

ARTIFACTS RETRIEVAL USING NETWORK FORENSIC APPROACH FOR SAAS CLOUD STORAGE ON ANDROID

Aqeel M Nezar¹, and Khairul Akram Zainol Ariffin^{1}*

¹Center for Cyber Security, Fakulti Teknologi dan Sains Maklumat
Universiti Kebangsaan Malaysia, 43600, Bangi, Selangor, Malaysia.

Emails: aqeeltamimi@hotmail.com¹, k.akram@ukm.edu.my^{1*}

ABSTRACT

The extensive adoption of cloud storage has significantly altered data governance; however, it has also brought about complex security challenges, especially for organizations implementing Bring Your Own Device (BYOD) policies. Although cloud storage enhances scalability and cost-effectiveness, it also creates opportunities for cyber threats and data breaches, making the development of robust digital forensic (DF) strategies essential. This study explored the critical need for DF experts to effectively retrieve and analyze data remnants from Android cloud storage applications given the ever-changing security landscape of the Android platform. The aim was to establish a digital forensic protocol for extracting data remnants from five specific Android cloud storage applications: BigMind, Degoo, FEX NET, File.fm, and Koofr, using network packet analysis as the main technique. By simulating various user activities, such as logging in, uploading, downloading, and deleting, this research compares the data remnants from both Android apps and mobile web browsers to highlight key forensic differences. These findings demonstrate the possibility of retrieving sensitive data, including user credentials, file metadata, and access tokens, thus providing DF professionals with crucial information for investigating cyberattacks and overseeing security. Additionally, this study highlights the challenges posed by advanced security measures in some applications, which complicate the processes of capturing and decrypting network packets. Ultimately, the results contribute to the development of improved BYOD security frameworks, enabling organizations to better manage cloud usage, detect unauthorized data access, and mitigate the security risks associated with the widespread use of cloud storage in the Android environment. This study adds to the growing body of knowledge necessary for securing cloud services and enhancing digital forensic techniques in response to the evolving cyberthreat landscape.

Keywords: Cloud forensic; cloud computing; android; SaaS; network forensic; mobile; BYOD; cybersecurity.

1. INTRODUCTION

Nowadays, the technological world has moved to virtualize people's daily lives, such as virtual social relations, virtual markets, virtual classes, virtual meetings, and virtual storage. This technology is available through computer devices and the Internet. Its concept brings to the surface the term cloud computing, where the user can utilize and customize computing resources (e.g., storage, applications, platform, and infrastructure) through network communication [1].

A cloud service provider (CSP) currently provides the user with storage, either at a free or predefined cost. Users can access their storage after creating accounts with a specific CSP, login to the system, and start uploading, downloading, and other actions on their files. Many CSPs allow users to access their accounts using various methods on different platforms such as web browsers, desktop applications, and mobile applications. Android has the largest OS market for smartphones [2], and it can also be considered a famous platform for cloud storage applications.

People use Android devices to connect with others, perform financial transactions, store private documents, and perform many other activities. In addition, Android devices have privileged over other devices (i.e., personal computers) because of their portability, ease of use, affordability, and other hardware and software manufacturing features. These features make Android devices a hub for data exposed to legitimate and malicious users [3]. Hence, security and DF practitioners must explore this realm to enhance security and provide evidence when required.

There are many Android cloud storage providers on Google's play; for example, Google Drive cloud storage applications have more than five billion installations [4]. As cloud-storage usage on Android has increased over the years, it indicates that users trust this platform to store and preserve their data. Nonetheless, cloud services remain the preferred choice for organizations because of their scalability, flexibility, and cost-effectiveness [5]. However, it also brings attention to the security risks and issues for both clients and CSPs. A study in [6] stated that cloud storage may allow criminals to launch a denial-of-service attack or utilize it to store malicious files. Hence, it raises security concerns regarding this issue as cyberattacks continue to increase and become more complex. In addition, users must better understand the security of CSPs to choose the best provider for securing their accounts and files.

Security and criminal activities in cloud applications have forced DF practitioners to be well equipped with knowledge to retrieve and extract data remnants [7]. This is to ensure that they are able to present evidence, help in cyberattack investigation, and monitor the organization's security measures. In the Software as a Service (SaaS) storage application, the data remnants do not merely reside in the Android device but are also transmitted between the client and server. These data remnants could be sensitive, as they are used to consume cloud-storage APIs, which are HTTP/HTTPS requests. Each request is used to perform actions, such as login, upload, download, view, and delete. Therefore, it is worth accessing these data from a forensic perspective and analyzing them to obtain artifacts that are not limited to user credentials, file information, and content [8].

In contrast, the Android platform updates its security measures regularly. This can be a minimal upgrade to the system, code, and user experience. However, this creates a challenge for DF practitioners in having a practical procedure for forensically extracting and analyzing artifacts [9, 10]. Hence, it is important to study the behavior of Android cloud storage applications and apply forensic techniques to capture and explore available artifacts. Despite empirical forensic studies on some popular Android cloud storage applications, many other cloud storage applications require the attention of DF practitioners. Applying a forensic approach to each CSP application provides DF practitioners with an overview of artifact discovery. It is also valuable for an organization to adopt a specific CSP, considering the security aspect.

Security assessment of cloud storage services can be performed by applying the DF principles and procedures to the Android device to assess its behavior and security. DF focuses on studies of digital resources' behavior, extracting evidential artifacts, and providing results to law enforcement. The application of DF procedures can help organizations assess electronic systems and device security in terms of the cybersecurity principles of Confidentiality, Integrity, and Availability [11].

Additionally, the proliferation of Bring Your Own Device (BYOD) policies in modern workplaces has introduced a new set of security challenges, particularly regarding the use of cloud storage services. While BYOD offers numerous benefits such as increased employee productivity and reduced hardware costs, it also exposes organizations to significant risks related to unauthorized access and data breaches. One of the primary concerns is the potential for employees to inadvertently or intentionally share sensitive company information through personal cloud storage accounts, thereby circumventing established security protocols. This practice, often referred to as "shadow IT," creates blind spots for IT departments, making it difficult to effectively monitor and control data flows [12, 13]. Therefore, this paper proposes a DF procedure to extract data remnants for several cloud storage applications on the Android platform from a network packet analysis perspective. It covers five Android cloud storage applications: BigMind, Degoo, FEX.NET, File.fm, and Koofr. Several scenarios such as login, logout, upload, download, and delete files are used in cloud storage applications. Subsequently, a comparison between retrieving data remnants from the Android cloud storage application and the mobile web browser is also performed to highlight the differences. The results of this experiment can then be applied by the organization to strengthen its BYOD policy.

The remainder of this paper is organized as follows. The Introduction section is followed by a review of the relevant literature related to forensic processes for SaaS cloud storage on Android. It then highlights the discussion on security in Android and mobile forensics. The methodology and experimental setup are outlined in section 4. Section 5 highlights the results of artifact retrieval for each Android storage application, from both the mobile application and web browser perspectives. The enhancement of BYOD security is discussed in Section 6. Finally, Section 7 concludes the paper.

2. RELATED WORKS

There are many contributions to mobile forensics in Android, such as empirical studies, creating tools and methodologies, and classifying artifacts. Agrawal et al. [14] presented a comparison between open source and commercial tools. It focused on the acquisition and analysis of data from Android devices using version 6.0.1. The study also analyzed unrooted devices to bypass the complexity of establishing root access to the (/data) partition. It is recommended that DF practitioners rely on open-source tools, as they produce results similar to those of commercial ones.

The application of various static analysis techniques to reverse engineering and artifact extraction was outlined in a study by Yoo et al. [15]. The obfuscation method has been applied in mobile applications. The method proposed in this study uses the de-compilation of the source code and then searches for a string to obtain the URL. Once the URL was found, the code flow was traced to sequentially determine the artifacts.

Most application developers implement a cross-platform development approach, which may help malicious users create malware on multiple platforms. This scenario was outlined in the study by Shim et al. [16]. The static analysis approach disassembles smali (assembled) applications; in other words, decompilation of compiled Java code to observe the flow of application behavior. On the other hand, a dynamic analysis approach was performed on a Unity Android application, which comprised a modification of the AndroidManifest.xml file, generating a debugger file, repacking the application, installing it on an Android device, and running the application. This study demonstrated the differences between native Android and Unity Android applications.

Faheem et al. [17] emphasized that it is essential for DF practitioners to equip themselves with the latest tools and procedures to cope with the rapid development of cloud computing and the talent of cyber attackers. The study highlighted the challenges of cloud forensics for mobile applications: (1) a variety of smartphone models, (2) numerous cloud storage devices, and (3) various structures and patterns for storing data. Thus, a unique forensic procedure is required for each cloud application and mobile device.

Several studies have illustrated the application of DF principles, phases, and methodologies for analyzing cloud storage applications. Satrya et al. [18] performed forensic analysis on the Android Google Drive application and applied multiple user interactions with the application: logged in to the application, downloaded, uploaded, and deleted a file. Thus, the study captured almost all the data remnants in the scenarios. Bhat et al. [19] highlighted that data remnants can be found by applying forensic analysis to the FlipDrive and Sync.com applications. These data remnants include user credentials, names of files being uploaded or downloaded, file locations on the device, and hypertext transmission protocol (HTTP) requests to the server.

Daryabar et al. [20] presented a mechanism for extracting information from cloud applications, namely Box, Google Drive, Dropbox, and OneDrive, from the Android device perspective. This study clarifies the locations and types of artifacts from Android devices. Network analysis was also conducted to retrieve Internet Protocol (IP) addresses, server names, timestamps, and provider certifications. Asim et al. [21] focused on extracting artifacts from mobile web browsers on an Android platform. They created a tool called AndroKit, by which they were able to capture forensically sound artifacts, such as user credentials, browsing history, and downloaded files.

In the following year, Daryabar et al. [22] adopted the framework of Martini et al. [23] to examine the MEGA cloud application on Android. The study discovered that the timestamps for the downloaded files were modified when compared to their original versions. However, the hash values of both files remain the same, and this scenario is also true for the uploaded files. Additionally, the work also captured the PCAP file and discovered information, such as URLs, IP addresses, server names, and certifications.

Martini et al. [24] introduced a method to acquire a live collection of Android devices for multiple cloud applications. This study showed that the box application encrypts its files with the key found in its internal folders. Moreover, the study highlighted that the tested cloud application stores the file metadata in the SQLite database.

