# DroidNative: A Greedy-Constructed Large-Scale Indexing for Android Native Libraries

Shiyang Zhang*, Chengwei Liu‡, Sen Chen†§, Lyuye Zhang‡, Yang Liu‡
*College of Intelligence and Computing, Tianjin University, China
†College of Cryptology and Cyber Science, Nankai University, China
‡College of Computing and Data Science, Nanyang Technological University, Singapore

***Abstract*—Native libraries are widely used in Android for performance optimization, but their integration also poses security risks. Although existing research works have investigated the adoption, management, and ecosystem evolution of third-party libraries (TPLs) in Android, studies specific to Android native libraries are still rare, which makes the potential threats of native libraries in Android less concerned. The biggest barrier is that, Android native libraries are usually provided by various suppliers in different ways and sources, leading to the lack of a comprehensive registry that indexes commonly used native libraries for further investigations.**

**To this end, by following a greedy strategy to identify possible repository sources and collect Android native libraries, we constructed the first comprehensive native library database $DroidNative$ for Android, with over 60K libraries and 292K versions well retained. Our experiments proved its completeness that 85.1% of binaries in real-world APPs can be successfully traced in $DroidNative$, with 10.1% of the rest suspicious to be not third-party native libraries. Moreover, $DroidNative$ is also evaluated to be useful regarding improving existing SCA detection (i.e., LibRARIAN) by outperforming existing state of the art tools with at least 78.4% recognition rate improvement.**

***Index Terms*—Android Native Library, SCA**

## I. Introduction

Native libraries are widely used in Android development to enhance performance. A study [1] analyzing over 1.2 million Android applications found that approximately 40% of them incorporated native code, underscoring its widespread adoption. However, the integration of native libraries also introduces challenges, such as security vulnerabilities [2].

Many existing research works [3]–[5] have examined the adoption, identification, and ecosystem analysis for third party libraries in the Android ecosystem. However, although they incorporated different solutions, such as static analysis and machine learning approaches, all of these existing works are focused on TPLs that are published and indexed in package managers, while none of them specifically investigated the supply chain of native libraries in Android Apps.

There are some major challenges that block the analysis of Android native libraries like other TPLs. **1) Lack of central indexes.** Native libraries are scattered across multiple sources, with no standardized repository or indexing services. **2) Lack of effective identification techniques.** Unlike Java/Kotlin dependencies, native libraries are often distributed as precompiled binaries with stripped symbols, obfuscation, and custom modifications, making it difficult to identically distinguish binaries to their libraries and versions for dataset construction.

Therefore, in this paper, we proposed a greedy approach to construct the first comprehensive dataset for Android native library detection. Specifically, for Challenge 1, by analyzing existing ground truths of Android native library importing, we summarized three major types of Android native libraries: Directly-Included, Self-Compiled, and Remotely-Integrated Libraries, and collected as many potential repository sources as possible. For Challenge 2, when validating the identities of binaries found in APKs, we combined the proofs of identical mappings from both corresponding tools and similarity confirmation by code clone analysis, ensuring all identified importing ways are well explained and precisely verified.

After collecting all binaries from identified sources, we constructed a comprehensive dataset for Android native library called $DroidNative$, with 60,287 libraries and 292,797 versions, to facilitate downstream tasks. Our experiments showed that, against a well-constructed real-world testset of mainstream apps, $DroidNative$ has covered at least 85.1% of all binaries in these Apps, with 10.1% suspicious to be self-compiled libraries. Moreover, by adopting $DroidNative$ to LibRARIAN, it significantly improved the performance of LibRARIAN by 78.4%, which also outperforms the SOTA commercial tool BinaryAI by 82.9% on identifiable Native libraries. These results proved that the constructed $DroidNative$ can be a comprehensive indexing database for Android native libraries and benefit downstream tasks. We have open sourced the $DroidNative$ for public profit [6].

## II. Approach

In this section, we introduce a greedy approach to 1) studying existing native library importing mechanisms, 2) mining native library sources, and 3) collecting native libraries, with the aim of constructing a comprehensive Android native library dataset $DroidNative$, as presented in Figure 1.

