ELECTRONIC FRONTIER FOUNDATION

SUNSECCON

EFF Threat Lab's APK downloader

Bill Budington — *he/they*
Senior Staff Technologist
Email: bill@eff.org

Who are we?

- 501(c)3 nonprofit based in the SF Bay Area
- Technologists, Activists, and Lawyers fighting against the expansion of surveillance technologies and for your privacy rights.
- Conduct outreach, advise lawmakers, fight bad legislation and develop technologies to that end.

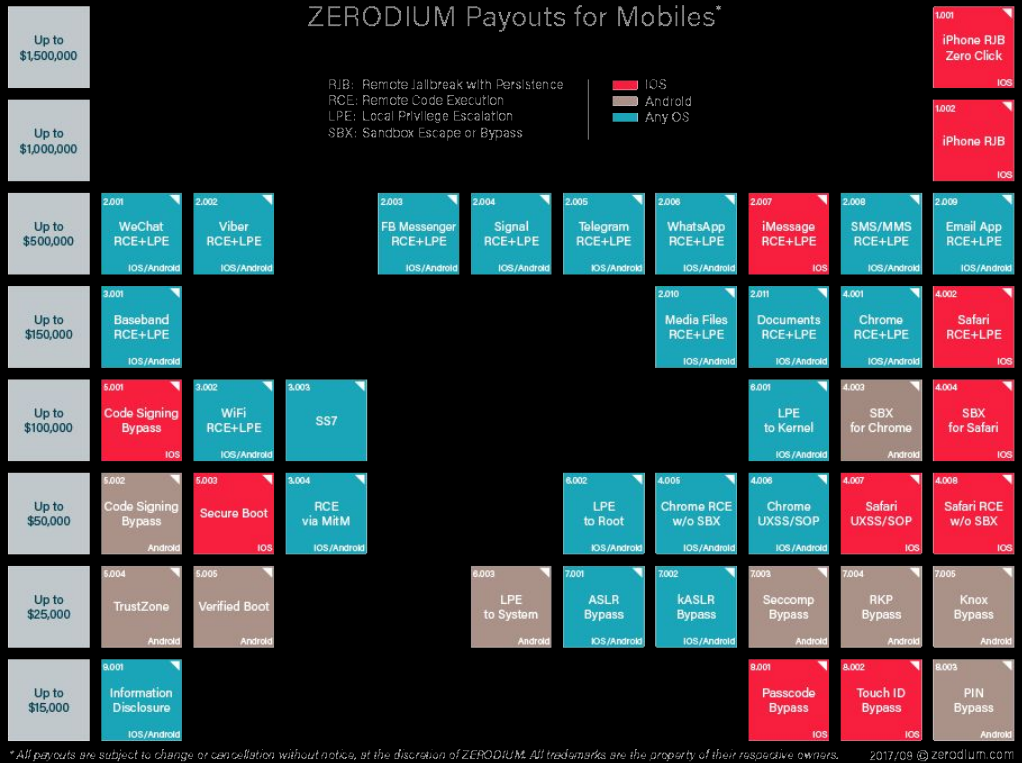
https://www.eff.org/
https://ssd.eff.org/ - Surveillance Self Defense
https://www.eff.org/donate/ - Help us help all!

ELECTRONIC FRONTIER FOUNDATION

Who am I?