The taxonomy for the forensic analysis of 31 Android cloud storage applications can be found in Amine et al. [25]. The study applied XRY version 6.16.0 to analyze Android cloud storage applications on Asus Nexus-7 with Android Kitkat OS. The PCAP files were also collected together with device analysis.

Network interception of mobile applications has been included in several empirical forensic studies. Jnox et al. [26] performed forensic analysis for a dating application on Android and iOS platforms. They used Fiddler Everywhere as a proxy to capture the HTTP/HTTPS traffic between the device and dating application server. Fiddler Everywhere failed to decrypt network packets for Android, but succeeded with iOS. Instead of Fiddler Everywhere, the Packet Capture Android application intercepts the network packets of the dating application. For the Android experiment, multiple rooted devices were used. The study obtains user information, plain text messages, and audio files by intercepting network packets.

Fiddler Everywhere Web proxy can implement a man-in-the-middle (MITM) attack to intercept and modify HTTP requests and decrypt the transmitted HTTPS packets of the Android application at runtime. The certificate of Fiddler Everywhere was installed on an Android device to ensure successful network interception. This scenario was explained by Jo et al. [27], in which Fiddler Everywhere was applied to examine the network traffic between cloud systems and Android devices when running an IoT system called an AI speaker ecosystem. The AI speaker ecosystem contains devices, appliances, and a cloud system that communicates through a network. This study applied network forensic analysis to mobile applications on Android devices and retrieved personal information, including email, user ID, profile image URL, birthdate, and gender.

It has become clear from the literature that Android cloud storage applications are a massive source of information for forensic investigators. Additionally, the number of cloud storage service providers (CSSPs) is increasing. Many of these applications must be addressed in academic research to identify evidence and present extraction and analysis methods.

This study significantly advances the field of mobile cloud forensics by providing a novel comparative analysis of five understudied cloud storage applications-BigMind, Degoo, FEX NET, File.fm, and Koofr-using network packet analysis as the primary forensic methodology. Unlike prior research, which predominantly focuses on mainstream applications such as Google Drive, OneDrive, and Dropbox, this investigation uniquely juxtaposes forensic artifacts retrieved from both Android applications and mobile web browsers, illuminating the critical security differentials between access methods. The methodology extends beyond previous approaches by systematically analyzing diverse user scenarios and capturing a comprehensive forensic footprint for each application. This study demonstrates the relevance of organizational BYOD frameworks by revealing extractable sensitive information-user credentials, file metadata, and access tokens-thereby providing security practitioners with actionable intelligence to identify unauthorized data access. Moreover, while earlier studies applied network interception techniques to specific application types, this study contributes to a systematic evaluation of cloud storage applications' security protocols, revealing how advanced protection mechanisms can obstruct network packet acquisition and decryption, thus establishing crucial benchmarks for the security assessment of emerging cloud storage services.

3. SECURITY IN ANDROID AND MOBILE FORENSICS

According to Meng et al. [28], the Android platform accounts for a vast majority of the smartphone market. This is supported by Wu et al. [29], who states that Android applications have become essential to people's lives and affect almost all people's habits. Therefore, security issues in Android applications must be considered to protect users from malicious intentions. However, Varol et al. [3] mentioned that relying on security standards and encryption techniques is inadequate to protect smartphone users from cyber threats and cyberattacks. Furthermore, the Open Web Application Security Project (OWASP) declared that Android has become the target for tampering with its application by reverse engineering owing to Android's characteristic as an open-source platform. This can be achieved by disassembling the Android package (APK), decompiling and changing its code, and then recompiling it to run on an Android device or emulator. However, the exact mechanism is not available for iOS applications [30].

Android implements restrictions to increase its security. One of the mechanisms is to set up a sandbox for each application to segregate it with other applications [31]. This application sandbox is installed in the kernel and uses user identification (UID) to segregate the interaction between applications, users, and the system. In addition, the Android system provides applications with a utility called permission to maintain privacy. Thus, if the application needs to access sensitive information, it requests permission from the user [32].

Permissions are divided into several protection categories: normal, signature, dangerous, and special. The OS directly grants normal permission as it has no significant hazard to privacy. Signature permission was applied during the installation of the application. By contrast, dangerous permission relates to an application trying to

access or modify a user's private information, data, or resources. In this situation, permission must be declared in the manifest file, and permission is required to show a request screen for the user at runtime. If the user approves of such a request, the information can be retrieved and modified. An example of dangerous permission is when an application requests to read the contact from the device. Unique signatures may affect the Android API level, which is lower than level 23. In such a situation, permission can be granted without user consent; thus, it permits the application to forcibly display a pop-up screen and view unwanted media or advertisements [33].

Owing to the threat of security attacks on Android devices, there is a need for an unbiased study of the behavior of Android applications. The purpose is to observe the mechanism of preserving and treating data and user information through the application. A study on the types and amount of data retrieved from Android devices is essential for application developers, users, and law enforcement officers.

NIST defines mobile forensics as "the science of recovering digital evidence from a mobile device, with accepted methods and under forensically sound manners" [34]. A piece of forensically sound evidence is a well-preserved artifact that is neither damaged nor altered. Because it will be applied in the investigation, acquiring it from a digital device must be acknowledged by the Court of Law [35]. McKemmish et al. [36] outlines the main attributes of forensically sound evidence as follows: (1) a bit-by-bit clone of the digital disk, (2) authenticity is ensured using a hashing algorithm, and (3) documentation of the investigation activities. The reported activities on the system may help unauthorized users discover malicious activities or find a responsible actor in any activity [37].

In addition, a chain of custody must support the investigation to ensure the acceptability of evidence in the court of law. The chain of custody may involve listing evidence with its details chronologically (e.g., type of evidence, time and place of seizing, method of acquisition, person in charge, and activities conducted). When involving mobile forensics, some studies [38, 39] recommend using extensible markup language (XML) as a format for recording details on the chain of custody. This is because it guarantees that the records are readable and interoperated. However, it depends on the investigators to implement the format that is suitable for them.

While dealing with mobile devices, [40] insisted that the investigator should avoid tampering with the evidence and minimize any changes while extracting it. Furthermore, protection against anti-forensic methods is recommended to protect evidence. As a precaution in mobile forensics, it is mandatory to protect artifacts from alteration; otherwise, there will be a loss of integrity in the evidence. The investigator must have adequate details about the case to steer him toward the most relevant evidence. For example, if an investigation is related to fraudulent emails, it must focus on artifacts related to the use of emails, such as web browser history. If the case focuses on data stored in illegal media, the investigation will focus on hard drive images and memory. Therefore, it is recommended to use any technique that preserves the evidence integrity, such as using a hashing algorithm for the acquired evidence, comparing it with the hash value of the original evidence before the acquisition and analysis phases [19].

4. EXPERIMENT IMPLEMENTATION FOR CLOUD APPLICATION

The selection of appropriate tools for digital forensic investigations is crucial to ensure the reliability and validity of the findings. In this study, Fiddler Everywhere was chosen as the primary network packet analysis tool owing to its comprehensive capabilities in capturing and decrypting HTTP/HTTPS traffic, which is essential for examining cloud storage applications. Unlike Wireshark, which provides broader protocol coverage but less specialized HTTP analysis, Fiddler Everywhere offers superior capabilities specifically for web debugging and HTTP/HTTPS traffic manipulation [4]. Additionally, Fiddler Everywhere's ability to implement man-in-the-middle attacks for intercepting encrypted HTTPS traffic from Android devices provides a significant advantage for the forensic analysis of mobile applications, as demonstrated by Jo et al. [27] in their study of AI speaker ecosystems [27]. For APK modification, APK Editor Studio was selected for its specialized capabilities in editing Android package files and network security configurations, which enabled the research team to bypass security restrictions and facilitate the capture of encrypted traffic, similar to the approaches used in the recent Android security research by Meng et al. [28]. This toolset combination optimizes the forensic workflow for cloud storage investigation, while maintaining the scientific rigor necessary for evidence collection.

This study adopted mobile forensic phases to retrieve several SaaS cloud storage artifacts on an Android platform. It is based on the recommendation of the Open Web Application Security Project (OWASP) guideline [41] for mobile security testing and the NIST forensic guideline [34]. Five SaaS cloud storage applications were selected for the experiments, as listed in Table 1. Account registration for each cloud storage application was performed

on its website through a web browser. All of the created accounts were free versions and used "aqeeltamimi@hotmail.com" as a logging username, and each application had a different password.

Table 1: Cloud storage application details

Application Name	Android version
BigMind	3.0.7.600
Degoo	1.57.39.200709
FEX.NET	3.0.5
Files.fm	3.6.0
Koofr	3.8.2

The dataset for each application was prepared, and contained multiple files in different formats. This dataset was used in the investigation process to search for artifacts. The folders and files of the dataset are presented in Fig. 1.

Name	Kind	Size
bigmind	Folder	
bm_audio.mp3	Folder	86 KB
bm_image.jpg	JPEG audio	36 KB
degoo	Folder	
degoo_audio.mp3	MP3 audio	1.6 MB
degoo_image.png	PNG image	1.6 MB
filesfm	Folder	
ffm_audio.mp3	MP3 audio	539 KB
ffm_image.jpg	JPEG image	47 KB
ffm.word_document	Folder	
fx_audio.mp3	MP3 audio	148 KB
fx_image.jpg	JPEG image	32 KB
flex	Folder	
ffm_audio.mp3	MP3 audio	64 KB
kfr_image.jpg	JPEG image	46 KB
koofr	Folder	
kfr_audio.mp3	MP3 audio	64 KB
kfr_image.jpg	JPEG image	46 KB

Fig. 1: Create files and folders for each cloud application, which will be applied in the user scenario.

In terms of the testbed environment, the experiment was conducted on a machine with macOS Mojave 10.14.6 (18G3030) operating systems, 2.2 GHz Intel i7 processor, 16 GB 1600 MHz DDR3 RAM, and Intel Iris Pro 1536 MB graphic card. A list of software or tools used for Android simulation and analysis is presented in Table 2.

Table 2: Software tools for experiments in user scenarios.

Name	Version	Purpose
Android Assistant	23.76	Extract Android APK files.
Android Debug Bridge (ADB)	1.0.41	Transfer files between workstation and Android emulator.
Android Virtual Device (AVD)	-	Create Android emulator.
APK Editor Studio	1.4.0	Edit Android application package (APK) files.
Fiddler Anywhere	1.0.1	Capture network packet.
Pixel 3a API 27 Emulator	API 27	Test emulator device.
Android Chrome web browser	-	Test cloud scenario activities and install certificate authority (CA).

