A LARGE-SCALE DATA COLLECTION AND EVALUATION FRAMEWORK FOR ANDROID DEVICE SECURITY ATTRIBUTES

Ernst Leierzopf, Michael Roland, René Mayrhofer

Johannes Kepler University Linz, Austria / Institute of Networks and Security

{leierzopf, roland, rm}@ins.jku.at

Florentin Putz

Technical University of Darmstadt, Germany / Secure Mobile Networking Lab
fputz@seemoo.tu-darmstadt.de

DOI: 10.35011/IDIMT-2023-63

Keywords

Android, Data Mining, Security and Privacy, Webscraping, Crowdsourcing

Abstract

Android's fast-lived development cycles and increasing amounts of manufacturers and device models make a comparison of relevant security attributes, in addition to the already difficult comparison of features, more challenging. Most smartphone reviews only consider offered features in their analysis. Smartphone manufacturers include their own software on top of the Android Open Source Project (AOSP) to improve user experience, to add their own pre-installed apps or apps from third-party sponsors, and to distinguish themselves from their competitors. These changes affect the security of smartphones. It is insufficient to validate device security state only based on measured data from real devices for a complete assessment. Promised major version releases, security updates, security update schedules of devices, and correct claims on security and privacy of pre-installed software are some aspects, which need statistically significant amounts of data to evaluate. Lack of software and security updates is a common reason for shorter lifespans of electronics, especially for smartphones. Validating the claims of manufacturers and publishing the results creates incentives towards more sustainable maintenance and longevity of smartphones. We present a novel scalable data collection and evaluation framework, which includes multiple sources of data like dedicated device farms, crowdsourcing, and webscraping. Our solution improves the comparability of devices based on their security attributes by providing measurements from real devices.

1. Introduction

Android is a rapidly developing open-source mobile operating system. Due to Android's fast-paced development cycles, the heterogeneity of device manufacturers, and the large number of devices with vendor-specific software modifications, it is challenging to evaluate and compare devices with regard to security and functionality. AOSP follows strict release cycles with one major version update per year and monthly security updates including updates for the Android platform, the upstream Linux kernel and fixes from system-on-chip (SOC) manufacturers. These security updates are defined in security patch levels and can be found in the Android Security Bulletins. Smartphone manufacturers include their own software on top of AOSP to improve user experience, to add their own pre-installed apps or apps from third-party sponsors, and to distinguish themselves from their competitors. Consequently, smartphone manufacturers need to adapt these additions and customizations of the AOSP codebase before releasing (security) updates to end user devices. Therefore, a fully automated rollout process from AOSP to the end user is not yet possible. However, Google has started to address this issue. Starting with Android 8, *Project Treble* (Malchev, 2017) aims to separate device-specific vendor implementations from the Android OS framework, which allows for faster Android releases without modification of hardware-specific parts of Android. *Project Mainline* (Siddiqui, 2020) adds modular system components and introduces the Android Pony EXpress (APEX) file format for system component packaging in Android 10. Since then, more and more system components are modularized in later Android versions. Modular system components are a major improvement on the update process, as Google is able to remove dependencies on manufacturers by providing updates of system modules over existing infrastructure like Google Play Store. These developments and increasing commitment to continuous maintenance by smartphone manufacturers for up to 5 years of security updates (Atanassov, 2021; Google, 2023; Samsung Mobile Security, 2022) improve the lifespan of Android devices, as lack of security updates often leads to restricted use in applications with high security requirements. Given such commitments, it is vital to verify if manufacturers keep their promises and if there are discrepancies in the quality of the maintained software across different manufacturers.

In addition to the functional aspects of a device, more-recent Android versions offer improved security and privacy features, which users may want to consider for their purchase decisions. The ever-growing number of Android devices and device manufacturers increase the number of options for consumers ranging from low-budget to high-end flagship devices. Psychology suggests that choice overload decreases the ability of consumers to properly compare and evaluate their options due to decreasing commitment and oversight (Schwartz, 2003). Web services like GSMArena provide the possibility to search for and compare multiple devices from different manufacturers. Most of these websites scrape manufacturer websites for devices and update their internal database. Users are often not able to validate the information from such third-party websites.

