VIDE - Vault App Identification and Extraction System for iOS Devices

By:
Gokila Dorai (Augusta University), Sudhir Aggarwal (Florida State University), Neet Patel (Florida State University), and Charisa Powell (Florida State University)

From the proceedings of
The Digital Forensic Research Conference
DFRWS USA 2020
July 20 - 24, 2020

Forensic Science International: Digital Investigation 33 (2020) 301007

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

DFRWS 2020 USA — Proceedings of the Twentieth Annual DFRWS USA

VIDE - Vault App Identification and Extraction System for iOS Devices

Gokila Dorai [b, *], Sudhir Aggarwal [a], Neet Patel [a], Charisa Powell [a]

[a] Department of Computer Science, Florida State University, FL, 32304, USA
[b] School of Computer and Cyber Sciences, Augusta University, GA, 30912, USA

ARTICLE INFO

Article history:

ABSTRACT

Content hiding (or vault) apps are a class of applications that allow users to hide photos, videos, documents and other content securely. A subclass of these applications called decoy apps further supports secret hiding by having a mode which mimics standard apps such as calculators but can turn into a vault app through entering a specific input. In this work we focus on iOS devices and first describe how to identify content hiding applications from the App Store. We consider not only the US Store but also give results for App Stores in Russia, India and China. We show an effective and very fast identification of content hiding apps through a two-phase process: initial categorization using keywords followed by more precise binary classification. We next turn to understanding the behavior and features of these vault apps and how to extract the hidden information from artifacts of the app's stored data. Based on this work, we have designed and built a fully automated vault app identification and extraction system that first identifies and then extracts the hidden data from the apps on an iOS smartphone. Using our vault identification and data extraction system (VIDE), law enforcement investigators can more easily identify and extract data from such apps as needed. Although vault apps are removed regularly from the App Store, VIDE can still identify removed apps as our system continues to maintain information on such apps in our vault database.

1. Introduction

The term *content hiding app* refers to apps that allow users to hide photos, videos, documents and other files secretly and in a secure way on their smartphones. Content hiding apps use hidden folders, locking mechanisms and sometimes encryption to implement this secure hiding. Such apps are also called vault apps, safe box apps, or app lockers. A more general term used for applications that compromise the availability of evidence is “anti-forensics” (see Harris (2006); Distefano et al. (2010); Azadegan et al. (2012)). A particular type of content hiding app called a *decoy app* is particularly interesting. This app operates in two modes: (1) a standard mode functioning for example as a calculator and (2) a decoy mode in which the app behaves as a secure private hidden storage app for data. An example is Calculator++ available from Apple's App Store. Calculator++ looks and performs like a simple calculator in almost all respects. However, on first installing the app, the user is prompted to enter a 4-digit code (a key). Whenever a user enters this key into Calculator++, the app immediately changes behavior. A hidden folder opens up and the app no longer looks or behaves like a calculator. Instead, the app now lets the user capture and store photos and videos. These photos and videos do not even show up in the Photos app of the iPhone. The data is secretly stored with access being available only through the key. In the article by Murphy (2016), the author describes various uses of content hiding apps by types of users ranging from teenagers to criminals. Teenagers use these apps to hide pictures, media and chat messages from their parents. Criminals use these apps to hide evidence of criminal acts from law enforcement. Many obvious criminal uses would be drug dealing, spying, etc.

In this work we focus on iOS phones. We first describe how to identify vault apps on the App Store. We consider not only the US Store but also give results for App Stores in Russia, India and China. We develop an effective and very fast identification of vault apps through a two-phase process: initial categorization using standard keyword search, followed by a more precise binary classification using machine learning techniques. Identified apps are stored in a database. Using the database we can determine which apps on an iPhone are in fact vault apps. Next, a logical acquisition of the smartphone is done using iOS backup utility tools and from this set

* Corresponding author.
E-mail addresses: gdorai@augusta.edu (G. Dorai), sudhir@cs.fsu.edu (S. Aggarwal), nhp15@my.fsu.edu (N. Patel), ckp13b@my.fsu.edu (C. Powell).

https://doi.org/10.1016/j.fsidi.2020.301007

of data, hidden data and other information from each suspect vault app is automatically extracted and presented as a report. We call our system the *Vault Identifier and Data Extraction* (VIDE) system. VIDE is designed for iOS devices but we believe that our approach can easily be extended to Android based devices.

