc = AccessControlSource{ fromApi = ’android. service. securespaces. SecureSpacesService. getDeviceOwnerUserRestrictions() Ljava/ util/ List;’, ac = ProgrammaticAccessControl{acType=Uid, level=SYS_OR_SIG, operator=GT, values=[10000]}}, accessType=GET, weight=0.95}

Table 1: Access Control Annotations for SecureSpaces APIs

(E2): [2 human-hours + 120-1200 compute-hours (depending on underlying system used for the experiment) + 50GB disk]: Analyze custom ROMs to identify the inconsistencies detected by Ariadne. For each of the custom ROMs included in the dataset, perform the following:

Preparation: Create ‘Input’ and ‘Output’ folders if they do not already exist. Copy the decompiled custom ROM from dataset to ‘Input’ folder. Clear the ‘Output’ folder (if not already cleared).

Execution: Run the command as described in the README:

```
 ./gradlew startAnalysis
-Dpath="./Input" -DoutPath="./Output"
-Dorg.gradle.java.home="<JAVA_HOME>"
-Duser.dir="."
```

Results: The execution should produce an apis.csv file for the given ROM. This file can be interpreted as specified in Section A.2.6. Running the above command to analyze the Xiaomi Mix 2S ROM included in the dataset results in Ariadne identifying the two inconsistent APIs shown in Table 1, among other new APIs. This result confirms that Ariadne can identify these new access control inconsistencies.

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/usenixsec2025/.