

Hybrid Analysis Approach for Detecting Mobile Security Threats

Yusfrizal¹, Mutiara Sovina², Faisal Amir Harahap³, Ivi Lazuly⁴


¹Department of Information Management, Gihon Polytechnic, Indonesia

^{2,3,4}Department of Informatics, Potensi Utama University, Indonesia

ABSTRACT

As technology continues to advance rapidly, smartphones are becoming increasingly powerful, drawing a large number of users with innovative features provided by mobile operating systems like Android. However, the security vulnerabilities of these systems make Android devices frequent targets for hackers and cyber criminals. As a result, research on effective and efficient mobile threat analysis has become a critical topic in the field of cyber security, employing methods such as static and dynamic analysis. This paper proposes a hybrid approach that combines static and dynamic analysis to detect security threats and attacks in mobile applications. The proposed method integrates data states and software execution along critical test paths. Initially, static analysis is used to identify potential attack paths based on Android APIs and existing attack patterns. This is followed by dynamic analysis, which executes the program along these paths within a focused scope to determine the likelihood of an attack by comparing detected paths with known attack patterns. In the runtime phase of dynamic analysis, the approach reports types of attack scenarios related to confidential data leakage, such as web browser cookies, while ensuring no actual critical or protected data on mobile devices is accessed.

Keyword: hacker; cybercriminal; cyber security.

 This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

Corresponding Author:

Yusfrizal,
Department of Information Management
Gihon Polytechnic
Jl. Dalil Tani No.48, Tomuan, Siantar, Indonesia.
Email : yusfrizal80@gmail.com

Article history:

Received Oct 19, 2024
Revised Oct 23, 2024
Accepted Oct 30, 2024

1. INTRODUCTION

The widespread use of smartphones has also made them a prime target for hackers. In recent years, there has been a notable rise in malware specifically targeting the Android operating system. Android OS is often labeled as the "worst platform for malware" due to several factors (Ahvanooy et al., 2020). Firstly, the Android SDK is freely accessible to everyone, enabling hackers to invest time in creating and distributing malicious software. Secondly, the application validation process may not be robust enough to detect vulnerabilities inherent to the Java programming language. As a result, malware can be introduced either intentionally or unintentionally. Most of these malicious programs are sophisticated, multi-functional Trojans designed to intercept sensitive information stored on users' devices (Rutherford & Wu, 2023).

Android smartphones are particularly attractive to attackers due to their dual role as tools for increased mobility and as repositories for sensitive information. The prevalence of malicious applications in the Android ecosystem stems from weaknesses in Android information security (Acharya et al., 2022). The CIA triad—Confidentiality, Integrity, and Availability—offers a framework to understand these security issues. Confidentiality issues arise when sensitive information or resources are exposed, often due to insecure programming practices like privilege escalation, which grants unnecessary permissions to applications. Such vulnerabilities may unintentionally leak data or intentionally open backdoors for spyware, allowing theft of private information such as contact lists and SMS (Syed et al., n.d.).

The integrity principle focuses on ensuring that data remains unaltered and protected from interception and modification, especially during transmission. For instance, preventing Man-in-the-Middle attacks is a key concern under this principle. Availability ensures that information is accessible without disruption or intentional delays. However, maintaining all three principles simultaneously is challenging, and attackers often exploit these gaps, leading to information leaks (Mitsarakis, 2023). To

address these threats, two primary techniques are used to detect malicious applications: static analysis and dynamic analysis.

Static Analysis is a white-box method that examines an application's source code to identify vulnerabilities, malicious code, and security threats without executing the program. This involves lexical and semantic analysis of source code, followed by constructing an Abstract Semantic Tree (AST) for further examination. Common static analysis techniques include lexical analysis, safety rule enforcement, type inference, constraint analysis, and data flow analysis (Melo et al., 2020). However, static analysis alone cannot uncover all threats or provide detailed insights into potential attack scenarios without program execution.

Dynamic Analysis is unlike static analysis; dynamic analysis operates without access to source code. It tests applications in a runtime sandbox environment, running test cases to observe behavior and identify subtle vulnerabilities or partial attack paths. Although dynamic analysis can reveal runtime issues, it is resource-intensive and impractical for evaluating entire programs exhaustively. Instead, it is often tailored to specific scenarios (Melo et al., 2020).