We expect that VIDE would be used by forensic investigators to quickly identify and extract relevant data from vault apps on a suspect's iPhone. It could be used for forensic analysis of a live device with appropriate caution related to data stored in the cloud. VIDE could also be useful for any user concerned about potential vault apps with private data hidden on a smartphone. Additionally, the determination of vault apps on the App store could itself be of independent interest to forensic analysts.

We believe our novel contributions are the following:

- The first in-depth exploration of content hiding apps for iOS devices.
- Capability to do rapid identification of content hiding apps from any App Store, including app stores such as in Russia, India and China.
- A lightweight vault app identification system with fairly high accuracy and using only textual description of apps.
- An automated vault app evidence extraction system for content-hiding apps installed on iOS devices.

The paper is organized as follows. In section 2, we discuss related work and in section 3, we describe the high-level design of VIDE. In section 4, we describe our work on the identification subsystem including design, implementation and evaluation. In section 5, we briefly digress to explain our work in understanding artifacts hidden in vault apps. In section 6, we describe the design and implementation of the extraction subsystem. We conclude with some comments in section 7. In the paper we try to include all information needed to support reproducible research.

2. Related work

Commercial forensic tools such as Cellebrite UFED (Cellebrite, 2018), (Paraben, 2018) and (Oxygen, 2018) are capable of performing physical/logical acquisition and typically extract all of the data from the device. They display the data in terms of files and folders. As far as we are aware, however, there are no commercial forensic tools that identify which apps are content hiding apps in order to direct the investigator to the relevant files and folders. Furthermore, even if one knows the targeted app of interest, these tools do not automatically extract the relevant data. Simply exploring the complete set of files and folders causes extensive overload for the investigator. In the thesis by Zhou (2016) the author categorizes apps by a set of features ("claimed functionality") by mining app descriptions as well as API calls from.apk files in Android devices. The goal of the work was to then classify apps based on these features. It was proposed that this approach would help developers decide in which App Category to place a new app. For iOS,.ipa files are not readily available for third-party applications and thus categorization by API calls is not possible. Furthermore, the goal of the work is an inverse problem to ours: place the derived clusters in the proper Google Play Store category. This may not be relevant for identifying genres such as content hiding apps which span multiple categories.

Pandita et al. (2013) discussed steps towards keeping malware out of smartphone app markets. As a first step, the authors developed a framework called WHYPER which used NLP techniques to identify sentences in app descriptions and determine whether the description supports the need for particular permissions. In a related vein, Qu et al. (2014), Watanabe et al. (2018) and Gorla et al. (2014) used textual description of android applications to determine app behavior. In our work, the goal is to automatically and quickly identify the class of vault applications based on textual descriptions that describe the behavior.

There are very few papers that focus on forensics related to content hiding apps. The paper by Rughani (2017) is one of the first to analyze content hiding android applications. The paper does an analysis of data, encryption and file permissions for content hiding apps and shows examples of how frequently these features are used. Zhang et al. (2017) explore the details of several popular vault apps for Android devices using methods of reverse engineering and forensic analysis. Their goal was to determine how vault apps were designed and maintained, as well as to "break" into the vault by finding passwords in the data. Duncan and Karabiyik (2018) explored detection and analysis of vault applications on android devices. Their work differs from ours since they did not explore the issue of identification of vault apps using app descriptions nor did they automate the forensic extraction process. We believe that we are the first to focus on iOS devices and also the first to do comprehensive and automated identification and extraction.

