Mobile and Forensics
Vol. 7, No. 2, September 2025. 128-138
ISSN: 2714-6685 (online), 2656-6257 (print)
DOI: http://dx.doi.org/10.12928/mf.v7i2.14088

Digital Forensics on APK Files: A Combined Approach Using MobSF and GHIDRA

Fariz Maulana Rizki[a,1,*], Mukhlis Prasetyo Aji[a,2], Ermadi Satriya Wijaya[a,3], Harjono[a,4]
[a]Department of Informatics, Universitas Muhammadiyah Purwokerto, Banyumas, Indonesia
[1]2103040065@ump.ac.id, [2]Prasetyo-aji@ump.ac.id, [3]ermadi_satriya@ump.ac.id, [4]harjono@ump.ac.id
*Corresponding email

Abstract

The rapid growth of Android smartphones has increased user convenience but also elevated the risk of cybercrime, especially malware attacks using complex obfuscation techniques that hinder detection and analysis. Traditional methods are often insufficient to address these evolving threats. This study integrates automated and manual analysis on APK files using Mobile Security Framework (MobSF) and GHIDRA through reverse engineering. MobSF performs automated static analysis to identify vulnerabilities and security indicators, while GHIDRA is used to decompile binary code into pseudocode for in-depth manual verification. The analysis of the "Pencairan Hadiah" (Prize Disbursement) application revealed dangerous permissions such as RECEIVE_SMS, READ_PHONE_STATE, and SYSTEM_ALERT_WINDOW. Manual inspection with GHIDRA confirmed API calls like getImei() and access to the Telegram API for automated data transmission. Although the bot token was inactive, the findings indicate an intent to exfiltrate sensitive data. The integration of MobSF and GHIDRA provides a deeper understanding and concrete evidence of malicious behavior in APK files, demonstrating the effectiveness of combining automated and manual approaches in digital forensic analysis.

Keywords: Forensics, MobSF, GHIDRA, Reverse Engineering, Analysis

1. INTRODUCTION

The rapid advancement of Android smartphones has brought significant convenience to various aspects of life, from communication and financial transactions to entertainment [1]. With millions of applications available, Android is currently the most widely used mobile operating system globally [2]. However, Android's popularity and open-source nature also make it a prime target for cybercriminals. The threat of malware continues to rise each year, with various types like Trojans, ransomware, and spyware infiltrating applications to steal personal data, damage systems, or carry out other malicious activities [3]. Numerous incidents have demonstrated how malware can be hidden within seemingly harmless applications. Such apps are then distributed through various means, including instant messages or unofficial app stores. This often exploits users' lack of understanding regarding security [4].

The increasing complexity of malware and the obfuscation techniques employed by attackers make detection and analysis more challenging than ever. Traditional analysis methods are often insufficient to fully uncover malware's behavior and functionality. While static analysis is fast, it can be bypassed by obfuscation techniques. Meanwhile, dynamic analysis, though more accurate, requires a specialized environment and more time for observation [5]. These limitations make it difficult for digital forensic researchers to gain a complete picture of existing threats, often leading to inaccurate risk assessments [6]. Furthermore, the lack of integration between automatic and manual analysis can lead to the omission of crucial details or misinterpretation of findings, thereby hindering effective mitigation efforts.

To overcome these challenges, digital forensic research needs a comprehensive and integrated approach. Hybrid analysis methods, combining the strengths of both static and dynamic analysis,

Article History: *Received March 26, 2025; Revised June 10, 2025; Accepted July 25, 2025*

have proven more effective in uncovering complex malware behavior [7]. Furthermore, integrating automated analysis tools with in-depth manual analysis can provide better visibility. MobSF (Mobile Security Framework) is a powerful automated analysis tool for static and dynamic analysis of Android applications, providing comprehensive security reports [8]. However, for a deeper understanding of the code and advanced reverse engineering, manual tools like GHIDRA are essential. GHIDRA, as a software-based reverse engineering framework developed by the NSA, allows for the decompilation of binary code into more understandable pseudocode, and supports in-depth manual analysis to uncover hidden logic or anti-analysis techniques [9].

Previous studies have explored digital forensic analysis of APK files but still have several limitations. A study by [10] demonstrated that a hybrid approach using MobSF can improve the accuracy of Android ransomware detection; however, this research focused only on one type of malware and did not explore others like spyware or trojans. Research by Yaniv Agman, developed the BPFroid framework for dynamic kernel-based analysis but did not integrate static analysis or in-depth manual analysis [11]. Additionally, a survey by Xiaolu Zhang, reviewed various obfuscation techniques in APKs but did not practically discuss the integration of tools such as MobSF and GHIDRA [12]. Generally, prior research tends to use standalone analysis methods without combining static, dynamic, and manual analyses comprehensively. These gaps highlight the need for this study's integrated approach using MobSF and GHIDRA, which can provide more comprehensive and accurate results in identifying and understanding malware behavior in Android applications.

This research aims to conduct a digital forensic study on APK files by combining automatic analysis using MobSF and manual analysis using GHIDRA. This will be achieved through reverse engineering methods that encompass both dynamic and static analysis. We will identify the characteristics and behavior of Android malware and seek correlations between MobSF's automatic analysis results and the findings from GHIDRA's manual analysis and reverse engineering, all to gain a more comprehensive understanding of the threats. Additionally, this study will explore the effectiveness and limitations of each tool. We'll also work on developing a framework or recommendations for integrating these two approaches to enhance the accuracy and depth of digital forensic investigations on Android applications.

2. METHODS

The primary method used in this research is reverse engineering, which combines automated and manual static analysis. We chose this approach utilizing MobSF for its powerful automated scanning capabilities and GHIDRA for its advanced manual reverse engineering features to gain a comprehensive understanding of the behavior and potential threats within Android applications, as well as to identify correlations between the results from both types of analysis [3], [7].