This paper presents a scalable data collection and evaluation framework, which includes multiple sources of data like dedicated device farms, crowdsourcing and webscraping, and stores the collected data for analysis and presentation to end users. In addition, data is recorded in a raw file archive to maintain transparency and permit iterative re-processing upon improvements of analysis algorithms. The aggregated database should serve experts as well as average smartphone users, by providing detailed data measured from real devices (to verify manufacturer claims) and a simple total security score for fast comparison (to assist future buying decisions). An initial prototypical frontend for advanced users is available at https://www.android-device-security.org/.

2. Related Work

The Android ecosystem contains a complex, evolving architecture including different security mechanisms. Mayrhofer et al. (2021) describe mechanisms and concepts in AOSP and their development history up to Android version 11. Based on their research, there are many privacy and security related threats like the risk of pre-installed applications with pre-granted permissions.

Lau et al. (2020) propose the Uraniborg risk computation framework as an approach to calculate the security risk of an Android device. Their formula is based on the number of apps with signature permissions, pre-granted permissions, cleartext communication of apps, and a custom risk score for specific permissions defined by the authors.

Ozbay and Bicakci (2023) propose a total device security score based on metrics of pre-installed applications. Each metric has its own security score calculated by multiplying the number of affected pre-installed applications, the difficulty to exploit, and the impact of exploitation. The result is a score based on the normalized sum of all individual metric scores.

Other work focuses on static code analysis of firmware images from different vendors to evaluate the security of Android devices based on pre-installed software, security patch rollout time, verification of security update promises, and the number of vulnerable apps with open CVEs (Elsabagh et al., 2020; Gamba et al., 2020; Hou et al., 2022).

Pöll and Roland (2022) investigated on the reproducibility of AOSP and proposed a framework to analyze the state of reproducibility of AOSP and AOSP-code used in device firmware images. The authors introduced accountable builds as a form of reproducibility permitting explainable differences. They found that pure AOSP is already close to full reproducibility and that Project Treble (Malchev, 2017) helped to get the core operating system (*system.img)* closer to AOSP in actual device firmware by eliminating hardware and manufacturer-specific components.

Wagner et al. (2014) conducted a large-scale crowdsourced data collection of low-level data in 33 different event categories from 12,500 Android users worldwide within a timespan of nearly two years. As a result, Thomas et al. (2015) proposed the Free-Update-Mean (FUM) security metric to rank the performance of device manufacturers and network operators, based on their provision of updates and exposure to critical vulnerabilities. Their conclusion was that the main bottleneck in the latency of the security update processes lies within the manufacturers.

Khokhlov and Reznik (2017) proposed the Overall Security Evaluation Score (OSES) to evaluate device security and to check the validity of sensor data. The metric assumes that the more permissions a user has, the higher the risk of sensor data manipulation.

Many manufacturers publish relevant security update promises, security bulletins, and other information about devices on their websites; often lacking a machine-readable format. Due to the vast quantity of existing device models, it is challenging to collect significant amounts of first-hand information about Android devices. Aside of crowdsourcing, manual or automated webscraping is usually the only way to obtain such information. Third-party websites also offer collections of information about Android devices, but this information is usually not obtained from actual devices, but through other, potentially wrong or outdated sources. Therefore, the information must be validated before usage in security evaluations. Jones et al. (2020) put effort into the investigation of rollout processes of Android security updates and OS upgrades. They used a pseudonymized dataset based on HTTP access logs from a social network app containing request date, hashed user account identifier, user-agent string (including OS and build version, phone model, etc.), country code (derived from IP address), combined with data scraped from Android security bulletins, carrier and manufacturer security update announcements, and device release dates from GSMArena and PhoneArena. Particularly the analysis of metadata from Android security bulletins and CVEs is interesting for this work, because it shows what security updates need to be implemented by manufacturers to fulfill their security update promises (Farhang et al., 2020; Wu et al., 2019).

Previous work provides a solid basis for the evaluation of Android devices from different manufacturers. However, most approaches do not offer an actionable way to evaluate the security of specific devices. Consequently, the results are short-lived and often incomplete. Our approach creates and provides access to a database of security-relevant attributes for specific devices.

3. Collecting Security State of Android Devices

In order to collect and aggregate data about update distribution, supported security and privacy features, device state in general, and promised device features, we create a data collection and processing framework that collects data from controlled lab devices (in the form of device farms), in-field devices (through future crowdsourcing) and from webscraping. A scalable data ingestion pipeline accepts data from all the different sources, stores it for long-term archival and for subsequent post-processing. A modular post-processing pipeline analyzes the data and derives attributes about device security and privacy features to be stored in a database for presentation through a web-based user interface.