3. VIDE system design

Although vault apps might not want to be identified as such on a smartphone, it is important that the app in the App Store be reasonably named so that a user would be able to find it for use. Information related to an app is available from the *title*, *subtitle* and *description* which are all displayed for the app in the App Store. The title and subtitle contain brief information related to the app but are usually sufficient for a user search. The description gives more detailed information about functionality. Note that the display name of the app (technically called *Bundle display name* by Apple) on the phone need not be the same as the title found on the app store and thus could hide the functionality of the app. For example, a vault app with display name Calculator + had the title Calculator App Lock - Keep Secret Photo Album Safe. In this work we assume that the full text information of an app can be used by users to identify it as a vault app or not. Apple has strict review guidelines, but it certainly might be possible to circumvent these and post false descriptive information about a vault app on the App Store. We speculate that the user base would likely be small for such an app.

Apps in the App Store are currently organized into about twenty five categories (such as Books, Business, etc.). In preliminary work we mimicked a user search for content hiding apps and they seemed to be located in only four categories: *Photo & Video, Productivity, Utilities and Business*, which we term the *significant* categories. Examining the App Store review guidelines for app developers (Apple, 2019b) it was clear that these four categories should have all the content hiding apps and that it was quite unlikely that other categories would have any such apps. Thus we restricted our focus for identification of content hiding apps to these four significant categories.

3.1. General vault app functionality

The possible interfaces of most content hiding apps are various such as a calculator, a calling keypad, a pattern-entry keypad, a stock trading interface, a gaming interface, etc. Entering a passcode typically unlocks the app. Storage of pictures and videos is common but other data is also often stored. The apps often function in two different modes: main mode and decoy mode. A second passcode is required to enter decoy mode. The apps may also support encryption of the data stored. In section 5 we discuss more details related to the functionality and analysis of these apps.

3.2. High-level design

Our system design is shown in Fig. 1. The goal of the VIDE system is to extract artifacts from vault apps and not from other apps on the phone. The system consists of two major components: an identification system and an extraction engine. The identification system consists of the following components: a phase one initial categorization system, the Vault Apps Database (DB) and a phase two classification system. The extraction engine consists of the following components: a logical acquisition system, a vault app detection system, an artifacts extraction system and a report generator. The vault app detection system of the extraction engine takes input from the Vault Apps DB.

The phase one initial categorization system does a keyword-based search on the title and subtitle text to determine all potential vault apps (PVAs) from the identified significant categories in the App Store iTunes Preview. A potential vault app is one that is likely to be a vault app but not certain to be one. All additional information related to the PVAs, and particularly the descriptions, are also downloaded. The phase two secondary classification system then classifies the potential vault apps more precisely into vault or non-vault using binary classification. Results from classification are stored in the Vault Apps DB. An example DB table with only a subset of the attributes is illustrated in Fig. 2. The implementation of the identification system was mostly done using Python.

In the extraction engine, we first do a logical acquisition of the iOS device. From this data, the Info.plist file of the device is parsed to obtain the list of apps installed and their respective bundle identifiers. The Vault Apps DB built using the identification system stores the app title, bundle-id and other information for each app classified as a vault app. The vault detection system and the artifact extraction system can thus identify vault apps on the phone based on their bundle-id and then extract artifacts from the detected apps. In section 6, we describe in detail the vault detection and automated extraction system.

3.3. International app stores

In our system design we also support identification and extraction from content hiding apps in all App Stores beyond simply the US App Store. Apple maintains App Stores in five regions of the world. For example the US, Canada and Puerto Rico are in one region. Each region supports a number of separate App Stores (the sub-regions). There are a total of 134 App Stores as of this time. A smartphone when connected to the internet in a certain region can only download apps from the specific App Store related to the billing address associated with the Apple ID account (Apple, 2019c). It is thus not possible to download such an app from a different region. However, in our approach downloading is not required to access the relevant information needed to identify content hiding apps in those regions, as we show in section 4. Besides the US App Store, we illustrate work related to examining the App stores in Russia, China and India.