### A. Study on Android Native Library Importing

We first investigate the possible ways in which Android native libraries can be imported into user projects, by inspecting the mapping of source codes and APKs that are well maintained in the F-Droid database.

**Data Preparation.** F-Droid provides Android APKs along with their hosted source code, which strongly supports the

§ Sen Chen is the corresponding author (senchen@nankai.edu.cn).



DOI 10.1109/ASEW67777.2025.00028

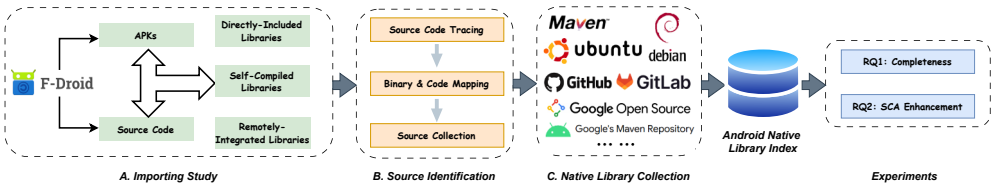


Fig. 1. Overview of the construction of $DroidNative$

subsequent Android native library source analysis. To this end, we first collect APKs that have integrated at least one Android native library, resulting in 2,979 APKs selected out of all 10,404 APKs that are available on F-Droid. After that, we further filter out APKs that are with valid source code links, resulting in 2,571 APKs, containing 17,165 Android native libraries, as the final dataset for this study.

**Mapping Analysis.** To exhaustively identify all possible ways to import Android native libraries, we randomly select 100 APKs for each of the first three authors to label the corresponding import approaches. After cross-validation among three authors, we identified three major types of imported Android native libraries that are packed into user APKs.

**1) Directly-Included Libraries.** The binaries of Android native libraries are directly included in the source code repositories under certain folders, such as *libs/* or *jniLibs/*.

**2) Self-Compiled Libraries.** The source code or links to source code are placed in the source code repositories, and they are (downloaded and) compiled to corresponding binaries using Android Native Development Kit (NDK) or CMake when building Android Apps.

**3) Remotely-Integrated Libraries.** The Android native libraries are downloaded by building tools like Gradle from remote repositories and packed into the APK files when building Android Apps.

### *B. Source Identification for Android Native Libraries*

Next, based on the importing approaches we identified in the study, we design an automated tool $SourceFinder$ to parse each APK file and its corresponding source code project and identify possible sources for the 17,165 Android native libraries. Specifically, for each Android APK, $SourceFinder$ first decompresses the APK file and locate all binary file (i.e., *.so* files). Then, $SourceFinder$ traverses its source code and search evidences for Android native library importing.

**1) For Directly-Included Libraries:** During the traversal, considering that some directories, such as *libs/* and *jniLibs/*, may contain multiple binary files of the same library for different system architecture support, $SourceFinder$ records these names and their locations only once for further mappings.

**2) For Self-Compiled Libraries:** $SourceFinder$ identifies all configuration files that are possible to download source code from external repositories, such as shell scripts that have executed git command. Apart from this, $SourceFinder$ also goes through *Android.mk* (for Android NDK) and *CMakeLists.txt* (for CMake) and identify the corresponding configurations. To this end, $SourceFinder$ collects the configured names and locations of corresponding source code throughtout the source code projects as records for further mappings.

**3) For Remotely-Integrated Libraries:** $SourceFinder$ identifies two folds of possible configurations in source code repositories. For Android native libraries that are introduced by build tools like Gradle, $SourceFinder$ inspects specific configuration file, such as *build.gradle* to identify the imported packages. Then, considering that after Gradle 7.0, Version Catelogs are introduced, $SourceFinder$ also identified the *libs.version.toml* file to trace the package names declared in *build.gradle* back to their original GAVs. After that, the identified package names and repository sources are all logged by $SourceFinder$ for further mappings.