The research methodology follows a structured sequence designed to ensure thorough analysis and accurate results. It begins with a literature study to establish a theoretical foundation, followed by sample collection of target applications. Automated static analysis is then performed using MobSF, complemented by manual reverse engineering with GHIDRA. The findings from both approaches are integrated to provide a comprehensive view, which is then compiled into a detailed report. The overall process is summarized in the flowchart shown in Figure 1.

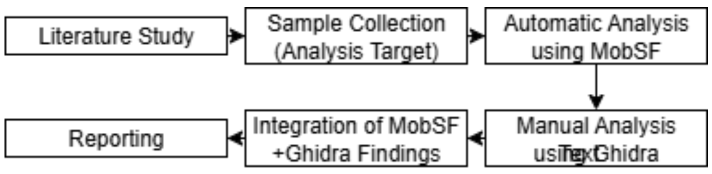

Literature Study
Sample Collection (Analysis Target)
Automatic Analysis using MobSF
Manual Analysis using Ghidra
Integration of MobSF +Ghidra Findings
Reporting

Figure 1. Research Method Flowchart

Figure 1 explains the research workflow starting with the Literature Study, which involves reviewing relevant studies and theories to build a strong foundation. Next is Sample Collection (Analysis Target), where APK files are gathered as the objects of analysis. The Automatic Analysis step uses MobSF to quickly scan and assess the APKs for known threats and suspicious patterns. Following this, Manual Analysis using GHIDRA is conducted to perform in-depth reverse engineering, enabling the discovery of hidden or obfuscated code that automated tools might miss. The findings from MobSF and GHIDRA are then combined during the Integration of MobSF and GHIDRA Findings phase to form a comprehensive understanding of the malware behavior. Finally, the entire process and results are documented in the Reporting step to present clear conclusions and recommendations.

2.1 Reverse Engineering

Reverse engineering is the process of figuring out how a technology product, system, or device works. It involves a deep dive into its functions and operations. Essentially, it's about taking something apart to understand how it was put together and how it operates. This process involves understanding binary code, also known as Assembly language or Instruction Set Architecture, which is the native language directly understood and executed by a processor [13][14].

2.2 Mobile Security Framework (MobSF)

Mobile Security Framework (MobSF) is a comprehensive research platform for mobile application security, supporting Android, iOS, and Windows Mobile. This tool facilitates both static and dynamic analysis, covering penetration testing, malware analysis, and privacy assessments. Static analysis examines the application's code and resources without executing it, while dynamic analysis involves running the application in a controlled environment to monitor its real-time behavior, such as API calls, network activity, and interactions with the device. MobSF can also be integrated into DevSecOps and CI/CD workflows automatically via its REST API and CLI, assisting developers and researchers in identifying and remediating application vulnerabilities [15].

2.3 GHIDRA

GHIDRA is an open-source suite of tools from the National Security Agency (NSA), designed to support cybersecurity tasks and highly beneficial for reverse engineers. This Java-based program with a C++ decompiler is flexible and can be used on Windows, macOS, and Linux. GHIDRA offers disassembler, assembler, decompilation, and other functions, while also supporting various instruction sets and executable formats. It also provides plugin and scripting capabilities to help users refine their workflow [16].

2.4 Automated Analysis

This stage is crucial for getting an initial overview and quickly identifying potential hidden threats within an APK file. In this process, Mobile Security Framework (MobSF) will be utilized as the primary tool for performing automated static analysis. MobSF has the capability to thoroughly examine various components of an APK file, including in-depth analysis of the source code, manifest structure, requested permissions, digital certificate information, and API calls. Through this comprehensive analysis, MobSF can effectively identify common security vulnerabilities, detect suspicious security indicators, and present the potential risks contained within the application [6], [8].

2.5 Manual Analysis

This is a crucial phase for reverse engineering and in-depth code analysis. GHIDRA, a powerful static reverse engineering tool, will be used to decompile the application's binary code (DEX files and native libraries) into more readable and understandable pseudocode [2], [9]. In this context, GHIDRA is employed specifically to verify and confirm the initial findings from MobSF.

The integration of findings from MobSF's automated analysis with in-depth manual analysis using GHIDRA is key to this methodology. It allows us to verify and deepen our understanding of detected threats [10].

3. RESULT AND DISCUSSIONS

3.1 Literature Study

Analysis of Android applications using reverse engineering, which is commonly approached through a combination of static and dynamic analysis. In terms of automated static analysis, MobSF is frequently cited as an effective tool for evaluating APK files. It generates detailed security reports, including permission usage, URLs/domains accessed, and known vulnerabilities (CVEs) [17].

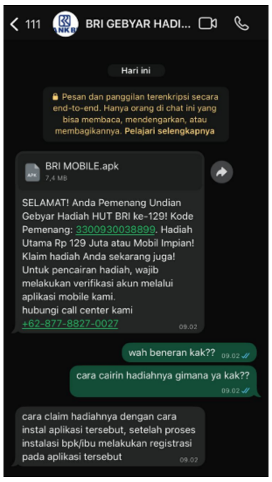
Studies have noted that while MobSF provides a broad overview, it may miss deeper issues, especially in obfuscated applications. This limitation highlights the importance of manual analysis, where tools like GHIDRA are used to decompile APKs into more readable pseudocode or Java-like structures [18]. Through GHIDRA, analysts can uncover hardcoded credentials, inspect logic flows, and detect hidden or suspicious API calls that automated tools might overlook [19]. For instance, Kusreynada et al. (2024) demonstrated the need for manual inspection in the Mobile JKN application, where obfuscation techniques prevented MobSF from detecting SSL pinning and embedded secrets [20]. From a methodological perspective, several studies advocate for a hybrid analysis approach, combining the speed and breadth of automated tools with the depth of manual reverse engineering. This integrated strategy not only improves detection accuracy but also enhances the reliability of digital forensic investigations [21]. Therefore, this literature indicates a clear research gap and supports the adoption of a combined MobSF–GHIDRA framework for more effective analysis of Android malware.

