Casing the Vault: Security Analysis of Vault Applications

Margie Ruffin
University of Illinois Urbana-Champaign
USA

Israel Lopez-Toldeo
University of Illinois Urbana-Champaign
USA

Kirill Levchenko
University of Illinois Urbana-Champaign
USA

Gang Wang
University of Illinois Urbana-Champaign
USA

ABSTRACT

Vault applications are a class of mobile apps used to store and hide users' sensitive files (e.g., photos, documents, and even another app) on the phone. In this paper, we perform an empirical analysis of popular vault apps under the scenarios of unjust search and filtration of civilians by authorities (e.g., during civil unrest). By limiting the technical capability of adversaries, we explore the feasibility of inferring the presence of vault apps and uncovering the hidden files *without employing sophisticated forensics analysis*. Our analysis of 20 popular vault apps shows that most of them do not adequately implement/configure their disguises, which can reveal their existence without technical analysis. In addition, adversaries with rudimentary-level knowledge of the Android system can already uncover the files stored in most of the vault apps. Our results indicate the need for more secure designs for vault apps.

CCS CONCEPTS

• **Security and privacy** → **Privacy protections**.

KEYWORDS

Privacy; Vault App; Android

ACM Reference Format:
Margie Ruffin, Israel Lopez-Toldeo, Kirill Levchenko, and Gang Wang. 2022. Casing the Vault: Security Analysis of Vault Applications. In *Proceedings of the 21st Workshop on Privacy in the Electronic Society (WPES '22), November 7, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3559613.3563204

1 INTRODUCTION

Mobile devices are increasingly used to carry sensitive data. While security mechanisms such as PIN codes and Face ID can prevent unauthorized access, in certain circumstances, users may be forced to give away their access to the phone. A common example is authorities may seize and search the phones of journalists and civilians during civil unrest. In 2021, police officers confiscated the phones and money from journalists of the Belarusian Association of Journalists (BAJ) [13]. Recently during the Russia-Ukraine war, civilians have been stopped at the border, and their phones have been searched as part of a "filtration" process [3, 8, 9, 11].

To protect user privacy, a class of "vault apps" is introduced. These apps can store and hide various user data on the phone, including images, videos, audio, documents, and sometimes even other sensitive apps [7]. Some vault apps can appear inconspicuously on a device by disguising themselves as typical applications such as a calculator or a clock. This feature can be helpful in times of civil unrest or intimate partner violence (IPV) when adversaries may obtain physical access to the victim's phone to inspect it.

Prior works have explored using *forensic analysis* to detect and break into vault apps [6, 7, 14]. The forensic analysis includes searching through the mobile device's storage image and file system for signs of app vulnerabilities, performing code analysis, and rooting the mobile device to uncover hidden files. Such in-depth forensic analysis is primarily used for criminal investigations, which is not directly applicable to the aforementioned scenarios. This is because (1) adversaries (e.g., police at the border) may not have the time to perform sophisticated forensic analysis during a quick inspection, and (2) the adversaries may not be technically capable of analyzing the phone storage or app code.

In this paper, we align our analysis closer to the threat model of phone inspection during civil unrest or IPV. We seek to understand how well vault apps maintain their disguise or hide user files when facing adversaries with *much limited* knowledge or technical capabilities. We ask two questions: can someone infer the presence of a vault app by simply browsing the apps on the phone? What does it take for them to uncover the hidden files once the vault app is identified? These questions are meaningful because authorities may only have a few minutes to snoop through the phone in situations such as unjust searches, seizure of property, or filtration of civilians. Only after the existence of a vault app is confirmed will they consider further technical approaches to break into it.

To answer these questions, we select 20 vault apps from Google Play, combining highly popular apps and those studied by prior works. Then we simulate adversaries with varying levels of technical capability to inspect the apps. These include (1) *novice-level* adversaries who have little to no knowledge of the Android system and only check the apps to determine if a vault app exists; (2) *intermediate-level* adversaries who have some knowledge of the Android System to pull files from apps without rooting the phone, and (3) *advanced-level* adversaries who have the ability to root the device. Note that both the intermediate- and advanced-level analyses still rely on the basic Android developer tools such as `adb` without using advanced forensics analysis.

