Dynamic Security Analysis On Android

[1]Dr. R. Dinesh Kumar, [2]Akshaya Mulinti, [3]Niveditha Bitla
[1]Associate Professor, Department of CSE, Bhoj Reddy Engineering College for Women, India
[2,3]B. Tech Students, Department of CSE, Bhoj Reddy Engineering College for Women, India

ABSTRACT

Dynamic analysis is a method used to comprehensively comprehend the internal workings of a system during execution. Dynamic security analysis on Android entails real-time evaluation and proactive modification of an application's behavior, used for diverse functions such as network surveillance, system call tracking, and taint analysis. Recent years have seen substantial advancements in the field of dynamic analytic research. Nonetheless, to our knowledge, there is a deficiency of secondary studies that examine the innovative concepts and prevalent limits in contemporary security research. The primary objective of this study is to comprehend dynamic security analysis research pertaining to Android, to delineate the current level of knowledge, identify research deficiencies, and provide insights into the available literature in an organized and methodical manner. We provide a thorough literature study on dynamic security analysis pertaining to Android. The systematic review formulates a taxonomy, delineates a categorization framework, and examines the influence of sophisticated Android app testing tools on security solutions within software engineering and security research. The study's principal results focus on tool use, research aims, limitations, and trends. Instrumentation and network monitoring technologies are essential, with research objectives centered on application security, privacy, malware detection, and the automation of software testing. Recognized limits include code coverage restrictions, challenges in security analysis, suitability of application selection, and non-deterministic behavior.

1. INTRODUCTION

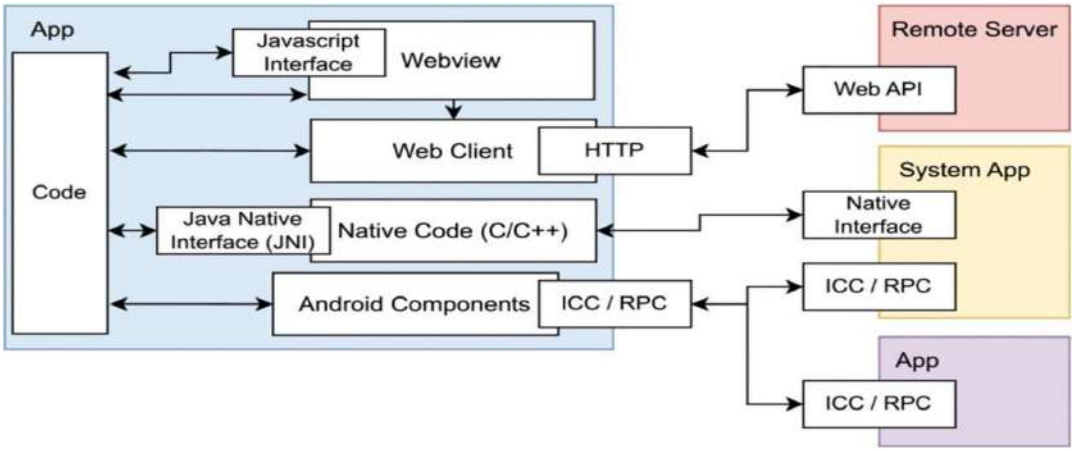
Dynamic analysis is a technique that is used to fully understand the internals of a system at runtime. On Android, dynamic security analysis involves real-time assessment and active adaptation of an app's behaviour, and is used for various tasks, including network monitoring, system-call tracing, and taint analysis. The research on dynamic analysis has made significant progress in the past years. However, to the best of our knowledge, there is a lack in secondary studies that analyse the novel ideas and common limitations of current security research. The main aim of this work is to understand dynamic security analysis research on Android to pre- sent the current state of knowledge, highlight research gaps, and provide insights into the existing body of work in a structured and systematic manner. We conduct a systematic literature review (SLR) on dynamic security analysis for Android.

The systematic review establishes a taxonomy, defines a classification scheme, and explores the impact of advanced Android app testing tools on security solutions in software engineering and security research. The study's key findings centre on tool usage, research objectives, con- straints, and trends. Instrumentation and network monitoring tools play a crucial role, with research goals focused on app security, privacy, malware detection, and software testing auto- mation. Identified limitations include code coverage constraints, security-related analysis ob- stacles, app selection adequacy, and

non-deterministic behaviour.

2-BACKGROUND ON TESTING ANDROID APPS

It is only logical that automated Android app testing plays a crucial role in most dynamic security analysis approaches as it is one of the fundamental building blocks for collecting data points. To fully grasp the capabilities of an Android app, it is necessary to interact with the app's Graphical User Interface (GUI) as well as with the APIs offered by the Android framework.