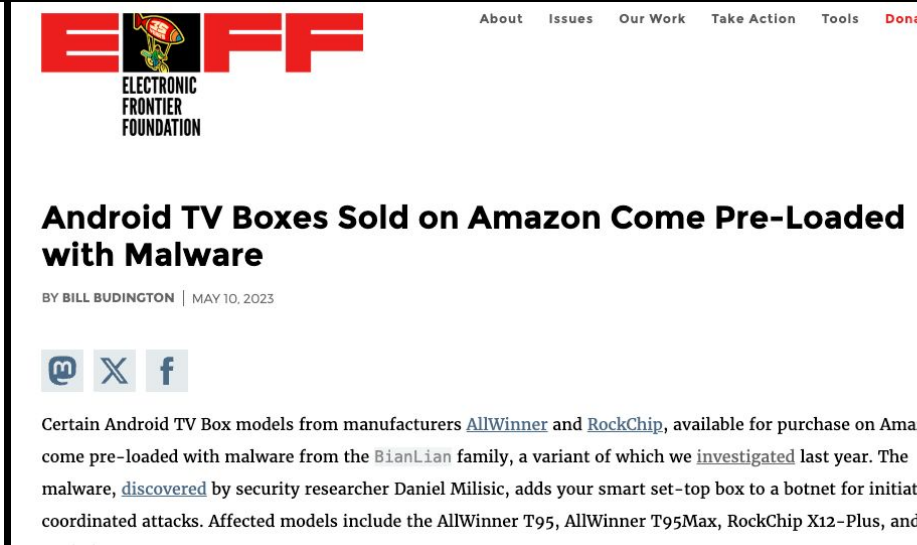
- Senior Staff Technologist
  - Public Interest Technology (Team)
  - Threat Lab & Cybersecurity Policy (Working Groups)
- Previous work:
  - HTTPS Everywhere Lead (2015-2018)
- Current:
  - Panopticlick: Browser Fingerprinting Research (ongoing)
  - Malware Reversing, focus on Android

Android Malware



Android Malware

- Initial infection usually comes from Android packages: apks.
  - May later infect e.g. `zygote` process to propagate.



Background:

- You download an app on the Google Play Store
  - Stored as a file with the extension `.apk`
    - This file is an android package, which runs on:
      - Dalvik VM (legacy)
      - Android Runtime (ART)
    - Many apps are split into multiple `.apk` files, for localization *and*
    - May contain code for running natively on specific android CPU architectures:

x86_64
arm64-v8a
x86
armeabi-v7a

Uses of obtaining a local `apk`

1. Analysis of an app
    a. Static:
        i. Auditing permissions
        ii. Uploading to analysis tools *(e.g. MobSF)*
        iii. Reversing *(tools e.g. jadx, apktool)*
    b. Dynamic:
        i. Installing exact same copy in multiple places
        ii. Upload to a cloud VM (e.g. corellium) for analysis
2. Comparison of an app variants
    a. How does the version on Google Play differ from other app stores?
    b. How does this app change its [code, behavior] when upgrading versions?

Uses of obtaining a local `apk`

```
[bill@Williams-MacBook-Air ~ % apkeep -l -a com.instagram.android                ]
Versions available for com.instagram.android on APKPure:
| 278.0.0.22.117, 360.0.0.52.192, 361.0.0.0.84, 361.0.0.46.88, 362.0.0.0.162, 36
2.0.0.33.241, 363.0.0.29.80, 364.0.0.18.86, 364.0.0.35.86, 365.0.0.40.94, 366.0.
0.34.86, 367.0.0.43.89, 368.0.0.45.96, 369.0.0.46.101, 370.0.0.42.96, 370.1.0.43
.96, 371.0.0.36.89, 372.0.0.0.0, 372.0.0.48.60, 373.0.0.46.67, 374.0.0.43.67, 37
5.0.0.38.66, 376.0.0.51.68, 376.0.0.52.68, 376.1.0.55.68, 377.0.0.40.63, 377.1.0
.48.63, 378.0.0.52.68, 379.0.0.41.80, 379.1.0.43.80, 380.0.0.49.84, 381.0.0.48.8
4, 381.2.0.53.84, 382.0.0.49.84, 383.1.0.48.78, 384.0.0.46.83, 385.0.0.47.74, 38
6.0.0.46.84, 387.0.0.33.85, 387.1.0.34.85, 388.0.0.31.75, 388.0.0.34.75, 389.0.0
.49.87, 390.0.0.43.81, 391.0.0.42.82, 392.0.0.51.78, 392.1.0.57.78, 392.2.0.58.7
8, 393.1.0.50.76, 394.0.0.46.81, 395.0.0.56.165, 396.0.0.46.242, 397.0.0.46.81,
397.1.0.52.81, 398.0.0.45.77, 398.1.0.53.77, 399.0.0.51.85, 400.0.0.49.68, 401.0
.0.48.79, 402.0.0.49.70, 403.0.0.49.74, 404.0.0.48.76, 405.0.0.52.77, 405.1.0.57
.77, 406.0.0.53.159, 406.0.0.58.159, 407.0.0.55.243, 408.0.0.51.78, 409.0.0.48.1
70, 409.1.0.49.170, 410.0.0.53.71, 410.1.0.63.71, 411.0.0.23.257, 412.0.0.35.87,
 413.0.0.41.84, 414.0.0.40.83, 415.0.0.36.76, 415.1.0.46.76, 416.0.0.47.66, 417.
0.0.54.77, 418.0.0.51.77, 419.0.0.49.71
```

