Chapter 1

GRIEF IN THE GRAY ZONE: IDENTIFYING AND ANALYZING VAULT APPS

Seth Barrett, Alex Salontai, Rajon Bardhan, Gokila Dorai, Esra Akbas, and Patrick Woodell.
e-mail: sebarrett@augusta.edu, asalontai1@student.gsu.edu, rbardhan@augusta.edu, gdorai@augusta.edu, eakbas1@gsu.edu, pwoodell@augusta.edu

Abstract In the digital age, concerns about privacy have notably elevated the popularity of vault apps, especially on alternative platforms like Aptoide. Designed to conceal media, these apps often imitate familiar tools, such as calculators, serving as decoys. While they serve to enhance user privacy, they present significant challenges for digital investigators. Encrypted or deleted data, ranging from contraband to sensitive documents, becomes difficult to access. Building upon previous research from mainstream stores like the iOS App Store and Google Play Store, our study focuses on Aptoide, a significant gray market app platform. We employ a comprehensive framework named Grey-market vault app Identification, Extraction, and Forensic analysis (GRIEF). This innovative framework leverages feature extraction from app descriptions coupled with advanced machine learning techniques. Our research provides insights into the prevalence of vault apps in gray app markets and sets the stage for in-depth studies on various Android platforms.

Keywords: Digital Forensics, Mobile Forensics, Android Apps, Grey Market App Stores

1. Introduction

In the age of digital pervasiveness, the quest for privacy has led to the proliferation of applications designed to conceal user content. Termed ‘vault apps’, these applications serve as digital lockers, allowing users to hide sensitive photos, texts, videos, and other data. While, on the surface, many of these apps masquerade as mundane utilities or games, their underlying functionality is geared towards ensuring user privacy. For instance, an app that appears to be a simple calculator might double as a storage vault for private photos. In our previous work, we examined the iOS App Store [1] and the Google Play Store [2] to understand the landscape of content-hiding applications.

The rise of vault apps is particularly pronounced in third-party app markets, with Aptoide being a notable example. Such platforms, often termed 'gray markets', operate outside the purview of big tech's mainstream app stores, like Google Play and Apple's App Store, thereby escaping the rigorous scrutiny and security checks typically associated with official marketplaces. This has made gray markets a fertile ground for the proliferation of vault apps, some of which might be used for nefarious purposes.

While vault apps address genuine privacy concerns, they also pose significant challenges for digital forensics and security research. Encrypted data, cloud-synced content, and remote wiping abilities make the retrieval of concealed or deleted data a daunting task. Furthermore, the deceptive nature of these apps makes them hard to identify, let alone analyze.

This paper delves deep into the world of vault apps in the gray market, with a particular focus on apps found on Aptoide. Building upon our prior research in mainstream app stores, we introduce GRIEF, a novel system designed to identify, extract, and forensically analyze vault apps. Through a combination of feature extraction techniques and advanced machine learning algorithms, GRIEF offers a promising approach to tackle the challenges posed by vault apps in the gray market.

The ensuing sections provide a detailed exploration of our methodology, findings, and the implications of our research for the broader Android ecosystem and the digital forensics community.

2. Related Work

In the realm of Android application analysis and security, the past few years have witnessed a plethora of insightful research. These studies have encompassed a broad spectrum of techniques, with static and dynamic analyses being the most predominant. At the forefront of static analysis, Feng et al. introduced Apposcopy [4]. This framework employs a formal app description language to discern and counteract malicious app behaviors. Notwithstanding its groundbreaking approach, the framework might have benefited from a sharpened focus on the nuances of vault apps. Complementing with Feng et al.'s efforts, Arzt et al.

launched FlowDroid [3]. This open-source tool, renowned for its comprehensive static taint analysis, is adept at pinpointing privacy intrusions within Android apps. However, its arsenal misses out on specific vault app detection functionalities. Further enriching the discussion on behavior analysis, Hou et al. presented a method to reconstruct Android app behavior graphs [5]. While their methodology holds promise for unmasking clandestine app behaviors, its application to vault apps remains uncharted. Zooming into the niche of encrypted or vault apps, Votipka et al.'s [6] study stands out. Their expansive investigation into Android ransomware has elucidated myriad behavioral patterns. While past research has illuminated various aspects of Android application security and analysis, our study takes a unique trajectory by diving deep into the realm of vault apps on alternative platforms like Aptoide, bridging the existing gaps and offering machine learning based methodologies for detection and traditional forensic analysis.

3. Background and Preliminaries

This section offers a foundational understanding of Android application permissions, which is pivotal to our research. The advent of mobile technology has spurred a need for secure data storage solutions on mobile devices, resulting in the proliferation of vault applications. These apps are tailored to store and encrypt user data, fortifying it against unauthorized breaches.

Vault apps manifest in diverse avatars — from photo lockers and camouflaged calculators to covert notepads. They frequently operate under the radar, disguised as commonplace apps on a device [13]. Accessing their concealed functionalities typically necessitates a password or some other authentication method.

While these applications serve the legitimate purpose of securing personal data, they also pose a potential threat to security and privacy. This is because they often require a wide range of permissions, giving them access to sensitive data and resources on the device. Moreover, when a device is involved in illegal activities, these vault apps can hinder law enforcement by making the data forensics process more challenging [14].

When advertising apps across various app market stores, developers typically use app titles, screenshots, and detailed text descriptions. We leverage this information to pinpoint content-hiding applications and discern the permissions these apps request on user devices. Android application permissions play a critical role in app security by regulating access to sensitive user data and system resources [9]. Each Android app lists in its manifest file the permissions it requires, and for the app to function properly, users need to grant these permissions. While some permissions correspond to security-sensitive tasks like accessing the camera or reading contacts, others relate to system-level functions, such as keeping the device awake [10]. In the context of vault applications, the permissions they seek often hint at the nature of the data they aim to protect.