Android apps can include native code via the Java Native Interface (JNI) or JavaScript code via the webview interface Apps can directly communicate with companion apps to exchange data or interact with system services via Inter-Component-Communication (ICC) and it is as well possible for apps to use JNI to interact with native system services.

Android apps make use of a wide variety of interfaces and can share data via services, content providers, or broadcast listeners. Android app with possible communication partners to give an overview of frequently used interface. Apps can directly communicate with companion apps to exchange data or interact with system services via Inter-Component-Communication (ICC) and it is as well possible for apps to use JNI to interact with native system services.

The usage of these interfaces in combination with ICC often makes an in-depth analysis of Android apps challenging because these dependent components need to be analysed in addition to the app under test.

TESTING QUALITY METRICS

To evaluate and compare the performance of Android testing tools, two quality metrics have been primarily used in the past

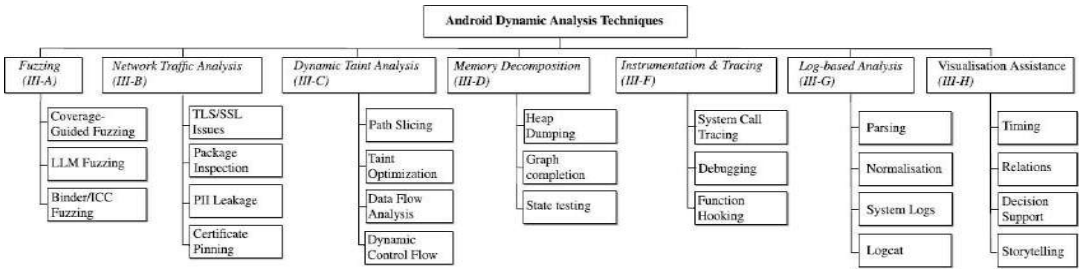
Code coverage : It is the primarily used metric to measure which parts of the code have been executed during testing. In addition to line coverage sometimes the class, activity, or method coverage are measured as well.

Fault detection : It is the other main metric applied when comparing testing tools. It is used to determine if a tool is capable of generating an input so that the Application under test (AUT) would crash or enter an unwanted state.

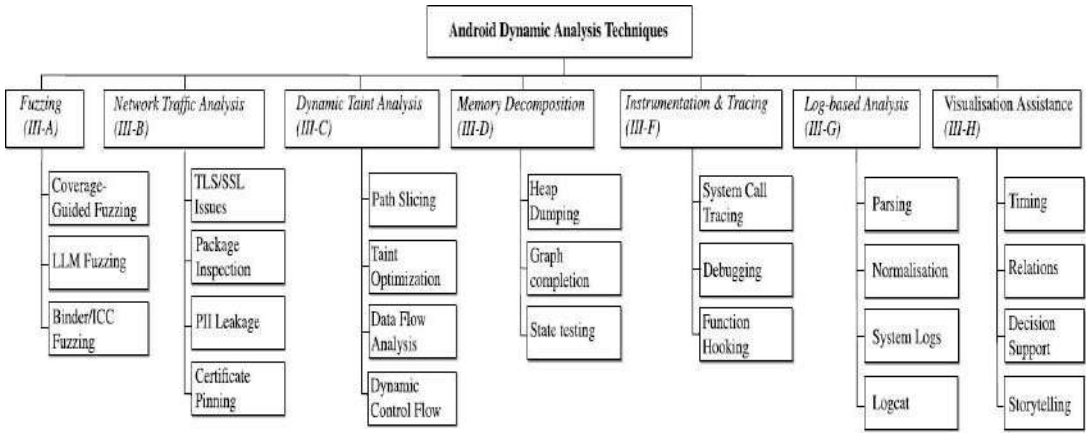
STATE-OF-THE-ART ANDROID APP TESTING TOOLS

Many dynamic analysis methods heavily rely on a

number of testing tools from industry and academia. These tools are often the basis for security researchers to test certain states of an application. Thus, we give a brief overview of common Android app testing tools and their limitations. The tools were chosen after a thorough review of publications on Android app testing, but the list is not comprehensive.