EFF

SUNSECCON

😵😱 No good tools for it! 😭😔

- A few projects out there but they were in rough shape *or* meant to clone Play Store functionality on-device (e.g. Aurora Store)

- Not having such a tool was interrupting our own work on Threat Lab projects!

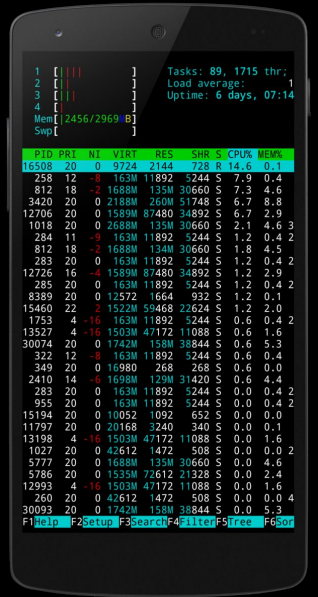
When it doesn't exist, make it!

Design Decision: Rust

- Previous EFF Project in Rust:
  - HTTPS Everywhere's `lib-core`
    - HTTPS Everywhere: list of all sites offering HTTPS (2010-2022)
      - Problem: By 2018, 🐌 lookups
      - Solution: Move core lookup functionality into new Rust module, include in (JavaScript) addon. Allow lookups for 'simple' rules in bloom filter. How?
        - Rust target: WebAssembly, tooling `wasm-bindgen`
      - Result: Dramatic performance increase. Systems-level speeds in browser extension(!!)

Design Decision: Rust

- Allowed targeting all android platforms (including architectures already mentioned) - directly run APK downloader on android hardware via Termux
  - Memory safety guarantees (ownership & borrowing)
  - Type safety, strict compiler
  - Speed
- Requires cross-compilation infrastructure, but 🤷



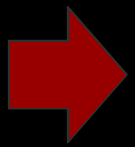
Design Decision: Multiple App Stores

- Goal: `apkeep` used for multiple purposes: backup, analysis
  - Can't know how (or from where) people want to source `apk` files.
- Solution: Support multiple stores.
  - Mainline: Google Play Store
  - Redundant: ApkPure
  - Purposeful: F-Droid
    - F-Droid Mirrors, Non-mirror F-Droid protocol repositories
  - Georegional: Huawei AppGallery

Design Decisions: Cont.