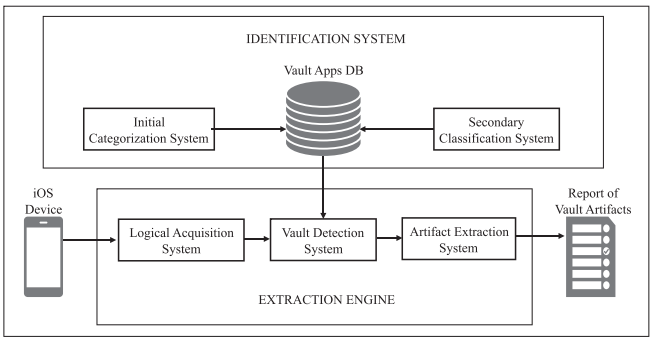

Fig. 1. Vide System DDesign.

4. Identification of vault apps

For developing the identification system, we used the US App Store which contains approximately 2 million apps. We focused only on the significant categories: Photo & Video, Productivity, Utilities and Business containing approximately 0.5 million apps. Even the significant categories, however, have a very large number of apps and thus we needed to develop our two-phased methodology of initial keyword categorization followed by detailed binary classification. Traditional classification techniques require having the full text data available which would require having title, subtitle and description available for each of the 0.5 million apps. Apple organizes the title and subtitle differently from the description and the time taken to download the description, is substantially greater than for title and subtitle. It is essentially prohibitive to download all three information components from all the 0.5 million apps.

Thus we developed our heuristic keyword search to scan only for title and subtitle to drastically reduce the number of descriptions we needed to download. This initial categorization gives us the set of apps termed potential vault apps (PVA). Our goal for this initial categorization was to: (1) ensure that most vault apps are in the PVA set (few false negatives); (2) have a reasonable sized PVA set but allow some number of false positives (PVA apps that are not vault apps) to be included; and (3) determine the PVA set very quickly. For a more precise classification of the PVA set into vault or non-vault, we initially explored using a Boolean rule-based classifier. In our brief testing we found that we seemed to get too many false negatives using heuristically derived Boolean (and, or, not) expressions. Work on learning constraints from examples (De Raedt et al., 2018; Kolb et al., 2017) indicates that there is no known approach to get an optimal Boolean expression for Boolean classification.

Once we have this PVA set, we complete the identification of a PVA by using classification based on the full information available about an app (title, subtitle and description). The techniques we used for classification are Gaussian Naive Bayes (NB), Support Vector Machine (SVM) and Decision Trees (Tan, 2018). These are straight forward and an optimal rule-based classifier in not likely.

4.1. Initial categorization and analysis

For the initial categorization, we first looked at the complete text information for a small set of vault applications. We then chose a set of keywords we felt would cover any content hiding app in the sense that at least one of the keywords must be in the app title for a user to even find a vault app. The list of eleven keywords we chose for the initial categorization was [private, sensitive, censor, protect, decoy, privacy, secret, hide, vault, secure, safe]. We could have chosen to explore fewer or greater numbers of keywords but as we describe later, this set seems sufficiently large. See Table 3. We call this set of keywords the *scan set*. When doing the initial categorization for a non-US App Store we use both the English words as well as the equivalent translated primary language word(s). For the India App Store we used only the English scan set since most of the App titles seemed to be in English rather than Hindi or other regional languages. For the Russian App store we used both the English and Russian scan set and for the Chinese App Store we used both the English and Chinese scan set.

