A3S3 - Automated Android Audit of Safety and Security Signals*

Guillaume Nguyen[1][0000−1111−2222−3333] and Xavier Devroey[1][0000−0002−0831−7606]

NADI - University of Namur, Namur, Belgium
`{guillaume.nguyen,xavier.devroey}@unamur.be`

Abstract. Android devices and related applications are increasingly prevalent in our daily routines. Furthermore, these technologies are being used for more than just connecting people around the world. Indeed, Android devices are more and more connected to external sensors or used as sensors, directly gathering data from their environment, which brings them closer to *Cyber Physical Systems (CPS)*. When used for specific purposes such as health, Android devices and related applications can be life-critical (insulin pumps, heart monitoring, etc.), requiring guarantees specific to the application domain. Interestingly, when considering the technical security in domains related to operational technologies, we can see that many *standards* are available while not directly intended for Android applications. Other *regulatory texts* can also be valuable to drive an audit process, although they need more effort to reach technically testable requirements from the legal requirements they define. In particular, Android applications are developed using various device permissions (i.e., resource access), external libraries, etc. In this paper, we present A3S3 a tool to link requirements from industry standards and regulatory texts to Android features to drive a security audit. Following research in the cyber-security community, we suggest an approach based on static code analysis of Android applications to retrieve good and bad *signals*, denoting potential violations, related to non-functional requirements.

Keywords: android · compliance · static analysis · tool · safety · security.

1 Introduction and Motivation

The Android Operating System (OS) makes up approximately 70% of the market share in terms of mobile operating systems [23]. With its development, Android-based devices are now commonly used for many different purposes than initially intended. Indeed, it started as an OS for mobile devices such as phones and tablets when Google released it in 2007, 2 years after their acquisition [7]. And now *Android is an open source, Linux-based software stack created for a wide array of devices and form factors* [11]. This new definition includes way more device types than previously. Many of those devices are part of our daily routines and are being used as means of data collection, among other things. This includes serving as an intermediary with Internet of Things (IoT) devices such as smart home devices, smartwatches, smart TVs, projectors, etc.

Highlighting the sensor capabilities of Android devices and their integration with external sensors, we can see that their use is closer to *Cyber Physical Systems (CPS)*, which relies on sensor data and real-time computing to have an effect on the real world than regular Android applications. CPSs share similarities with *Embedded Systems*, *Internet of Things (IoT)* or *Operational Technology* (OT) in the industry. Gartner defines OT as *hardware and software that detects or causes a change through the direct monitoring and/or control of industrial equipment, assets, processes, and events* [10].

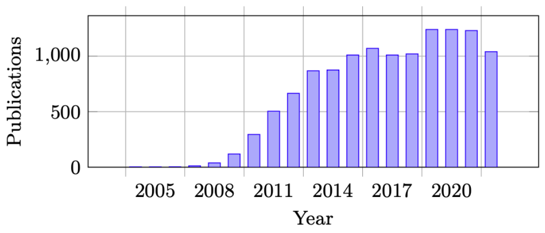

Fig. 1. Evolution of the number of papers based on search results for *"Android device" + "health"* on Google Scholar.

In particular, the number of Android devices used in the health sector for medical purposes is rising. Indeed, as shown in Fig. 1, we can see that the number of papers associated with *Android device* and *Health* has been growing quite steadily since Google released Android. Of course, this is only a light indicator of evolution. Yet, it remains interesting to focus our efforts on ensuring patients' safety when such technologies are used for medical purposes. Furthermore, when talking about Android Smartphones, we can see that they also have their utility as sensors. Indeed, they could also be used as *Fall Detection systems*, a significant issue for elderly people regarding life risks and hospital costs [3]. Other well-known embedded sensors, such as the camera and the microphone, can also serve various purposes in the health sector (e.g., respiratory monitoring) [18]. We can already see Android applications supporting the control of Unmanned Aerial Vehicles (UAV).