Each method has its strengths and limitations, making a combination of both approaches more effective for comprehensive security analysis.

2. RELATED WORK

The rising prevalence of malware necessitates the development of tools capable of automatically detecting malicious applications. Malware refers to software intentionally designed to harm systems or perform undesirable actions (Narwal et al., 2019). Various techniques have been proposed to differentiate malware from legitimate software, with permission management and information leakage being primary concerns in more than 60% of related research papers. Two widely used methods for detecting Android malware targeting users' private information are taint analyzers and signature-based detectors. Taint analyzers trace the flow of sensitive data to detect potential leaks, while signature-based detectors identify malware based on predefined patterns or characteristics. Additionally, different techniques have been developed to address threats against the CIA principles (Confidentiality, Integrity, and Availability), ensuring robust security measures to protect sensitive information and maintain system functionality (Mughal, 2020).

For information confidentiality, also known as information leakage, TaintDroid is one of the widely used tools that employs taint analysis to track the flow of sensitive data through third-party applications. Its primary objective is to detect if sensitive data is being leaked from the system via untrusted applications. Another tool that applies taint analysis is FlowDroid. FlowDroid is a context-, flow-, field-, object-sensitive, and lifecycle-aware static taint analysis tool designed specifically for Android applications (Zhang et al., 2021). It monitors the path from data sources to sinks to detect malicious behaviors.

The first Android analysis tool, ScanDroid, developed by Adam et al., incrementally checks applications as they are installed by extracting security specifications from application manifests and verifying that data flows according to the defined specifications. Additionally, tools like Apposcopy propose a semantics-based approach for identifying various classes of Android malware, focusing on private user information (Alswaina & Elleithy, 2020). It combines pattern-based detection with signature matching algorithms to detect Android malware effectively.

For information integrity, techniques are designed to examine whether private data remains unchanged as it flows during execution. Integrity ensures that information arrives at its destination exactly as it was at the source, without unauthorized modifications. Since additional permissions are often required for extra operations, static analysis techniques such as DroidRay check for privilege escalation to detect malware (Alzubaidi, 2021).

For information availability, techniques focus on ensuring the accessibility of data, which can be achieved by constructing control flow graphs to propagate intents from APIs to top-level functions. A variety of techniques have been developed to detect threats against the CIA principles. In this paper, we focus on information confidentiality attacks and the corresponding strategies used to mitigate these risks.

A. Vulnerability in Android Platform

Mobile application security has become one of the most pressing issues in the world of cybersecurity, becoming increasingly critical with the rise in mobile applications. Smartphones, acting as personal information repositories that store data such as phone numbers, SMS, and even financial details, naturally attract the attention of attackers. According to guidelines from Carnegie Mellon University, secure programming practices on the Android platform aim to address noncompliant coding practices that could lead to vulnerabilities being exploited. In this section, we will first analyze vulnerabilities in the Android platform and then provide common examples found in real-world applications to highlight the patterns used by malware attackers (Yadav et al., 2022).

Android applications are built using four primary components: Activities, Services, Content Providers, and Broadcast Receivers. Each component serves different purposes within Android apps. Activities provide a user interface for interaction, Services run in the background to perform long-term tasks, Content Providers allow manipulation of shared personal data, and Broadcast Receivers respond to system-wide broadcasts (Cai et al., 2020). These components can communicate with each other through Intents, which are categorized as implicit or explicit intents, depending on how the intent is declared. Intents act like messengers, carrying messages between components, making them potential points of vulnerability that could lead to information leakage, as previously noted in DRD.

Content Providers are designed to manage and share sets of app data across different applications. However, this makes them a common target for malware, which often exploits them to access personal information when appropriate permissions are granted. By default, app data is private to the app, but Content Providers serve as an interface for sharing data between apps. They interact with databases using methods such as `insert()`, `query()`, `update()`, and `delete()`, utilizing URI starting with "content://". Improper implementation of Content Providers can lead to vulnerabilities, such as SQL injection and path traversal attacks.