Before simulating the scenario on the Android cloud application, the testbed underwent five preparation steps:

- Extracting Android cloud storage application (APK) files: We performed Android cloud storage application (APK) file extraction using the Android assistant installed on the Android emulator. In this study, the Android emulator environment was Pixel 3a API 27 with Android OS version 8.1. This process was required to access the APK file of each cloud storage application for a subsequent reverse engineering process.
- Reverse engineering of the extracted APK files: This implies editing the APK, adding or modifying the network security configuration file, and recompiling the APK using APK Editor Studio. In addition, the network security config.xml file was allocated to the resources in the values folder.
- Installing APK and datasets: A modified APK file was installed for each cloud storage application and copied to the Android emulator. Simultaneously, the dataset folder for the cloud storage analysis was copied to the emulator's storage using the ADB tool.
- Installing the Certificate Authority (CA) file of Fiddler Everywhere: Using the Chrome web browser on the Android emulator, we used the link (ip4.fiddler:8866) to install the CA. Port 8866 was set up for Fiddler Everywhere.
- Modifying WiFi settings: This involves including the laptop's Internet Protocol (IP) address in the wireless connection settings of the Android emulator, which would allow Fiddler Everywhere to capture the network packets that were transmitted to the emulator.

In this experiment, the adopted forensic technique was real-time network traffic analysis on a virtual mobile device. Therefore, some of the regular mobile forensic phases were skipped, such as device seizures and hashing evidence. The focus is on the acquisition and analysis phases. An experiment was conducted for each CSSP on its website using a Chrome mobile web browser and Android cloud storage applications. The test included five user scenarios that produced network packet sessions on Fiddler Everywhere: captured, exported, and encrypted. The scenarios are listed below:

- Login using predefined user credentials with aqeeltamimi@hotmail.com as the username.
- Upload a file from the dataset folder on an Android emulator
- A file was downloaded from the user account.
- Delete a file from the user account.
- Logout user account.

The flow of this scenario is illustrated in Fig 2. The upload, download, and delete file scenarios were applied to the files in the dataset either as a group or individually. The acquisition stage of the proposed network forensic approach was performed at runtime, represented by capturing the Fiddler Everywhere session and then exporting the captured session as an SAZ file. The naming of the saved files was started by the CSSP initials, followed by the side of the testing (i.e., either APK or website) and the scenario's name (e.g., ffm apk login.saz, degoo website download.saz).

After saving all scenario sessions, a network analysis was implemented by viewing the saved SAZ file content. SAZ is a compressed file comprising three files: a) session# c.txt, which contains the HTTP/HTTPS requests of the saved session. b) Session # m.xml, which contains session information, including session ID, host IP, client device IP, request and response time, and others. c) Session # s.txt, which contains the HTTP/HTTPS response of the save session. The captured and extracted packets were encrypted with AES256 to protect them from modification and unauthorized access.

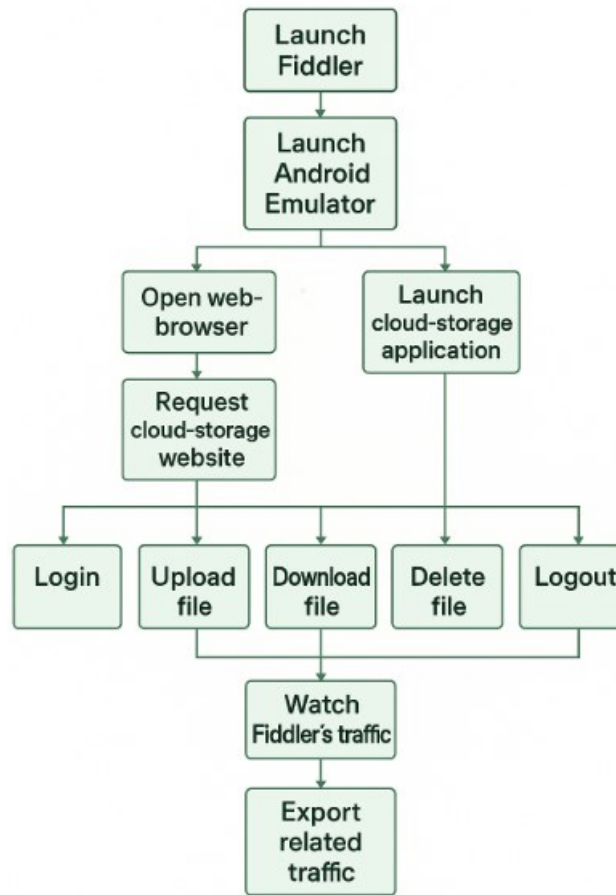


Fig. 2: Created user scenarios for activities in cloud storage applications and web browsers. This result was replicated for each CSSP.

5. RESULT OF ANALYSIS

Before experimentation, the APK files of the five cloud storage applications were edited using the Android APK Editor Studio to modify the network security configuration file. The purpose of this step was to authorize Fiddler Everywhere's certificate authority (CA) and capture and decrypt the application's network traffic. For some cloud storage applications that use their network security configuration, Fiddler Everywhere failed to crack the captured HTTP/HTTPS traffic, even after editing the network security config.xml file. The results of APK reverse engineering are listed in Table 3. Degoo and FEX.NET used network security configuration files.

Table 3: Reverse engineering results for cloud applications.

Application Name	Own network security configuration	Details
BigMind	No	-
Degoo	Yes	Allow trusting customs CA for specific domain.
FEX.NET	Yes	Allow trusting custom CA when the application is tagged as debug able
Files.fm	No	-
Koofr	No	-

Degoo APK uses a custom network security config file, which is modified to accept the CA from a specific domain. Hence, it prevents Fiddler Everywhere from decrypting the captured network traffic. Similarly, FEX.NET APK uses a custom network security config file, which is modified to accept CA when the application is tagged as debugging. A debug able tagging of the application is made in the AndroidManifest.xml file by adding the following line to the application tag -android:debuggable="true."

The overall results for the artifacts of Android cloud storage applications are presented in Table 4. It appears that artifacts can be retrieved from the scenarios of downloading, uploading, and deleting files in Android cloud storage applications.

Table 4: Artifact retrieval from user scenarios for cloud applications

Cloud Access	Login artifact	Upload artifact	Download artifact	Delete artifact	Logout artifact
BigMind (APK)	No	No	No	No	No
BigMind (web browser)	Yes	N/A	Yes	Yes	Yes
Degoo (APK)	No	No	Yes	No	No
Degoo (web browser)	Yes	Yes	Yes	Yes	No
FEX.NET (APK)	Yes	Yes	Yes	Yes	Yes
FEX.NET (web browser)	Yes	Yes	Yes	Yes	Yes
Files.fm (APK)	No	Yes	No	No	No
File.fm (web browser)	Yes	Yes	Yes	Yes	Yes
Koofr (APK)	Yes	Yes	Yes	Yes	Yes
Koofr (web browser)	Yes	Yes	Yes	Yes	Yes

5.1 Result of analysis on Koofr application

For the login scenario, the approach was able to detect the related artifacts to assist forensic investigation from the perspectives of Koofr APK and the web browser. In the simulated scenario with Koofr APK, the approach retrieved artifacts such as user credentials (e.g., username and password), client ID, request link, open standard for authorization (OAuth) tokens, redirect uniform resource identifiers (URIs), and other significant information. Artifact retrieval from the web browser also showed a similar result to the Koofr APK, with no significant difference. Figs 3 and 4 outline the artifacts retrieved from the web browser and the Koofr APK, respectively.

The results of the artifact retrieval from the uploading scenario are shown in Fig 5 and 6. For the Koofr APK, Fiddler Everywhere can detect the HTTP POST request link along with the header request, file name, file type, file size, file hash value, and file content. However, the file content for text documents, MP3, and PNG files was encoded and could not be read as plaintext. Another critical piece of information is the mount ID, which represents the folder name of the user files. For the Koofr website, the detected request and response comprised the same information as the APK, with no significant differences.

The process of downloading files from the Koofr account was detected through network packet tracking for the APK and web browsers. After analyzing the requests and responses, it was noticed that there was no significant difference between the two downloading files. Hence, the captured network packets comprise the entire request path, file name, file type, file size, file hash, request ID, user ID, HTTP request, entity tag, and encoded file content, as illustrated in Fig. 7.

After implementing the delete scenario for both the web browser and the APK, it was observed that the two methods presented similar results, as shown in Fig 8 and 9. The results indicated that the requested link for deletion was detected, representing the removed file path with all its attributes, such as file name, file type, and mount ID (i.e., identifying the user's account files). This mount ID is critical information, as it also exists in other scenarios (i.e., upload and download files). With this finding, the mount ID can present two outcomes for DF investigation. It can be applied to find any correlation between file activities such as uploading, downloading, and deleting. However, at the same time, it can be used as an anti-forensic agent where the suspect can apply this information by modifying or creating a new deletion request to delete any file. If this happens, there is a possibility of tampering with pieces of evidence.

In terms of the logout scenario, the results showed different behaviors between the APK and web browsers. When a user clicked on a logout on the website, a request was detected with an attribute in the path of the link. Once the browser navigated the login screen, another request with a login attribute and empty credentials was detected. In the case of the APK, only an empty login request is detected, and the user navigates to the application's login screen.