**4) Source code & Binary Mapping:** After collecting all these clues in source code project that can indicate the integration of Android native libraries, $SourceFinder$ then maps these clues back to the binary files (i.e., *.so* files) in APKs to ensure that the source of all binary files are properly identified. 1) For Directly-Included Libraries, $SourceFinder$ directly matches them back to binaries in APK by hash. 2) For Self-Compiled Libraries, after collecting all source code from external links, $SourceFinder$ maps the source code locations to binary files by the defined names in configuration files. 3) For Remotely-Integrated Libraries, $SourceFinder$ downloads the corresponding packages from remote repositories, and decompresses them to retrieve the binary files inside, then compare with those binaries identified in APKs by names and hashes. After this, we can obtain the detailed mappings between binary files in APKs and the corresponding source code locations or remote addresses.

**Results.** We applied our tool to the 2,571 APKs. The experimental results show that, in total, 15,666 out of these 17,165 binaries in APKs (91.3%) are successfully traced to their sources with confirmed evidences. In detail, 96 (0.6%), 6,344 (37.0%), and 9,226 (53.7%) of them are directly-included, self-compiled, and remotely-integrated libraries, respectively.

For self-compiled libraries, although some of these libraries are from external source code repositories, and integrated after local compilation, it does not mean we can directly collect the corresponding binary files from these repositories. Moreover, considering that some Android native libraries are directly kept in source code repositories, we add these source code repositories as possible sources for further collection, such as GitHub, GitLab, and Gitdab.

For remotely-integrated libraries, we examined the distribution of their sources (Figure 2), covering 9 repositories. Google Maven and Maven Central are the top sources of Android native libraries. Jcenter and Bintray have since been

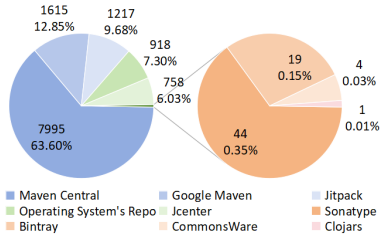


Fig. 2. Date Source Distribution

TABLE I
ANDROID NATIVE LIBRARIES DATASET DISTRIBUTION

| Data Source | #Lib. | #Ver. | Data Source | #Lib. | #Ver. |
|---|---|---|---|---|---|
| Debian | 32,261 | 120,993 | Google Maven | 69 | 4,872 |
| Ubuntu | 8,040 | 48,373 | Google Source | 617 | 2,480 |
| Maven Central | 18,358 | 112,849 | Git Repositories | 942 | 3,230 |

shut down and merged into Maven Central [7]. OS repositories like Ubuntu and Debian also contribute significantly. Jitpack, though not a central repository, is commonly used to integrate GitHub projects. Sonatype serves as an alternative to Maven. Other sources, include Bintray, CommonsWare, and Clojars, rarely provide common Android native libraries.

Based on these results, we further summarize the collected sources as comprehensively as possible. After filtering out duplicate sources and closing service sources, we collected a set of sources that maintain the mainstream Android native libraries, including Git-repositories (i.e., GitHub, GitLab, Gitdab, Google Source, etc.), OS repositories (i.e., Ubuntu and Debian, etc.), Maven-alike TPL repositories (i.e., Maven Central and Google Maven, etc.).

### C. Native Library Collection

After identifying these possible sources, we collect these Android native libraries correspondingly.

• **OS Package Repositories:** We mainly collect Android native libraries from Ubuntu and Debian repositories. We first recursively download all *.deb* files, then decompress and extract the native libraries by BinWalk. We further filter relevant files and record metadata, including package name, version, compiler architecture, and release time. This approach resulted in the collection of 48,373 Android native libraries from Ubuntu and 120,993 from Debian.

• **TPL Repositories:** This type of data sources mainly include Maven Central and Google Maven. We recursively download, extract, and filter native libraries from *.aar* packages and recording relevant metadata such as package name, version, and release timestamp. This process resulted in the collection of 112,849 Android native libraries from Maven Central and 4,872 from Google Maven.

• **Source Code Repositories:** To collect Android native libraries from Google Source, GitHub, GitLab, and other Git-based platforms, we used automated downloads, crawlers, and filtering, ultimately collecting 2,480 libraries from Google Source and 3,230 from Git repositories.

Overall, we collect a total of 60,287 native libraries with 292,797 versions from various sources and build a complete dataset $DroidNative$ of native libraries along with their data sources. The specific component and version distribution for each data source are presented in Table I.