3-DYNAMIC SECURITY ANALYSIS TECHNIQUES



Fuzzing :

Fuzzing, or fuzz testing, is an automated software testing technique used in cybersecurity to discover vulnerabilities, bugs, and other issues in software systems. By feeding unexpected, malformed, or random data as input to a program, fuzzing aims to cause unexpected behavior, crashes, or security breaches, highlighting potential weaknesses that could be exploited by attackers . How Fuzzing Works

1. **Input Generation:** A fuzzing tool generates a wide range of test cases, which could include random strings, boundary values, or specially crafted payloads designed to exploit specific vulnerabilities.
2. **Input Delivery:** These inputs are fed into the target program, component, or interface (e.g., APIs, web services, file parsers).
3. **Observation:** The program's behaviour is monitored for anomalies, such as crashes, memory corruption, or unexpected outputs. Logging and debugging tools are often used to capture these events for analysis.

LOG-BASED ANALYSIS

Most applications and services log errors, warnings and informational events to a log facility. What exactly is logged is not standardised; however, it is

often information that helps diagnose and reproduce erroneous behaviour of an application or information that is needed for audit purposes, e.g. requests for access to protected resources. By default the access to app logs on Android is only granted to privilege processes and app logs should only be accessible when an app is in debug mode.

4-LITERATURE REVIEW METHODOLOGY

Step 1 : we defined the following exclusion and selection criteria to limit our study to peer- reviewed publications:

Language: Papers must be written in English.

Time: Only publications published between 2017 to 2023

Type of publication: Only peer-reviewed publications that were published in computer science conferences or journals. No patents, books, or demo publications.

Topic: Publications pertinent to our subject matter, which revolves around Android dynamic security analysis. In this context, "relevance" denotes the degree to which publications align with the specific focus of our study. Therefore, we will only include publications that directly contribute to or address aspects relevant to Android dynamic security analysis.

Step 2 : To identify publications pertinent to our topic, we conducted iterative test searches on Google Scholar. Through multiple iterations, we identified search terms that consistently produced relevant outcomes, as evidenced by the relevance of the first 100 search results.

Step 3 : We conducted keyword searches on Google Scholar and exported the results to machine-readable files. For this purpose we use the tool Publish or Perish . Previous studies have demonstrated that the usage of Google Scholar is sufficient to find relevant publications from top computer science conferences and journals. In addition, using Google Scholar prevents a bias towards a specific publisher and identifies publications based on a scoring system.

Step 4 : We apply our selection criteria to filter out irrelevant publications by year, language, and article form (only peer reviewed articles, no books, etc.) to reduce the number of search results. Following the application of filter criteria, we reviewed the top 1,000 search results, scanning for potentially relevant publications based on their titles and abstracts. This selection of the initial 1,000 results aimed to balance comprehensiveness with relevance, taking into account the practical consideration of managing a sizeable dataset for thorough analysis. Additionally, this number was chosen to ensure a broad exploration of the literature landscape while maintaining feasibility in terms of manual screening efforts. As a result, we identified 177 potential publications.

Step 5 : We review the methodology and results section of the potential publications to identify articles on Android dynamic security analysis. In case, we find a fitting article, we conduct an in-depth review and add it to the SLR.

Step 6 : The authors delve into the scope, quality, and relevance of the publications. This is done to prevent any potential bias from a single author's perspective. If a situation arises where most of the authors identify an inconsistency in relation to a particular publication, a decision is reached regarding its inclusion or exclusion. This determination is based on a majority vote among the authors.

5-TAXONOMY OFANDROID SECURITY RESEARCH

By analysing the publications in our dataset, three primarynresearch domains can be identified in the

realm of dynamic security analysis of Android Apps: (i) App Security, Privacy and Compliance, (ii) Malware, and (iii) OS & Framework. We use these three domains to group research publications with similar use cases and objectives.

It shows the three main research domains and their connections to related security topics. It should be noted that many studies have overlapping use cases or objectives and may fit into more than one category. However, we assigned publications to the most fitting category based on their main use-case or objective.

- App security, privacy and compliance (ASPC) encompasses publications that are primarily centred around the development of innovative techniques for conducting security testing on Android applications. Furthermore, it includes publications that explore specific categories of Android apps, such as, for instance, in-depth analyses of vulnerabilities in mobile banking apps.
- Malware includes publications on methods for detecting fraudulent or malicious applications and how to extract or collect features from apps for classification tasks that can be used for detection techniques.
- OS & Framework (OSF) is mainly about publications that study the Android framework and its components from a security perspective. This also includes research about the security of the Android operating system.

  The data source used by the publications, dynamic analysis techniques, tools used, and other factors are not explicitly discussed in the text but are recorded in the tables for the respective subsection.

  At the end of each section, we summarise the main takeaways and discuss the most important limitations.