Sequentially scanning each app in the significant categories, we

	id	name	description	url	lookup_url	bundle_id	tag
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1263488666	Redacted - Censor Private Content remo...	Redacted - Private Content Re...	https://itunes.a...	https://itunes.a...	jack.redacted	0
2	845266002	Private Apps - Secure personal data man...	*** PRIVATE APPS *** *** PIN ...	https://itunes.a...	https://itunes.a...	com.mainhung....	1
3	792410101	Private Photo - Protect Privacy, Photo a...	Private Photos protects photos...	https://itunes.a...	https://itunes.a...	ch.b-eng.private...	1
4	951306034	Private video locker - safety protect you...	Import from a PC or Mac and pl...	https://itunes.a...	https://itunes.a...	private-video-lo...	1
5	651767664	Contact Lock - Secure and Protect Your ...	Don't want other guys view yo...	https://itunes.a...	https://itunes.a...	com.stickto.Sec...	1

Fig. 2. Screenshot of the *apps* Table from the Vault Apps Database.

check if the app title contains one or more of our scan set keywords and if so we store the title and app_id as a PVA in our Vault App DB. Our Vault App DB consists of seven tables, with the most important being the *apps table*.

In this table, for each PVA found in the initial scan, we populate all the fields except bundle_id, subtitle and tag. We get the name (title), description, url and also form a lookup_url which is then used to get the bundle_id and subtitle. The *tag* field stores our classification of a PVA as vault or not vault based on the further classification. The *app_id* is the primary index for the Apps Table as is unique to an app. Our system takes into account the usage of different character sets such as Russian, Korean, etc. The database also contains other tables which help in obtaining statistical data on apps such as which region an app was found in, or using which keyword(s).

Fig. 3 shows the numbers of potential vault apps that were found using each of the scan set keywords in the legend (only English version shown) for each region. Remember that a PVA is found by one or more keywords so the numbers shown do not add up to the actual numbers of PVAs found. It seems clear that the pattern of success based on a keyword search is similar for each region. Table 1 show the actual counts for the Russian App Store. The keyword "decoy" in the title is surprisingly not very useful. Note that each English scan set keyword also has a translated word in Russian.

Table 2 shows the initial scan statistics for each of the regions we analyzed. The reduction using the scan set from the original numbers of apps is uniformly substantial. For the US store with 477,002 apps in the significant categories, the initial scan reduces this set to 2364 PVAs, which is 0.50%. The time taken for the initial scan is 48 min running on a MacBook Pro with Mac-OS Mojave 10.14.4, 2.3 GHz Intel Core i5 processor and 16 GB RAM. Table 3 shows the number of vault apps found using each scan set element on the US store, divided up by the significant categories. Each app in the App Store can belong to at most two categories

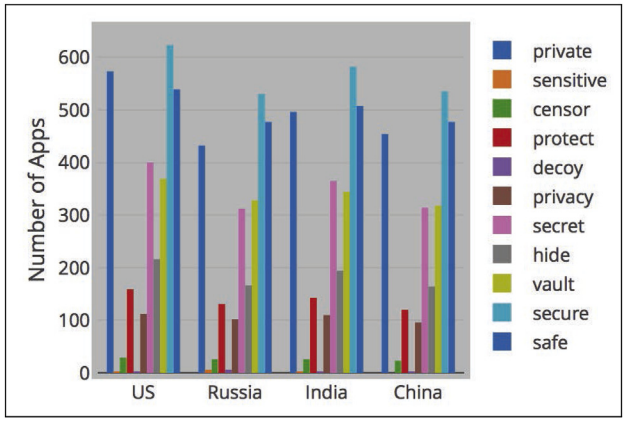

Fig. 3. Content hiding app search by keywords for various regions.

called the "primary category" and the "secondary category" (Apple, 2019b). Note that if we do not follow our two-phased approach and we download descriptions for all of the 0.5 million apps for classification, the time taken would have been 9,700 min just for the download.

Table 4 shows statistics for the numbers of PVA apps found in the US store as well as other regions. 1735 apps were common to all the four regions: US, Russia, India and China. Of the 2364 PVAs from the US App Store, we found that 178 were removed within a period of 3 months from the date of the initial scan. However, these apps still continued to be fully functional on iOS devices on which they were previously installed. Users are unfortunately not informed about the app removal nor the cause for the removal. According to Apple, the cause of app removal from the App Store could be due to various reasons such as the app being found to be malicious (Apple, 2019d; Davey Winder, 2019), outdated or crashing on launch (Apple, 2019a). The fact that 7.5% of PVAs were removed by Apple within 3 months of our initial scanning shows that many apps being used in practice no longer have any information about them in the App Store. In practice, our VIDE system would periodically scan an App Store to identify new vault apps. However, we would continue to maintain a list of previous vault apps found even if they have been removed from the App Store. Investigators, without using VIDE, manually trying to identify vault apps on phones by checking the App store, may not find these removed apps.