## III. EXPERIMENTS

In this section, we evaluate the completeness and applicability by answering two research questions: **1) Completeness.** How complete is $DroidNative$ in terms of Android native libraries? **2) SCA Enhancement.** To what extent can $DroidNative$ enhance existing SOTA SCA tools in terms of identifying Android native libraries?

### A. *Completeness of* $DroidNative$

Given the cutting edge performance of LibRARIAN, we use it to verify the completeness of $DroidNative$. Considering the popularity and timeliness of the projects, we extracted Android native libraries from all apks published on the top-list of 24 categories on APKCombo within the past five years. In total, we downloaded 14,734 APKs from 477 Android projects and extracted 150,914 binaries (i.e., *.so* files) from them. To avoid the rarely used instance and focus on mainstream libraries, we retained only those found in more than 10 Android projects, specifically totaling 63,008 binaries.

**Result:** Among these 63,008 binaries, features were successfully extracted from 61,325 of them by LibRARIAN, denoted as *Real-world dataset*. Therefore, we compared these binaries against $DroidNative$ by LibRARIAN. The experimental results showed that 52,194 binaries, out of 61,325 (85.1%) can be successfully mapped to $DroidNative\_F$, which means these binaries are well covered in our $DroidNative$ dataset.

Subsequently, we conducted an in-depth analysis of all instances that fail to achieve successful matching:

• **Self-Compiled Android Native Libraries.** Inspired by the Self-Compiled libraries, we compared the hashes and filenames of untraceable Android native libraries with those of self-compiled ones. Out of the 9,131 untraceable Android native libraries, 3,039 exhibited identical filenames and SHA-256 hashes, while 3,349 shared identical filenames but differed in their SHA-256 hashes, indicating in total 6,388 (10.1%) libraries were likely introduced through self-compilation.

• **Incomplete Version Collection Due to Data Source Updates.** In certain instances, identification results of Android native libraries with the same filename from different versions of the same Android project were inconsistent. Given that Android apps usually show version continuity in using TPLs, we hypothesized these failures may stem from incomplete version collection rather than total lack of data source. This data loss may result from data source updates. Of the remaining 2,743 failed cases, 446 showed the above phenomenon.

### B. *Enhancement for SOTA Tools*

Given the large coverage of $DroidNative$, we further assess the extent to which $DroidNative$ enhances the performance of existing Android native library identification tools (LibRARIAN with $DroidNative$) with two selected baselines, the SOTA model (LibRARIAN with its public dataset) and leading commercial tool (BinaryAI).

To compare with Baseline 1, we applied the pre-constructed *Real-world dataset*. To compare with Baseline 2, due to the upload file size and API limitations, we randomly selected 1,000 Android native libraries from the *Real-world dataset*.

**LibRARIAN:** The experimental results demonstrated an increase from 4,121 libraries (6.7%) to 52,194 libraries (85.1%) beyond LibRARIAN's original dataset, and all libraries identified by LibRARIAN can be successfully identified by LibRARIAN with $DroidNative$, indicating that $DroidNative$ can significantly enhances the capability of LibRARIAN due to the larger volume and a broader range of data sources.

**BinaryAI:** According to our manual inspecting, BinaryAI can successfully identified 151 official links corresponding to the 1000 native libraries. In comparison, LibRARIAN with $DroidNative$ can successfully identify 980 of them with detailed original sources, achieving a traceability success rate improvement from 15.1% to 98.0%. Moreover, LibRARIAN with $DroidNative$ successfully identified 150 out of the 151 succeed cases of BinaryAI, and even for the missed one, LibRARIAN actually has identified its source but this result is rejected due to its own threshold of required similarity.

## IV. Limitations and Threats to Validity

**1) Sample Selection:** We analyzed APKs from popular apps on APKCombo(2019–2024), which may overlook older or less popular apps. However, since popular apps tend to be well-maintained and widely used, they provide a representative view of real-world dependency management. **2) Data Collection Constraints:** GitHub's API restrictions limited our ability to retrieve all Android native libraries. To mitigate this, we included libraries referenced in build.gradle files, ensuring that our dataset still captures the most commonly used dependencies.