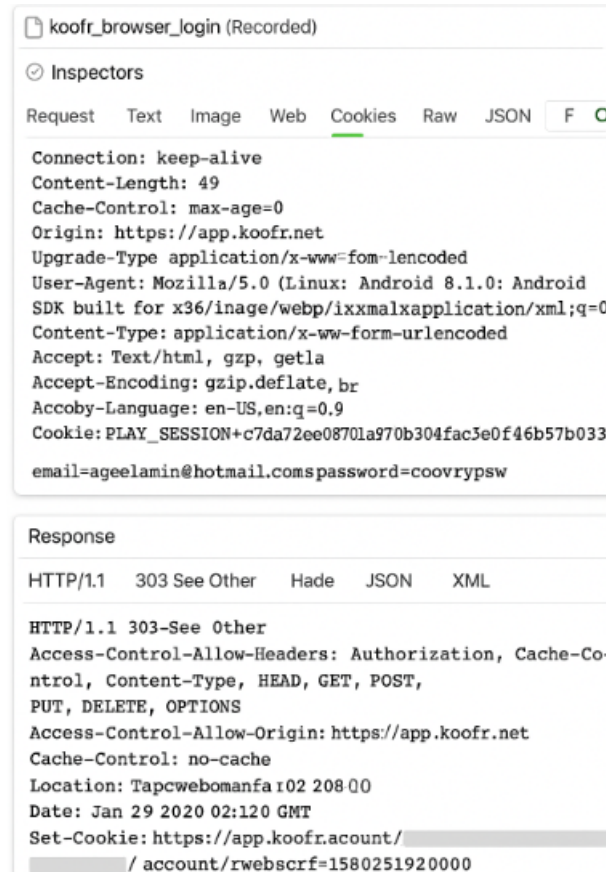


Fig. 3: Artifacts for Koofr login session (web browser)

Key	Value
redirect	/oauth2/auth? redirect_uri=urn:ietf:wg:oauth:2.0:oob&resp...
csrfToken	1bdf9e8a44c8ea692d0a2428a8f8e1b87a30a... 1597285279598-f2595407cd11b5748453214e
Key	Value
email	aqeeltamimi@hotmail.com
password	XXXXXXXXXX

Fig. 4: Artifacts from Koofr login session (APK)

```
POST https://app.koofr.net/content/api/v2/monts/fa982797-95e4-4ff4-8a/
files/putPeth
Authorization 2.21
Authorizator: Bearer ODBKLPFTCWNZGBERFZQDSSZWZGT7KUVECBTGA3PCREA3ZXQ
Accept-Encoding: gzip
Content-Type: multipart/form-data; boundary=JavaKooFrp13-fbd441b2-555d-
41d0-82c9-9b32e03cb50a
Transfer-Encoding: chunked
User-Agent: Dalvik/2.1.0 (Linux; U; Android 8.1.0
Connection: Keep-Alive

HTTP/1.1 200 OK
Content-Type: application/json
Date: Thu, 13 Aug 2020 04:23:05 GMT
X-Request-Id: 55aa310b-be91-465e-4c5cc-11667d0d44a333
X-User-Id: dc682f03-4a4f-48c6-492c-82374bd72e6
Content-Length: 155
{"name":"kfr_audio.mp3", "type":"file","modified":1597258388594,"size":
"hash":"dfebd3578ea30acc518d3666abaaafd"},
"tags":[]
```

Fig. 5: Artifact from Koofr upload session (APK)

```
POST https://app.koofr.net/content/api/v2/monts/
fa902797-95e4-4f44-8a82-cbb
429652d69/files
Host: app.koofr-net
Connection: keep-alive
Origin: https://app.koofr.net
X-Koofr-Version 2.1
User-Agent: Mozilla/5.0 (Linux; Android 8.1.0;
Android SDK built for x86 Build/OSM1.180081.)
Content-Type: multipart/form-data; boundary=-
WebKitFormBoundarye1ehsbllCRyGtA
Accept: */*
Referer: https://app.koofr.net/app/storage/
fa902797-95e4-4f44-8a82
Accept-Encoding: gzip, deflate, br
Cookie: PLAY_SESSION=f9210f3-A3Sa55c9:17a9Zc143id00aJ5a=

Responder
Headers Text Image Web Cookies XML

HTTP/1.1 200 OK
Access-Control-Allow-Headers: Authorization, Cache-Control
Content-Type, If-Modified-Since, If-None-Match
Access-Control-Allow-Methods: HEAD, GET, POST, PUT, DELETE,
OPTIONS
Access-Control-Allow-Origin: https://app.koofr.net
Content-Type: application/json
Date: Tue, 04 Jul 2020 11:48:13 GMT
X-Request-ID: f9210f3b-4a56-9db9-54573bd00aaae01
Etag: c34ba03d3f0601ac346bc0e2-8271dab72e6
"name":"xfr_word_document.docx","type":"file","19"2614607",
"size":11569,"id":60dc8
```

Fig. 6: Artifacts from Koofr upload session (web browser)

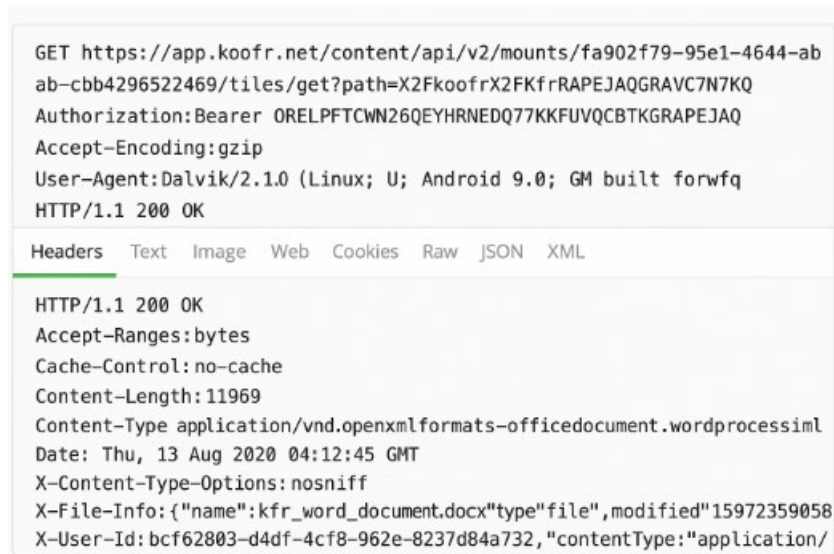


Fig. 7: Artifacts from Koofr download session (APK)

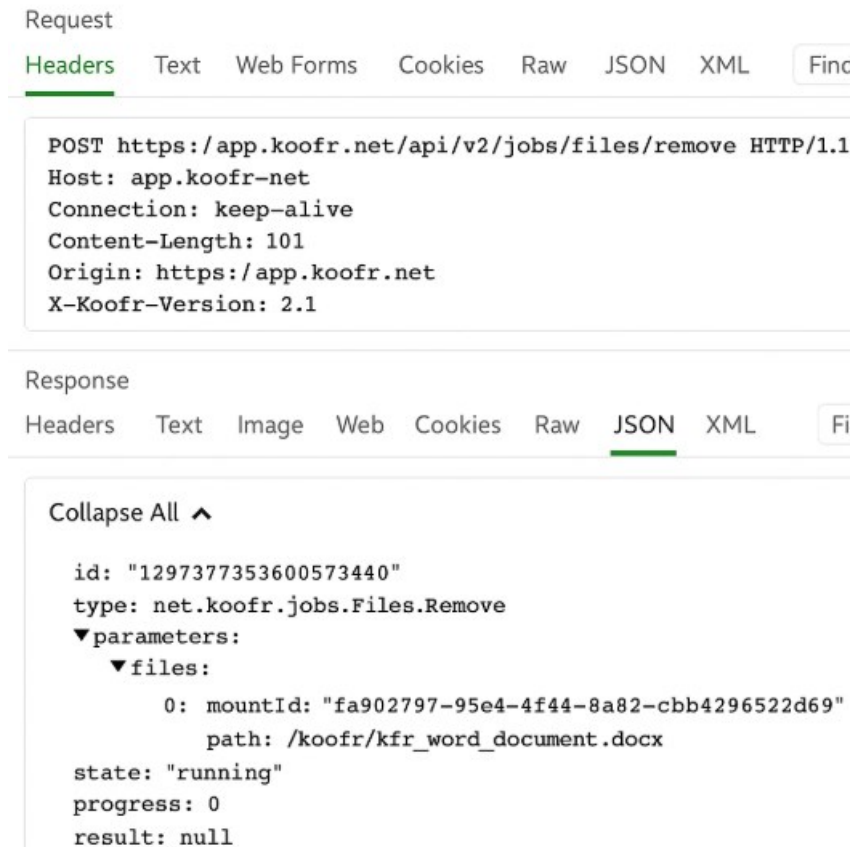


Fig. 8: Artifacts from Koofr delete session (web browser)



Fig. 9: Artifacts from Koofr delete session (APK)

5.2 Result of analysis on BigMind application

As shown in Table 4, Fiddler Everywhere failed to detect any decrypted versions of all implemented user scenarios for the Android APK. However, it can retrieve information on the scenario from the web browser, except when uploading the file. In login using the BigMind APK, each time the login is requested, a new session is detected, but it does not hold any valuable information. It only shows the user agent as the library's name used to request an HTTP called okhttp. In contrast, when login in from the web browser (Fig 10), Fiddler Everywhere detects and decrypts the login session, displaying the request path, username, and password.

In the upload scenario, BigMind does not provide the upload file option through a mobile web browser. Using the application APK, the uploading session is encrypted and cannot be read in plaintext.


hfProductUserInfo
hfProductLink
hfUserName
txSocialEmail
txtEmail aqeeltamimi@hotmail.com
txtPassword 
cboxStayLogin

Fig. 10: Artifacts from BigMind login session (web browser)

For the downloading scenario, it was found that the information on downloaded files could be detected by Fiddler Everywhere, but it was URL-encoded. Thus, after decoding it with a URL decoder, it seems to hold information such as the downloaded file name, file path, file ID, and others can be obtained. The results for the download scenario are shown in Fig 11. As for the deleting scenario, the related session can be captured, after which it shows the deleted file as a hashed object, and the HTTP Action attribute is set as DeleteFile (i.e., as shown in Fig 12). Finally, through network packet tracking with Fiddler Everywhere, the logout session in BigMind can be retrieved, which comprises the HTTP request path, but without any other sensitive information.