- Download multiple apks and multiple versions (specified in e.g. CSV file)
- (Relatively) easy to use, versatile, swiss army knife for downloading apks
  - Default to `APKPure` Google Play mirror so no need for account.

```
com.openai.chatgpt,true,true
com.einnovation.temu,true,true
com.zhiliaoapp.musically,true,true
com.shopify.arrive,true,true
com.whatsapp,true,true
easy.launcher,true,true
com.wbd.stream,true,true
com.instagram.android,false,true
com.walmart.android,true,true
com.zzkko,true,true
com.squareup.cash,true,true
com.tubitv,true,true
com.paypal.android.p2pmobile,true,true
com.amazon.mShop.android.shopping,true,true
com.alibaba.aliexpresshd,true,true
com.cvs.launchers.cvs,true,true
com.spotify.music,true,true
com.facebook.katana,true,true
com.peacocktv.peacockandroid,true,false
com.onefinance.one,true,true
com.facebook.orca,true,true
org.telegram.messenger,true,true
com.whatnot_mobile,true,true
com.disney.disneyplus,true,true
com.indeed.android.jobsearch,true,true
```



```
Downloading com.openai.chatgpt...
Downloading com.einnovation.temu...
Downloading com.zhiliaoapp.musically...
Downloading com.shopify.arrive...
com.einnovation.temu downloaded successfully!
Downloading com.whatsapp...
com.openai.chatgpt downloaded successfully!
Downloading easy.launcher...
com.shopify.arrive downloaded successfully!
Downloading com.wbd.stream...
com.zhiliaoapp.musically downloaded successfully!
Downloading com.instagram.android...
com.whatsapp downloaded successfully!
Downloading com.walmart.android...
easy.launcher downloaded successfully!
Downloading com.zzkko...
com.wbd.stream downloaded successfully!
Downloading com.squareup.cash...
com.zzkko downloaded successfully!
Downloading com.tubitv...
com.instagram.android downloaded successfully!
Downloading com.paypal.android.p2pmobile...
com.squareup.cash downloaded successfully!
Downloading com.amazon.mShop.android.shopping...
com.tubitv downloaded successfully!
Downloading com.alibaba.aliexpresshd...
com.amazon.mShop.android.shopping downloaded successfully!
Downloading com.cvs.launchers.cvs...
[00:00:29] ████████████████████ 262.02 MiB/288.49 MiB | com.walmart.android.apk
[00:00:11] ████████████▌        96.32 MiB/147.09 MiB | com.paypal.android.p2pmobile.apk
[00:00:04] ██████████▌          40.59 MiB/58.87 MiB | com.alibaba.aliexpresshd.apk
```

Implementation Challenges: Google Play

- Closed source!
- Little to no documentation on how e.g. Google Play API implementation works...
  - Early Python reference API no longer available, was used at first to build out upstream `gpapi` crate (2021)
  - Significant work analyzing network traffic via (`mitmproxy`) and doing bitwise payload comparison:
    - Determine where there's some mismatch between my API calls and Google Play's API calls

Implementation Challenges: Google Play

- Ciphersuite fingerprinting!? Why!?

gpapi/src/consts.rs +3

```
@@ -5,6 +5,9 @@ pub const GOOGLE_PUB_KEY_B64: &'static str = "AAAAgMom/1a/v0lblO2Ubrt60J2gcuXSlj
5  5  pub const GOOGLE_PUB_MODULUS_HEXSTR: &'static str =
   "ca26ff56bfbf495b94ed946ebb7ad09da072e5d296318541781cc995af7962c4c28ea9af0822de224865da1dca129942b356a799ca277b2b4577145be175043ddb684546726120a9a2d950d0639b4e7ba4a448d7a901d18a69786c79a884394232b3
   b11f044d06ca2cd5a0458d1044d573df890c251dcffcb8076b1ffaae67f9";
6  6  /// Google Play Public exponent
7  7  pub const GOOGLE_PUB_EXP: u32 = 65537;
   8  + /// Exact ciphersuite specification is needed, see:
   9  + /// https://stackoverflow.com/questions/22832104/how-can-i-see-hidden-app-data-in-google-drive
   10 + pub const GOOGLE_ACCEPTED_CIPHERS: &'static str =
   "TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256:ECDHE+AESGCM:ECDHE+CHACHA20:DHE+AESGCM:DHE+CHACHA20:ECDH+AESGCM:DH+AESGCM:ECDH+AES:DH+AES:RSA+AESGCM:RSA+AES:!aNULL:!eNUL
   L:!MD5:!DSS";
8  11
9  12 pub mod defaults {
10 13     pub const DEFAULT_LANGUAGE: &str = "en_US";
```