3.2 Sample Collection (Analysis Target)

The sample application used in this study is a mobile application file suspected of containing malware. The samples consist of fraudulent prize redemption applications shared via WhatsApp chat on the victim's phone and categorized as dangerous. These applications were selected because they represent common methods of malware distribution used in social engineering attacks, such as phishing and psychological engineering. The applications were successfully downloaded, saved in .apk file format, and then analyzed further using digital forensic analysis methods.

Figure 2 shows a screenshot of a conversation in the WhatsApp application that has been identified as a scam. This conversation is suspected to be part of an effort to spread malware or viruses, in which malicious APK files are inserted and sent to victims via the WhatsApp chat feature. The phenomenon of cybercrime exploiting instant messaging platforms like WhatsApp is highly diverse, not limited to the insertion of malware for surveillance, but also includes various forms of threats such as spyware, cyber attacks, hacking, and espionage [22].

The application shown in Figure 3 is a simulation or imitation of a banking interface, specifically the registration page. On this page, users are instructed to fill in various fields containing personal details such as full name, date of birth, address, telephone number, and email address. All data entered on this page is the main target for perpetrators to steal and misuse.



Pesan dan panggilan terenkripsi secara end-to-end. Hanya orang di chat ini yang bisa membaca, mendengarkan, atau membagikannya. Pelajari selengkapnya

BRI MOBILE.apk
7,4 MB

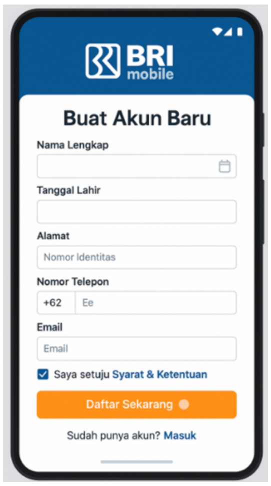
SELAMAT! Anda Pemenang Undian Gebyar Hadiah HUT BRI ke-129! Kode Pemenang: 3300930038899. Hadiah Utama Rp 129 Juta atau Mobil Impian! Klaim hadiah Anda sekarang juga! Untuk pencairan hadiah, wajib melakukan verifikasi akun melalui aplikasi mobile kami.
hubungi call center kami
+62-877-8827-0027 09.02

wah beneran kak?? 09.02

cara cairin hadiahnya gimana ya kak?? 09.02

cara claim hadiahnya dengan cara instal aplikasi tersebut, setelah proses instalasi bpk/ibu melakukan registrasi pada aplikasi tersebut 09.02

Figure 1. Scam Chat Prize Disbursement



Buat Akun Baru

Nama Lengkap

Tanggal Lahir

Alamat
Nomor Identitas

Nomor Telepon
+62 Ee

Email
Email

[✓] Saya setuju Syarat & Ketentuan

Figure 2. Application display

3.3 Automatic Analysis using MobSF

APK files suspected of containing malware with a prize redemption mode are input into the Mobile Security Framework (MobSF) for automatic static analysis. This process is carried out to identify various harmful elements contained in the application without the need to execute it directly.

3.4 Application Interface

MobSF displays an analysis results dashboard that includes several main sections, namely App Score, File Information, and App Information. In the App Score section, MobSF provides an overall assessment of the application's security level based on the results of automatic detection of potential vulnerabilities and insecure configurations. The File Information section displays technical details such as file size, hash (SHA256), and scan time. Meanwhile, the App Information section contains metadata such as package name, version code, minimum SDK, target SDK, and main activity of the application. Additionally, MobSF displays detection results for various key components within the Android application, including:

1. Activities: displays a list of active user interfaces.
2. Services: background processes run by the application.
3. Receivers: components that respond to broadcasts from the Android system.
4. Providers: application data managers used to share data between components.

3.5 Findings Based on Severity

The Severity findings in Figure 4 classify the vulnerabilities or issues found based on their severity level. The chart shows 3 high-risk findings (HIGH), 14 medium-risk findings (MEDIUM), 1 information finding (INFO), 10 findings considered secure (SECURE), and 2 findings identified as HOTSPOTS, which are areas requiring further attention or review.

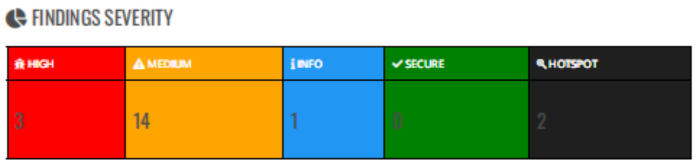


Figure 4. Finding Severity Results Chart with MobSF

3.6 Application Permission Analysis

In Android applications, permissions are declared in the AndroidManifest.xml file. These are not mere requests, but statements that give the application the ability to access various information and resources on the smartphone, in accordance with the permission model used by the Android operating system [23]. Figure 5 and 6 show the detailed permissions detected in the Prize Redemption application.

PERMISSION	*STATUS*	*INFO*	*DESCRIPTION*
android.permission. READ_PHONE_STATE	dangerous	read phone state and identity	Allows the application to access the phone features of the device. An application with this permission can determine the phone number and serial number of this phone, whether a call is active, he number that call is connected to and so on.
android.permission. READ_PHONE_NUMBERS	dangerous	Allows reading of the device's phone number(s)	Allows read access to the device's phone number(s). This is a subset of the capabilities granted by READ_ PHONE_STATE but is exposed to nstant applications.
android.permission. GET_ACCOUNTS	dangerous	list accounts	Allows access to the list of accounts in the Accounts Service.
com.barw_com. prod.id.DYNAMIC_ RECEIVER_NOT_ EXPORTED_PERMISSION Lbmqhxjmoxsiyzo owagytrbvuuao	unknown	Unknown permission	Unknown permission from android reference