URL Decoder/Encoder

```
Action=DownloadFiles&DownloadedFiles=["FileName:"  
"ffm_word_document.docx"  
FileAmazonPath:"Zoolz/109140/EC/64B758842DABD545DF37Ac"  
FileParentPath:"1/40/  
Download/filesfm/  
FileSize:"11956,"1671089536 30"  
FileModificationDate:"07/22/2020"  
FileModificationDateTime:"7/22/2020, 8:00:04 PM,"  
FileModifisationDateTime:""  
FilePathHash:""  
FilePath:"1/40/Download/filesfm/ffm_word_document.docx"  
MachineID:"824914"  
IsMachineMacc>false,"FileNewName:""  
FileNewParentPath="Internal Storage  
/Documents/Download/filesfm)
```

Fig. 11: Artifacts from BigMind download session after decoded

```
POST https://intelli.zoolz.com/Services/SearchDownloadHandler.ashx HTTP/1.1
Host: intelli.zoolz.com
Connection: keep-alive
Content-Length: 799
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Linux; Android 8.1.0; Android SDK built for x86)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.111 Mobile Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Origin: https://intelli.zoolz.com
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://intelli.zoolz.com/Discover
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9

Action=DownloadFiles&DownloadedFiles=58B7B2A2F1&Name=%22%3A%22
tfm_word_document.docx
%22%2C%22
FileAmazonPath=%2F%2Fzoolz%2F10891402FEF20FC64875B8442DAB5D45DF37AB6E73C%22%2C%22
FileParent=%2F
Download
files%2F
FileSize%3A311956%2C%22
FileModificationDateTime=2020-08-13T08%3A04PM%22%2C%22
FileCreatedDateTime=2020-08-13T08%3A04PM%22%2C%22
FileReference=%2F
tfm_word_document.docx
tfm_word_document.docx
%2F%2Fzoolz%2F10891402FEF20FC64875B8442DAB5D45DF37AB6E73C%22%2C%22
Internal%2FStorage%2FDocuments%2FDownload%2Ffiles%2F58B7B2A2F1
```

Fig. 12: Artifact from BigMind delete session (web browser)

5.3 Result of analysis on Degoo application

As shown in Table 4, by analyzing the network packet through Fiddler Everywhere, all the user scenario artifacts except for logout can be detected if the user uses a mobile web browser for Degoo. From the APK perspective, it fails to capture artifacts from all user scenarios, except for the downloading files.

The results from the login scenario showed that tracking artifacts through network packets did not reveal any significant artifacts. In the case of Degoo APK, Fiddler Everywhere could not detect a login request. Instead, it captures post-login requests that show login success. These requests contain information such as the device type, OS version, and other insignificant details. In contrast, tracking the network packet in the Degoo login session with a mobile web browser can capture the login HTTP request path without any user credentials.

The upload scenario had the same results as the login for the Degoo APK, indicating that Fiddler Everywhere did not detect the request from the application. Instead, it detects requests from libraries used in applications with insignificant information. In the case of interacting with Degoo through a mobile web browser, packet capture with Fiddler Everywhere can retrieve the upload session, comprising the list of uploaded files, their names, sizes, and types, as shown in Fig 13.

```

operationName: "SetUploadFile2"
▼ variables
  Token: eeyJhbGColllUzl1NiislnR5cckcTlKpxXJ9, eyj124Y
  5VUCluAXYAA
▼ FileInfos
  [0] ParentID: "13055387813"
    ▪ Size: 115552
    ▪ CreationTime: "15962890000"
    ▪ Checksum: "CRFqoXHYm5IOoOclMXDz MMZZzIF18AA"
  [1] Name: "degoo_audio.mp3"
    ▪ ParentID: "13055387813"
    ▪ Size: 119577
    ▪ CreationTime: "1597019881000"
    ▪ Checksum: "CTTILSOtZAeU5+ ZLyCqogNKQ3SSRRA"
  [2] Name: "degoo_image.png"
    ▪ ParentID: "13055387813"
    ▪ Size: 163726
    ▪ CreationTime: "1597030549000"
    ▪ Checksum: "CTVoSLxoAeU5+ ZLyCqOCtaRaq
query: mutation SetUploadFile2 ($Token: String!,
$FileInfos: [FileInfos: [FileInfoUpload2]]) { seUplodafe2)

```

Fig. 13: Artifacts from Degoo upload session (web browser)

In the download scenario using the mobile web browser and APK, network packet tracking with Fiddler Everywhere captured significant artifacts, including information about the downloaded file, its contents, requests, and file names, as shown in Fig 14. In the deletion scenario with a mobile browser, network packet tracking revealed several vital data: function name for deletion, file ID, and hashed tokens. Finally, the Degoo APK does not provide functionality for the logout, whereas when using the mobile web browser, no network packet is detected with Fiddler Everywhere.

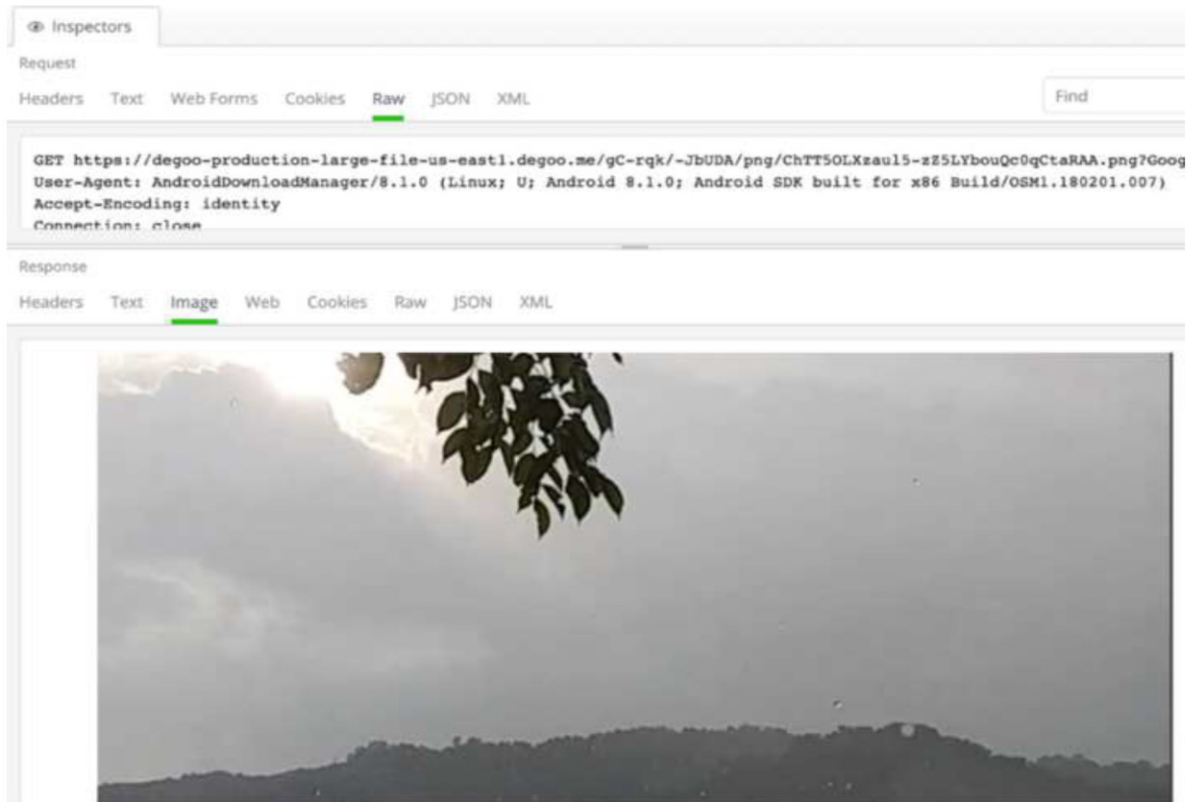


Fig. 14: Artifacts from Degoo download session (APK)

5.4 Result of analysis on FEX.NET application

As shown in Table 4, the network packet tracking for FEX.NET APK and the mobile web browser can detect all user-scenario artifacts. From the perspective of the login scenario, network packet tracking can capture critical and sensitive information regarding the login session for FEX.NET, as shown in Fig 15. This information comprises the usernames, passwords, tokens, emails, user IDs, and HTTP request paths.

In addition, the upload file scenario revealed the possibility of retrieving significant artifacts such as filenames, types, sizes, and directory IDs on the server-based and authorization tokens. On the other hand, artifacts such as download links, filenames, type, and file contents can be retrieved through network packet tracing, as shown in Fig 16. The HTTP request path for file deletion and deleted file ID can be obtained through simulation and analysis of the network packet for the delete file scenario. Finally, the requested link and authorization token are revealed by capturing the network packet during the logout scenario.

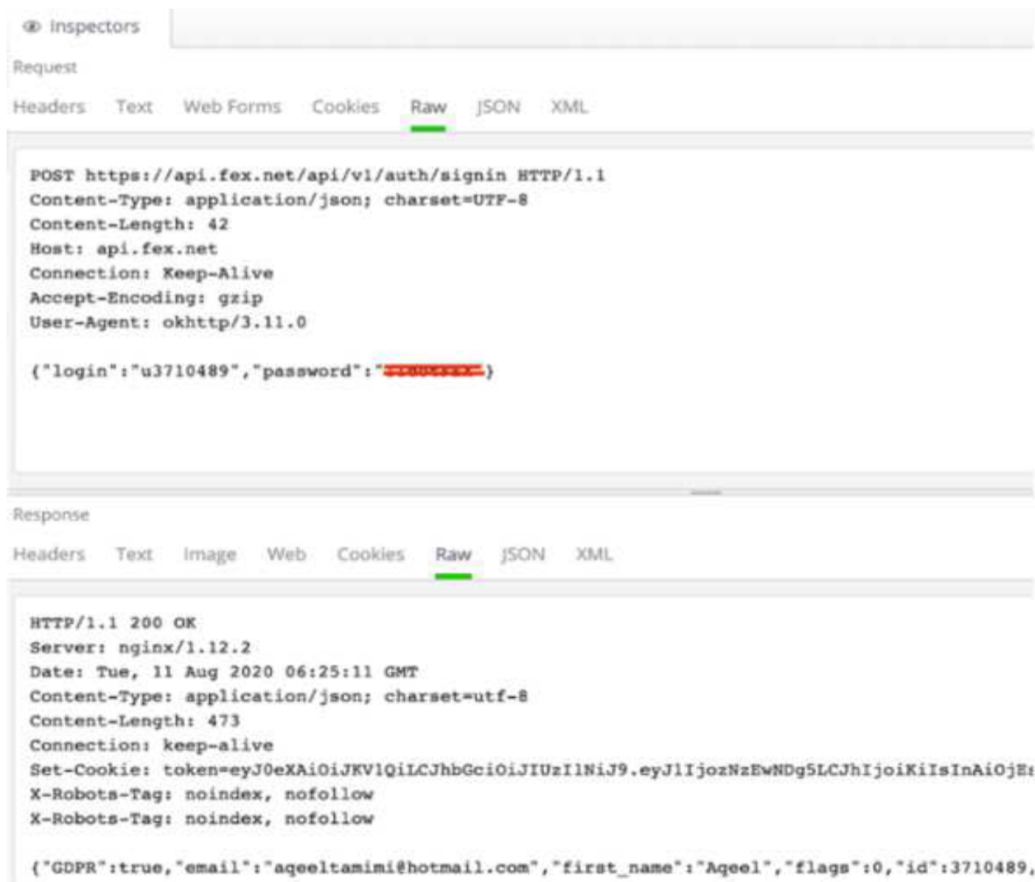


Fig. 15: Artifacts from FEX.NET login session (APK)