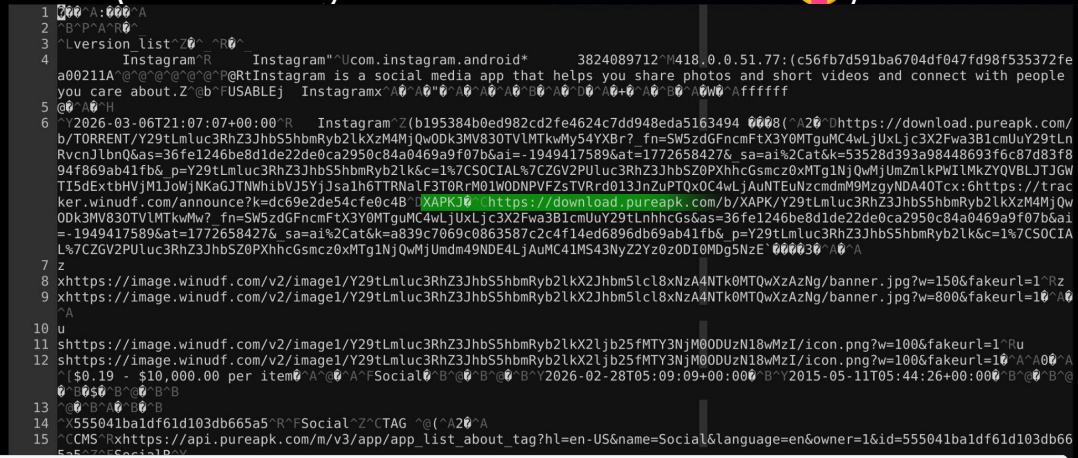
- Added dependency: OpenSSL (via `hyper-openssl`) 😭
  - (RusTLS doesn't support arbitrary ciphersuite offer ordering)

Implementation Challenges: Google Play

- OpenSSL dependency *means* you have to cross-compile it for all platforms you want targeted which don't include `libssl-dev` package support, and even platforms which do, but not available on cross-compilation machine (all android targets)
- For Windows support, easier to statically compile OpenSSL on Windows and include resulting files in binary 🫠
- Entire API exchange is in ProtoBuf
  - Have to obtain `.proto` files for Google Play

Implementation Challenges: APKPure

- Again, 0 documentation…
- Strange serialization format… (`file body.bin` returns “data” 🤪)
- Not easily parseable,
  - Regex to the rescue



Code Blame 7 lines (7 loc) · 833 Bytes Raw

```
pub const APKPURE_VERSIONS_URL_FORMAT: &str = "https://api.pureapk.com/m/v3/cms/app_version?hl=en-US&package_name=";
pub const APKPURE_DOWNLOAD_URL_REGEX: &str = r"(X?APKJ)..(https?://(www\.)?[-a-zA-Z0-9@:%._\+~#=]{1,256}\.[a-zA-Z0-9()]{1,6}\b([-a-zA-Z0-9()@:%_\+.~#?&//=]*))";
pub const FDROID_REPO: &str = "https://f-droid.org/repo";
pub const FDROID_INDEX_FINGERPRINT: &[u8] = &[67, 35, 141, 81, 44, 30, 94, 178, 214, 86, 159, 74, 58, 251, 245, 82, 52, 24, 184, 46, 10, 62, 209, 85, 39, 112, 171, 185, 169, 201, 204, 171];
pub const FDROID_SIGNATURE_BLOCK_FILE_REGEX: &str = r"^META-INF/.*\.(DSA|EC|RSA)$";
pub const HUAWEI_APP_GALLERY_CLIENT_API_URL: &str = "https://store-dre.hispace.dbankcloud.com/hwmarket/api/clientApi";
pub const PROGRESS_STYLE: &str ="[{elapsed_precise}] {bar:40.cyan/blue} {bytes}/{total_bytes} | {msg}";
```

Implementation Challenges: F-Droid

- Finally, documentation!
  - Very helpful dev community
- Verification of apps in large manifest (`index.json` => `index-v2.json`) signed by f-droid maintainers.
- Support for additional repos means verifying against *those* maintainers keys as well.
- Want to have the same verification of signatures built into `apkeep`!

Introducing “apkeep,” EFF Threat Lab’s new APK Downloader

BY **BILL BUDINGTON** | SEPTEMBER 2, 2021

To track state-sponsored malware and combat the stalkerware of abusive partners, you need tools. Safe, reliable, and fast tools. That’s why EFF’s Threat Lab is proud to announce our very own tool to download Android APK files, apkeep. This enables users to download an Android APK or number of APKs directly from the command-line—either from the Google Play Store (with Google credentials) or from a third-party which mirrors the Play Store apps (no credentials needed).