WPES '22, November 7, 2022, Los Angeles, CA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9873-2/22/11…$15.00
https://doi.org/10.1145/3559613.3563204

Our analysis returns several important findings. *First*, novice-level analysis can already detect the presence of vault apps for most of the analyzed apps. For example, we find that only 10 of the 20 apps present a decoy app as a disguise (e.g., a lock or a calculator). Only 7 of the decoy apps have implemented the decoy functionality, and only 5 of them would turn on the disguise automatically upon installation. Also, only 3 apps maintain the consistent decoy app name/icon in the App Library (file system). In other words, adversaries can uncover the presence of most of the vault apps without technical analysis. *Second*, intermediate analysis (without rooting the phone) can successfully retrieve files for 15 out of the 20 vault apps, and advanced analysis (rooting the phone) can retrieve the files for the remaining 5 apps. Common techniques used to hide files include creating hidden folders, changing file extensions, and modifying header bytes. Finally, only 5 of the 20 apps attempted encrypting the files. This means most of the files in vault apps can be easily recovered.

In summary, we present a multi-level analysis of 20 vault apps by varying the level of knowledge and technical capabilities of adversaries. Our threat model is more aligned with scenarios such as unjust search and filtration of civilians (e.g., during civil unrest). We show that without performing sophisticated forensic analysis, adversaries can infer the existence of most of the popular vault apps (downloaded by millions of people) and retrieve the stored files with the rudimentary-level knowledge of the Android system.

2 RELATED WORK

Mobile vault applications (vault apps) are designed to be privacy-enhancing as they allow users to protect their personal data through encryption, camouflage, and hiding [12]. Prior works have studied their behaviors and security postures. Still, most studies are considering the threat model where *adversaries can perform in-depth technical forensics analysis* on the device and apps, which is different from our threat model.

Forensics analysis usually has two goals: (1) to determine whether an app is a vault app and (2) to recover the data hidden in the vault. This is done by analyzing potential artifacts left by applications on a device. Take the Android phone, for example; such artifacts include app-generated folders (that may contain hidden photos, videos, messages, or passwords), and the APK files of the target apps (which can be used to reverse engineer the executable code) [6, 12, 14].

Recent works developed forensic suites to detect vault apps (based on package names) and recover non-encrypted data [7] and compared the encryption schemes of popular vault apps [12]. To infer the behaviors of vault apps, researchers combined static analysis of the app code and dynamic analysis of the runtime behavior (e.g., using simulated user interactions [1, 4]). Such analysis searches for indicators of content hiding, system partitioning, data encryption, and application disguise [6]. Zhang et al. [14] combine several forensics techniques to analyze 18 vault apps from Google Play store to uncover the private data inside.

3 METHODOLOGY

3.1 Threat Model and Data Collection

Threat Model. Unlike prior works, we do not assume adversaries have the time and technical capability to perform in-depth forensic analysis on the file system and encryption scheme once they have seized the victim's phone. Instead, we assume adversaries (e.g., police) only temporarily take over the victim's phone (e.g., an activist) to perform a quick examination. Only after the vault app is identified will adversaries attempt to uncover the hidden files with *rudimentary* methods. Our goal is to explore how likely the readily available side channels can reveal the presence of vault app and the hidden files. The target scenarios could be police searching protesters' phones, customs searching travelers' phones, and abusers searching their victims' phones under intimate partner violence (IPV).