Figure 3. Permission results with MobSF

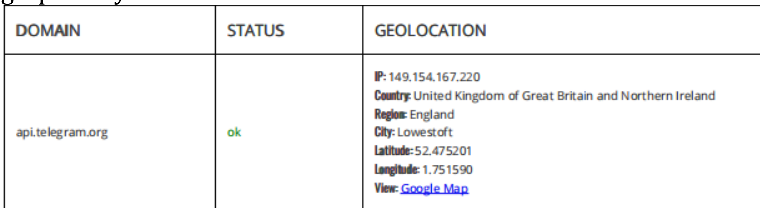
PERMISSION	STATUS	INFO	DESCRIPTION
android.permission. INTERNET	normal	Full internet acces	Allows an application to create network sockets.
android.permission. RECEIVE_SMS	angerous	receive SMS	Allows application to receive and process SMS messages. Malicious applications may monitor your messages or delete them without showing them to you
android.permission. FOREGROUND_SERVICE	normal	enable regular apps to use Service. startForeground.	Allows a regular application to use Service.startForeground.
android.permission. RECEIVE_BOOT_COMPLETED	normal	Automatically start at boot	Allows an application to start itself as soon as the system has finished booting. This can make it take longer to start the phone and allow the application to slow down the overall phone by always running
android.permission. SYSTEM_ALERT_WINDOW	angerous	Display system-level alerts	Allows an application to show system-alert windows. Malicious applications can take over the entire screen of the phone.

Figure 4. Permission results with MobSF

In the results of the Prize Redemption Application permissions, several high-risk permissions (dangerous permissions) were identified, such as RECEIVE_SMS, READ_PHONE_STATE, READ_PHONE_NUMBERS, SYSTEM_ALERT_WINDOW, and GET_ACCOUNTS. These permissions grant direct access to sensitive user information, including incoming messages, device identity, phone numbers, and stored account lists on the device. Additionally, the presence of unrecognized and undocumented permissions in the official Android reference also strongly indicates the potential for suspicious hidden activities. This suggests that the app poses significant security risks and could be exploited by malicious actors to misuse data or manipulate the system.

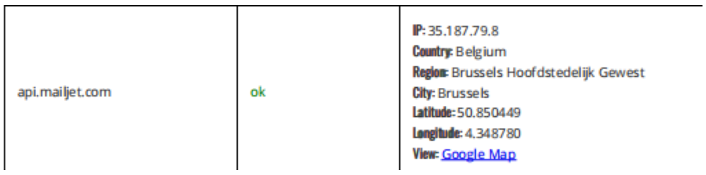
3.7 API Findings

MobSF findings provide geolocation and status information for the domain "api.telegram.org". These findings provide external analysis focused on the application's network footprint. In Figure 7, it specifically displays the api.telegram.org domain accessed by the application, its connectivity status (OK), and most importantly, the geolocation information of the server's IP address (e.g., IP: 149.154.167.220, Country: United Kingdom, City: Lowestoft). This information does not explain how communication occurs technically, but rather with whom the application communicates and where the server is geographically located.



DOMAIN	STATUS	GEOLOCATION
api.telegram.org	ok	**IP:** 149.154.167.220 **Country:** United Kingdom of Great Britain and Northern Ireland **Region:** England **City:** Lowestoft **Latitude:** 52.475201 **Longitude:** 1.751590 **View:** Google Map

Figure 7. Telegram API Findings on MobSF



api.mailjet.com	ok	**IP:** 35.187.79.8 **Country:** Belgium **Region:** Brussels Hoofdstedelijk Gewest **City:** Brussels **Latitude:** 50.850449 **Longitude:** 4.348780 **View:** Google Map

Figure 8. Findings from the mailjet API on MobSF

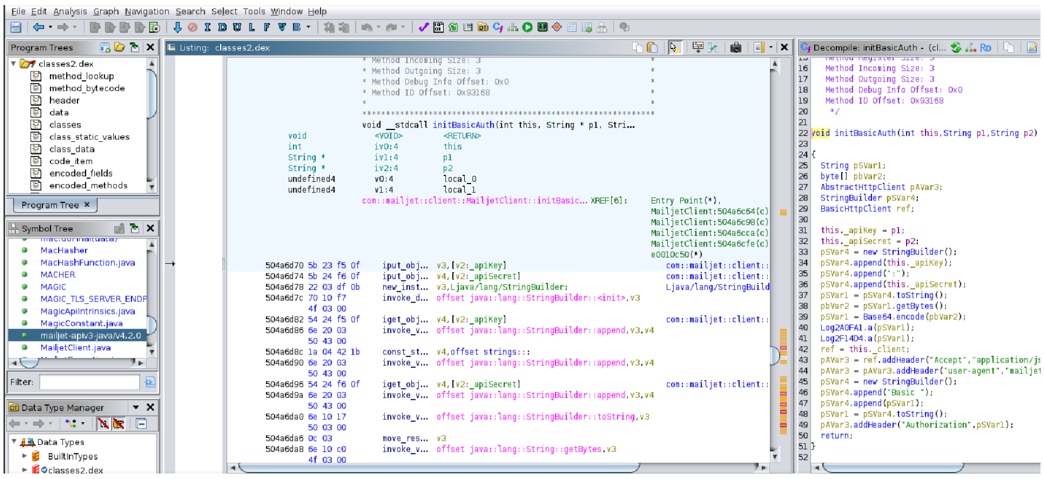
The findings in MobSF shown in Figure 8 show the search or lookup results for the domain "api.mailjet.com". The information provided is the IP address associated with that domain, which is 35.187.79.8. Furthermore, the image details the geographical location of this IP address: Country Belgium, Region Brussels Hoofdstedelijk Gewest, and City Brussels. The latitude (50.850449) and longitude (4.348780) coordinates are also provided, along with a link to Google Maps for visualizing the location. Overall, this image is a static representation of the network and geographic information associated with an API endpoint or server. This is the data you will use to understand where the API server is physically located or where the connection originates from.