3.1. Data Ingestion and Processing

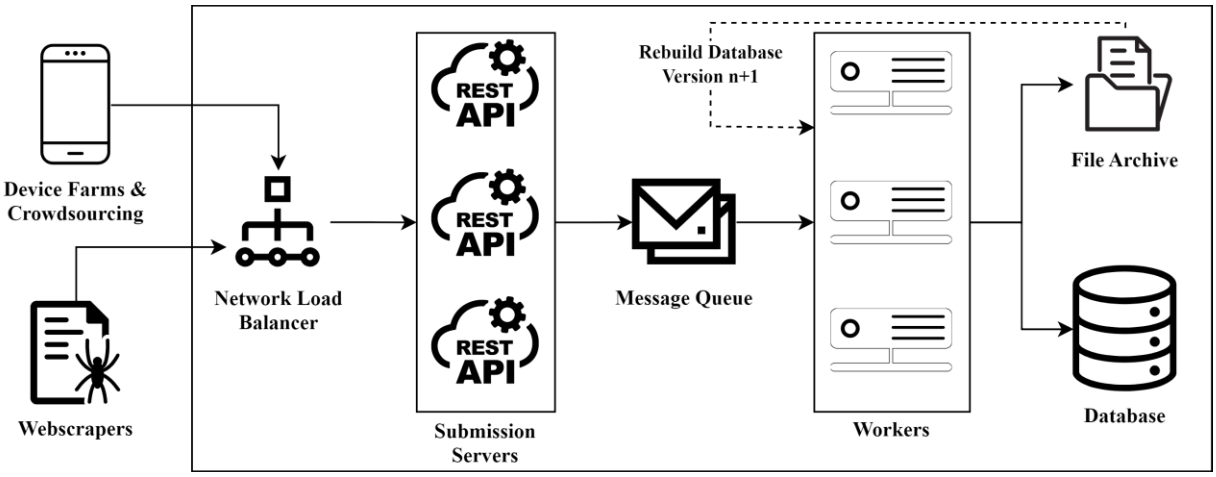

Figure 1. Data Ingestion and Processing Pipeline

The data ingestion and processing pipeline consists of modular server components that receive data from controlled lab devices in device farms, from crowdsourcing, and webscrapers. All components are designed in a modular and scalable manner with the aim of high-throughput processing. To achieve this goal, a load balancer receives distributes incoming requests to one or more submission servers. The submission servers implement a simple REST API, which receives the submissions as JSON objects and writes them into a message queue (MQ). The MQ reduces the risk of dropping incoming requests due to servers being overloaded with processing data.

Depending on its source, the data is labeled with different trust levels distinguishing between trusted data from controlled lab devices, controlled webscrapers and (potentially manipulated) crowdsourced data. An HMAC (hash-based message authentication code) based authentication with shared secrets is used to identify data submitted by trusted sources.

Workers are responsible for the offline processing of data from the MQ. From experience, we have learned that errors may occur in data or the processing implementation that result in inaccurate post-processed data. Manual fixes are time-intensive and the change history is often incomplete and opaque. To tackle this problem, we implemented a file-based archive, which stores all raw data and permits later rebuilding of the post-processed data by feeding the collected raw data into the MQ again. When a new iteration of the database needs to be built, the down time of the frontend can be minimized by reprocessing the raw data into a fresh database, which replaces the old one once the build process is finished. Unique file names based on content hashes allow for an efficient synchronization to backups and between multiple instances of the archive. Figure 1 shows the full pipeline.

3.2. Data Storage

All collected raw JSON data is stored in a file archive. A special folder structure derived from the hexadecimal representation of a SHA-256 hash function allows for efficient searching through the archive. Half of the hexadecimal characters are grouped in pairs, each representing one level in the folder hierarchy. This strategy permits faster searches of specific messages. The filename consists of the archival timestamp, a message-specific index token for file-lookup, and the hexadecimal representation of the authentication HMAC (if one exists).