  Android security research can be categorized into key domains, each addressing different aspects of securing the platform:

- Application Security: Focuses on detecting vulnerabilities in Android apps, including malware analysis, code obfuscation, and secure coding practices.
- Platform Security: Examines the Android OS, including kernel security, permission models, and system updates.

6-APP SECURITY, PRIVACY AND COMPLIANCE RESEARCH

The App Security, Privacy and Compliance domain. We analysed these publications in terms of the used dynamic analysis technique, testing methodology, data sources, as well as number of tested apps.

NETWORK ANALYSIS

There have been several methods described to test the used network protocols and to identify implementation errors in common security mechanisms such as TLS. In this section we discuss studies with a strong focus on network analysis and summarise their objectives and findings.

There have been several methods described to test the used network protocols and to identify implementation errors in common security mechanisms such as TLS.

- App Security: Identifies vulnerabilities such as insecure APIs, improper data storage, and weak encryption, ensuring robust defenses against threats.
- Privacy Protection: Examines practices to safeguard user information, including permissions management, data anonymization, and preventing unauthorized data sharing.
- Compliance: Ensures apps meet legal and industry regulations like GDPR, HIPAA, and CCPA, addressing requirements for data collection, storage, and user consent.

OPEN PORTS

Wu et al. conducted an in-depth study of open ports on Android. 3,293 users in 136 countries worldwide contributed to their research by allowing the researchers to continuously monitor open ports on their smartphones. By installing an Android network monitoring app the researchers were able to detect and analyse open ports for vulnerabilities and identified five vulnerable patterns for open ports. Consequently, the researchers found vulnerabilities in several popular Android apps (e.g., Instagram, Samsung Gear, Skype) due to open ports. Moreover, Wu et al. found out that many of the open ports are solely from SDKs integrated into these apps, which raises the concern that the app developers are unaware of these open ports.

One of the main use cases for static and dynamic analysis is the examination and detection of fraudulent apps (mainly referred to as malware). The definition of what is considered to be malware is often blur. Solely analysing the capabilities of a program is not sufficient to determine if an application is considered malicious or not. As malware has many facets and uses similar or even the same capabilities as genuine programs, it is often hard to categorising malware. Consequently, different classification definitions for malware exist. These standards have their own definitions on how to categorise and identify malware a specific malware type we refer mainly to the definitions by the Google Play Protect schema for Potentially Harmful Applications (PHAs).

7-MALWARE RESEARCH

Tool / Publication	I	N	V	L	D	M	F	H	A	SC	DS	NA	ST	TE	TD
StaDart [70]	✓				✓			✓	✓	✓	Google App Store, Drebin	1,000/1,000	Monkey, Droidbot	Physical	Dynamic code loading Reflection
EspyDroid+ [71]	✓				✓			✓	✓	✓	F-Droid, Ripple, Virustotal, MalGenome	660	Robotium (custom)	Emulator	Dynamic code loading Reflection
DL-Droid [72]				✓				✓	✓		McAfee Labs	19,620/11,505	Monkey, DroidBot, DynaLog	Physical	DL input generation
D'Angelo et al. [73]			✓	✓					✓		Contagion, MalGenome, VirusShare, Playdrone, Google Play Store	25,500/23,200	MSF, Matlab	Unknown	API-images and autoencoders
VizMal [74]	✓		✓						✓		Drebin, Google Play Store	250/250	Monkeyrunner	Physical	Visualisation of traces
Droidcat [65]	✓			✓					✓	✓	AndroZoo, VirusShare, Google Play Store, Drebin, MalGenome	17,365/16,978	Monkey, DroidFax, VirusTotal, Soot	Emulator	ICC and method call classification
SAMADroid [66]	✓			✓				✓	✓		Drebin, MalGenome	123,453/5560	AAPT backsmali, MonkeyRunner	Physical	Multiple folds
EnDroid [68]	✓			✓					✓		Google Play Store, Drebin, AndroZoo	8,806/5,213 5,000/5,000	Droidbox, MonkeyRunner	Emulator	Ensemble learning
DAMBA [75]				✓		✓		✓	✓		Genome, Contagion, Baidu App Store	1,000/564	-	Emulator	ORGB
Bernardi et al. [69]	✓			✓					✓		Genoma, Drebin, Google Play	200/1,801	Droidchameleon, ADAM	Emulator	Process mining
Cai et al. [76]	✓			✓					✓		AndroZoo, VirusShare, Google Play Store	15,451/15,183	Money, Droidfax	Emulator	Run-time evolution
Bhat et al. [77]	✓			✓				✓	✓		CICMalDroid 2020	1,795/9,803	CopperDroid	Emulator	Ensemble learning

TRACE-BASED DETECTION

Malware detection based on system call logs is a form of behavioural model analysis. The key idea behind such detection mechanisms is that malicious behaviour is represented by a sequence of system calls. It is based on the assumption that genuine and malicious apps have distinguishable system call sequences. Identifying the potential malicious

sequences is subject to research and many classification systems and models have been proposed.