## V. Related Work

Third-party libraries are a cornerstone of Android application development, prompting extensive research on the identification of third-party library versions. In early time, machine-learning based approaches [8], [9] are widely adopted. After that, clustering technique [10] are introduced. Subsequently, characteristics of code syntax and semantics [4], [5] are also used to enhance performance. However, the primary focus of these works remain on JVM-based libraries, with insufficient attention to Android native libraries. To the best of our knowledge, there are two papers most related to our scope. Liu et al. [11] is the only existing work that systematically discussed the major building process of Android Apps, while they focused more on the practices of configurations of mainstream building tools. Another work is conducted by Almanee et al. [12], who introduced LibRARIAN, specifically for identifying versions of native libraries. It employs $bin^2sim$, a novel similarity metric that leverages features extracted from library metadata and identifies strings in read-only sections to enable similarity-based matching. However, their feature dataset was constructed using only 200 Android apps containing 7.2K binaries, which limits its real-world applicability.

## VI. Conclusion

In this paper, we lay the foundation for studying native libraries in Android apps by constructing the first comprehensive indexing dataset, $DroidNative$, through the incorporation of a greedy strategy for searching, identification, and collection. $DroidNative$ is proved to achieve a high coverage (85.1%) of binaries in real-world apps and boosts SOTA SCA detection with at least a 78.4% improvement on detection rate.


## Acknowledgements

This work was partially supported by the National Natural Science Foundation of China (No. 62472309) and the National Research Foundation Singapore and the Cyber Security Agency under the National Cybersecurity R&D Programme (NCRP25-P04-TAICeN). Any opinions, findings and conclusions, or recommendations expressed in these materials are those of the author(s) and do not reflect the views of National Research Foundation, Singapore, Singapore.


## References


[1] M. Polino *et al.*, "Going native: Using a large-scale analysis of Android Apps to create a practical native-code sandboxing policy," in *The Network and Distributed System Security Symposium 2016*, 2016, pp. 1–15.

[2] F. Zhang, L. Fan, S. Chen, M. Cai, S. Xu, and L. Zhao, "Does the vulnerability threaten our projects? automated vulnerable API detection for third-party libraries," *IEEE Transactions on Software Engineering*, 2024.

[3] X. Zhan, L. Fan, T. Liu, S. Chen, L. Li, H. Wang, Y. Xu, X. Luo, and Y. Liu, "Automated third-party library detection for Android applications: Are we there yet?" in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 2020, pp. 919–930.

[4] X. Zhan, L. Fan, S. Chen, F. We, T. Liu, X. Luo, and Y. Liu, "ATVHUNTER: Reliable version detection of third-party libraries for vulnerability identification in Android applications," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1695–1707.

[5] Y. Wu, C. Sun, D. Zeng, G. Tan, S. Ma, and P. Wang, "{LibScan}: Towards more precise {Third-Party} library identification for Android applications," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 3385–3402.

[6] "Android native library," [Online; accessed 2025-03-14]. [Online]. Available: https://sites.google.com/view/hi-library

[7] adia, "Jcenter sunset on august 15th, 2024 — jfrog," 7 2024, [Online; accessed 2025-03-14]. [Online]. Available: https://jfrog.com/blog/jcenter-sunset/

[8] A. Narayanan, L. Chen, and C. K. Chan, "AdDetect: Automated detection of Android ad libraries using semantic analysis," in *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. IEEE, 2014, pp. 1–6.

[9] B. Liu, B. Liu, H. Jin, and R. Govindan, "Efficient privilege de-escalation for ad libraries in mobile Apps," in *Proceedings of the 13th annual international conference on mobile systems, applications, and services*, 2015, pp. 89–103.

[10] M. Backes, S. Bugiel, and E. Derr, "Reliable third-party library detection in Android and its security applications," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 356–367.

[11] P. Liu, L. Li, K. Liu, S. McIntosh, and J. Grundy, "Understanding the quality and evolution of Android App build systems," *Journal of Software: Evolution and Process*, vol. 36, no. 5, p. e2602, 2024.

[12] S. Almanee, A. Ünal, M. Payer, and J. Garcia, "Too quiet in the library: An empirical study of security updates in Android Apps' native code," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1347–1359.