Each attribute derived from raw data through post-processing is stored as a separate measurement in the database without linking back to the ingested raw dataset. This reduces the risk of device fingerprinting and privacy-relevant data leaks from the processed data through unique attribute combinations. Measurements only include a reference to the device model and its specific firmware version, a validity score, the origin (if from a trusted source), and the measured value. They are grouped into multiple categories. The definition of these categories and the determination of the validity score, which represents the trustworthiness of the measured value based on the source of the data, are subject to future evaluation. By using measurements from actual devices and by comparing those with update expectations (from scraped data), we are able to check how trustworthy manufacturers are with their update promises.

3.3. Android Device Farm

Using controlled lab devices in organized in device farms for measuring state of untainted devices is an effective way to ensure integrity of collected data. It also allows running tests that cannot easily be run on in-field devices via crowdsourcing, e.g. due to relying on access through Android Debug Bridge (ADB) or due to being intensive in terms of screen time, storage, etc.

We have built a device farm consisting of about 30 Android devices with Android versions ranging from 7 to 13. All devices are connected via a set of USB hubs to a Mini PC (Zotac ZBOX CI547 Nano) running Debian/Linux, which gives access to all devices through ADB. Due to their heterogeneity, Android devices have subtle differences in interaction methods and commands required to fulfill a task. Additionally, ADB on its own not efficient for the management of a whole device farm. To overcome these challenges, we implemented a Device Farm Manager with a web interface to monitor the state of devices and to create scheduled tasks for running our attribute collection.

The farm manager allows connecting to and managing multiple distributed device farms simultaneously. In future, the infrastructure will consist of multiple device farms operated by trusted institutions and organizations that contribute measurements to the submission pipeline.

As a first take-away from operating a device farm for several years is that battery management is troublesome. Keeping devices continuously attached to an external USB power supply, resulted in damaged (bloated) batteries for several devices. As a solution, we decided to keep the devices attached to USB (and thus charging) for only one hour per day. We achieve this by switching the power supply of the USB hubs. However, this creates a range of new problems including loss of ADB connectivity (due to unusual default USB configuration after restarting the USB hubs). Also, the battery management of some devices could not handle this alternative charging mode resulting in yet more damaged batteries. Eventually, we replaced batteries on affected phones with supercapacitors by removing the battery, extracting the protection circuit from the battery pack and replacing the Lithium cells with 5-farad capacitors with a dielectric strength of more than 5 volts.

3.4. Attribute Scanner App

We created an app to collect measurements from Android devices. The app can be run on lab devices as well as on devices of volunteers (crowdsourcing). It consists of a lightweight frontend and a modular scanner library. In future, the library may be included in third-party apps to boost crowdsourcing. A plug-in system simplifies the addition of new scanner modules and allows scans with pre-defined subsets of scanner modules using the App settings or ADB arguments. When run via ADB, the app also accepts the HMAC secret for authenticated submission of results.

In the device farm, the farm manager schedules different types of scans at different intervals (e.g. daily, weekly, or monthly), based on the likelihood that scanned data may change (e.g. due to system updates), expected collection effort (processing time to get results) and addition of new scanner modules. The scheduled process interacts with devices over ADB and consists of unlocking the device, installing the latest version of the scanner app, and starting it with scan-specific parameters. After scan completion, the app directly submits the collected data into the ingestion pipeline.

3.5. Crowdsourcing

Despite the advantages of device farms, it is infeasible to achieve full device coverage due to cost and management effort. In addition, staged update rollout processes, regional differences, mobile network operator customizations, etc. may influence measurements. Crowdsourced measurements can cover a broader spectrum of smartphone models across different regions, including less popular or older models that are not part of the device farms. Thus, crowdsourcing helps to fill that gap and to increase the scope of the evaluations.

In an initial attempt towards future crowdsourcing, we designed the scanner app with a GUI that permits volunteers to participate in crowdsourcing by collecting and submitting measurements from their devices. The scanners permitted in the crowdsourcing case are designed to minimize required permissions and to avoid collection of potentially personally identifying or otherwise privacy-sensitive data. Nevertheless, unlike with ADB-based automation, the app informs the user about the contents of collected data and explicitly requests user-consent before submitting any data.

To evaluate our prototype implementation of the crowdsourcing process, we have conducted a pilot measurement campaign with 67 volunteers, covering 52 distinct smartphone models from 12 vendors. These devices were manufactured between 2013 and 2020, and ran Android versions 6 to 11. Most devices were manufactured by Samsung, Huawei, and Google, but we also tested our implementation on smartphones from Sony, OnePlus, and Xiaomi, and even on less common Android smartphones such as the Blackview BV5500 Pro, the Fairphone 3, or the Realme X50 5G.