Practically, A3S3 is developed as a platform to generate, using static code analysis [17], a report enabling an auditor to decide whether an Android application passes specific audit controls based on the collected *signals* (i.e., indications of a potential violation, extracted using static analysis). Concretely, we want to help developers (or quality assurance) tick audit boxes (called *audit controls*) based on automatically retrieved Android features in specific applications. Once checked, the report produced by A3S3 can be used as an additional input for an audit process supporting the production of a proof of (non-)conformity. In short, we aim at detecting non-conformity risks similar to the efforts of the Software Engineering community in detecting security risks through audits [22]. A non-referenced demonstration video of the tool is available [1], and the source code is openly available [2]. A3S3 is a static code analyser for non-conformity risks on Android application with respect to a specific framework (regulation or standard) based on URL's, Imports and Permissions found in the decompiled code of an application.

2 Related Work

In this section we discuss the various work related to assessing an android application such as previous work in malware detection, static security analysis, non-functional properties and medical applications.

Malware detection - Most Android application assessment focuses on malware detection [6]. For example, if a video game is requesting access to text messages, to the storage of the device, the camera, the Bluetooth devices connected to the phone, etc., it might be a signal that this video game is trying to collect data unlawfully [25]. Exploiting this relationship between permissions and malware, Aswini et al. [2] created *Droid Miner* to extract relevant permissions as *features* and determine whether those *features* are signals of a *malicious* or a *benign* application using machine learning classification techniques.

Static security analysis - Going deeper into Android application code analysis, we can see that there is an extensive collection of tools. Indeed, when trying to determine if an application is *benign* or *malicious*, popular tools like *Flow-Droid* perform a taint analysis using *sources* (data, user input, etc.) and *sinks* (where it goes) [1]. There is also the *FEST* tool that acts as an extractor of API calls and URLs within the code [26]. As a helping tool within the *Intellij* IDE, *SWANassist* takes known *Common Weakness Enumeration* (CWE) from the *MITTRE corporation* [4] and identifies Security Relevant Method from a Java source code [20]. Finally, *SootFX*, a tool developed as a Java API, extracts a tremendous amount of features, manifest, packages, and methods from Android *.apk* files[15].

Non-functional properties - When speaking about non-functional requirements, work is emerging from the well-known (European) *General Data Protection Regulation* (GDPR) [8] and its privacy concerns. Indeed, Tan et al. [24] use Flow-droid to find discrepancies between the declared privacy policies of some applications and the actual collection and processing of personal data within the `.apk`

[1] `https://youtu.be/8GB6Fcq9rII`
[2] `https://github.com/sabredefable/A3S3_python`

file. They actually found that application providers sometimes did not declare everything or even did not classify personal data as such. On a more precise matter, Khedar et al. [16] propose the use of static analysis on `.apk` files to identify where data anonymization was necessary and not performed or not sufficiently performed. At a higher abstraction level, Sacre et al. use metamodels to assess the conformity of a system by comparing a theoretical and compliant model to the model of the system under conformity assessment [21].

Medical applications - Looking more specifically at the security of medical applications, Hussain et al. [12] proposed a framework accompanied by a set of tools to help developers enhance the security maturity of their applications.

As we can see, the main concern is identifying malware or data leakage within the applications. All the existing tools either provide ways of retrieving features, classifying *malicious* and *benign* applications, or finding more general *bugs*. Encouraging work is being done concerning privacy matters on Android applications. However, we look for conformity signals and propose a way to audit non-functional requirements.

3 Approach

Our solution aims to retrieve *signals* for audit controls, presented in a structured data format, and produce an audit report. In a nutshell, the assessment of conformity to a regulation or industrial standard (i.e., the audit) is usually carried out by non-technical auditors who require: (1) inputs from technical experts and (2) thorough documentation from the audited party or both. For example, conformity audits in the European Union are based on technical reports, which must include "*proof of conformity*" based on a specific legal framework [5]. Our approach allows technology-agnostic audit controls of the legal requirements by external non-technical auditors to be interpreted in technology-specific signals (here, Android).