B. WebView Vulnerability

More and more Android applications use WebView to display web content and enable user interactions. While WebView provides significant convenience, it also introduces vulnerabilities that can be exploited by attackers. This is due to how WebView is implemented. When an HTML page is loaded in WebView, the application generates two folders: one for data cache and another for browsing history. The cache stores URLs, while cookies are saved in `webview.db`, as shown in Figure 1. To view the structure of the database, it's possible to export `webview.db` from the device to the local file system.

The code below demonstrates malware that leverages Web View in Android applications to display web pages to users, where cookies are stored in `webview.db` within the application package. Attackers can query this database to retrieve sensitive data and transmit it using the necessary permissions. The structure of the tables within `webview.db` is depicted in Figure 1. User browsing history records are categorized and stored in the database. While this data can be useful for application providers, such as performing shopping habit analysis, it can also be exploited by attackers.

In this example, the malware functions as a browser, allowing users to surf the internet, during which it collects cookies stored by Web View. Once it has access to the browsing history, the malware sends out the collected data using a standard HTTP POST request.

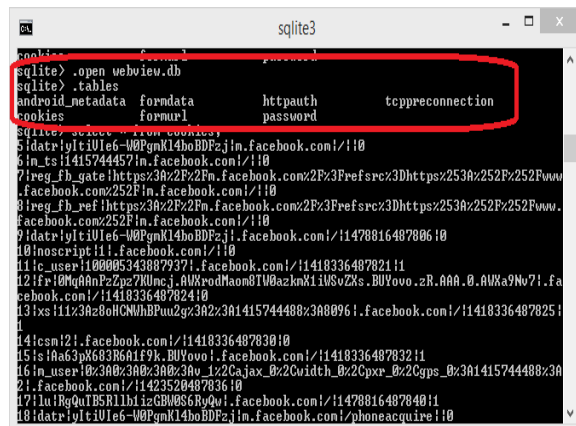


Fig.1 webview.db

In this attack scenario, attackers target the browsing history stored in the application database. Due to the use of Web View, the history records and cookies are saved within the application package, allowing malicious code to be embedded to extract data from the database. The codes provided in the example demonstrate at least two methods for spying on cookies. Attackers can focus on the cookie history stored in the application database, as well as intercept the current session to steal sensitive information.

The only permission required is Internet, which is commonly used even in legitimate Android applications. In this case, it becomes challenging to detect malware simply by examining the app's manifest. However, dynamic analysis can reveal the data flow path, showing how data is transferred from the storage location to an untrusted website, raising suspicion.

Get Cookie History

```
// usage of Web View in application
webView = (Web View)findViewById(R.id.webView);
//get cookies history SQLiteDatabase database =
SQLiteDatabase.openDatabase(getDatabasePath("webvie w.db").getAbsolutePath(), null,
SQLiteDatabase.OPEN_READONLY);
Cursor cursor = database.rawQuery("select * from cookies", new String[] {});
StringBuilder sb = new StringBuilder();
if(cursor.moveToFirst()){
do{
sb.append(cursor.getString(cursor.getColumnIndex("name")) + "=" +
cursor.getString(cursor.getColumnIndex("value")) + ";");
} while(cursor.moveToNext());
}
//HttpPost to send out sensitive data HttpClient client = new DefaultHttpClient(); HttpPost post = new
HttpPost("http://ip:port/Mytest/cookie/stealinginfo.php");
post.setEntity(sb);
HttpResponse response = client.execute(post);
```

Get Current Session Cookie

```
// usage of Web View in application webView = (Web View)findViewById(R.id.webView);
webView.addJavaScriptInterface(new MyJavaScriptInterface, "MyInterface");
//get cookies of current session via JavaScript String js =
"document.forms[0].onsubmit=function(event){var email =
document.getElementById("email").value;var pwd =
document.getElementById("pass").value;window.M
yInterface.saveInfo(email,pwd);window.MyInterface.s aveCookie(document.cookie);return true;}";
//execute JavaScript to get current session cookie view.loadUrl("javascript:"+js);
//send out via HTTP POST request
```

Use Permission

```
<uses-permission android:name="android.permission.INTERNET"/>
```

As outlined in the analysis, the general methods used to gather personal data—whether stored on devices or generated by applications—follow two main phases. First, they request the necessary permissions in the manifest file, which grant access to specific data and allow the app to send that data externally. Second, they exploit the APIs provided by the Android SDK to extract and communicate data.