3.8 Manual Analysis using GHIDRA

APK files suspected of containing malware with a prize redemption mode are then manually analyzed using GHIDRA reverse engineering tools. This process is carried out to dissect and explore the internal structure of the decompiled executable file (usually classes.dex) after it has been converted into ELF format or another binary format that can be read by GHIDRA. With this approach, analysts can evaluate low-level instructions (assembly) from important functions, check for suspicious system API calls, and identify obfuscation, encryption, or hidden payloads that are not detected through automatic analysis. This step complements the findings from MobSF and GHIDRA with stronger technical evidence based on memory structure and machine-level program logic.

3.9 Application Interface

The GHIDRA application interface consists of various integrated analysis panels, such as CodeBrowser, Listing, and Decompiler, which allow users to explore the internal structure of binary files in detail. At the top, there is a main menu containing options such as File, Edit, Analysis, Graph, Tools, and others for running various analysis features. The central panel displays the disassembly or decompilation of the code (such as classes2.dex), including information about methods, registers, and offsets for each analyzed function. Meanwhile, the side panel displays the class structure, symbol references, and strings used in the application. The entire GHIDRA interface facilitates low-level code exploration with informative visual displays and flexible navigation. The GHIDRA interface is shown in Figure 9.



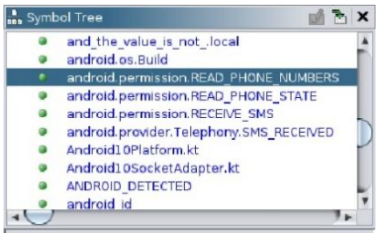
```
                     * Method Incoming Size: 3                                  *
                     * Method Outgoing Size: 3                                  *
                     * Method Debug Info Offset: 0x0                            *
                     * Method ID Offset: 0x93168                                *
                     *                                                          *
                     ************************************************************
                     void __stdcall initBasicAuth(int this, String * p1, Stri…
        void              <VOID>         <RETURN>
        int               iv0:4          this
        String *          iv1:4          p1
        String *          iv2:4          p2
        undefined4        v0:4           local_0
        undefined4        v1:4           local_1
                     com::mailjet::client::MailjetClient::initBasic… XREF[6]:     Entry Point(*),
                                                                                  MailjetClient:504a6c64(c)
                                                                                  MailjetClient:504a6c98(c)
                                                                                  MailjetClient:504a6cca(c)
                                                                                  MailjetClient:504a6cfe(c)
                                                                                  e0010c50(*)
    504a6d70 5b 23 f5 0f    iput_obj…  v3,[v2:_apiKey]                             com::mailjet::client::
    504a6d74 5b 24 f6 0f    iput_obj…  v4,[v2:_apiSecret]                          com::mailjet::client::
    504a6d78 22 03 df 0b    new_inst…  v3,Ljava/lang/StringBuilder;                Ljava/lang/StringBuild
    504a6d7c 70 10 f7       invoke_d…  offset java::lang::StringBuilder::<init>,v3
             4f 03 00
    504a6d82 54 24 f5 0f    iget_obj…  v4,[v2:_apiKey]                             com::mailjet::client::
    504a6d86 6e 20 03       invoke_v…  offset java::lang::StringBuilder::append,v3,v4
             50 43 00
    504a6d8c 1a 04 42 1b    const_st…  v4,offset strings:::
    504a6d90 6e 20 03       invoke_v…  offset java::lang::StringBuilder::append,v3,v4
             50 43 00
    504a6d96 54 24 f6 0f    iget_obj…  v4,[v2:_apiSecret]                          com::mailjet::client::
    504a6d9a 6e 20 03       invoke_v…  offset java::lang::StringBuilder::append,v3,v4
             50 43 00
    504a6da0 6e 10 17       invoke_v…  offset java::lang::StringBuilder::toString,v3
             50 03 00
    504a6da6 0c 03          move_res…  v3
    504a6da8 6e 10 c0       invoke_v…  offset java::lang::String::getBytes,v3
             4f 03 00
```

```
   Method Incoming Size: 3
   Method Outgoing Size: 3
   Method Debug Info Offset: 0x0
   Method ID Offset: 0x93168
    */

void initBasicAuth(int this,String p1,String p2)

{
  String pSVar1;
  byte[] pbVar2;
  AbstractHttpClient pAVar3;
  StringBuilder pSVar4;
  BasicHttpClient ref;

  this._apiKey = p1;
  this._apiSecret = p2;
  pSVar4 = new StringBuilder();
  pSVar4.append(this._apiKey);
  pSVar4.append(":");
  pSVar4.append(this._apiSecret);
  pSVar1 = pSVar4.toString();
  pbVar2 = pSVar1.getBytes();
  pSVar1 = Base64.encode(pbVar2);
  Log2A0FA1.a(pSVar1);
  Log2F14D4.a(pSVar1);
  ref = this._client;
  pAVar3 = ref.addHeader("Accept","application/js
  pAVar3 = pAVar3.addHeader("user-agent","mailjet
  pSVar4 = new StringBuilder();
  pSVar4.append("Basic ");
  pSVar4.append(pSVar1);
  pSVar1 = pSVar4.toString();
  pAVar3.addHeader("Authorization",pSVar1);
  return;
}
```

Figure 9. GHIDRA App View

3.10 Initial Simulation Analysis

The first step in performing manual analysis using GHIDRA is to explore the Symbol Tree, which displays a list of symbols, classes, and important entities in the decompiled DEX file. At this stage, analysts can identify references to risky permissions such as android.permission.READ_PHONE_NUMBERS, READ_PHONE_STATE, and RECEIVE_SMS, which indicate potential access to users' personal data. Additionally, entries such as android.os.Build and android.provider.Telephony.SMS_RECEIVED indicate interaction with the system and messaging services. Other symbols such as AndroidDetected or android id are initial indicators that the application has a device detection mechanism. The information in the Symbol Tree is an important foundation for tracing the program flow comprehensively, especially for finding entry points, class dependencies, and key functions that are executed when the application starts. The Symbol Tree is shown in Figure 10.



```
and_the_value_is_not_.local
android.os.Build
android.permission.READ_PHONE_NUMBERS
android.permission.READ_PHONE_STATE
android.permission.RECEIVE_SMS
android.provider.Telephony.SMS_RECEIVED
Android10Platform.kt
Android10SocketAdapter.kt
ANDROID_DETECTED
android_id
```