Assessment of medical application. As mentioned earlier, life-critical data analysis safety concerns based on sensor collection share CPS-related concerns. So, for example, in terms of technical security, we can look at manufacturing industry standards such as IEC 62443-4-2 [13], which has been declined into multiple industry-specific variations and lays out seven technology-agnostic *Foundational Requirements* (FR's): *Identification and Authentication Control*, *Use Control*, *System Integrity*, *Data Confidentiality*, *Restrict Data Flow*, *Timely Response to Events* and *Resource Availability*. Those FR's have multiple controls that must be performed for a FR's to be validated. For each control, the auditor has to decide on a Security Level for each of the following categories (as defined in IEC 62443-4-2): the *target level*, the *achieved level*, and the *capability level* of a specific component. The levels range from 0 to 4, the latter being the highest level of security, indicating resilience to cyber attacks using *sophisticated means*, *great resources*, and *motivation*.

For this first application of A3S3, we focus on Android applications working with medical data gathered from a device connected to the Android phone using Bluetooth, any other Android Application Programming Interface (API) used to retrieve data from existing services, or the device itself. Fortunately, the IEC 62443 standard has been adapted into different versions to suit better the needs of specific manufacturing industries (here, the production of medical devices). Specifically, we use the IEC 60601-4-5 [14] that specifies the technical security requirements for Medical Devices (MD). While costs to access such documents quickly add up in the balance, we can assume that it includes the relevant security levels for each MD use case. Thus, we can use the foundational requirements as an audit base. Despite industry standards being easier than regulatory texts to use when assessing the conformity of a *product*, they do not constitute a basis for (technical) audit documents. While they give valuable information to engineers or any interested party on what to aim for when building, developing, producing, testing, etc. *products*, standards must be interpreted for each specific use case to link FR's to implementation. In our example, this means linking FR's related to Data Confidentiality and Restrict Data Flow to specific Android API calls gathering data from connected devices.

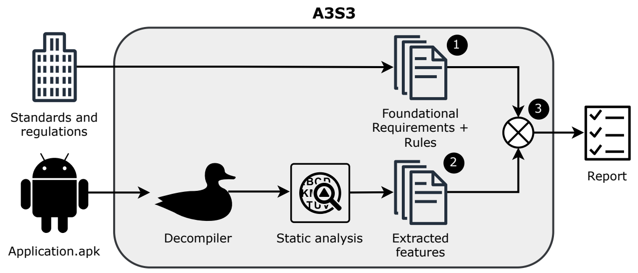

Fig. 2. A3S3 analysis pipeline overview.

Assessment process using A3S3. Fig. 2 presents an overview of the A3S3 approach. First, we must identify the various **FRs** and their link to the different *features* that can be extracted based on defined **rules** (1 in Fig. 2). For instance, Android permissions can be easily collected and linked to the various FR's related to data management, like Data Confidentiality and Restrict Data Flow. Second, we need to apply static analysis to the Android application to collect various features (2 Fig. 2) that can be used to identify signals based on the rules defined in (1). We consider Android executables stored as `.apk` and `.xapk` files. Those files must be decompiled (from executable in binary to human-readable files) before being analyzed. For instance, for our evaluation, we consider Android applications from the Google Play store and search for applications with medical purposes to download a set of `.apk` and `.xapk` files. Finally, A3S3 assesses each

FR individually based on its rules (3 Fig. 2) and produces a report with the various signals (i.e., combinations of extracted features) for each FR. In this paper, we reach level 3 on the Technology Readiness Level (TRL) scale [19].

4 Preliminary Evaluation

For our preliminary evaluation, we performed a reduced audit of an application with medical purposes to test our tool. Specific applications might have to comply with MD requirements [9]. Indeed, when looking at smartphones, we can quickly see that multiple applications offer "medical" follow-up in one form or another. While many applications include a disclaimer saying users should not be considered a replacement for a health professional, they still gather and correlate health data to provide recommendations.