Written in async Rust, this tool prioritizes simplicity of use, memory safety, reliability, and speed. It has also been compiled to a number of architectures and platforms, including Android’s `armv7` and `aarch64` platforms to download apps directly from an Android device using Termux. It is available right now for you to use.

In the future, we hope to expand apkeep’s functionality by adding support for the Amazon Appstore, allowing downloads of older app versions, and adding additional architectures.

We are proud to give back to the pool of tools that the application security community has created and that we use every day. We hope our own

RELATED TAGS:

THREAT LAB MOBILE


```
Downloads APKs from various sources

Usage: apkeep <-a app_id[@version] | -c csv [-f field] [-v version_field]> [-d download_source] [-r parallel] OUTPATH

Arguments:
  [OUTPATH]  Path to store output files

Options:
  -a, --app <app>
          Provide the ID and optionally the version of an app directly (e.g. com.instagram.android)
  -c, --csv <csv>
          CSV file to use
  -f, --field <field>
          CSV field containing app IDs (used only if CSV is specified) [default: 1]
  -v, --version-field <version_field>
          CSV field containing versions (used only if CSV is specified)
  -l, --list-versions
          List the versions available
  -d, --download-source <download_source>
          Where to download the APKs from [default: apk-pure] [possible values: apk-pure, google-play, f-droid, huawei-app-gallery]
  -o, --options <options>
          A comma-separated list of additional options to pass to the download source
  -i, --ini <ini>
          The path to an ini file which contains configuration data
      --oauth-token <google_oauth_token>
          Google oauth token, required to retrieve long-lived aas token
  -e, --email <google_email>
          Google account email address (required if download source is Google Play)
  -t, --aas-token <google_aas_token>
          Google aas token  (required if download source is Google Play)
      --accept-tos
          Accept Google Play Terms of Service
  -s, --sleep-duration <sleep_duration>
          Sleep duration (in ms) before download requests [default: 0]
  -r, --parallel <parallel>
          The number of parallel APK fetches to run at a time [default: 4]
  -h, --help
          Print help
  -V, --version
          Print version
```

EFF

SUNSECCON

Threefold Goals

- Organizational: Facilitates our own work
  - Android malware reversing, also
  - Location data brokers project working on now
- Personal:
  - Understand how app distribution mechanisms work
  - Cutting teeth on async Rust
- External: Gives back to the open source infosec community!
  - Used in a variety of projects

EFF

SUNSECCON

Milestones: Google Play

- Upgraded to Google Play API v3
  - Remove `hyper-openssl`, moved to `reqwest`
- Ability to download “split APKs”
  - Separate APK for locale, CPU architecture
- Download additional ‘expansion’ files: `.obb`
  - Used for games and other large downloads > 200MB
- As well as DexMetadata (`.dm`) files, which include
  - Android ART profiles (Google Cloud Profile)
- Device configurations maintained by Aurora Store

Aurora

Open Source Software

Milestones: More ways to install...

- Download binaries directly from https://github.com/EFForg/apkeep/releases/
- From `cargo`:
  - Latest release:
    - `cargo install apkeep`
  - Latest commit:
    - `cargo install --git https://github.com/EFForg/apkeep.git`
- Termux: `pkg install apkeep`
- Github's Docker Container Registry:
  - `docker pull ghcr.io/efforg/apkeep:stable`

EFF

SUNSECCON

Milestones: Cont.

- F-Droid:
  - Introduced newer entrypoint specification, move to support that *by default* with a fallback to older standard
  - Allow listing of versions
  - JSON output for versions, downloads
- UX:
  - Nice progress bar!
  - Sweet new logo

Where is apkeep used?

- Exodus Privacy

Where is apkeep used?