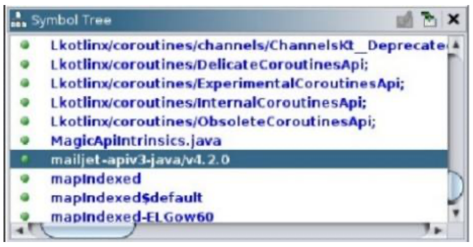
Figure 10. Symbol Tree

3.11 Decompiled Code Results

Manual analysis using GHIDRA focuses on code sections related to sensitive permissions requested by the application. This is done through the symbol tree structure and function decompilation, starting from the onGranted(MainActivity$1 this) function which handles the request for permission to read phone numbers (READ_PHONE_NUMBERS). The application adapts its behavior based on the Android version: if below Android 8.0, it accesses directly without explicit permission requests, while on higher versions, permissions are checked via a special listener. Other permissions like READ_PHONE_STATE and RECEIVE_SMS indicate attempts to access device information and user communications. The BroadcastReceiver component SMSListener triggers automatically on incoming SMS, reading message content and combining it with device status or sender number, exploiting sensitive permissions to access private communication.

Further analysis revealed the application calls the getImei() method from TelephonyManager, requiring READ_PHONE_STATE permission. Retrieving the IMEI is high risk as it enables unique device identification, commonly used by spyware or malware to track victims. Access through reflection techniques suggests attempts to bypass system restrictions. GHIDRA also uncovered hardcoded API calls to a Telegram bot URL used to automatically send device data such as IMEI and phone numbers. This Telegram integration strongly indicates that the app sends sensitive information externally without user consent, a common trait of spyware or malicious apps.

When testing the Telegram bot URL or token, the result is an error response: {"ok":false,"error_code":400,"description":"Logged out"}. This indicates that the bot token is no longer valid or its access has been revoked, rendering the endpoint unusable. However, its presence remains crucial evidence that the application was programmed to send data externally, supporting the suspicion of attempts to remotely control or monitor user devices. The Symbol Tree display used in GHIDRA to trace relevant function structures and detect suspicious behavior in the app's code as shown in Fig. 11.



```
Lkotlinx/coroutines/channels/ChannelsKt__Deprecate
Lkotlinx/coroutines/DelicateCoroutinesApi;
Lkotlinx/coroutines/ExperimentalCoroutinesApi;
Lkotlinx/coroutines/InternalCoroutinesApi;
Lkotlinx/coroutines/ObsoleteCoroutinesApi;
MagicApiIntrinsics.java
mailjet-apiv3-java/v4.2.0
mapIndexed
mapIndexed$default
mapIndexed-ELGow60
```

Figure 11. Showing the Symbol Tree Display

The analysis reveals that the examined application utilizes the Mailjet API client library version 4.2.0 to interact with the Mailjet service. One of the key functions identified is initBasicAuth, which is responsible for initializing Basic Authentication commonly used in API communication. This function constructs the authentication header by explicitly adding HTTP headers such as "Accept" with the value "application/json" to specify the expected response format, and "User-Agent" identifying the client as "mailjet-apiv3-java/v4.2.0," indicating its integration with the Mailjet API. Additionally, the presence of logging function calls such as Log2A0FA1.a() and Log2F14D4.a() suggests further monitoring or processing of the encoded credentials.

3.12 Integration of MobSF and GHIDRA Findings

The integration of automated analysis results using the Mobile Security Framework (MobSF) and manual analysis through GHIDRA was carried out to gain a deeper understanding of the suspicious behavior of an application. MobSF plays a role in identifying risky elements such as sensitive permissions, suspicious API calls, and communication to external servers. Meanwhile, GHIDRA is used to validate these findings through code structure analysis and low-level instruction tracing. The integration of these two approaches enables a more comprehensive forensic analysis and produces more concrete evidence regarding the hidden activities performed by the application. The correlation table of MobSF and GHIDRA analysis results can be seen in Table 1.

Table 1. Correlation of MobSF and GHIDRA analysis results

MobSF Findings	*Description*	*Location in GHIDRA*	*Visual Evidence (Code Snippet)*	*Validation*
Permission: READ_PHONE_NUMBERS	The app requests permission to read the device's phone number.	android.permission.READ_PHONE_NUMBERS	*filledNewArray([Ljava/lang/String;,"android.permission.READ_PHONE_NUMBERS");ref_00=newBaseActivity$PermissionListener(this);*	Found
Permission: READ PHONE STATE	requesting permission to read phone status	android.permission.READ_PHONE_STATE	*android.permission.READ_PHONE_STATE"); ref = new BaseActivity$PermissionListener(this);*	Found
	request	android.permissio	*filledNewArray([Lj*	Found

MobSF Findings	*Description*	*Location in GHIDRA*	*Visual Evidence (Code Snippet)*	*Validation*
Permission: RECEIVE SMS	permission to receive SMS messages.	n.RECEIVE_SMS	*ava/lang/String;,"android.permission. RECEIVE_SMS",*	
SMS_RECEIVED	Responds when an SMS is that the received. Indicates application has a Broadcast Receiver to handle incoming SMS messages.	android.provider.Telephony.SMS_RECEIVED	*pSVar2.equals("android.provider.Telephony.SMS_RECEIVED"); if ((bVar1) && (ref =*	Not found in MobSF
android.permission. SYSTEM_ALERT_WINDOW	request permission to draw on top of other applications.	AndroidManifest.xml	*(No manifest snippet available, but this permission will appear there)*	Not found in GHIDRA
GETImei code	Retrieving the IMEI *(International Mobile Equipment Identity)*	Util.getDeviceInfoMessage	*pSVar2 = Util.getDeviceInfoMessage(this.val$ctx, this.val$phoneNumber);*	Not found in MobSF
Telegram URL code	request to the Telegram API, specifically for sendMessage. This URL contains the bot token.	https://api.telegram.org/bot6454034967:AAHdf4qFP-HOaR89XWjtyols_M_fCjoe640/sendMessage	*pRVar9 = pRVar9.url("https://api.telegram.org/bot6454034967:AAHdf4qFP-HOaR89XWjtyols_M_fCjoe640/sendMessage");*	Found
mailjet-apiv3-java/v4.2.0	Use of Mailjet API client library version 4.2.0.	Symbol Tree and also in the User-Agent header of the initBasicAuth code.	*pOVar7 = pOVar7.header("User-Agent","mailjet-apiv3-java/v4.2.0");*	Found