Using requirements extracted from the MDR for devices with a medical purpose (monitoring, prevention or monitoring of disease) and requirements laid out in points 17 and 18 of Annex I we decided to focus on 17.2 stating "*For devices that incorporate software or for software that are devices in themselves, the software shall be developed and manufactured in accordance with the state of the art taking into account the principles of development life cycle, risk management, including information security, verification and validation*" [9, p. 100]. As previously introduced, we decided to use IEC 60601 as state of the art which has a specific application for medical devices. We focused on the fifth one: *Restrict Data Flow (RDF)*, stating that a limited number of data sources is desirable. While not an exhaustive list, we identified 8 possible data sources. By auditing 12 randomly selected health-related applications available on *Google Play*, we found that the average number of data sources per application is 3.75 and that the most common sources are Internet (12/12), Bluetooth (11/12) and Firebase (10/12). Google supports all APIs identified in the source code.

The auditor can decide whether the FR is met for each application based on the data collected. For instance, when considering *FibriCheck*, we identify the three data sources using the following features collected by the static analyzer: (i) Internet, identified from the manifest that has the '`android.permission.INTERNET`' permission;[3] (ii) Bluetooth, identified from the import of the '`android.bluetooth`' library somewhere in the code;[4] (iii) Firebase API, identified from the presence of the '`https://firebase.google.com`' URL in the code and the import of the '`com.google.firebase`' library.[5] Based on those signals, an auditor can assess if the RDF requirement is satisfied.

5 Conclusion and Future Work

We presented A3S3, a tool that performs static code analysis on Android applications following specific controls from various sources to look for *signals* that

[3] https://developer.android.com/develop/connectivity/network-ops/connecting
[4] https://developer.android.com/reference/android/bluetooth/package-summary
[5] https://firebase.google.com/docs/build?hl=fr

an auditor or any interested party might interpret. We showed that linking non-functional controls (i.e., FRs) to specific Android features was possible. Indeed, by auditing 12 Android applications, we looked into various data sources by performing a static code analysis and helping an auditor decide if the control was satisfied. The compliance or non-compliance with a specific control remains within the auditor's power.

While we believe this tool has great potential in assessing Android applications, it will be upgraded to extract even more features. For static code analysis, we will use SootFX, which allows a broader range of static analyses, like the extraction of methods called within the application. For signal generation, we will look for additional technical regulations. For instance, we believe that the controls from IEC 62443 might act as a good cornerstone between high-level and technical requirements for CPS-related safety and security concerns.

We will also work with users to develop coherent dashboards to display the audit results in a user-friendly and adapted way (e.g., for auditors, developers, managers, etc.). We will add information on the various signals found in the assessed application to limit as much as possible the technical knowledge required to understand what the signals mean in the context of a specific regulation.

Acknowledgments. This research was funded by the CyberExcellence by Digital-Wallonia project (No. 2110186), funded by the Public Service of Wallonia (SPW Recherche).

References

1. Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Le Traon, Y., Octeau, D., McDaniel, P.: Flowdroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. ACM SIGPLAN Notices **49**(6), 259–269 (Jun 2014). `https://doi.org/10.1145/2666356.2594299`
2. Aswini, A.M., Vinod, P.: Droid permission miner: Mining prominent permissions for android malware analysis. In: The Fifth International Conference on the Applications of Digital Information and Web Technologies (ICADIWT 2014). IEEE (Feb 2014). `https://doi.org/10.1109/icadiwt.2014.6814679`
3. Casilari, E., Luque, R., Morón, M.J.: Analysis of android device-based solutions for fall detection. Sensors **15**(8), 17827–17894 (Jul 2015). `https://doi.org/10.3390/s150817827`
4. Corporation, M.: CWE - Common Weakness Enumeration - cwe.mitre.org. `https://cwe.mitre.org/`, accessed 10-09-2024
5. EC: EUR-Lex - commission notice the 'blue guide' on the implementation of eu product rules 2022 (2022), `https://eur-lex.europa.eu/`, accessed 19-11-2024
6. Ehsan, A., Catal, C., Mishra, A.: Detecting malware by analyzing app permissions on android platform: A systematic literature review. Sensors **22**(20), 7928 (Oct 2022). `https://doi.org/10.3390/s22207928`
7. Elgin, B.: Google Buys Android for Its Mobile Arsenal - web.archive.org (2005), `https://shorturl.at/Qoh2p`, accessed 06-09-2024
8. EU: Regulation - 2016/679 - EN - gdpr - EUR-Lex - eur-lex.europa.eu. `https://eur-lex.europa.eu/eli/reg/2016/679/oj` (2016), accessed 20-09-2024