Vault App Selection. We select a list of 20 Android vault applications (Table 1), combining those that are commonly studied in prior works and those that are highly popular in the Google Play store. First, we perform a keyword search on the Google Play store using the keywords "vault", "hide media", and "hide apps". This returns many apps that claim to provide privacy-preserving functionality. Then we select 10 most downloaded free vault apps, as shown in Table 4 in the Appendix. Each of the applications chosen has ten million or more downloads from users. Then, we select the other 10 apps from previous works [7, 14], as shown in Table 3 in the Appendix. When selecting apps from prior works, we find that the Video Locker app (`com.handyapps.videolocker`) is already in the top 10 list (and thus, we do not need to include it again). We also exclude apps no longer available on the Google Play store. We select these 10 apps based on their higher download count.

Setups. After each app is downloaded from the Google Play store, we need to perform the basic setup. First, a password or pin is usually required to store media files in the vault. We use a random number generator to select a 4-digit or 6-digit pin depending on the vault app's requirement. Then we store relevant files (.jpg, .mp3, mp4, .txt) and/or a target android app (e.g., Facebook) depending on the vault app's features. Under this setup, we will emulate different types of adversaries to examine the vault app's behavior, ranging from behaviors exhibited to users and the operations behind the scenes at a lower level. As stated, we try to avoid using any advanced forensics tools and instead use the basic android file system, APKtool, and adb to gather all the necessary information for the analysis. The android devices used include Samsung Galaxy S9+, LG K31 Rebel, and the Bluestacks Android emulator. Both Galaxy S9+ and LGK31 Rebel are running on Android OS Version 10. The Bluestacks Android emulator is running on Android 11.

3.2 Security Analysis

We emulate different adversaries to analyze each app's security measures and identify their vulnerabilities. The vault apps are analyzed in three settings: Novice, Intermediate, and Expert. The goal is to identify the level of expertise needed to exploit their vulnerabilities. The Novice level represents an attacker with little to no technical knowledge of Android Systems; this actor can only perform a qualitative analysis to determine if a vault application is on the phone. The Intermediate level represents an attacker that has some knowledge of Android Systems. This actor can access regular (or hidden) files stored by an app. Finally, the Expert level represents an attacker that is highly knowledgeable about Android and even has the ability to root the device in question. This actor can effectively use adb (with and without root) to retrieve files and perform backups, can decompile apks, and has the basic knowledge of encryption schemes.

Novice Level Security Analysis. To simulate a technologically unsavvy attacker, we seek to identify a vault app by examining the app icon, the displayed name, and the app size. These features may give away the presence of a privacy-preserving vault app. More specifically, we first check whether the disguise (e.g., vault app is presented as a calculator) is enabled automatically upon app installation and whether the icon and app name are displayed properly on the phone screen. If the disguise is not automatically enabled, we further examine whether it can be enabled or configured through settings. In addition, we check whether the decoy app's name and icon are properly displayed in the phone's "app library" (file system). If the displayed name or icon in the app library does not match those shown on the phone screen, it can break the disguise. Finally, we examine the app size. If the app is abnormally large (e.g., for a clock or a calculator app), it may raise suspicion.

Intermediate Level Security Analysis. At this level, we assume the attacker can collect some files generated by the vault apps (also referred to as artifacts). To do so, we use the Android Debug Bridge (ADB), which is a basic command-line tool provided to Android developers [2]. For this attack level, we first perform basic file retrieval without rooting the phone, using the command `adb pull` to obtain the app's files from `/sdcard`. Another file retrieval method (without rooting the phone) is to perform a backup operation using `adb backup`. For a vault app, the developer is supposed to set the `Android:allowBackup` flag to "false". However, if the developer did not set the flag properly, the attacker may create a backup for the application's files. Using these two methods, we examine whether the private media files stored in the vault app can be discovered.

Advanced Level Security Analysis. At this level, we emulate a strong attacker who can root the phone of interest (using the rooted Bluestacks Android emulator). With the root access, the attacker can further obtain files from `/data/data/<package name>` together with those pulled from `/sdcard` and `adb backup` for a thorough analysis. In addition, the attacker can obtain the app's apk file and use APKTool [10] to decompile the APK to obtain the source code and the android manifest file. This allows the attacker to perform a basic static analysis to understand the app's workflow.