4.2. Classification and analysis

In this section we describe manually labelling apps and developing our phase two classification system.

4.2.1. Ground truth

To further explore the best way to identify vault apps, we first manually labeled the 2364 PVAs. For the manual labeling, three members of our team classified each app using the app URL and all the information available about each app which was title, subtitle, description and the app icon. Each labeller used this full information to classify each app as vault or non-vault. Any app not unanimously labelled was checked by the group and then labelled after discussion. Our basic guidelines for deciding an app was a vault app or not was whether or not the goal of the app was to hide any information, essentially whether it was used for anti-forensic purposes. Each labeller also noted some other features of each app: decoy, spying, dual mode, supports encryption, and password storage. The manual labeling of the PVA set resulted in 1118 apps with label "vault" and 1246 apps with label "non-vault."

We decided to also check the performance of our initial scan on a set of randomly chose 600 apps from the App Store in the significant categories; we stored and manually labeled those in the same way as for the PVAs. Note that vault apps seem to be a very small fraction of all the apps in the significant categories. Of these 600 manually labeled apps there was only one that we had manually labeled as a vault app. On further checking we found that it was also in our initial scan set. Thus there were no false negatives

Table 1
Vault apps by keywords (Russian app store).

Region: "Russia"			
KID	(RID)	English Keyword (Russian Keyword)	Count
1	(12)	private (ЧастНый)	432
2	(13)	sensitive (ЧувСТвИТеЛЬНый)	4
3	(14)	censor (цеНЗОР)	26
4	(15)	protect (ЗащИщаТ)	131
5	(16)	decoy (МаНОК)	4
6	(17)	privacy (ОНфИдеНциаЛЬНОСТЬ)	102
7	(18)	secret (СеКреТ)	312
8	(19)	hide (скРЫваТЬ)	166
9	(20)	vault (СвОд)	328
10	(21)	secure (бе3ОПасНЫй)	530
11	(22)	safe (бе3ОПаСНО)	477
Total number of apps = 1968			

Table 2
Statistics of initial categorization.

Item	USA	Russia	India	China
Apps in significant categories	477002	448389	451432	460551
Number of potential vault apps	2364	1968	2169	1986
Run Time initial categorization	48m2s	38m1s	39m43s	39m37s

in the randomly chosen 600 apps, which gave us some confidence in our approach for the initial scan.

4.2.2. ML based binary classification

For the second phase binary classification, we considered three different binary classifiers: Gaussian Naive Bayes (GNB), Support Vector Machine (SVM) and Decision Tree (DT). For the purpose of binary classification, we heuristically added additional keywords (features) to the previous list of 11 keywords resulting in a total of 20 keywords that we term the feature set. The features we added were designed to try to minimize false positives such as banking related apps and safety apps that we noted were picked up in the initial scan but were not vault apps. We also wanted to keep the number of features reasonably tractable. Thus, we did not try to learn the features automatically.

The binary classification into vault/non-vault is based on occurrence or not in full text information available for a PVA app: *app title, app sub-title and app detailed description*. The full feature set is ['private', 'sensitive', 'censor', 'protect', 'decoy', 'privacy', 'secret', 'hide', 'vault', 'secure', 'safe', 'photos', 'videos', 'notes', 'passwords', 'contacts', 'password-protected', 'password protected', 'browser', 'private browser']. Note that when using a different language, the occurrence is true if either the keyword or the translated keyword appears.