3. RESULTS AND DISCUSSION

Among the vast array of malware, some pose direct harm to devices, while others focus on eavesdropping on personal data. These malicious applications can take on various forms—some

appear as standalone apps, while others disguise themselves as malicious versions that exploit vulnerabilities in existing applications. To differentiate malware from legitimate software, it is crucial to first analyze the patterns of different types of applications, as summarized in Table 1.

A. Opinion Survey of Workshop Participants

A survey was administered to participants at the conclusion of the workshop to assess its effectiveness and gather their feedback. The overall satisfaction rating averaged 3.93 out of 5, with 5 representing the highest level of satisfaction. Table 1 provides the average ratings for various aspects of the workshop, including the effectiveness of presentations and lab sessions, as well as the relevance of the information to the participants' teaching and research. These ratings were also measured on a 5-point scale, where 5 indicates maximum effectiveness and usefulness.

Table 1. Patterns to Send Out Data

Ways to send out data	Patterns	Permissions
By Internet	S1 create HttpClient	INTERNET
	S2 make POST request to the given URL	
	S3 build data	
	S4 set HttpPost entity	
	S5 execute POST request to the given URL	
By SmsManager	S1 get SmsManager object	SEND_SMS
	S2 send text message to specified mobile number	
By builtin Intent	S1 use ACTION_VIEW action to launch SMS	No
	S2 set Intent data as "smsto"	
	S3 set Intent type as "vnd.androiddir/mms-sms"	
	S4 set phone number and SMS body	
	S5 start the Intent to send SMS	
By Bluetooth	S1 get the BluetoothAdapter	BLUETOOTH
	S2 enable Bluetooth	
	S3 find paired devices	
	S4 connect as client	
	S5 send data to server	
By Socket	S1 set up socket connection by specifying server IP and port number	INTERNET WIFI
	S2 connect as client	
	S3 send data to server	

There are numerous methods, beyond the patterns mentioned above, that can be used to exfiltrate personal data from users' devices. The most common methods involve the use of the Internet or SMS. By leveraging the appropriate permissions and the capabilities of the Android SDK API, attackers can consistently find ways to eavesdrop on sensitive information. The general process typically follows these steps: 1) Determine the method of data extraction and request the necessary permissions. 2) Prepare the targeted data for exfiltration. 3) Utilize API calls to transmit the data to the attackers.

This paper introduces a hybrid method that combines the strengths of static and dynamic analysis to examine the data flow paths of suspicious applications. The method begins with APK files as inputs, representing the applications to be analyzed. It is composed of two main phases: 1) Static Analysis Phase : During this phase, the APK files are disassembled to extract the `AndroidManifest.xml`. The configuration files are meticulously analyzed to gather basic application information, such as the declared permissions and core components utilized. This information is crucial for constructing a pattern library to facilitate further analysis. 2) Dynamic Analysis Phase (Sandbox Testing): In this phase, the application is executed in a controlled environment to track the flow of confidential information through the system. The features identified during the static analysis phase serve as input for this dynamic testing phase.

Malicious applications are often disguised as legitimate ones using various patterns. Another research highlights that malicious apps are frequently found in the game category of the official Android Google Play Store. Their analysis of the top 80 legitimate game apps was conducted to uncover differences in system event patterns between normal and malicious applications.

Generally, normal applications only request permissions necessary for their functionality, unlike malicious applications that often request excessive permissions. For instance, a game application requesting permissions such as `READ_SMS` or `SEND_SMS` should raise suspicion, whereas permissions like `LOCATION` or `INTERNET` might be more justifiable. One common attack

pattern in malware is privilege escalation, where additional permissions are exploited to perform malicious actions. These static patterns can be effectively detected through static analysis.

In the first phase of the hybrid analysis method, static analysis is utilized to gather essential information about the application by disassembling its APK file. The primary focus is on analyzing the `AndroidManifest.xml` file to extract critical details, such as declared permissions, the usage of core components, and the launching activity. This foundational information serves as the basis for further analysis.