4 RESULTS

Novice Level Analysis. For the novice qualitative analysis, the adversaries look for clear giveaways/indicators to identify a vault application. As shown in Table 1, these indicators appear in many of these applications. Three example screenshots are presented in Figure 1–3) in the Appendix.

First, only 10 out of 20 apps have a decoy app as a disguise. The most common decoy applications used for the disguise are either "clocks" or "calculators". Some apps, such as `com.hideitpro` have multiple decoy applications to choose from. However, only 5 of the 20 apps employ automatic disguises (i.e., enabling the disguise immediately upon app installation). For all 10 apps, users can enable or disable the disguise manually after the installation (i.e., configurable disguise). Only 7 of the decoy apps are truly functional (i.e., a calculator decoy can work as a calculator). *Second*, only 3 of the 20 apps have kept their disguise in the application file system (i.e., app library). In other words, if the adversary browses the apps at the app library and compares them with the same apps on the phone screen, 17 out of the 20 vault apps do not match up. *Third*, the app sizes can be a potential giveaway. Vault apps can be used to hide different types of media files, and their storage size grows as more files are added. For example, a real calculator app requires 6.10 MB of storage on an Android phone. However, as shown in Table 1, the smallest vault app disguised as a calculator can take as much as 35.17 MB of storage *even before we add any files*. The app sizes grow quickly once we add a few files (a photo, a video, and a text file) to them.

Overall, the results indicate the vast majority of the vault apps can be identified via novice-level analysis by checking the app names and icons and comparing them with those displayed in the app library. The better-disguised apps can be potentially identified by the abnormally large sizes of the app.

Intermediate Level Analysis. For the intermediate analysis, the adversary leverages their basic knowledge of the Android file system to extract private information from the suspected vault app *without rooting the phone*.

First, we find that 10 of the 20 vault apps have the `allowBackup` flag set to "true", meaning that a complete backup of the app in any state could be captured without root permission. For the remaining 10 apps where a backup is not an option, we attempt to retrieve its files from the SD card (via `adb pull`). The attempt is successful for 5 of these apps. *Second*, by analyzing the files pulled using the above two methods, we have a few observations, some of which echo those mentioned in [14]. For example, some app developers only change the extension of the file or simply add a header to disguise it (without performing any encryption on the files). Other developers may corrupt the file so that the files cannot be directly opened. Some apps (e.g., `ws.clockthevault`) simply leave the files in cleartext. Overall, we can uncover the app's PINs and recovery emails in plain text for 7 of the applications (see Table 2 in Appendix).

Overall, an adversary with a rudimentary knowledge of the Android file system can easily recover private files from 15 out of the 20 apps without rooting the phone.

Advanced Level Analysis. At this level, adversaries can root the victim's phone (and decompile the APK). Recall that there are still 5 apps whose files cannot be retrieved without root. With root access, we can recover the files for 2 of the apps, as they only hide the files by changing the file extensions and modifying the header bytes. The files from the remaining three apps are difficult to verify since they are corrupted/modified in an unknown manner.

Combining the results from the intermediate and advanced analysis, only 5 out of 20 apps have used some encryption schemes to protect the files. For example, `com.ushareit.lockit` has encrypted the files using a proprietary method and has changed the file extensions to ".kcol". A more common approach adopted by