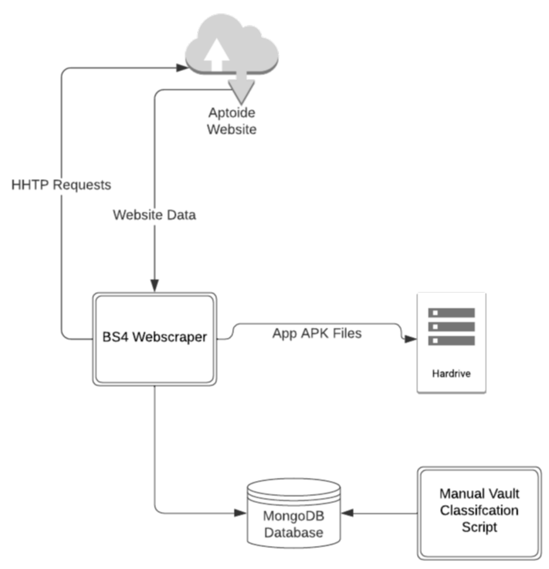


Figure 1: Aptoide Scraping Architecture

4. Methodology

To empirically evaluate our proposed approach, we established an experimental environment encompassing a wide range of Android applications. We detail our setup in the stages that follow:

4.1 Data Collection

App Collection and Labeling: Initially, we undertook the task of gathering an extensive dataset of Android applications. We targeted the Aptoide third-party app store and scraped it using keywords pertinent to vault applications. These keywords encompassed terms such as ‘private,’ ‘sensitive,’ ‘censor,’ ‘protect,’ ‘decoy,’ ‘privacy,’ ‘secret,’ ‘hide,’ ‘vault,’ ‘secure,’ ‘safe,’ ‘photos,’ ‘videos,’ ‘notes,’ ‘password,’ ‘contacts,' 'browser,' 'lock,' 'gallery,' 'calculator,' 'fingerprint,' 'password-protected,' 'fake,' and 'steganography.'

Upon finding matching applications, we proceeded to download their APKs, subsequently extracting basic details like the app's name, developer, country of origin for the developer, version, download count, and user rating.

After securing the APKs, we undertook a manual review of each application's webpage. This allowed us to sort the apps into two distinct categories: vault and nonvault, a classification hinging on the functionalities delineated on their respective webpages.

4.2 Identification System

The Vault-App Identification System utilizes a fusion of feature vector generators and classifier models to discern vault applications. We train a machine learning model using features derived from the applications, and the trained models subsequently determine whether an application qualifies as a vault or non-vault. The model's accuracy is contingent upon the quality of the feature vectors. For the generation of these feature vectors, we harness the titles, text descriptions, and the permissions requested by the applications.

4.2.1 Feature Extraction From Text - Binary Vector. The binary vector stands as a fundamental and widely-used text feature representation, indicating the words present within a given text. However, considering every word from the applications' titles and descriptions would result in an exceedingly high-dimensional binary vector. Thus, drawing parallels to DECADE [2], we chose 21 frequently-occurring words from titles and another 24 from the descriptions of the applications. Leveraging these chosen words, we craft binary vectors for titles and descriptions of the applications, designating the presence or absence of these words. Finally, merging these vectors, we fashion 45-length binary vectors for all applications, serving as their feature vector for the ML model.

4.2.2 Feature Extraction From Text - Document Embedding:. While binary vector representation is created with selected representative words, it is challenging to determine which word is crucial for vault app detection. Excluding certain words might lead to information loss about applications. Contemporary ML models for text employ text embedding to extract features automatically without requiring user input. By leveraging all words and their interrelations, text embedding

Table 1: Selected Words List from Title and Description of applications

Text type	**Selected Words**
Description	private, sensitive, censor, protect, decoy, privacy, secret, hide vault, secure, safe, photos, videos, notes, password, contacts, browser, lock, gallery, calculator, fingerprint, password-protected, password-protected, fake
Title	private, sensitive, censor, protect, decoy, privacy, secret, hide, vault, secure, safe, photos, videos, notes, password, contacts, browser, lock, gallery, calculator, fingerprint

models produce a compact representation of words and documents suitable for use as features in a machine learning model. To craft representations for titles and descriptions, we utilize Doc2Vec [17], an adaptation of Word2Vec.

4.2.3 Feature Extraction From Permission Request.

As mentioned above, each application requests certain permissions during installation, and users must grant these permissions for the app to function properly. Some permissions are linked with security-sensitive operations, such as accessing the camera or reading contacts, while others relate to system-level operations, like keeping the device awake. These permissions offer vital insights into the app's functionality. Consequently, we incorporate them as features of the application. After gathering permissions from all applications, we identified 75 unique permissions. Thus, we generate a 75-length binary vector indicating whether the application seeks that specific permission or not.

4.3 Forensic Analysis

In this section, we conduct a detailed forensic analysis of five Android vault apps to comprehend the pertinent artifacts. These applications are Wire, Twinme, Private Notebook, 3C Sensitive Backup, and Steganography Master. We employed *Magnet AXIOM - Process and Examine*, a comprehensive digital forensics toolkit [15] designed for the collection, analysis, and reporting of digital evidence [16]. We have structured this section into five subsections where we present the analyses of these five applications individually.