The downside of crowdsourcing is that measurements are not possible on-demand and instead depend on volunteers' ad-hoc participation and consent. Crowdsourcing is best suited for lightweight measurements rather than long-running scans, in order to encourage volunteers to participate in the collection process. Moreover, in order to attract volunteers, the app will need to offer some incentive. E.g., the app might present a total security score for comparison with other devices or recommendations for device-specific security and privacy improvements.

3.6. Webscraping

Not all data that we aim to collect about Android devices is measurable directly on-device. For instance, some manufacturers distribute data on update promises through their websites; some data about version-specific Android features is available in the AOSP documentation; device certifications (if available) are published by independent labs; additional data about devices is available through third-party device information websites. Webscraping is an obvious choice to acquire such information and to enhance the database with information not (yet) collected and verified through measurements from actual devices. The trustworthiness of data is determined based on its origin. Generally, we consider first-party websites more trustworthy.

We developed a Python-based webscraping framework to collect data about the Android platform (Android versions, builds of Google devices, manifest permissions, permission groups), smartphone certifications (Common Criteria, Samsung Knox), security update promises and update schedules from first-hand source (Google, Samsung, Oppo, Huawei, Nokia, Xiaomi, Motorola, Vivo), and basic device information from third-party websites (GSMArena, PhoneDB, AndroidEnterpriseSolutionsDirectory). We use BeautifulSoup to render websites and and Selenium to automate crawling. Scraped data is normalized into a JSON structure and fed into the ingestion pipeline. To assess the requirements for maintenance of the webscraper components due to structural page-changes, we tested our solution against the history of a website based on data from the Wayback Machine.

3.7. Frontend

One of the main outcomes of the project is a browsable frontend to our database. It can be accessed at https://www.android-device-security.org/ and allows users to query information inferred about device models. Data can be filtered based on a set of basic and advanced filters. Basic filters include manufacturers, device names, device model names, and searched attributes. Advanced filters allow for filtering based on selected attributes and ranges of release dates, Android API levels, and security patch levels. The current design allows users to research attributes of one or multiple smartphones. Future work will include additional selection fields and a configurable scoring algorithm, where users can define their own score by defining feature/attribute sets and their impact to compare devices. Future work may also add different presentations of the data to target different experience levels of users or specific comparison use-cases, eventually aiming for a broad target audience ranging from researchers and experienced users evaluating the capabilities of devices to end users making buying decisions.

4. Conclusion

This research focuses on the very important aspect of security on Android devices. We present a scalable architecture for data mining, evaluation, and presentation purposes. The main aim of this architecture is to create a basis for future research, by collecting large amounts of data on the security and privacy capabilities and features of Android devices. The results of this research may impact future purchase decisions of end users and businesses, and tech-reviewers might use the raw data to include a security comparison, in addition to the feature reviews.

In future work we aim to extend the frontend by including more attributes of devices and grouping them into intuitive clusters. Also, there may be different presentation forms based on the knowledge of the users. Further, we aim for a customizable security evaluation score, where the user is able to select and weight specific attributes or clusters in the calculation of the security score. New types of scanners will check if manufacturers provide correct implementations of security-relevant code like for example EncryptedSharedPreferences.

Acknowledgements

This work has been carried out within the scope of ONCE (FFG grant FO999887054) in the program "IKT der Zukunft" funded by the Austrian Federal Ministry for Climate Action, Environment, Energy, Mobility, Innovation and Technology (BMK). This work has been co-funded by the LOEWE initiative (Hesse, Germany) within the emergenCITY center. We thank Patrick Schmidt (TU Darmstadt) for his contributions to the scanner app and the submission pipeline.

References

Atanassov, M. (2021). OnePlus extends software support for its smartphones. https://www.gsmarena.com/oneplus_extends_software_support_for_its_smartphones-news-49868.php (Retrieved 2023-04-16)

Elsabagh, M., Johnson, R., Stavrou, A., Zuo, C., Zhao, Q., & Lin, Z. (2020). FIRMSCOPE: Automatic Uncovering of Privilege-Escalation Vulnerabilities in Pre-Installed Apps in Android Firmware. 29th USENIX Security Symposium (USENIX Security 20), 2379–2396.