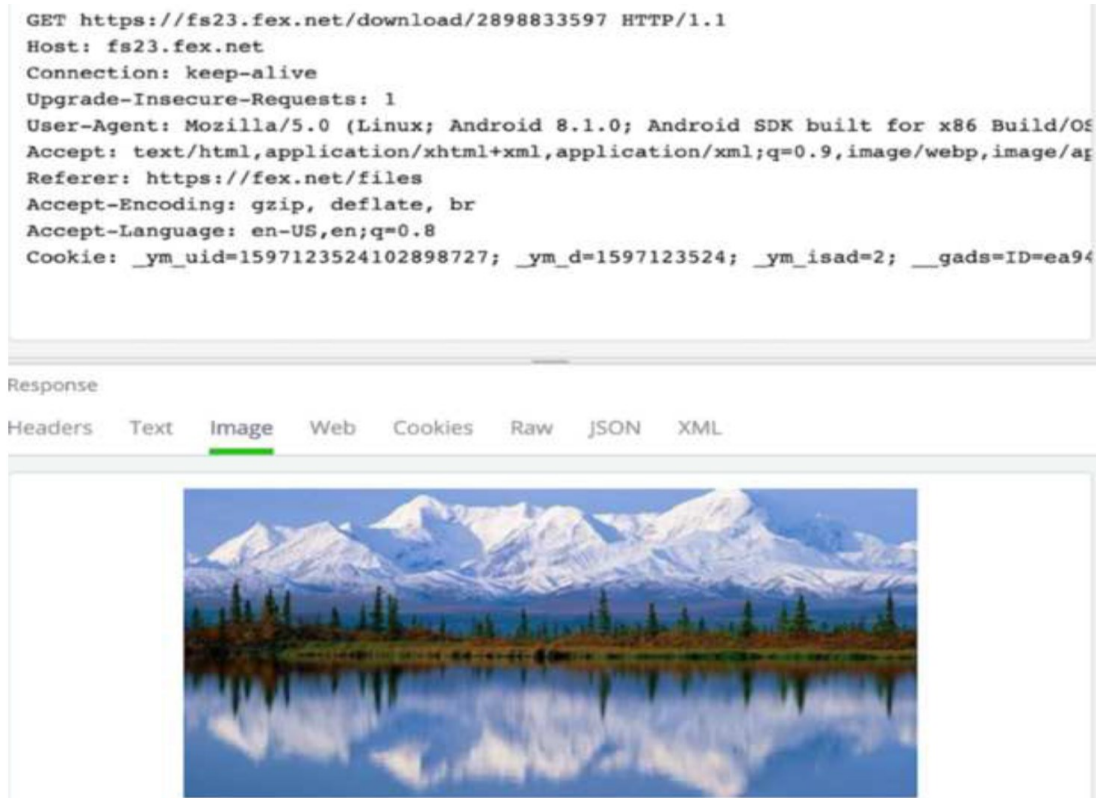


Fig. 16: Artifacts from FEX.NET download session (web browser)

5.5 Result of analysis on Files.fm application

As shown in Table 4, network packet tracing can capture all the user scenarios' artifacts when accessing Files.fm through a mobile web browser while only being able to retrieve artifacts for uploading in the case of the APK version. From the perspective of the login session, network packet tracing through Fiddler Everywhere reveals the username and password. The uploading session can capture artifacts such as the requested link, filenames, type, content, and file hash, as shown in Fig 17. Similar artifacts can also be retrieved in the downloading session by relying on network-packet tracking. Both delete the file, and logout sessions do not contain any significant forensic artifacts, as the network packet only provides information on the requested link.

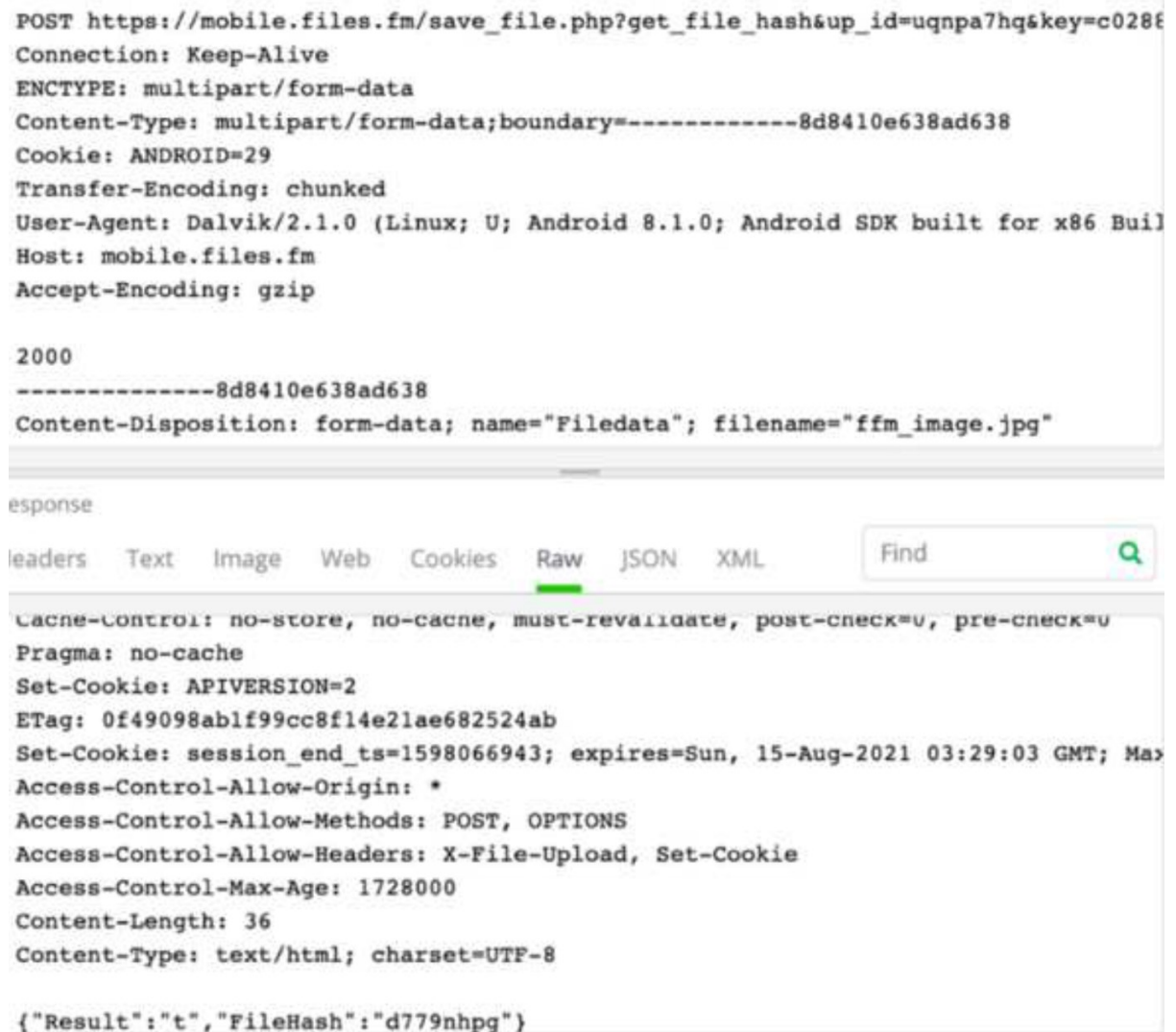


Fig. 17: Artifacts from Files.fm upload session (APK)

5.6 Overall result of analysis and discussion

The artifacts found during the analysis stage hold some significance from the mobile cloud forensics perspective because they revealed some sensitive information that had been transmitted through the network, such as the following:

- Login credentials: usernames and passwords.
- User information: user ID and email.
- Complete URL links were used to upload, download, login, logout, and delete. It also contains the HTTP/HTTPS header requests and responses.
- Folder names were used to save user files on the server.
- Contents of the transmitted files, file name, file type, file size, and file content.
- Other information such as session ID, authorization tokens, timestamps, and file hash values.

Fig18 shows a graphical representation of the results of implementing the proposed user scenarios for the five CSSPs on both the web browser and the Android APK.

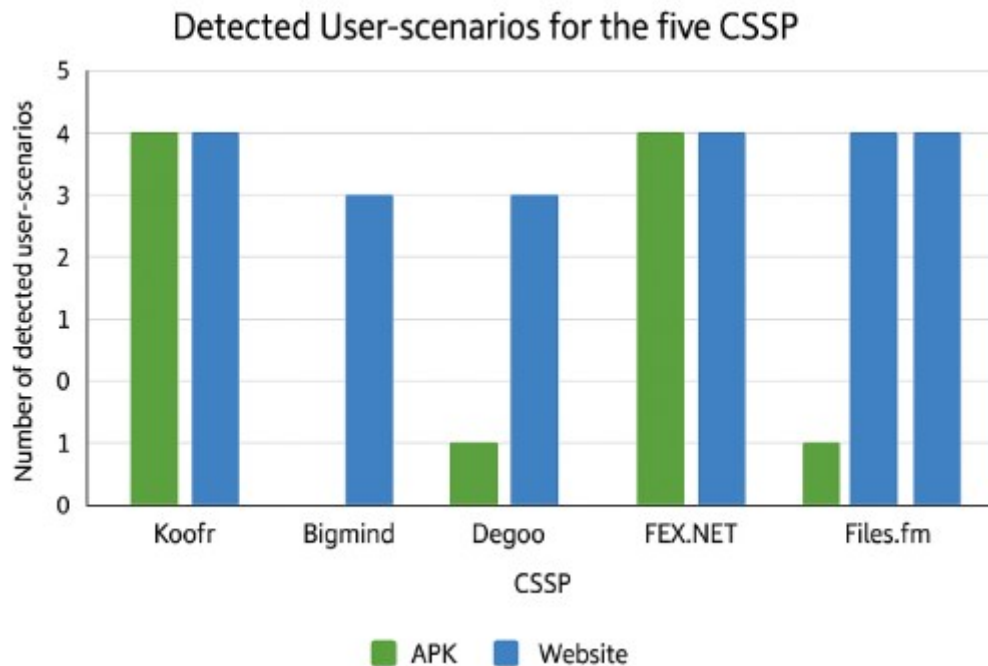


Fig. 18: Total detected user scenarios for five CSSPs

The results confirmed that network packet tracking through Fiddler Everywhere captures fewer or equal artifacts for simulated user scenarios performed on the mobile web browser and application APK. The Fiddler Everywhere program implicitly uses the MITM technique to capture network packets. Therefore, the failure to capture and decrypt the data in user scenarios through Fiddler Everywhere indicates that application developers may consider security measures to develop their application. This situation may negatively affect forensic investigation because it results in no significant artifacts in that type of application.