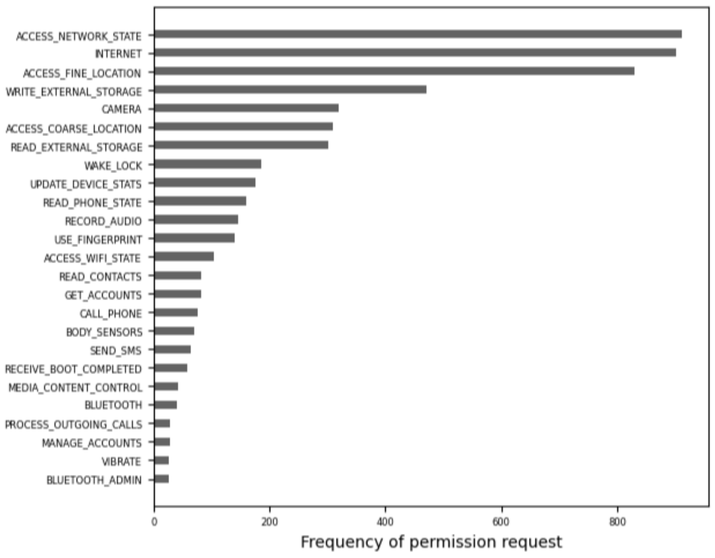
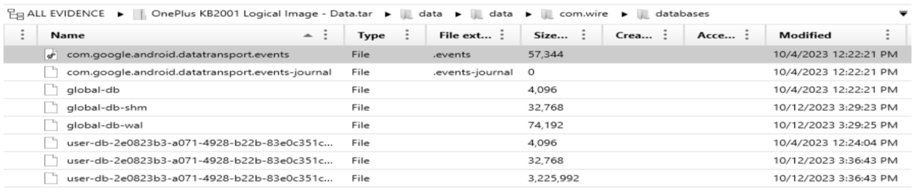


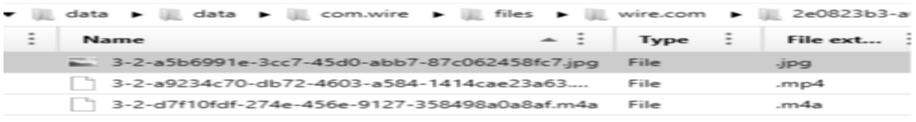
Figure 2: Top 25 Frequent Permissions Requested by Applications

4.3.1 Wire App. Wire is a messaging and communication platform. While it shares similarities with other messaging applications, it provides several unique features for users. It offers end-to-end encryption for all forms of communication, including messaging, voice calls, and file sharing. Our forensic analysis revealed that all the files associated with the Wire application are stored in the data/data/com.wire file structure. We successfully retrieved all the databases and media files related to this application, as illustrated in Figure 3a and Figure 3b.


ALL EVIDENCE ▸ OnePlus KB2001 Logical Image - Data.tar ▸ data ▸ data ▸ com.wire ▸ databases

Name	Type	File ext...	Size...	Crea...	Acce...	Modified
com.google.android.datatransport.events	File	.events	57,344			10/4/2023 12:22:21 PM
com.google.android.datatransport.events-journal	File	.events-journal	0			10/4/2023 12:22:21 PM
global-db	File		4,096			10/4/2023 12:22:21 PM
global-db-shm	File		32,768			10/12/2023 3:29:23 PM
global-db-wal	File		74,192			10/12/2023 3:29:25 PM
user-db-2e0823b3-a071-4928-b22b-83e0c351c...	File		4,096			10/4/2023 12:24:04 PM
user-db-2e0823b3-a071-4928-b22b-83e0c351c...	File		32,768			10/12/2023 3:36:43 PM
user-db-2e0823b3-a071-4928-b22b-83e0c351c...	File		3,225,992			10/12/2023 3:36:43 PM

(a) Databases Files

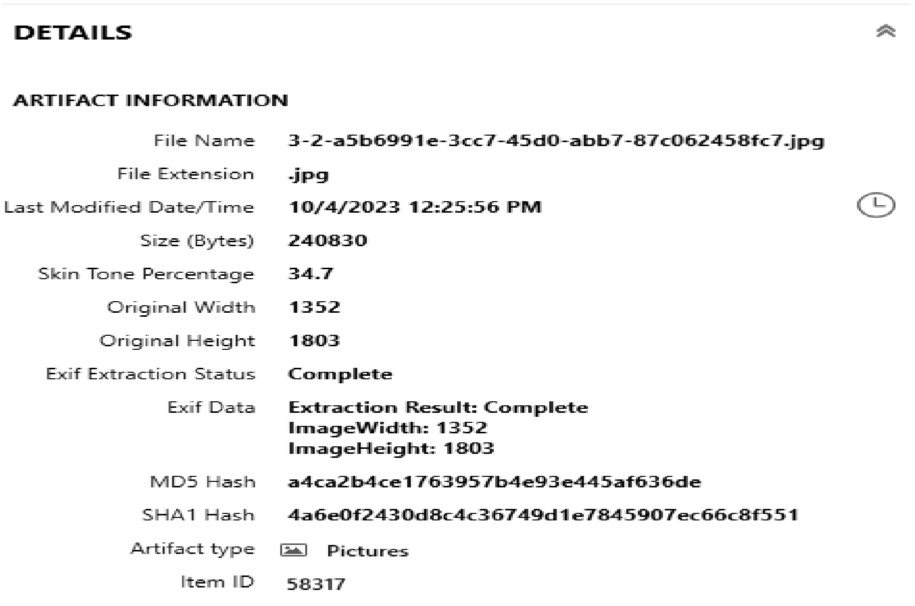

data ▸ data ▸ com.wire ▸ files ▸ wire.com ▸ 2e0823b3-a

Name	Type	File ext...
3-2-a5b6991e-3cc7-45d0-abb7-87c062458fc7.jpg	File	.jpg
3-2-a9234c70-db72-4603-a584-1414cae23a63....	File	.mp4
3-2-d7f10fdf-274e-456e-9127-358498a0a8af.m4a	File	.m4a

(b) Media Files

Figure 3: Recovered from Wire App

We also obtained the metadata information for each of the media files, as shown in Figure 4a. However, we were unable to recover any message and call history from this application due to its use of end-to-end encryption. Additionally, we identified the specific permissions utilized by the Wire application in Figure 4b.