8-ANDROID OS AND FRAMEWORK RESEARCH

Analysis techniques that focus on examining the Android OS and parts of the framework for security purposes with dynamic analysis techniques.

Tool / Publication	I	N	V	L	D	M	F	H	A	SC	DS	NA	ST	TE	TD
CoMe [78]	✓			✓					✓		Zimperium	15,000	MEDA MSNM	Emulator	Android media server and media files
Dynamo [85]	✓			✓			✓		✓	✓	Android images	-	Arcade ARF Pscout Kratos	Mixed (Physical + Emulator)	Android framework
PREV [80]	✓							✓	✓	✓	Google Play Store	1,258 595	Weka IceTA Covert	Emulator	Android framework (permissions)
Fans [81]				✓		✓	✓	✓	✓	✓	Android images	-	-	Physical Device	Android framework (native services)
Partemu [82]				✓		✓	✓		✓		TEE	-	Panda QEMU LLVM TriforceAFL	Emulator	Trusted Execution Environments
Sifter [79]	✓						✓		✓	✓	Public CVEs	-	-	Physical Device	Android Kernel
EASIER [83]							✓		✓		62 drivers	-	-	Evasion Kernel	Android Kernel (drivers)
DaVinci [84]	✓	✓							✓		Hardened apps	14	RootBeer	Mixed (Physical + Emulator)	App detection and evasion techniques
Teezz [87]	✓					✓	✓	✓	✓	✓	TEE from devices	-	adb vendor test suite	Physical	Trusted Execution Environments
U2-I2 [61]	✓							✓	✓	✓	Android images (popular devices)	-	adb	Physical	User-unresettable Identifiers
Gopper [86]	✓								✓	✓	-	-	-	Unknown	Detecting Frida

FUZZING OS AND FRAMEWORK COMPONENTS

Analysing the framework's permission is a challenging task as there are several hurdles to overcome.

- Permissions frequently change from one Android version to another, making it elaborate to keep track of new permissions as there exists over 600 permissions by default.
- The framework itself is a large code base which makes it challenging to map the guarded functions to the specific permissions and to verify that the guarded function cannot be reached without having the correct permissions at runtime.
- Smartphone vendors customise the Android framework and integrate custom permissions into the framework which makes it challenging to have generic tooling.

Generating tests to detect permission re-delegation vulnerabilities is challenging as the Android framework is too large to be instrumented with standard tools. Therefore, several researchers have proposed solutions for test generation and fuzzing.

9-CONCLUSION

In this systematic literature review (SLR), we conducted a thorough exploration of dynamic analysis in Android security research, illuminating significant trends and key aspects within the field. We give an overview of the applied analysis techniques in the field and show which techniques are frequently combined. By meticulously examining 43 carefully selected publications from diverse venues, we identified and analysed innovative methodologies, tools, and ideas related to dynamic analysis for security purposes. The development of a taxonomy yielded three primary research domains, providing a structured framework for comprehending the dynamic analysis landscape in Android security.

REFERENCES

1. Proton AG. (2023). Complete Guide to GDPR

Compliance. MISC. Accessed: Apr. 26, 2023. [Online]. Available: https://gdpr.eu/

2. PState of California Department of Justice. (2023). California Consumer Privacy Act (CCPA). MISC. Accessed: Apr. 26, 2023. [Online].
3. S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, ‘‘FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps,’’ ACM SIGPLAN Notices, vol. 49, no. 6, pp. 259–269, 2014.
4. L. Wartschinski, Y. Noller, T. Vogel, T. Kehrer, and L. Grunske, ‘‘VUDENC: Vulnerability detection with deep learning on a natural codebase for Python,’’ Inf. Softw. Technol., vol. 144, Apr. 2022, Art. no. 106809.
5. A. Lyons, J. Gamba, A. Shawaga, J. Reardon, J. Tapiador, S. Egelman, and N. Vallina- Rodriguez, ‘‘Log: It’s big, it’s heavy, it’s filled with personal data! Measuring the logging of sensitive information in the Android ecosystem,’’ in Proc. Usenix Secur. Symp., 2023, pp. 2115–2132.