Typically, normal applications consume personal data retrieved from the device for legitimate purposes, whereas malware seeks to steal confidential information for malicious use. To determine whether data exits the system, a symbolic execution technique is employed to trace the application's internal logic. This approach is a form of dynamic analysis, also known as sandbox testing. In the second phase of the proposed hybrid method, features of the source code are extracted and replaced with utility functions to monitor the data flow path, enabling the detection of suspicious activity.

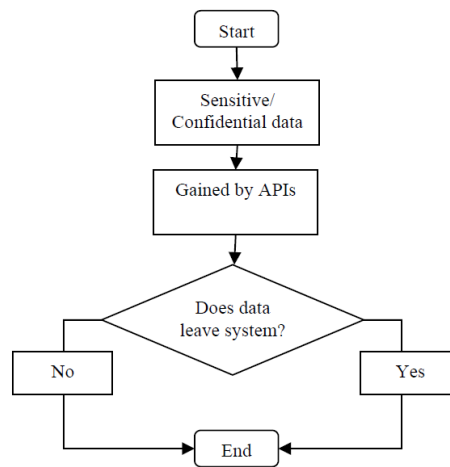


Fig. 2 Data Flow Path in Normal Apps vs. Data Flow Path in Malware

The analysis is divided into two primary phases. Phase One: This involves static analysis, where the Apktool is utilized to disassemble the packaged APK file. The input for this phase is the APK file, which contains the manifest file, dex file, and all required resources. Apktool extracts these components from the APK file. The main focus in this phase is the analysis of the manifest file. For instance, as described in section three, the manifest file indicates that only the INTERNET permission is granted, along with identifying the package information and launching activity. This extracted information is crucial for identifying sensitive API calls, which are further analyzed in the second phase. A custom tool previously developed is employed at this stage for tasks such as reading, writing, and searching files. This tool helps pinpoint the location where data exits the system and inserts code fragments to trace API calls. Phase Two: Dynamic analysis is conducted in this phase. The application, modified in the previous step, is executed in a controlled environment. This method is referred to as symbolic execution because it simulates the application's behavior to trace the API calling path without performing any actual malicious actions. This phase focuses on understanding the application's data flow and identifying potential points of vulnerability or data leakage.

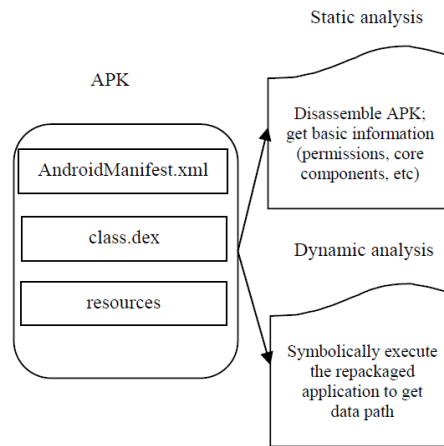


Fig. 3 Hybrid Analysis

```

StackTraceElement indext i=1,threadID=1,threadName=main,fileName=Thread.java,className=java.lang.Thread,methodName=getStackTrace,linenumber=788
-----
StackTraceElement indext i=2,threadID=1,threadName=main,fileName=SMMainActivity.java,className=com.wyou.cookiespyv3.SMainActivity$2,methodName=execute,linenumber=102
-----
StackTraceElement indext i=3,threadID=1,threadName=main,fileName=SMMainActivity.java,className=com.wyou.cookiespyv3.SMainActivity,methodName=queryDatabase,linenumber=219
-----
StackTraceElement indext i=4,threadID=1,threadName=main,fileName=SMMainActivity.java,className=com.wyou.cookiespyv3.SMainActivity,methodName=prepareCookie,linenumber=197
-----
StackTraceElement indext i=5,threadID=1,threadName=main,fileName=SMMainActivity.java,className=com.wyou.cookiespyv3.SMainActivity$1,methodName=onPageFinished,linenumber=86
-----
StackTraceElement indext i=6,threadID=1,threadName=main,fileName=CallbackProxy.java,className=android.webkit.CallbackProxy,methodName=handleMessage,linenumber=274
-----
StackTraceElement indext i=7,threadID=1,threadName=main,fileName=Ha
    
```

Fig. 6 APIs calling report

The APIs HttpPost and DefaultHttpClient are required in this context to send data to a malicious server. In our approach, the data exit point is substituted with custom-prepared code to monitor the sequence of method calls. In the following tables, the environment setup and technologies used are listed.