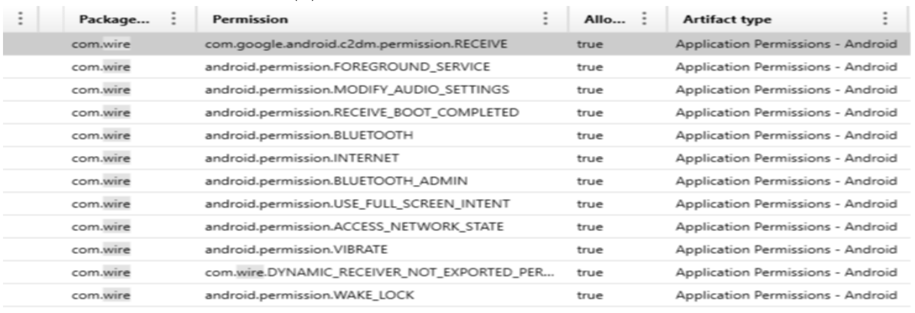


DETAILS

ARTIFACT INFORMATION

File Name	3-2-a5b6991e-3cc7-45d0-abb7-87c062458fc7.jpg
File Extension	.jpg
Last Modified Date/Time	10/4/2023 12:25:56 PM
Size (Bytes)	240830
Skin Tone Percentage	34.7
Original Width	1352
Original Height	1803
Exif Extraction Status	Complete
Exif Data	Extraction Result: Complete ImageWidth: 1352 ImageHeight: 1803
MD5 Hash	a4ca2b4ce1763957b4e93e445af636de
SHA1 Hash	4a6e0f2430d8c4c36749d1e7845907ec66c8f551
Artifact type	Pictures
Item ID	58317

(a) Media Metadata from Wire



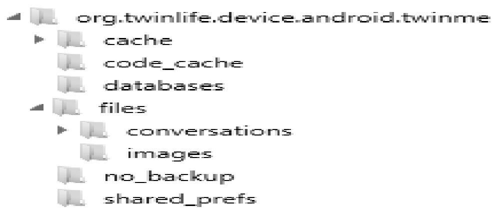
Package...	Permission	Allo...	Artifact type
com.wire	com.google.android.c2dm.permission.RECEIVE	true	Application Permissions - Android
com.wire	android.permission.FOREGROUND_SERVICE	true	Application Permissions - Android
com.wire	android.permission.MODIFY_AUDIO_SETTINGS	true	Application Permissions - Android
com.wire	android.permission.RECEIVE_BOOT_COMPLETED	true	Application Permissions - Android
com.wire	android.permission.BLUETOOTH	true	Application Permissions - Android
com.wire	android.permission.INTERNET	true	Application Permissions - Android
com.wire	android.permission.BLUETOOTH_ADMIN	true	Application Permissions - Android
com.wire	android.permission.USE_FULL_SCREEN_INTENT	true	Application Permissions - Android
com.wire	android.permission.ACCESS_NETWORK_STATE	true	Application Permissions - Android
com.wire	android.permission.VIBRATE	true	Application Permissions - Android
com.wire	com.wire.DYNAMIC_RECEIVER_NOT_EXPORTED_PER...	true	Application Permissions - Android
com.wire	android.permission.WAKE_LOCK	true	Application Permissions - Android

(b) Permissions Used by Wire

Figure 4: Media Metadata and Permissions of Wire

4.3.2 Twinme App.

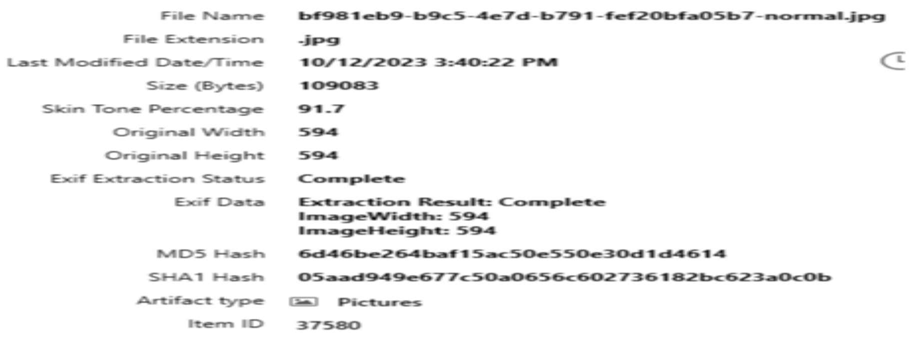
Twinme is another private and secure messaging platform that offers users end-to-end encryption. This cross-platform Android application doesn't store messages centrally and doesn't require any personal information like emails or phone numbers to maintain user anonymity. Twinme supports various communication methods, including text messaging, video calls, and voice calls. In our forensics analysis, we observed that Twinme stores databases, files, conversations, and images locally in the data/data/org.twinlife.device.android.twinme structures in an encrypted manner, as shown in Figure 5.



```
◢ org.twinlife.device.android.twinme
  ▸ cache
    code_cache
    databases
  ◢ files
    ▸ conversations
      images
    no_backup
    shared_prefs
```

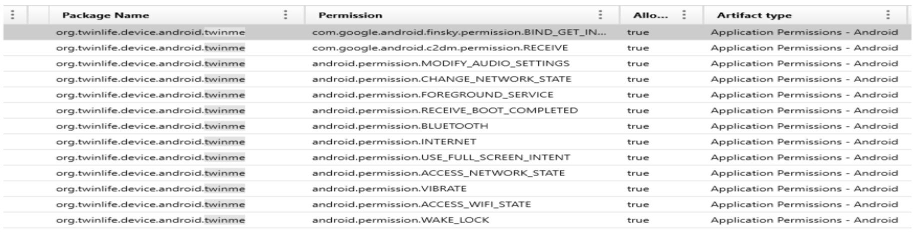
Figure 5: File Structure of Twinme

However, we managed to retrieve metadata information for all media files associated with the application, as depicted in Figure 6a. Additionally, we gathered insights into the permissions used by this specific application to understand its functionality and security features, presented in Figure 6b.