9. EU: Regulation - 2017/745 - EN - Medical Device Regulation - EUR-Lex - eur-lex.europa.eu. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32017R0745 (2017), accessed 18-09-2024
10. Gartner: Definition of Operational Technology (OT) - Gartner Information Technology Glossary - gartner.com, https://www.gartner.com/en/information-technology/glossary/operational-technology-ot, accessed 04-Jul-2023
11. Google: Platform architecture | Android Developers - developer.android.com. https://developer.android.com/guide/platform?hl=en, accessed 05-09-2024
12. Hussain, M., Zaidan, A., Zidan, B., Iqbal, S., Ahmed, M., Albahri, O., Albahri, A.: Conceptual framework for the security of mobile health applications on android platform. Telematics and Informatics **35**(5), 1335–1354 (Aug 2018). https://doi.org/10.1016/j.tele.2018.03.005
13. IEC: IEC 62443-4-2:2019 (2019), https://webstore.iec.ch/en/publication/34421, accessed 30-08-2024
14. IEC: IEC TR 60601-4-5:2021 (2021), https://webstore.iec.ch/en/publication/64703, accessed 30-08-2024
15. Karakaya, K., Bodden, E.: Sootfx: A static code feature extraction tool for java and android. In: 2021 IEEE 21st International Working Conference on Source Code Analysis and Manipulation (SCAM). vol. 14, pp. 181–186. IEEE (Sep 2021). https://doi.org/10.1109/scam52516.2021.00030
16. Khedkar, M., Bodden, E.: Toward an android static analysis approach for data protection (2024). https://doi.org/10.48550/ARXIV.2402.07889
17. Louridas, P.: Static code analysis. IEEE Software **23**(4), 58–61 (Jul 2006). https://doi.org/10.1109/ms.2006.114
18. Majumder, S., Deen, M.J.: Smartphone sensors for health monitoring and diagnosis. Sensors **19**(9), 2164 (May 2019). https://doi.org/10.3390/s19092164
19. Mankins, J.C., et al.: Technology readiness levels. White Paper, April **6**(1995), 1995 (1995)
20. Piskachev, G., Nguyen Quang Do, L., Johnson, O., Bodden, E.: Swanassist: Semi-automated detection of code-specific, security-relevant methods. In: 2019 34th IEEE-ACM International Conference on Automated Software Engineering (ASE). IEEE (Nov 2019). https://doi.org/10.1109/ase.2019.00110
21. Sacre, A., COLIN, J.N., Hosselet, B.: ARRCIS: évaluation et renforcement de la conformité réglementaire d'un système d'information, pp. 159–176. No. 52 in Collection du CRIDS, Larcier (2021)
22. Sanchez-Garcia, I.D., Rea-Guaman, A.M., Gilabert, T.S.F., Calvo-Manzano, J.A.: Cybersecurity Risk Audit: A Systematic Literature Review, p. 275–301. Springer Nature Switzerland (2024). https://doi.org/10.1007/978-3-031-50590-4_18, http://dx.doi.org/10.1007/978-3-031-50590-4_18
23. Statista: Mobile OS market share worldwide 2009-2024 | Statista - statista.com (2024), https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/, accessed 09-09-2024
24. Tan, Z., Song, W.: Ptpdroid: Detecting violated user privacy disclosures to third-parties of android apps. In: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). pp. 473–485. IEEE (May 2023). https://doi.org/10.1109/icse48619.2023.00050
25. Zhang, Y., Yang, M., Xu, B., Yang, Z., Gu, G., Ning, P., Wang, X.S., Zang, B.: Vetting undesirable behaviors in android apps with permission use analysis. In:

Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS '13. pp. 611–622. CCS '13, ACM Press (2013)

26. Zhao, K., Zhang, D., Su, X., Li, W.: Fest: A feature extraction and selection tool for android malware detection. In: 2015 IEEE Symposium on Computers and Communication (ISCC). IEEE (Jul 2015). https://doi.org/10.1109/iscc.2015.7405598