Table 2. Environment Setup

	PC Settings	IDE
Basic Environment	64-bit Operating System Intel (R) Core (TM) i3 4.00 GB memory	Eclipse Android ADT Bundle

Table 3. Patterns to Send Out Data

	Patterns	Permissions
Static Analysis	Apktool-2.0.0	Reverse engineering for application disassembly
Static Analysis	String Analysis Tool	A self implemented Java application to read file and string pattern
Dynamic Analysis	Auto-Sign Tool	Sign the repackaged application

The generated report provides a clear printout of the stack status maintained by the compiler during the application's execution. This allows us to determine whether the application in question is malicious or not.

4. CONCLUSION

Android applications have rapidly developed and gained widespread popularity, largely due to the open-source nature of the Android SDK. However, this openness also enables malicious applications to infiltrate the Android market, often disguised as legitimate apps, impacting users' daily lives. This security vulnerability puts sensitive data and critical information stored on Android devices at constant risk of attack by malicious software, highlighting the need for effective methods to differentiate malware from regular software. This paper outlines secure programming guidelines and introduces a hybrid data path tracing approach to identify malware by analyzing noncompliant coding practices and gathering attack patterns that exploit vulnerabilities in Android APIs. Future work will focus on refining this method into an integrated malware analysis system, adapting it to accommodate various mobile app formats, enhancing the precision of malware detection, and fostering a safer environment for mobile users.

REFERENCES

- Acharya, S., Rawat, U., & Bhatnagar, R. (2022). [Retracted] A Comprehensive Review of Android Security: Threats, Vulnerabilities, Malware Detection, and Analysis. *Security and Communication Networks*, 2022(1), 7775917.
- Ahvanooey, M. T., Li, Q., Rabbani, M., & Rajput, A. R. (2020). A survey on smartphones security: software vulnerabilities, malware, and attacks. *ArXiv Preprint ArXiv:2001.09406*.
- Alswaina, F., & Elleithy, K. (2020). Android malware family classification and analysis: Current status and future directions. *Electronics*, 9(6), 942.
- Alzubaidi, A. (2021). Recent advances in android mobile malware detection: A systematic literature review. *IEEE Access*, 9, 146318–146349.
- Cai, H., Fu, X., & Hamou-Lhadj, A. (2020). A study of run-time behavioral evolution of benign versus malicious apps in android. *Information and Software Technology*, 122, 106291.
- Melo, L. T. C., Ribeiro, R. G., Guimarães, B. C. F., & Pereira, F. M. Q. (2020). Type inference for c: Applications to the static analysis of incomplete programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 42(3), 1–71.
- Mitsarakis, K. (2023). *Contemporary Cyber Threats to Critical Infrastructures: Management and Countermeasures*.
- Mughal, A. A. (2020). Cyber Attacks on OSI Layers: Understanding the Threat Landscape. *Journal of Humanities and Applied Science Research*, 3(1), 1–18.
- Narwal, B., Mohapatra, A. K., & Usmani, K. A. (2019). Towards a taxonomy of cyber threats against target applications. *Journal of Statistics and Management Systems*, 22(2), 301–325.
- Rutherford, D., & Wu, N. (2023). Cybersecurity Risks in the Deployment and Use of Digital Business Cards: Implications for Organizations and End-Users. *2023 International Conference on Computational Science and Computational Intelligence (CSCI)*, 765–770.
- Syed, M., Dettaille, M., & Sadre, R. (n.d.). "A detection system for the sources of information leaks on networked smart devices.
- Yadav, C. S., Singh, J., Yadav, A., Pattanayak, H. S., Kumar, R., Khan, A. A., Haq, M. A., Alhussen, A., & Alharby, S. (2022). Malware analysis in IoT & android systems with defensive mechanism. *Electronics*, 11(15), 2354.
- Zhang, J., Wang, Y., Qiu, L., & Rubin, J. (2021). Analyzing android taint analysis tools: FlowDroid, Amandroid, and DroidSafe. *IEEE Transactions on Software Engineering*, 48(10), 4014–4040.