File Name	bf981eb9-b9c5-4e7d-b791-fef20bfa05b7-normal.jpg
File Extension	.jpg
Last Modified Date/Time	10/12/2023 3:40:22 PM
Size (Bytes)	109083
Skin Tone Percentage	91.7
Original Width	594
Original Height	594
Exif Extraction Status	Complete
Exif Data	Extraction Result: Complete ImageWidth: 594 ImageHeight: 594
MD5 Hash	6d46be264baf15ac50e550e30d1d4614
SHA1 Hash	05aad949e677c50a0656c602736182bc623a0c0b
Artifact type	Pictures
Item ID	37580

(a) Media Metadata

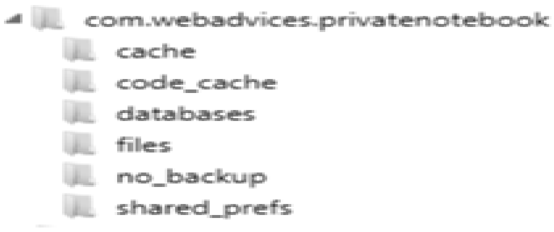


Package Name	Permission	Allo...	Artifact type
org.twinlife.device.android.twinme	com.google.android.finsky.permission.BIND_GET_IN...	true	Application Permissions - Android
org.twinlife.device.android.twinme	com.google.android.c2dm.permission.RECEIVE	true	Application Permissions - Android
org.twinlife.device.android.twinme	android.permission.MODIFY_AUDIO_SETTINGS	true	Application Permissions - Android
org.twinlife.device.android.twinme	android.permission.CHANGE_NETWORK_STATE	true	Application Permissions - Android
org.twinlife.device.android.twinme	android.permission.FOREGROUND_SERVICE	true	Application Permissions - Android
org.twinlife.device.android.twinme	android.permission.RECEIVE_BOOT_COMPLETED	true	Application Permissions - Android
org.twinlife.device.android.twinme	android.permission.BLUETOOTH	true	Application Permissions - Android
org.twinlife.device.android.twinme	android.permission.INTERNET	true	Application Permissions - Android
org.twinlife.device.android.twinme	android.permission.USE_FULL_SCREEN_INTENT	true	Application Permissions - Android
org.twinlife.device.android.twinme	android.permission.ACCESS_NETWORK_STATE	true	Application Permissions - Android
org.twinlife.device.android.twinme	android.permission.VIBRATE	true	Application Permissions - Android
org.twinlife.device.android.twinme	android.permission.ACCESS_WIFI_STATE	true	Application Permissions - Android
org.twinlife.device.android.twinme	android.permission.WAKE_LOCK	true	Application Permissions - Android

(b) Permissions Used

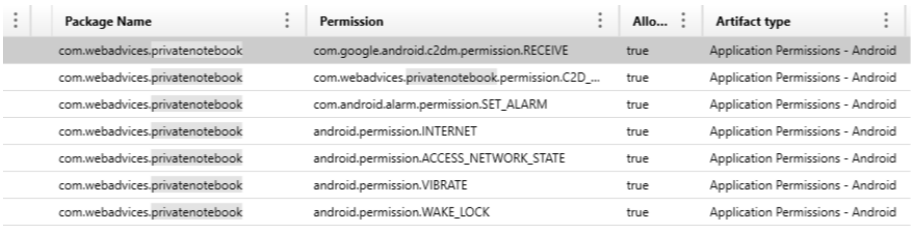
Figure 6: Media Metadata and Permissions of Twinme

4.3.3 Private Notebook App. Private Notebook is an Android application designed to assist users in creating and maintaining notes, documents, and personal information. It prioritizes data security through the implementation of several secure features such as encryption and password protection. During our analysis, we noted a specific characteristic of the application: it stores files and databases in the data/data/com.webadvices.privatenotebook structure in an encrypted manner, as depicted in Figure 7a. Consequently, we could not retrieve any note file or document from this application. However, we did manage to obtain information regarding the application's permissions, as shown in Figure 7b.



```
com.webadvices.privatenotebook
    cache
    code_cache
    databases
    files
    no_backup
    shared_prefs
```

(a) File Structure



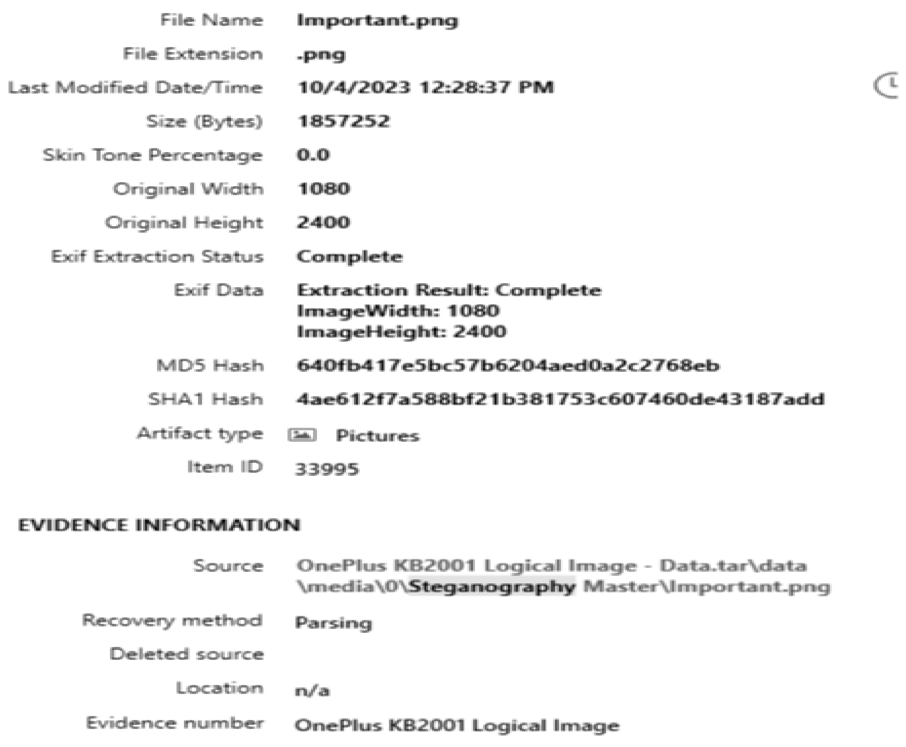
	Package Name	Permission	Allo...	Artifact type
	com.webadvices.privatenotebook	com.google.android.c2dm.permission.RECEIVE	true	Application Permissions - Android
	com.webadvices.privatenotebook	com.webadvices.privatenotebook.permission.C2D_...	true	Application Permissions - Android
	com.webadvices.privatenotebook	com.android.alarm.permission.SET_ALARM	true	Application Permissions - Android
	com.webadvices.privatenotebook	android.permission.INTERNET	true	Application Permissions - Android
	com.webadvices.privatenotebook	android.permission.ACCESS_NETWORK_STATE	true	Application Permissions - Android
	com.webadvices.privatenotebook	android.permission.VIBRATE	true	Application Permissions - Android
	com.webadvices.privatenotebook	android.permission.WAKE_LOCK	true	Application Permissions - Android