App Package Name	Default Name on Phone Screen	Decoy App	Automatic Disguise?	Functional Disguise?	Configurable Disguise?	Disguise in App Library?	Hide Photos	Hide Videos	Hide Audio	Hide Documents	Lock Apps	Hide Apps	Intruder Alert	Allow Backup?	ADB Pulled?	App Size (MB) Before Vs After
com.domobile.applockwatcher	AppLock	Calculator Compass Spirit Level			✓		✓	✓	✓	✓	✓		✓		✓	45.17 → 95.63
com.netqin.ps	Vault	N/A					✓	✓			✓				✓	42.92 → 57.12
com.kii.safe	Keepsafe	Anti Virus			✓		✓						✓			52.39 → 62.32
com.thinkyeah.galleryvault	Gallery Lock	Calculator		✓	✓		✓	✓								47.19 → 62.58
com.cyou.privacysecurity	LOCX	N/A					◐				✓		✓	✓		13.97 → 26.69
com.app.calculator.vault.hider	Calculator	Calculator	✓	✓	✓	✓	✓	✓				✓		✓		35.17 → 99.10
com.theronrogers.vaultyfree	Vaulty	N/A					✓	✓							✓	13.98 → 21.61
com.morrison.gallerylocklite	Gallery Lock	N/A			◐		✓	✓							✓	31.63 → 41.38
com.xcs.piclock	Pic Lock	N/A			◐		✓	✓					✓	✓		19.79 → 28.68
com.handyapps.videolocker	Video Locker	N/A						✓						✓		34.28 → 43.07
com.apusapps.launcher	APUS	N/A									✓	✓				67.18 → 96.59
com.alpha.applock	AppLock	N/A					✓	✓			✓			✓		17.65 → 49.48
com.flatfish.cal.privacy	Calculator	Calculator	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓	72.09 → 237.0
com.ushareit.lockit	Lockit	N/A					✓	✓			✓			✓		27.54 → 36.86
com.ultra.applock	ULTRA APPLOCK	Calculator			✓						✓					30.35 → 39.66
ws.clockthevault	Clock	Clock	✓	✓	✓		✓	✓	✓	✓	✓			✓		24.12 → 72.59
com.sp.protector.free	AppLock	N/A									✓		✓	✓		07.47 → 17.13
com.hld.anzenbokusucal	Calculator	Calculator	✓	✓	✓	✓	✓	✓		✓						36.20 → 39.07
com.app.hider.master.pro	App Hider	Calculator		✓	✓		✓	✓				✓		✓		32.92 → 40.93
com.hideitpro	Audio Manager	Audio Manager Calculator Currency Converter No Icon Joke of the Day	✓	✓	✓		✓	✓	✓					✓		32.99 → 48.19

Table 1: Qualitative Analysis of Vault Applications—We perform a qualitative review of each vault app highlighting its security and privacy features. "✓" means a given feature is supported and it is functional; "◐" means the feature exists but it is nonfunctional during our test.

vault apps is to store files in hidden folders (9 out of 20 apps, see Table 2 in Appendix). We argue that the hidden folder is not a secure measure since adversaries can easily turn on the ability to view hidden files.

Automating The Process. The above analysis (especially file analysis) is largely done manually. However, some parts of the analysis can be automated. To improve the analysis efficiency (for future works), we have implemented a tool to automate the decompilation process, the backup generation, and the artifact retrieval. Given an APK file of interest, the tool first uses APKTool to obtain the decompiled files. Then, based on the android manifest file, it checks whether the backup flag is set to "true" and retrieves the app's package name. If a backup is allowed, it utilizes the adb tool to generate a backup. Otherwise, the tool will perform adb pull to retrieve files from `/sdcard` and `/data/data/<package name>` directory.

5 CONCLUSION AND DISCUSSION

In this paper, we empirically analyze popular mobile vault apps under the threat model of unjust search and filtration of civilians. We show that adversaries with limited technical capability can already identify the presence of vault apps via various side channels (e.g., app library, displayed name and icon, app interface). The main problem is that most developers did not implement truly functional disguises (e.g., a working calculator) or failed to maintain the disguise across different interfaces. In addition, we show that with rudimentary-level knowledge of the Android system, adversaries can uncover the files hidden in vault apps. The main problem is developers are using insecure methods (e.g., hidden folders, modifying file extensions) to hide files. We made a list of recommendations to developers in Appendix A. Future work can extend the analysis scope to cover more vault apps (including the less popular ones) or further automate the analysis procedure (based on the efforts of Section 4). In addition, our current analysis is focused on the Android system–many of these apps also have an iOS version with similar characteristics. In other words, the novice-level analysis should still be able to identify their presence. Another interesting direction is to perform a user study to emulate the device inspection process and explore how likely lay users can identify the presence of a vault app. Finally, further work can explore how to systematically improve the security of vault apps, e.g., by implementing more complete decoy apps and using advanced encryption algorithms (e.g., deniable encryption [5]) to hide the files from both novice-level and advanced-level adversaries.