Farhang, S., Kirdan, M.B., Laszka, A., and Grossklags, J. (2020). An Empirical Study of Android Security Bulletins in Different Vendors. Proceedings of The Web Conference 2020. https://doi.org/10.1145/3366423.3380078

Gamba, J., Rashed, M., Razaghpanah, A., Tapiador, J.E., & Vallina-Rodriguez, N. (2020). An Analysis of Pre-installed Android Software. 2020 IEEE Symposium on Security and Privacy. https://doi.org/10.1109/SP40000.2020.00013

Google (2023). Learn when you'll get software updates on Google Pixel phones. https://support.google.com/pixelphone/answer/4457705 (Retrieved 2023-04-16)

Hou, Q., Diao, W., Wang, Y., Liu, X., Liu, S., Ying, L., Guo, S., Li, Y., Nie, M., & Duan, H. (2022). Large-scale Security Measurements on the Android Firmware Ecosystem. 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE), 1257–1268. https://doi.org/10.1145/3510003.3510072

Jones, K.R., Yen, T.-F., Sundaramurthy, S.C., & Bardas, A.G. (2020). Deploying Android Security Updates: An Extensive Study Involving Manufacturers, Carriers, and End Users. Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20). https://doi.org/10.1145/3372297.3423346

Khokhlov, I., & Reznik, L. (2017). Data Security Evaluation for Mobile Android Devices. 2017 20th Conference of Open Innovations Association (FRUCT), 154–160. https://doi.org/10.23919/FRUCT.2017.8071306

Kumar, R., Virkud, A., Raman, R.S., Prakash, A., & Ensafi, R. (2022). A Large-scale Investigation into Geodifferences in Mobile Apps. 31st USENIX Security Symposium (USENIX Security 22), 1203–1220.

Lau, B., Zhang, J., Beresford, A.R., Thomas, D.R., & Mayrhofer, R. (2020). Uraniborg's Device Preloaded App Risks Scoring Metrics. https://www.android-device-security.org/publications/2020-lau-uraniborg/Lau_2020_Uraniborg_Scoring_Whitepaper_20200827.pdf

Malchev, I. (2017). Here comes Treble: A modular base for Android. https://android-developers.googleblog.com/2017/05/here-comes-treble-modular-base-for.html (Retrieved 2023-04-16)

Mayrhofer, R., Vander Stoep, J., Brubaker, C., & Kralevich, N. (2021). The Android Platform Security Model. ACM Trans. Priv. Secur., 24(3). https://doi.org/10.1145/3448609

Ozbay, A., & Bicakci, K. (2023). Should Users Trust Their Android Devices? A Scoring System for Assessing Security and Privacy Risks of Pre-Installed Applications. https://doi.org/10.48550/arXiv.2203.10583

Pöll, M., & Roland, M. (2022). Automating the Quantitative Analysis of Reproducibility for Build Artifacts Derived from the Android Open Source Project. Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '22), 6–19. https://doi.org/10.1145/3507657.3528537

Samsung Mobile Security (2022). Announcing up to five (5) years support for Samsung Security Updates on select Galaxy devices. https://security.samsungmobile.com/securityPost.smsb (Retrieved 2023-04-16)

Schwartz, B. (2003). The Paradox of Choice: Why More Is Less. HarperCollins.

Siddiqui, A. (2020). Everything you need to know about Android's Project Mainline. https://www.xda-developers.com/android-project-mainline-modules-explanation (Retrieved 2023-04-16)

Thomas, D.R., Beresford, A.R., & Rice, A. (2015). Security Metrics for the Android Ecosystem. Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM '15), 87–98. https://doi.org/10.1145/2808117.2808118

Wagner, D.T., Rice, A., & Beresford, A.R. (2014). Device Analyzer: Large-Scale Mobile Data Collection. ACM SIGMETRICS Performance Evaluation Review, 41(4), 53–56. https://doi.org/10.1145/2627534.2627553

Wu, D., Gao, D., Cheng, E.K.T., Cao, Y., Jiang, J., & Deng, R.H. (2019). Towards Understanding Android System Vulnerabilities: Techniques and Insights. Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security (Asia CCS '19), 295–306. https://doi.org/10.1145/3321705.3329831