(b) Permissions Used

Figure 7: Media Metadata and Permissions of Private Notebook

4.3.4 3C Sensitive Backups App. The Android application *3C Sensitive Backups* allows users to securely back up and restore essential data, including call logs, contacts, SMS, MMS, and calendar data. Users can choose their preferred backup location, whether it be local storage or remote cloud services such as Google Drive, Dropbox, or FTP servers. During our forensics analysis, we faced a significant challenge. Despite our efforts, we could only recover basic installation details and no other information. This result highlights the application's robust data protection and security measures.

4.3.5 Steganography Master App. The Android application *Steganography Master* is designed to encode messages within images. Users can either save the resulting picture or share it with friends. However, only the same application can decode the encoded message. For enhanced security, users can set a password. Notably, this application does not use external databases for storage; instead, it stores all pictures locally. During our forensic analysis, we successfully identified and retrieved all encoded pictures from the "Steganography Master" folder located in the media folder. Additionally, we gathered valuable metadata associated with these images as shown in figure 8.



File Name	Important.png
File Extension	.png
Last Modified Date/Time	10/4/2023 12:28:37 PM
Size (Bytes)	1857252
Skin Tone Percentage	0.0
Original Width	1080
Original Height	2400
Exif Extraction Status	Complete
Exif Data	Extraction Result: Complete ImageWidth: 1080 ImageHeight: 2400
MD5 Hash	640fb417e5bc57b6204aed0a2c2768eb
SHA1 Hash	4ae612f7a588bf21b381753c607460de43187add
Artifact type	Pictures
Item ID	33995

EVIDENCE INFORMATION

Source	OnePlus KB2001 Logical Image - Data.tar\data \media\0\Steganography Master\Important.png
Recovery method	Parsing
Deleted source	
Location	n/a
Evidence number	OnePlus KB2001 Logical Image

Figure 8: Media Metadata from Steganography Master

5. Results and Evaluation

In this section, we present our experimental results. We first evaluate the effectiveness of our text-based features and permission features using various machine learning techniques. Then, we combine text and permission features to create a feature set for each application. This set is then used to train machine learning models and produce results. We employ accuracy, precision, recall, and F1 measures for comparison. In our experiments, we partition the data into training and testing sets, allocating 60% for training and 40% for testing.

5.1 Features Comparison

We use feature vectors extracted from text and permissions to train a classification model. We create four different feature sets using these vectors: (1)*BV*, the binary vector obtained from word presence, (2)*Doc2Vec*, document embeddings, (3)*Perm*, permissions, and (4)*PermBV*, permissions combined with the binary vector. We train our classifier using four different machine-learning algorithms, including Logistic Regression, Support Vector Machines, Gaussian Naive Bayes, and Random Forests. Table 2 presents the summary of results and the accuracy of the four classifiers for each feature set.

The results for each feature varied across different machine-learning algorithms. Among them, the *BV* feature exhibited the best performance, achieving approximately 86% accuracy when paired with the Support Vector Machine. Notably, *BV* consistently maintained accuracy levels above 80% when used in conjunction with Logistic Regression and Random Forest.

On the other hand, *Doc2Vec* reached its peak performance when paired with Logistic Regression, boasting an impressive accuracy rate of 90%. In contrast, the *Perm* feature experienced a significant drop in accuracy, falling into the 70% range across all machine-learning models. This data underscores the limited effectiveness of the *Perm* feature when used on its own. It becomes evident that a more effective approach is to incorporate the *Perm* feature with the combined *PermBV* feature to achieve better accuracy scores. When comparing *BV* to *PermBV*, the latter showed a notable 15% increase in accuracy, especially with the

Table 2: Accuracy Comparison of Machine Learning Models Based on Various Feature Sets

Model	*BV*	*Doc2vec*	*Perm*	*PermBV*
NB	65.9	84.2	74.7	69.9
RF	83.5	75.6	71.4	85.7
SVM	86.8	85.5	74.7	79.7
LR	85.7	90.0	69.2	79.1

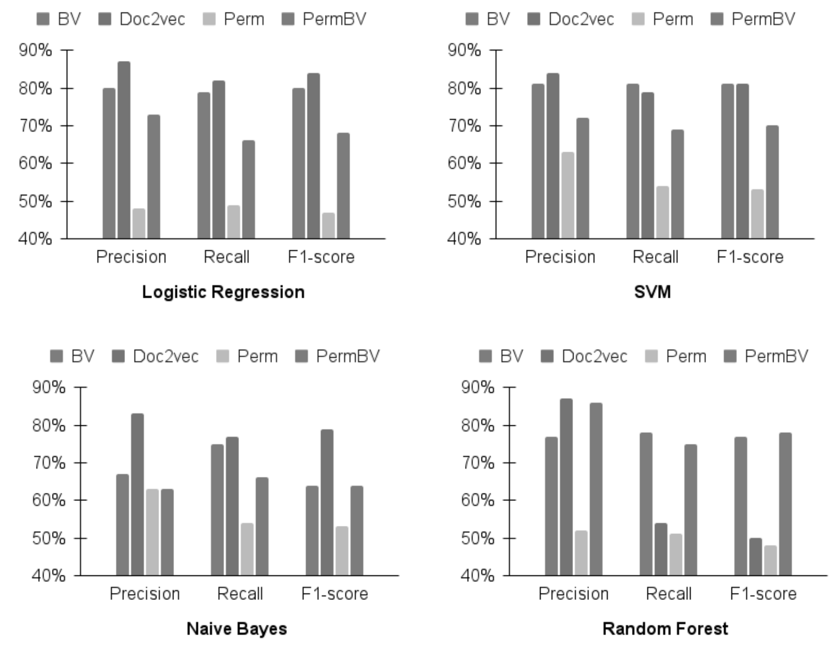


Figure 9: Precision, recall, and F1 score comparison of ML models based on different feature sets.