ACKNOWLEDGMENTS. This work was supported in part by NSF grants 2030521, and the Graduate Research Fellowship Program under Grant No 21-46756.

REFERENCES

[1] 2020. UI/Application Exerciser Monkey. https://developer.android.com/studio/test/monkey.

[2] 2021. What is ADB? How to Install ADB, Common Uses, and Advanced Tutorials. https://www.xda-developers.com/what-is-adb/.

[3] Katie Balevic. 2022. Moscow police are stopping people and demanding to read their text messages, reporter says. https://www.businessinsider.com/russian-

police-are-demanding-to-read-peoples-text-messages-reporter-2022-3.

[4] Patrick Carter, Collin Mulliner, Martina Lindorfer, William Robertson, and Engin Kirda. 2017. Curiousdroid: Automated user interface interaction for android application analysis sandboxes. *Lecture Notes in Computer Science* 9603 LNCS (2017), 231–249. https://link.springer.com/chapter/10.1007/978-3-662-54970-4_13

[5] Bing Chang, Fengwei Zhang, Bo Chen, Yingjiu Li, Wen-Tao Zhu, Yangguang Tian, Zhan Wang, and Albert Ching. 2018. MobiCeal: Towards Secure and Practical Plausibly Deniable Encryption on Mobile Devices. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 454–465. https://doi.org/10.1109/DSN.2018.00054

[6] Gokila Dorai, Sudhir Aggarwal, Neet Patel, and Charisa Powell. 2020. VIDE - Vault App Identification and Extraction System for iOS Devices. *Forensic Science International: Digital Investigation* 33 (Jul 2020), 301007. https://doi.org/10.1016/j.fsidi.2020.301007

[7] Michaila Duncan and Umit Karabiyik. 2018. Detection and Recovery of Anti-Forensic (VAULT) Applications on Android Devices. *Annual ADFSL Conference on Digital Forensics, Security and Law* 6 (2018). https://commons.erau.edu/adfsl/2018/presentations/6

[8] Joyce Sohyun Lee and Jonathan Edwards. 2022. Video shows Russian filtration camp, Mariupol mayor's office says. urlhttps://www.washingtonpost.com/world/2022/05/06/ukraine-mariupol-russian-filtration-camp-video/.

[9] Hrihoriy Pyrlik. 2022. Bribes of cash, cigarettes pave escape for Ukrainians under Russian occupation. https://www.rferl.org/a/cash-cigarettes-can-pave-escape-ukrainians-russian-occupation/31904352.html.

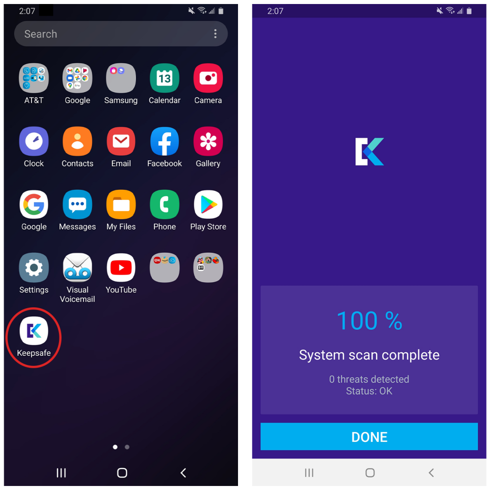
[10] Connor Tumbleson and Ryszard Wisniewski. 2022. Apktool: A tool for reverse engineering 3rd party, closed, binary Android apps. https://ibotpeaches.github.io/Apktool.

[11] Peter Weber. 2022. Russia is sorting Mariupol 'evacuees' at 'filtration camps,' based on social media posts, Ukrainians say. https://theweek.com/russo-ukrainian-war/1011541/russia-is-sorting-mariupol-evacuees-at-filtration-camps-based-on-social.