- Researchers, hackers…

Profile Coverage: Using Android Compilation Profiles to Evaluate Dynamic Testing

Jakob Bleier
jakob.bleier@seclab.wien
TU Wien
Vienna, Austria

Felix Kehrer
felix.kehrer@seclab.wien
TU Wien
Vienna, Austria

Jürgen Cito
juergen.cito@tuwien.ac.at
TU Wien
Vienna, Austria

Martina Lindorfer
martina@seclab.wien
TU Wien
Vienna, Austria

***Abstract*—The rising complexity of Android apps makes comprehensive dynamic testing infeasible, especially for third-party apps. Knowing which methods are exercised by real users typically requires costly user studies or access to usage telemetry. We show that Android's *compilation profiles*, specifically *Cloud Profiles* collected by the Google Play Store, offer a readily available, underutilized source of such information. These operational profiles aggregate which methods are commonly executed across users and guide ahead-of-time compilation during app installation. We provide the first in-depth characterization of *Baseline Profiles* and *Cloud Profiles* and show that over 99.89 % of the top 1,000 apps include usage-derived cloud profiles.**

Based on this insight, we introduce *profile coverage*, a novel metric that measures how well dynamic testing exercises the methods real users interact with. This metric builds on the idea of operational coverage and enables a more holistic evaluation of automated test input generators. To enable profile coverage measurements, we develop a lightweight tracer, ProfTrace, based on Linux kernel uprobes that requires no app or system modifications. We demonstrate its utility by comparing three tools and a no-interaction baseline on 50 popular apps, showing that profile coverage reveals differences that traditional code coverage misses. For instance, in *Candy Crush*, automated testing achieves only 2.22 % method coverage, but 21.39 % profile coverage—indicating better alignment with user behavior than traditional code coverage would suggest.

I. Introduction

Dynamic analysis of Android apps plays an important role in both quality assurance and security assessments. From developers trying to detect bugs and crashes [54, 65, 76], to researchers studying entire app ecosystems to detect and characterize malware [10, 45, 46] and assessing different aspects of app privacy [38, 56, 60–62, 66, 75], automating the interaction with apps is a task with many applications.

Limitations of Traditional Coverage Metrics. As app complexity increases, test input generation techniques are typically evaluated using code coverage metrics, which report the share of code executed during testing. These metrics range in granularity from instruction-level to method, class, or activity important: some methods are critical to core functionality, while others are rarely executed in real-world usage. Thus, traditional coverage cannot distinguish between coverage of highly relevant vs. rarely used parts of the app.

Understanding Profiles. To address this gap, we turn to Android's *compilation profiles*, which represent actual usage patterns at scale. In particular, *Cloud Profiles* are aggregated from telemetry data collected by the Google Play Store during real-world use and are used by the Android Runtime (ART) to optimize app startup. Despite being widely deployed, they remain underdocumented. We begin by answering **RQ1:** *What profiles exist, how are they created, and for what are they used?*, based on a deep dive into Android's source code.