Random Forest model, and consistently delivered results in the 70% to 80% range across various machine learning models.

Based on this thorough analysis and the accompanying data table, the most effective feature-model combinations are as follows: Logistic Regression with $Doc2Vec$, Support Vector Machine with either BV or $Doc2Vec$, and Random Forest with $PermBV$. These combinations consistently yielded the highest overall accuracy levels.

In addition to accuracy, we provide an overview of precision, recall, and F1 scores for each feature set across various machine-learning models. These results are visualized in Figure 9. Examining the figure, it becomes evident that across most metrics, *Doc2vec* consistently delivers the most favorable outcomes, with BV ranking as the second-best performer, and $PermBV$ securing the third position. Conversely, $Perm$ consistently yields the lowest scores across all evaluation measures.

5.2 Parameter Analysis

$Doc2Vec$ incorporates multiple parameters within its model, including vector size, window size, epochs, and min count. Here, we scrutinize the impact of these parameters. Our analysis reveals that the majority of these parameters do not exert a significant influence on the results. However, two parameters, vector size, which specifies the dimensionality of the embedded vectors, and window size, which specifies the size of the sliding window, exhibit a subtle impact.

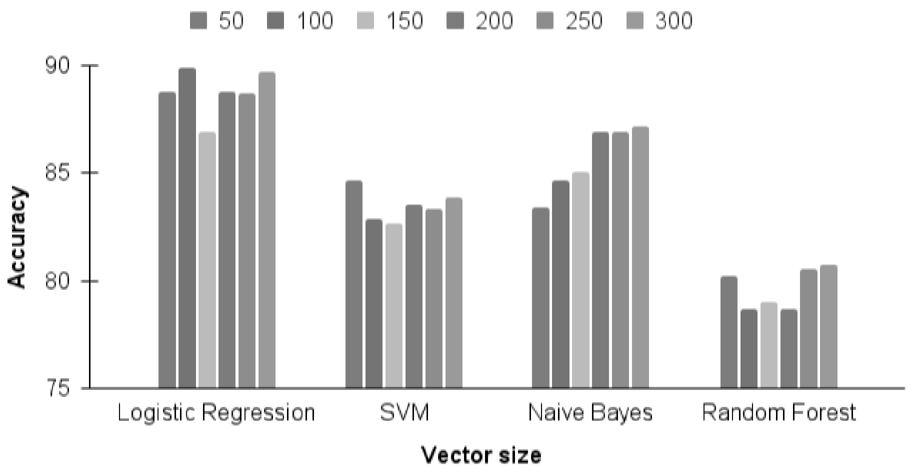


Figure 10: Comparison of ML models accuracy based on the vector size within Doc2Vec.

For the vector size parameter, we experiment with varying size, ranging from 50 to 300. As illustrated in Figure 10, we observe that increasing the vector size enhances the accuracy of the Naive Bayes model, reaching its peak when the size is 300. Conversely, larger vector sizes correspond to decreased accuracy for the Support Vector Machine (SVM). Notably, the vector size does not substantially affect the performance of Random Forest and Logistic Regression models.

For the window size parameter, we experiment with varying size from 1, 2, 4, 8, and 12. As illustrated in Figure 11, we observe that increasing the vector size enhances the accuracy of the Naive Bayes and Random Forest models, reaching their peak when the size is 12. Furthermore, the vector size does not substantially affect the performance of Logistic Regression and SVM models.

6. Discussion and Implications

The implications of this research are multi-faceted. Firstly, from a security analysis standpoint, our approach introduces a novel method for security analysts to pinpoint potential vault applications. This can

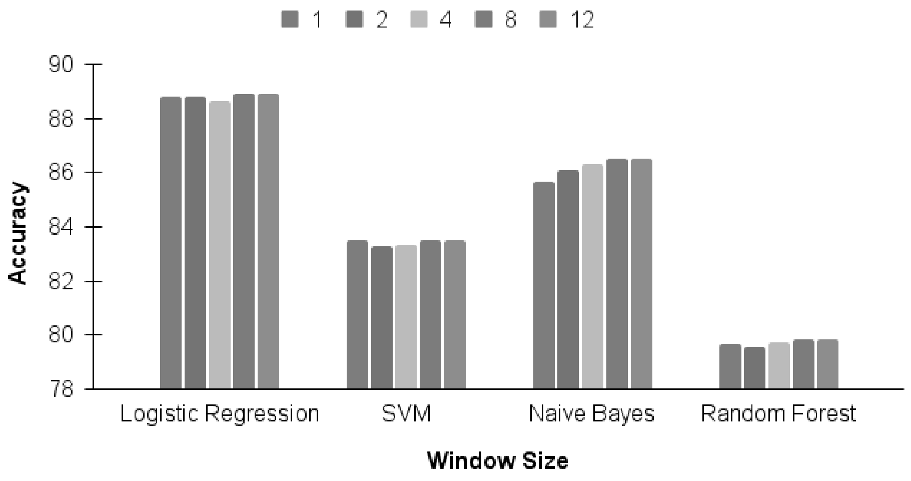


Figure 11: Comparison of ML models accuracy based on the window size within Doc2Vec.

significantly elevate the overall security analysis process. Secondly, in the realm of forensic investigation, our findings provide insights into data storage and data analysis of vault apps. Thirdly, by bringing attention to vault applications, users can make more informed decisions about the apps they install and use, contributing to improved data privacy. Lastly, our research insights can serve as valuable input for policy-makers and tech industry regulators. These insights can be instrumental in crafting more effective policies around app permissions and data security. In summary, our study not only breaks new ground in vault application detection but also sets the stage for further exploration in this domain. While we have made significant progress, there remains a vast expanse to be explored as we refine and evolve our approach.