[12] Nannan Xie, Hongpeng Bai, Rui Sun, and Xiaoqiang Di. 2020. Android Vault Application Behavior Analysis and Detection. *Communications in Computer and Information Science* 1257 CCIS (2020), 428–439. https://link.springer.com/chapter/10.1007/978-981-15-7981-3_31

[13] Naomi Zeveloff. 2021. Belarusian authorities raid Belarusian Association of Journalists Headquarters, journalists' homes. https://cpj.org/2021/02/belarusian-authorities-raid-belarusian-association-of-journalists-headquarters-journalists-homes/.

[14] Xiaolu Zhang, Ibrahim Baggili, and Frank Breitinger. 2017. Breaking into the vault: Privacy, security and forensic analysis of Android vault applications. *Computers and Security* 70 (2017), 516–531. https://doi.org/10.1016/j.cose.2017.07.011



100 %

System scan complete

0 threats detected
Status: OK

Figure 1: com.kii.safe—This vault app is disguised as an anti-virus application. When opening the app, an interface is implemented to mimic an anti-virus software.

App Package Name	Plaintext Info?	Encryption?	Hidden Folder?
com.domobile.applockwatcher		✓	✓
com.netqin.ps			✓
com.kii.safe	✓		✓
com.thinkyeah.galleryvault		✓	
com.cyou.privacysecurity			
com.app.calculator.vault.hider		✓	
com.theronrogers.vaultyfree			
com.morrison.gallerylocklite			✓
com.xcs.piclock	✓		✓
com.handyapps.videolocker	✓		✓
com.apusapps.launcher		✓	
com.alpha.applock			✓
com.flatfish.cal.privacy			✓
com.ushareit.lockit		✓	✓
com.ultra.applock	✓		
ws.clockthevault	✓		
com.sp.protector.free	✓		
com.hld.anzenbokusucal			
com.app.hider.master.pro			
com.hideitpro	✓		

Table 2: Additional Qualitative Analysis of Vault Applications—This table provides additional results from our qualitative review. "✓" means a given feature is supported and it is functional

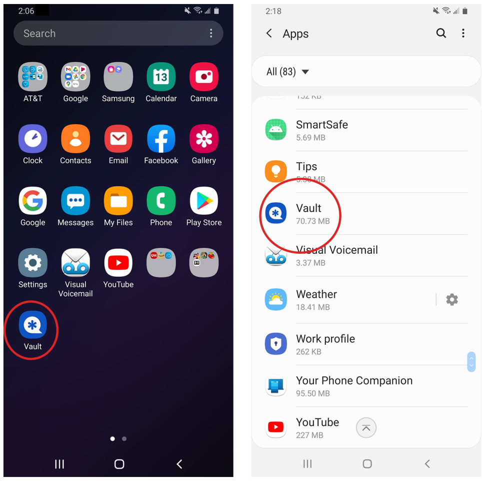


Figure 2: com.netqin.ps—This vault app does not employ a decoy app as a disguise. The app directly displays itself as a vault app.

A APPENDIX: RECOMMENDATIONS

We make the following recommendations to vault app developers to reduce the potential risks. First, vault apps should maintain a consistent icon disguise and name disguise everywhere on the phone (e.g., home screen, the application list in the setting panel, search screen). Second, developers should implement *functionally*