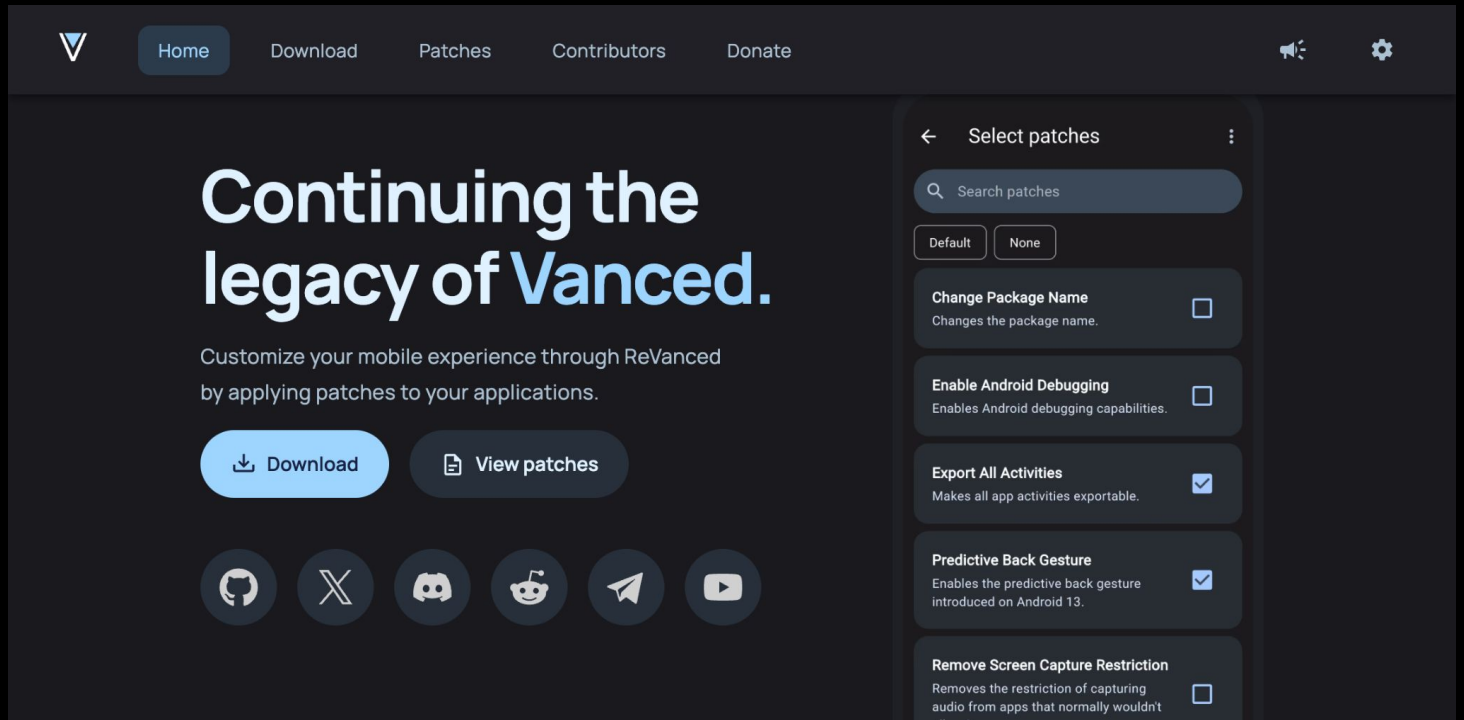
We further ask and answer **RQ2:** *How prevalent are Baseline Profiles and Cloud Profiles among popular apps on the Google Play Store?*, and **RQ3:** *Do developer-supplied Baseline Profiles differ from usage-derived Cloud Profiles?* To this end, we collect over 43,056 APKs and 43,009 *Cloud Profiles* from the top 1,000 apps. We show that profiles are widely available and often diverge, underlining the value of metrics based on telemetry for testing and optimization.

Profile Coverage for Testing Effectiveness. We propose *profile coverage* as a novel metric for evaluating dynamic testing techniques. It measures the proportion of methods listed in an app's *Cloud Profile* exercised during testing. This approach builds on the concept of *operational coverage* [53], which evaluates testing against actual usage behavior. Unlike operational coverage, however, profile coverage does not require app-specific instrumentation or large-scale user studies; it leverages usage data already collected and curated via the Google Play Store. Profile coverage provides a more grounded and holistic assessment of test effectiveness by contextualizing which parts of an app are covered. This perspective is especially valuable when analyzing third-party apps, where the developer's intent or internal structure is often unavailable.

Beyond large-scale evaluation, profile coverage is also

Where is apkeep used?

- App patchers...



What's being done on apkeep?

- Adding support for more app stores and architectures
  - Future app stores (e.g. Xiaomi, Baidu)
  - Future platforms (e.g. MacOS)
- Paid Apps for accounts which purchased them
- Integration into existing reversing toolboxes
- Additional repositories

EFF

SUNSECCON

What `apkeep` is (and isn't)

- Not some flashy, cool project that does everything
- Purpose-specific, meant to do one thing right
  - Yes: Flexible, extensible, and portable
  - What you want in a tool: do that one thing right and get out of the way!
- Didn't see an easy tool for this out there already, so why not make it?

Questions?

https://github.com/EFForg/apkeep
https://www.eff.org/pages/apkeep

Bill Budington
Senior Staff Technologist