7. Limitations and Future Work

While our study boasts significant findings, it's essential to acknowledge certain limitations that underscore our research. Firstly, the initial categorization of apps into vault and non-vault was executed manually, a method which might have inadvertently introduced a bias. The adoption of an automated or semi-automated categorization could address this concern. Secondly, our dataset's foundation is applications sourced from the third-party app store, Aptoide. Integrating apps from varied sources such as the Google Play Store or other third-party platforms might offer a broader, more representative spectrum of results. Lastly, our methodology leans considerably towards static analysis, sidelining dynamic behaviors. This emphasis might result in overlooked detections. Augmenting our approach with dynamic analysis might further enrich our findings.

Our research, while showing promising results, acknowledges the limitations inherent in our current approach, thus highlighting several avenues for future work. One direction worth exploring is automated categorization. By developing a machine learning model or an AI-based system, we could refine the process of categorizing apps into vault and non-vault categories, ensuring greater efficiency and accuracy. Further, our current reliance on static analysis could be complemented with dynamic analysis, offering insights into more nuanced or concealed behaviors typical of potential vault apps. Moreover, we are considering expanding our dataset to foster a more generalizable set of results. This would involve integrating applications from varied sources and perhaps even considering different operating systems.

In wrapping up, our research initiates a novel trajectory in Android vault application detection. We are committed to continually refining our methods, aspiring towards a holistic and encompassing approach to vault app detection and analysis.

8. Conclusion

Despite the inherent limitations and challenges, our methodology introduces a fresh perspective to Android application analysis, especially in the realm of vault app detection. Our findings lay the groundwork for subsequent research, aiming to enhance the proficiency of digital forensic investigators and Android security analysts. By highlighting the intricacies of vault apps, we aspire to foster a safer and more transparent Android ecosystem. In an era where user privacy is paramount, it's vital to comprehend the implications of installing and utilizing such vault applications. Our research sheds light on a relatively underexplored area in Android security analysis, paving the way for more advanced detection methods and deepening our understanding of vault apps.

9. Acknowledgment

The authors would like to acknowledge National Science Foundation (NSF) (Award number: 2043302) and CyberCorps Scholarship for Service (SFS) program for supporting this project.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

References

[1] G. Dorai, S. Aggarwal, N. Patel, and C. Powell, *Vide-vault app identification and extraction system for iOS devices*, *Forensic Science International: Digital Investigation*, vol. 33, p. 301007, 2020, Elsevier.

[2] M. Peng, M. Khanov, S. R. Madireddy, H. Chi, E. Akbas, and G. Dorai, *DECADE-Deep Learning Based Content-hiding Application Detection System for Android*, in *2021 IEEE International Conference on Big Data (Big Data)*, 2021, pp. 5430–5440, IEEE.

[3] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, *Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps*, *ACM SIGPLAN Notices*, vol. 49, no. 6, pp. 259–269, 2014.

[4] Y. Feng, S. Anand, I. Dillig, and A. Aiken, *Apposcopy: Semantics-based detection of Android malware through static analysis*, in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 576–587.

[5] S. Hou, Y. Fan, Y. Zhang, Y. Ye, J. Lei, W. Wan, J. Wang, Q. Xiong, and F. Shao, *αcyber: Enhancing robustness of Android malware detection system against adversarial attacks on heterogeneous graph-based model*, in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019, pp. 609–618.

[6] D. Votipka, S. Rabin, K. Micinski, J. S. Foster, and M. L. Mazurek, *An observational investigation of reverse {Engineers'} processes*, in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 1875–1892.

[7] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, *Riskranker: scalable and accurate zero-day Android malware detection*, in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, 2012, pp. 281–294.

[8] H. Gascon, F. Yamaguchi, D. Arp, and K. Rieck, *Structural detection of Android malware using embedded call graphs*, in *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security*, 2013, pp. 45–54.

[9] Android Developers. *Manifest.permission — Android Developers.* URL: `https://developer.android.com/reference/android/Manifest.permission`.

[10] Adam P. Fuchs, Avik Chaudhuri, and Jeffrey S. Foster, *Scandroid: Automated security certification of Android*, Technical Report, 2009.

[11] Steven Muchnick, *Advanced Compiler Design Implementation*, Morgan Kaufmann, 1997.

[12] Anthony Desnos and G. Gueguen, *Androguard Documentation*, 2018, [Online; accessed August 2023], Available: `https://androguard.readthedocs.io/en/latest`

[13] PCMag. *Vault App Definition — PCMag Encyclopedia.* URL: `https://www.pcmag.com/encyclopedia/term/vault-app`.

[14] X. Zhang, I. Baggili, and F. Breitinger, *Breaking into the vault: Privacy, security and forensic analysis of Android vault applications*, *Computers & Security*, vol. 70, pp. 516–531, 2017, Elsevier.

[15] Magnet AXIOM. URL: `https://www.magnetforensics.com/products/magnet-axiom/`, Accessed: 2023-06-05

[16] Javed, Abdul Rehman and Ahmed, Waqas and Alazab, Mamoun and Jalil, Zunera and Kifayat, Kashif and Gadekallu, and Thippa Reddy, *A Comprehensive Survey on Computer Forensics: State-of-the-Art, Tools, Techniques, Challenges, and Future Directions*, *IEEE Access*, vol. 10, pp. 11065-11089, doi = 10.1109/ACCESS.2022.3142508, 2022

[17] Le, Quoc, and Tomas Mikolov. *Distributed representations of sentences and documents*, *In International conference on machine learning*, pp. 1188-1196. PMLR, 2014.