USENIX Security '25 Artifact Appendix: Easy As Child's Play: An Empirical Study on Age Verification of Adult-Oriented Android Apps

Yifan Yao, Shawn McCollum, Zhibo Sun, Yue Zhang
Drexel University

A Artifact Appendix

A.1 Abstract

GUARD (Guarding Underage Access Restriction Detection) is an automatic tool that analyzes the existence of age verification mechanisms by determining relevant components (e.g., those that can accept the user's age or date of birth) based on the relationships of the components in a layout and tracking the data flows through taint analysis. GUARD contains both a static and a dynamic analyzer. Both analyzers are written in Java; therefore, they are compatible with all operating systems. The static analyzer is based on Soot, while the dynamic analyzer is based on Appium.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

To ensure the compliance of the community standards, it is imperative to strictly adhere to the following guidelines. The guidelines are as follows:

- **Responsible Collection of Data**: A significant number of Android Packages may be required for conducting large-scale research. Therefore, it is essential to ensure that this collection process does not interfere with the website owner's operations. Improperly managed web scraping can disrupt the website's normal functionality. We recommend utilizing AndroZoo; however, we also advise collecting data from additional sources to enhance diversity.
- **Analysis in Controlled Environment**: All analyses must be conducted within a controlled setting, using personal account and machines.

A.2.2 How to access

- Project source code is hosted on Zenodo: https://doi.org/10.5281/zenodo.14676162

A.2.3 Hardware dependencies

- Since GUARD unpacks Android packages into RAM for analysis, it is strongly recommended to run on machines with a minimum of 32GB of RAM.
- To execute the dynamic analyzer, it is required to have an Android emulator or a physical Android device. The emulator can be installed via Android Studio or the Android SDK Command-Line Tools.
- Although GUARD is compatible with all operating systems, it is strongly recommended to run the dynamic analyzer on an emulator (Mac computers with Apple Silicon is recommanded) or phone with an ARM64 architecture.
- At least 100GB of free disk space.

A.2.4 Software dependencies

Although this project is developed in Java, ensuring OS independence, the provided instructions have been written and tested specifically on macOS Sequoia (15.0+).

- Java Development Kit (JDK) 21: https://www.oracle.com/java/technologies/downloads/#java21
- Gradle: https://gradle.org/install/
- Android SDK Command-Line Tools (not necessary to have full Android Studio): https://developer.android.com/studio#command-tools
- Android Build Tools, Android Platform Tools, Android API 33 (Android 13) SDK Platform: `$ sdkmanager "build-tools;33.0.2" "platform-tools" "platforms;android-33"`
- Android System Images (Android 13): `$ sdkmanager "system-images;android-33;google_apis;arm64-v8a"`
- Node.js: https://nodejs.org/en/download/
- Appium: `$ npm i -location=global appium`
- Appium Driver (UiAutomator2): `$ appium driver install uiautomator2`
- (Optional) Docker for build and run container for static analysis.

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

```
# clone the repository
git clone https://github.com/Drexel-
    SePAL/AgeScope.git

cd AgeScope

# download sample apks
chmod +x download.sh
./download.sh

# build
gradle build
```

A.3.2 Basic Test

Static Analysier via Docker (results will be placed in `sample/result` folder):

```
# build container image
docker build -t agescope .

# run the container
docker run -it \
    --tmpfs /mnt/ramdisk:rw,size=8g \
    -v $PWD/sample/:/sample agescope

# check results
cat sample/result/*.txt
```

Dynamic Analysier with Android Emulator:

```
# create emulator
avdmanager create avd \
    -n "Pixel_6_0" \
    -d "pixel_6" \
    -k "system-images;android-33;
        google_apis;arm64-v8a"

# start emulator (new terminal window)
emulator -avd Pixel_6_0 \
    -partition-size 8192 \
    -wipe-data

# start Appium server (new terminal
    window)
appium

# run the dynamic analyzer
java -cp ./build/libs/AgeScope-1.0-
    SNAPSHOT.jar DynamicAnalyzer.Main \
    -i sample/apk_index.txt \
    -v 13 \
    -o sample/result \
    -u emulator-5554 # adb devices -l
```

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): GUARD utilizes a combination of static and dynamic analysis to detect the existence of age verification mechanisms in Android applications.

(C2): Static Analyzer unpacks Android packages and: 1) Extracts the app's XML layout structure to analyze age verification keywords; 2) Tracks input fields that may be used to detect the user's age or identity.

(C3): Dynamic Analyzer loads and processes app layouts in real time by running the app on an emulator or physical device. Once launched, it attempts to interact with the app extensively to trigger and identify the age verification mechanism.

A.4.2 Experiments

General Setup for Experiments:

- **Collect Android Packages**: Please download the Android APK samples we provide by executing `download.sh`. Or you can download the APK samples from the internet. When collecting packages from the internet, besides app identifier and content rating, ensure you gather metadata as much as possible for futher analysis, include but not limited to version number, developer information, and description.

- **Generate Index File for Android Packages**: The index file is a text file that includes the path of apk files.

(E1): *[Static Analysis] [30 human-minutes + 5 compute-minutes + 16GB memory]:*
Preparation: Please refer to **General Setup for Experiments**.
Execution: The static analyzer can be executed via Docker (see **Static Analyzer via Docker**) or natively on macOS by running the `macos_run.sh` script. The script automatically sets up the environment and runs the static analyzer. For more details, refer to `README.md`.
Results: The results of the static analysis are located in the `sample/result` folder, formatted as `<index_file>_result.txt`. The static analyzer outputs the analysis results, including the app's package name and the types of age verification mechanisms found in both layout XML files and activities in details.

(E2): *[Static Analysis] [60 human-minutes + 15 compute-minutes + 16GB memory]:*
Preparation: Please refer to **General Setup for Experiments** and **Dynamic Analysier with Android Emulator**.
Execution: The dynamic analyzer can only run natively and requires an Android emulator or a physical Android device. Before running the analyzer, ensure the emulator or device is connected to the computer, recognized by Android Debug Bridge (adb), and that the Appium server is running. The analyzer will then automatically install the app and perform the analysis.
Results: The static analysis results are stored in the `sample/result` folder, formatted as `<index_file>_exec_result.txt`. The dynamic analyzer outputs the analysis results, including the app's package name and the corresponding string found during execution, which is used to identify age verification mechanisms.

A.5 Notes on Reusability

Due to the varying implementations of age verification across regions, this project can be tailored to identify and adapt to different age verification mechanisms based on their distinct features.

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/usenixsec2025/.