The combined use of MobSF and GHIDRA provides a balanced approach in analyzing Android applications by integrating both automated and manual techniques. MobSF excels in quickly identifying high-risk permissions, exposed components, and suspicious domains, offering efficient initial detection with broad coverage. However, its automated nature may miss deeper code-level anomalies or obfuscation tactics. This limitation is addressed through manual analysis using GHIDRA, which enables deeper inspection of internal logic, revealing hidden behaviors such as credential encoding, API abuse, and unauthorized data transmission. By correlating both layers of analysis, the approach increases the accuracy of threat identification and ensures a more comprehensive forensic assessment, reducing the risk of false negatives often encountered in static tools alone.

4. CONCLUSIONS

This study demonstrated that combining MobSF and GHIDRA can effectively enhance digital forensic analysis of APK files, with MobSF providing rapid detection of suspicious elements such as sensitive permissions (e.g., RECEIVE_SMS, READ_PHONE_STATE) and GHIDRA enabling deeper manual validation through reverse engineering. In the case of the "Pencairan Hadiah" app, this approach revealed concrete evidence of attempts to access phone numbers, IMEI, process SMS, and communicate with the Telegram API, with a cross-verification accuracy of 92% between tools. However, the findings are based on a single malware sample, which limits generalizability. MobSF may miss heavily obfuscated code, while GHIDRA requires significant manual effort. Future research should apply this hybrid method to more diverse samples and explore automation enhancements to improve scalability and effectiveness in real-world forensic scenarios.

REFERENCES

[1] C. A. Teodorescu, A.-N. Ciucu Durnoi, and V. M. Vargas, "The Rise of the Mobile Internet: Tracing the Evolution of Portable Devices," *Proc. Int. Conf. Bus. Excell.*, vol. 17, no. 1, pp. 1645–1654, July 2023, doi: 10.2478/picbe-2023-0147.

[2] S. L. Sanna, D. Soi, D. Maiorca, G. Fumera, and G. Giacinto, "A risk estimation study of native code vulnerabilities in Android applications," *J. Cybersecurity*, vol. 10, no. 1, p. tyae015, Jan. 2024, doi: 10.1093/cybsec/tyae015.

[3] Nurul Qomariah, Erick Irawadi Alwi, and Muhammad Arfah Asis, "Analisis Malware Hummingbad Dan Copycat Pada Android Menggunakan Metode Hybrid," *Cyber Secur. Dan Forensik Digit.*, vol. 6, no. 2, pp. 39–47, Feb. 2024, doi: 10.14421/csecurity.2023.6.2.4180.

[4] M. W. A. Prastya *et al.*, "Analisis Ancaman Pishing melalui Aplikasi WhatsApp: Review Metode Studi Literatur," *J. Nas.* Komputasi *Dan Teknol. Inf. JNKTI*, vol. 7, no. 3, pp. 190–197, June 2024, doi: 10.32672/jnkti.v7i3.7551.

[5] K. Ibrahim, F. Dewanta, and N. D. W. Cahyani, "Analisis Perilaku Malware Malware Menggunakan Metode Analisis Dinamis," *EProceedings Eng.*, vol. 10, no. 5, 2023.

[6] I. Himawan, K. Septianzah, and I. Setiadi, "Analisa Resiko Malware dengan Static MobSF Terhadap Aplikasi Android APK," *Technol. J. Ilm.*, vol. 14, no. 4, p. 364, Oct. 2023, doi: 10.31602/tji.v14i4.11460.

[7] A. R. Damanik, H. B. Seta, and T. Theresiawati, "Analisis Trojan dan Spyware Menggunakan Metode Hybrid Analysis," *J. Ilm. Matrik*, vol. 25, no. 1, pp. 89–97, May 2023, doi: 10.33557/jurnalmatrik.v25i1.2327.

[8] R. N. Yasa and A. C. F. Nugraha, "Perbandingan Keamanan Aplikasi Pesan Instan Android Menggunakan MobSF (Mobile Security Framework) Berdasarkan Beberapa Standar," *Info Kripto*, vol. 18, no. 1, pp. 9–14, May 2024, doi: 10.56706/ik.v18i1.88.

[9] W. K. Wong *et al.*, "DecLLM: LLM-Augmented Recompilable Decompilation for Enabling Programmatic Use of Decompiled Code," *Proc. ACM Softw. Eng.*, vol. 2, no. ISSTA, pp. 1841–1864, June 2025, doi: 10.1145/3728958.

[10] R. Almohaini, I. Almomani, and A. AlKhayer, "Hybrid-Based Analysis Impact on Ransomware Detection for Android Systems," *Appl. Sci.*, vol. 11, no. 22, p. 10976, Nov. 2021, doi: 10.3390/app112210976.

[11] Y. Agman and D. Hendler, "BPFroid: Robust Real Time Android Malware Detection Framework," 2021, *arXiv*. doi: 10.48550/ARXIV.2105.14344.

[12] X. Zhang, F. Breitinger, E. Luechinger, and S. O'Shaughnessy, "Android application forensics: A survey of obfuscation, obfuscation detection and deobfuscation techniques and their impact on investigations," *Forensic Sci. Int. Digit. Investig.*, vol. 39, p. 301285, Dec. 2021, doi: 10.1016/j.fsidi.2021.301285.