Application Name	Name on Device	Package Name	Version	Number of Downloads
APUS Launcher: Themes, Hide Apps, Launcher App	APUS	com.apusapps.launcher	3.10.34	100M+
AppLock - Fingerprint & Password, Gallery Locker	AppLock	com.alpha.applock	4.0.1	50M+
Calculator Lock - App Hider & Photo Vault - HideX	Calculator	com.flatfish.cal.privacy	3.1.7.14	50M+
AppLock - Fingerprint	AppLock	com.sp.protector.free	7.9.2	50M+
Calculator- Photo Vault & Video Vault hide photos	Calculator	com.hld.anzenbokusucal	10.0.7	10M+
App Hider- Hide Apps Hide Photos Multiple Accounts	App Hider	com.app.hider.master.pro	2.9.2_703d758f7	10M+
Hide Photos, Video and App Lock - Hide it Pro	Audio Manager	com.hideitpro	8.4	10M+
LOCKit - App Lock, Photos Vault, Fingerprint Lock	Lockit	com.ushareit.lockit	2.3.58_ww	10M+
Ultra AppLock-Ultra AppLock protects your privacy	ULTRA APPLOCK	com.ultra.applock	6	10M+
Clock - The Vault : Secret Photo Video Locker	Clock	ws.clockthevault	9	10M+

Table 3: Most Downloaded Android Vault Applications (Google Play Store).

Application Name	Name on Device	Package Name	Version	Number of Downloads
AppLock	AppLock	com.domobile.applockwatcher	3.3.2	100M+
Vault - Hide Pics & Videos, App Lock, Free Backup	Vault	com.netqin.ps	Varies	50M+
Keepsafe Photo Vault: Hide Private Photos & Videos	Keepsafe	com.kii.safe	10.2.15	50M+
Hide Pictures & Videos - Vaulty	Vaulty	com.theronrogers.vaultyfree	Varies	50M+
Gallery Lock (Hide pictures)	Gallery Lock	com.morrison.gallerylocklite	5.1	10M+
Pic Lock- Hide Photos & Videos	Pic Lock	com.xcs.piclock	3.1	10M+
Video Locker - Hide Videos	Video Locker	com.handyapps.videolocker	2.1.3	10M+
Gallery Vault - Hide Pictures And Videos	Gallery Lock	com.thinkyeah.galleryvault	3.19.8	10M+
LOCX Applock Lock Apps & Photo	LOCX	com.cyou.privacysecurity	2.3.9	10M+
Calculator Vault : App Hider - Hide Apps	Calculator	com.app.calculator.vault.hider	2.9.2_f0f859a1f	10M+

Table 4: Android Vault Applications Tested in Previous Work.

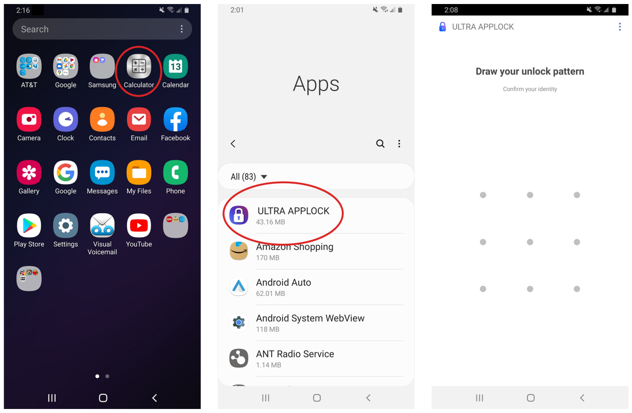


Figure 3: com.ultra.applock—This vault app is disguised as a calculator. However, when the inspector checks out the app from the app library, it reveals its original name "ULTRA APPLOCK" and its vault app icon. Also, when the inspector opens the app, it does not display the decoy app (calculator) interface. Instead, it directly displays the vault interface.

disguised apps so that they don't immediately raise suspicion. Third, the entry point to the hidden files within the disguised app should be oblivious to inspectors. For example, a calculator app can show the hidden files only after users enter a sequence of numbers into the calculator. This is more stealthy than popping up a dialog window asking for a PIN code. Fourth, disguises should be enabled immediately after app installation. Fifth, developers should apply encryption schemes to encrypt the stored files. Finally, developers should maintain a reasonable app size (e.g., by compressing the stored files) or by choosing disguises that are supposed to have a large size.

B APPENDIX: OTHER SUPPORTING MATERIALS

App examples are presented in Figure 1–3. The detailed app list is shown in Table 3 and 4. Table 2 shows additional analysis results.