In future work, it will be beneficial to expand the scope of using real devices, add more test cases, and include more Android versions, applications, and web browsers. In addition, the same experiment can be conducted on other mobile platforms such as iOS. It is also preferable to use a trusted hashing algorithm to protect artifacts from tampering and preserve the integrity of the files. This can be achieved by calculating the hash value of the files before applying the testing scenarios and calculating another hash value for the files after being acquired by the forensic tool. We then compared both hash values to detect tampering.

6. ENHANCING BYOD SECURITY THROUGH NETWORK FORENSIC ANALYSIS OF CLOUD STORAGE APPLICATION

The proliferation of Bring Your Own Device (BYOD) policies within modern organizations generates both potential benefits and significant challenges. Although BYOD may enhance employee adaptability and efficiency, it also introduces considerable security vulnerabilities, particularly in relation to the utilization of cloud storage services. The investigation delineated in this study provides a critical framework for organizations endeavoring to oversee cloud utilization and identify unauthorized data access within BYOD contexts. By employing network forensic methodologies, institutions can attain augmented visibility in employee interactions with cloud storage applications on personal devices, thereby facilitating the proactive detection and alleviation of prospective security threats. Central to this methodology is the capture and analysis of network traffic linked to cloud storage activities, which yields a comprehensive comprehension of data flows and user behaviors.

A pivotal discovery of this study is its capacity to extract a range of artifacts from network traffic, encompassing user credentials, file metadata, and access tokens. In a BYOD scenario, surveilling for anomalous login attempts or access originating from unfamiliar geographical regions is imperative. If an employee's credentials are compromised, an adversary may endeavor to gain access to cloud storage resources from an alternate location or device. Through an analysis of network traffic patterns, security teams can discern these irregularities and swiftly probe potential breaches. The analysis also draws attention to the vital role of file metadata, including aspects such as file name, type, and size. Within a BYOD framework, such information may be instrumental in identifying

occurrences where sensitive corporate data are stored or transmitted inappropriately. As a case in point, if an employee places confidential information in a personal cloud storage account, this move might be marked for subsequent analysis. The presence of specific identifiers, such as the "mount IDs" identified in the analysis of the Koofr application, can serve as valuable forensic markers to correlate file activities and reveal unauthorized data manipulation.

Nonetheless, this study also recognizes the escalating challenges introduced by the application-level security protocols. As cloud storage providers increasingly implement encryption and customize network security configurations, conventional network forensic techniques may encounter limitations. Several evaluated cloud storage applications employ encryption and bespoke configurations, which complicate the capture of network packets. For example, this study illustrates that altering the APK files of cloud storage applications to circumvent CA certificates is more complex than initially anticipated. Organizations must embrace a multi-faceted strategy that integrates network monitoring with endpoint security solutions and comprehensive employee training initiatives. Endpoint security measures can offer supplementary insights into user activities on personal devices, while employee training can enhance awareness regarding the risks associated with cloud storage and encourage responsible data-handling practices.

To effectively enact network forensic analysis within a Bring Your Own Device (BYOD) framework, organizations must deliberate over subsequent procedural steps. Initially, it was imperative to identify cloud storage applications that were predominantly utilized by employees. This identification can be accomplished through methodologies such as surveys and network traffic examination, or by scrutinizing application usage data on devices under organizational management. Subsequently, it is critical to deploy network-monitoring instruments that are capable of capturing and analyzing traffic pertinent to these applications. Tools such as Fiddler Everywhere, as demonstrated in this research, can prove advantageous for this objective; however, organizations should also assess alternative solutions based on their unique requirements and available resources. The selection of an appropriate tool is instrumental for extracting digital artifacts by capturing HTTP and HTTPS traffic, thereby enabling the detection and analysis of artifacts in a manner that adheres to forensic integrity. Therefore, it is necessary to establish baseline usage patterns for cloud storage applications. This process involves the observation of standard user behaviors over an extended timeframe to comprehend typical data flows, access behaviors, and file types. This investigation executed a variety of scenarios, including login, upload, download, and deletion, to formulate baseline usage patterns.

The subsequent step involved delineating explicit security policies pertaining to the utilization of cloud storage in BYOD environments. These policies should delineate acceptable data-handling protocols, impose restrictions on the retention of sensitive information on personal devices, and provide guidelines for the usage of cloud storage applications. An additional vital measure is the implementation of automated alerts for anomalous activity. It is essential to configure network monitoring tools to signal deviations from established baseline usage patterns, such as atypical login attempts, substantial file transfers, or access to sensitive information from unauthorized locales. Finally, conducting regular security audits is crucial for evaluating the efficacy of a network forensic strategy and identifying domains that require enhancement. Such audits should encompass the review of network traffic logs, analysis of security alerts, and assessment of employee adherence to security policies.

The outcomes of this research further underscore the necessity of acknowledging the distinctions between accessing cloud storage services via mobile applications and web browsers. The study elucidated that through network forensics, various login credentials, including usernames and passwords, as well as user information, such as user IDs and email addresses, and complete URL links utilized for uploading, downloading, logging in, logging out, and deletion, were extracted. In certain instances, accessing cloud storage through a web browser may provide enhanced visibility for network monitoring, whereas in other situations, mobile applications might provide superior security features. Organizations should meticulously evaluate these distinctions when formulating their BYOD security policies and executing network forensic procedures. By comprehending the distinctive characteristics of various cloud storage access modalities, organizations can customize their security strategies to optimize their effectiveness.

Network forensic analysis presents a formidable methodology for augmenting BYOD security and attenuating risks associated with cloud storage utilization. By implementing the techniques and strategies delineated in this

study, organizations can acquire critical insights into user activities, detect unauthorized data access, and enforce security policies with efficacy. As cloud storage services perpetually evolve and BYOD policies become increasingly ubiquitous, network forensic analysis assumes a pivotal role in safeguarding sensitive organizational data. This study provides an overview of artifact discovery and holds significant value for organizations in selecting a particular Cloud Storage Service Provider (CSSP) with a focus on security considerations.

7. CONCLUSIONS

In conclusion, the widespread integration of cloud storage solutions has fundamentally transformed data management practices while concurrently presenting substantial security challenges, particularly for entities implementing Bring Your Own Device (BYOD) policies. Although cloud storage offers benefits, such as scalability and financial efficiency, it also creates openings for cyber threats and data leaks, making it essential to adopt effective digital forensic practices. This research addressed the imperative for DF professionals to proficiently recover and analyze data remnants from Android cloud storage applications in light of the evolving security landscape within the Android ecosystem. The primary aim was to propose a digital forensic protocol for the recovery of data remnants from five distinct Android cloud storage applications, BigMind, Degoo, FEX NET, File.fm, and Koofr, utilizing network packet analysis as the principal methodological approach.

Through the simulation of diverse user interactions, including login, uploading, downloading, and deletion, this study conducted a comparative analysis of data remnants from Android applications and mobile web browsers, thereby elucidating significant forensic disparities. The results illustrate the feasibility of extracting sensitive information, such as user credentials, file metadata, and access tokens. These findings provide DF professionals with the essential intelligence to conduct cyberattack investigations and enhance security oversight. Furthermore, the study highlighted the challenges posed by advanced security protocols in certain applications, which obstruct the network packet acquisition and decryption processes.

Ultimately, these findings contribute to the development of enhanced BYOD security frameworks, thereby enabling organizations to manage cloud utilization more effectively. Organizations can significantly enhance their security posture by identifying unauthorized data access and addressing security vulnerabilities associated with the pervasive adoption of cloud storage within the Android environment. This study enriches the expanding corpus of knowledge vital for securing cloud services and fortifying digital forensic methodologies in response to the dynamic landscape of cyber threats. Future directions may focus on the scope by including authentic devices, a broader array of testing cases, extra Android variants, distinct app versions, and various web browsers, while also considering other mobile environments, such as iOS, to attain a more rounded perspective and elevate the generalizability of these observations.

The use of Android emulators in our methodology, rather than physical devices, offers significant advantages, including controlled experimental environments, reproducibility, and cost-effectiveness, for analyzing multiple cloud storage applications simultaneously. Despite these benefits, we acknowledge the inherent limitations associated with emulated environments that may impact our forensic findings. Varol et al. (2017) [3] indicated that Android applications often exhibit different behavioral patterns when executed in emulators versus real devices, particularly regarding security protocol implementation and anti-forensic techniques. This discrepancy arises because sophisticated Android applications can detect virtualization environments and subsequently modify their behavior, potentially concealing or altering forensically valuable artifacts. For cloud storage applications, these differences may manifest as altered authentication mechanisms, modified network traffic patterns, and variations in encrypted data storage methodologies. Although our network forensic approach yields significant insights into artifact retrieval possibilities, the forensic community should recognize that experiments conducted on physical Android devices may produce different results regarding data encryption handling, security certificate implementation, and network protocol behaviors.

ACKNOWLEDGEMENT

The authors would like to thank the Ministry of Higher Education for their support through the grant FRGS/1/2023/ICT07/UKM/02/. Additionally, the author would appreciate the help from Cybersecurity Malaysia for their support to grant TT-2024-04. Finally, we express our gratitude to the Universiti Kebangsaan Malaysia and the Faculty of Information Science and Technology.