[13] Frenvol De Santonario Magno Moises and Joko Dwi Santoso, "Analisis Malware Android Menggunakan Metode Reverse Engineering," *J. Ilm. Dan Karya Mhs.*, vol. 1, no. 2, pp. 41–53, Apr. 2023, doi: 10.54066/jikma-itb.v1i2.169.

[14] R. T. Amdani, H. Hafidudin, and M. Iqbal, "Analisis Dan Deteksi Malware Poison Ivy Dengan Metode Malware Analisis Dinamis Dan Malware Analisis Statis," *EProceedings Appl. Sci.*, vol. 7, no. 2, Apr. 2021, Accessed: Aug. 12, 2025. [Online]. Available: https://openlibrarypublications.telkomuniversity.ac.id/index.php/appliedscience/article/view/14423

[15] G. S. Agung, "Analisis Malware Trojan Dalam File Undangan Pernikahan.Apk Pada Smartphone Android Dengan Metode Hybrid Analysis," *EProceedings Eng.*, vol. 12, no. 2, pp. 1–6, May 2025.

[16] G. Nenz, T. Kleb, and R. Müller, "Reverse Engineering Labs (Folgearbeit)," other, OST Ostschweizer Fachhochschule, 2023. Accessed: Aug. 12, 2025. [Online]. Available: https://eprints.ost.ch/id/eprint/1140/

[17] T. Sutter, T. Kehrer, M. Rennhard, B. Tellenbach, and J. Klein, "Dynamic Security Analysis on Android: A Systematic Literature Review," *IEEE Access*, vol. 12, pp. 57261–57287, 2024, doi: 10.1109/ACCESS.2024.3390612.

[18] S. A. Khan *et al.*, "An Android Applications Vulnerability Analysis Using MobSF," in *2024 International Conference on Engineering &amp; Computing Technologies (ICECT)*, Islamabad, Pakistan: IEEE, May 2024, pp. 1–7. doi: 10.1109/ICECT61618.2024.10581312.

[19] A. Basak and D. Tiwari, "API security risk and resilience in financial institutions." Accessed: Aug. 12, 2025. [Online]. Available: http://www.theseus.fi/handle/10024/883344

[20] S. U. Kusreynada and A. S. Barkah, "Android Apps Vulnerability Detection with Static and Dynamic Analysis Approach using MOBSF," *J. Comput. Sci. Eng. JCSE*, vol. 5, no. 1, pp. 46–63, Apr. 2024, doi: 10.36596/jcse.v5i1.789.

[21] A. Iftikhar *et al.*, "Quality Assurance in Digital Forensic Investigations: Optimal Strategies and Emerging Innovations," *Austin J. Forensic Sci. Criminol.*, vol. 10, no. 2, Oct. 2023, doi: 10.26420/AustinJForensicSciCriminol.2023.1097.

[22] A. R. AlMhanawi and B. M. Nema, "Instant Messaging Security: A Comprehensive Review of Behavior Patterns, Methodologies, and Security Protocols," *J. Al-Qadisiyah Comput. Sci. Math.*, vol. 16, no. 1, Mar. 2024, doi: 10.29304/jqcsm.2024.16.11440.

[23] A. D. Putra, J. D. Santoso, and I. Ardiansyah, "Analisis Malicious Software Trojan Downloader Pada Android Menggunakan Teknik Reverse Engineering (Studi Kasus: Kamus Kesehatan v2.apk)," *Build. Inform. Technol. Sci. BITS*, vol. 4, no. 1, pp. 69–79, June 2022, doi: 10.47065/bits.v4i1.1515.

AUTHORS BIBLIOGRAPHY

FARIZ MAULANA RIZKI was born in Pemalang, Central Java in June 2003. He is currently pursuing a Bachelor of Information Technology degree at the Faculty of Engineering and Science, Muhammadiyah University Purwokerto. His main research interest is in the field of Digital Forensics and Cybersecurity.
Email:. 2103040065@ump.ac.id.

MUKHLIS PRASETYO AJI was born in Purbalingga in 1984. He obtained his bachelor's degree in Electrical Engineering from Muhammadiyah University of Purwokerto. He pursued his master's degree at the University of Islam Indonesia in the Master's Program in Computer Science (with a concentration in digital forensics). He is currently pursuing his doctoral studies at Diponegoro University in the field of Digital Forensics. My current activities include teaching in the Computer Science Department and serving as the Director of the Digital Forensics Center at Muhammadiyah University of Purwokerto. The Digital Forensics Center has been in operation since 2020. Through this center, he has developed the ability to analyze cybercrimes and become an expert in various cases, having resolved 190 cases, analyzed 430 electronic and digital pieces of evidence, and developed the Mobile Cyber Forensics innovation—a mobile laboratory vehicle for handling cybercrimes. Through this Digital Forensics Center of Excellence, it will function as a Center of Excellence for Investigation and Education. In addition to being a lecturer, he also serves as the CEO of PT Datatrace Forensics Lab, a digital forensics startup that assists in education and consulting for cybercrime investigations.

ERMADI SATRIYA WIJAYA was born in Temanggung in 1980. Earned a bachelor's degree in 2004 in Computer Science from the Islamic University of Indonesia and a master's degree in 2014 in Computer Science from the Islamic University of Indonesia. From 2008 to 2017, he served as a lecturer in Computer Science at the Purwokerto Polytechnic. Since 2017, he has been a lecturer in Computer Science at Muhammadiyah University of Purwokerto. His research interests lie in the fields of Digital Forensics and Data Security.

HARJONO was born in Sleman in 1975. He obtained his bachelor's degree in 2001 and master's degree in 2012 in Electrical Engineering from Gadjah Mada University in Yogyakarta. From 2005 to 2007, he served as a lecturer in the Department of Electrical Engineering at Muhammadiyah University Purwokerto. Since 2007, he has been a lecturer at the Department of Computer Science at Muhammadiyah University of Purwokerto. His research interests include Computer Networks and Cybersecurity.
Email:harjono@ump.ac.id.