REFERENCES

- [1] P. M., Mell and T., Grance, The NIST definition of cloud computing, Gaithersburg, MD, 2011.
- [2] M., Chau and R., Reith, Smartphone Market Share [Online]. Available: <https://www.idc.com/promo/smartphone-market-share/os>, 2020, Accessed on: Jan, 15, 2021.
- [3] N. Varol, A. F. Aydogan, and A. Varol, Cyber attacks targeting Android cellphones, *2017 5th Int. Symp. Digit. Forensic Secur. ISDFS 2017*, pp. 1–5, 2017. doi: 10.1109/ISDFS.2017.7916511
- [4] Google, Google Drive, and *play.google.com* [online]. Available: <https://play.google.com/store/apps/details?id=com.google.android.apps.docs>, 2020, Accessed: Aug, 11, 2020.
- [5] N. H., Ab Rahman, N. D. W., Cahyani, and K.-K. R., Choo, Cloud incident handling and forensic-by-design: cloud storage as a case study, *Concurr. Comput. Pract. Exp.*, vol. 29, no. 14, p. e3868, 2017. doi: 10.1002/cpe.3868
- [6] R. B., Bahaweres, N. B., Santo, and A. S., Ningsih, Cloud Based Drive Forensic and DDoS Analysis on Seafile as Case Study, *J. Phys. Conf. Ser.*, vol. 801, p. 012055, 2017.
- [7] N. H., Ahmad, A. S. S. A., Hamid, N.S.S., Shahidan and K. A. Z., Ariffin, Cloud Forensic Analysis on pCloud: From Volatile Memory Perspectives. In: Miraz M.H., Excell P.S., Ware A., Soomro S., Ali M. (eds) *Emerging Technologies in Computing. iCETiC 2020. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, vol 332. Springer, Cham, pp. 3-15, 2020. doi: 10.1007/978-3-030-60036-5_1
- [8] M. N., Yusoff, A., Dehghantanha, and R., Mahmood, Network Traffic Forensics on Firefox Mobile OS: Facebook, Twitter, and Telegram as Case Studies, *Contemp. Digit. Forensic Investig. Cloud Mob. Appl.*, pp. 63–78, 2017. doi: 10.1016/B978-0-12-805303-4.00005-8
- [9] K. A. Z., Ariffin and F. H., Ahmad, Indicators for maturity and readiness for digital forensic investigation in era of industrial revolution 4.0, *Computers & Security*, vol. 105, pp. 102237, 2021. doi: 10.1016/j.cose.2021.102237
- [10] M. H. Mohd Zainudin, K. A. Zainol Ariffin, D. A. Ahmad Zainaddin, A. T. Abdul Ghani and S. N. Huda Sheikh Abdullah, "Cloud-based Data Extraction From Mobile Devices: A Preliminary Analysis," *2022 International Conference on Cyber Resilience (ICCR)*, Dubai, United Arab Emirates, 2022, pp. 01-05, doi: 10.1109/ICCR56254.2022.9995930.
- [11] M. A., Majid and K. A. Z., Ariffin, Success Factors for Cyber Security Operation Center (SOC) Establishment, INCITEST, EAI, 2019. doi: 10.4108/eai.18-7-2019.2287841
- [12] S. De Capitani di Vimercati, S. Foresti, S. Paraboschi and P. Samarati, "Enforcing Corporate Governance Controls With Cloud-Based Services," in *IEEE Transactions on Services Computing*, vol. 17, no. 6, pp. 3583-3596, 2024, doi: 10.1109/TSC.2024.3451179.
- [13] M. Ali, L. T. Jung, A. H. Sodhro, A. A. Laghari, S. B. Belhaouri and Z. Gillani, A Confidentiality-based data Classification-as-a-Service (C2aaS) for cloud security, *Alexandria Engineering Journal*, vol 64, 749-760, 2023. doi: 10.1016/j.aej.2022.10.056.
- [14] A. K., Agrawal, A., Sharma, S. R., Sinha, and P., Khatri, Forensic of an unrooted mobile device, *Int. J. Electron. Secur. Digit. Forensics*, vol. 12, no. 1, pp. 118–137, 2020. doi: 10.1504/IJESDF.2020.103882
- [15] D., Yoo, Y., Shin, S., Kim, H., Kim, S., Kwon, and T., Shon, Digital Forensic Artifact Collection Technique using Application Decompilation. 2019 International Conference on Platform Technology and Service (PlatCon), IEEE, pp. 1–3, 2019. doi:10.1109/PlatCon.2019.8668959
- [16] J., Shim, K., Lim, S. J., Cho, S., Han, and M., Park, Static and Dynamic Analysis of Android Malware and Goodware Written with Unity Framework, *Security and Communication Networks*, 2018, doi:10.1155/2018/6280768.
- [17] M., Faheem, T., Kechadi, N. A., Le-Khac, and N., An Le-Khac, N. The state of the art forensic techniques in mobile cloud environment: A Survey, challenges and current trends. *International Journal of Digital Crime and Forensics*, vol. 7(2), pp 1–19, 2015. doi:10.4018/ijdcf.2015040101
- [18] G. B., Satrya, and S. Y., Shin, Proposed method for mobile forensics investigation analysis of remnant data on Google Drive client, *Journal of Internet Technology*, vol. 19(6), pp. 1741–1751, 2018. doi:10.3966/160792642018111906011
- [19] W. A., Bhat, M. F., Jalal, S. S., Khan, F. F., Shah, and M. A., Wani, Forensic analysis of Sync.com and FlipDrive cloud applications on Android platform, *Forensic Science International*, vol. 302, pp. 109845, 2019. doi:10.1016/j.forsciint.2019.06.003
- [20] F., Daryabar, A., Dehghantanha, B., Eterovic-Soric, and K. K-R., Choo, Forensic investigation of OneDrive, Box, GoogleDrive and Dropbox applications on Android and iOS devices, *Australian Journal of Forensic Sciences*, vol. 48(6), pp. 615–642, 2016. doi: 10.1080/00450618.2015.1110620

- [21] M. Asim, M. F. Amjad, W. Iqbal, H. Afzal, H. Abbas, and Y. Zhang, AndroKit: A toolkit for forensics analysis of web browsers on android platform, *Futur. Gener. Comput. Syst.*, vol. 94, pp. 781–794, 2019. doi: 10.1016/j.future.2018.08.020
- [22] F., Daryabar, A., Dehghantanha, and K. K-R., Choo, Cloud storage forensics: MEGA as a case study. *Australian Journal of Forensic Sciences*, vol. 49(3), pp. 344–357, 2017. doi:10.1080/00450618.2016.1153714
- [23] B. Martini, K. K-R., Choo, An integrated conceptual digital forensic framework for cloud computing, *Digital Investigation*, vol. 9(2), pp. 71–80, 2012. doi:10.1016/j.diin.2012.07.001
- [24] B. Martini, Q., Do, and K.K-R., Choo, K. K. R., Mobile cloud forensics: An analysis of seven popular Android apps, *The Cloud Security Ecosystem: Technical, Legal, Business and Management Issues*, Elsevier Inc, 2015. doi:10.1016/B978-0-12-801595-7.00015-X
- [25] M., Amine Chelihi, A., Elutilo, I., Ahmed, C., Papadopoulos, and A., Dehghantanha, An Android Cloud Storage Apps Forensic Taxonomy, *Contemporary Digital Forensic Investigations of Cloud and Mobile Applications*, pp. 285-305, 2017. doi:10.1016/B978-0-12-805303-4.00015-0
- [26] S., Knox, S., Moghadam, K., Patrick, A., Phan, and K. K-R., Choo, What’s really ‘Happning’? A forensic analysis of Android and iOS Happn dating apps, *Computers & Security*, vol. 94, pp. 101833, 2020. doi:10.1016/j.cose.2020.101833
- [27] W., Jo, Y., Shin, H., Kim, D., Yoo, D., Kim, C., Kang, J., Jin, J., Oh, B., Na and T., Shon, Digital Forensic Practices and Methodologies for AI Speaker Ecosystems, *Digital Investigation*, vol. 29, S80–S93, 2019. doi:10.1016/j.diin.2019.04.013
- [28] G., Meng, M., Patrick, Y., Xue, Y., Liu, and J., Zhang, Securing Android App Markets via Modeling and Predicting Malware Spread between Markets, *IEEE Transactions on Information Forensics and Security*, vol. 14(7), pp. 1944–1959, 2019. doi: 10.1109/TIFS.2018.2889924
- [29] S., Wu, and J., Liu, Overprivileged Permission Detection for Android Applications, *IEEE International Conference on Communications*, pp. 1–6, 2019. doi: 10.1109/ICC.2019.8761572
- [30] B., Mueller, S., Schleier, and J., Willemsen, OWASP Mobile Security Testing Guide 1.1.3, Available: <https://github.com/OWASP/owasp-mstg>, 2020. Accessed: Dec, 12, 2020.
- [31] Android, Secure an Android Device, Available: <https://source.android.com/security>, 2020, Accessed: Sept, 2, 2020.
- [32] Android, Permissions overview, Available: <https://developer.android.com/guide/topics/permissions/overview>, 2019, Accessed : July, 28 2020.
- [33] IMStudio, Android: SYSTEM_ALERT_WINDOW, Available: <https://medium.com/@imstudio/android-system-alert-window-1865a96db648>, 2020, Accessed: Oct 17 2020.
- [34] R., Ayers, W., Jansen, and S., Brothers, Guidelines on mobile device forensics (NIST Special Publication 800-101 Revision 1), *NIST Special Publication 1(1)*: 85, 2014. doi:10.6028/NIST.SP.800-101r1
- [35] M., Shah, S., Saleem, and R., Zulqarnain, Protecting Digital Evidence Integrity and Preserving Chain of Custody, *The Journal of Digital Forensics, Security and Law*, vol. 12(2), 2017. doi:10.15394/jdfsl.2017.1478
- [36] R., McKemmish, When is digital evidence forensically sound? *IFIP International Federation for Information Processing*, vol. 285, pp. 3–15, 2008. doi:10.1007/978-0-387-84927-0_1
- [37] M., Nieves, K., Dempsey, and V. Y., Pillitteri, NIST SP800-12 Revision 1 : An introduction to information security, *NIST special publication (800–12 (draft) revision 1)*, 2017. doi: 10.6028/NIST.SP.800-12r1
- [38] A.H, Lone and R. N., Mir, Forensic-chain: Blockchain based digital forensics chain of custody with PoC in Hyperledger Composer. *Digital Investigation*. 2019; 28: 44–55. doi:10.1016/j.diin.2019.01.002
- [39] Y., Prayudi, A, Ashari, and T. K., Priyambodo. The pseudo metadata concept for the chain of custody of digital evidence. *International Journal of Electronic Security and Digital Forensics*. 2019; 11(4): 395–419. doi:10.1504/IJESDF.2019.102554
- [40] O., Afonin and V., Katalov. *Mobile Forensics – Advanced Investigative Strategies*. (S. G. Punja & M. Raja, Eds.), Packt Publishing, 2016.
- [41] B., Mueller, S., Schleier and J., Willemsen J. OWASP Mobile Security Testing Guide 1.1.3. 2020