Our full labeled dataset consisted of 2963 applications. This labeled dataset has 1118 vault and 1845 non-vault apps. For our training and test set we used all the 1118 vault apps and 690 randomly chosen apps from the 1845 non-vault apps for a total of 1808 apps. It was more important to accurately classify a vault app compared to a non-vault app. The training and testing dataset was partitioned into 60% (Training Set) and 40% (Test Set) with random seeds. We used 4-fold cross validation for each classifier type.

The classification reports for each of the classifiers are shown in Tables 5–7. Table 8 shows the cross validation scores and accuracy values. The F1 scores show that all the classifiers perform reasonably well and that we achieve a 0.9 F1 score using SVM. It should be noted that when periodically doing another initial categorization

Table 4
Statistics of apps found in various regions.

Region	Count
Apps found in US region as well as Russian region	1831
Apps found in US region as well as Indian region	2015
Apps found in US region as well as Chinese region	1836
Apps found in US, Russia, India and China	1735

Table 5
Classification report of Gaussian NB classifier (Support = 724).

	Precision	Recall	F1-Score
0 (non-vault)	0.73	0.93	0.82
1 (vault)	0.95	0.79	0.86
Average	0.87	0.84	0.85

Table 3
Apps searched using keywords for popular app store categories (primary and secondary categories).

Region: "US"				
Keywords/App Categories	Business	Utilities	Productivity	Photo-Video
private	109	331	202	204
sensitive	0	2	1	1
censor	1	15	0	27
protect	37	110	43	26
decoy	0	1	0	0
privacy	14	80	54	22
secret	37	276	117	166
hide	8	158	56	135
vault	55	254	125	178
secure	216	367	324	72
safe	114	357	176	119
Total App Count	679	1760	1099	798

Table 6
Classification report of SVM classifier (Support = 724).

	Precision	Recall	F1-Score
0 (non-vault)	0.91	0.83	0.86
1 (vault)	0.90	0.95	0.92
Average	0.90	0.90	0.90

Table 7
Classification report of decision tree classifier (Support = 724).

	Precision	Recall	F1-Score
0 (non-vault)	0.82	0.89	0.85
1 (vault)	0.93	0.88	0.90
Average	0.89	0.88	0.88

Table 8
Cross Validation Scores of Classifiers and their Accuracy (4-fold).

Category	Run 1	Run 2	Run 3	Run 4
GNB Cross Validation Score	0.854304	0.783664	0.840707	0.764966
GNB Accuracy	0.81 (+/− 0.07)			
SVM Cross Validation Score	0.891832	0.891832	0.89601	0.889135
SVM Accuracy	0.89 (+/− 0.00)			
DT Cross Validation Score	0.87858	0.883002	0.878318	0.853658
DT Accuracy	0.87 (+/− 0.02)			

we can filter very quickly using the title/subtitle only, albeit getting many false positives. For the classification, once we have the models for each classifier type, we can use the best model whenever needed. We simply identify the new current set of vault apps from the new set of PVAs using our trained classification model. We used the three trained classifiers to check for vault apps in the Russian, India and China app stores based on initial scans of these stores. The estimated numbers of vault apps are shown in Table 9. For comparison we also show the estimated number based on our initial scan for the US store using these same trained classifiers.

5. Forensic analysis of vault apps

In this section we do a detailed analysis of several vault apps to understand the relevant artifacts and storage paths. There are mainly two popular types of acquisition methods: logical and physical. Logical acquisition is used to extract user data from the file system and does not include deleted files. An approach for leveraging the iPhone backup files created using the iTunes backup utility was presented by Bader and Baggili (2010), and Morrissey and Campbell (2011). Most of the allocated data can be retrieved using the backup methodology as stated by Hoog and Strzempka (2011). This method has also been considered as a leading logical acquisition method for iOS devices by Tso et al. (2012).

Our work is based on the investigation of files and folders obtained using logical acquisition of two iPad devices: one non-jailbroken device and one jail-broken device. A factory reset of the iOS device was performed to wipe all previous contents and settings before we conducted our experiments. iOS 9.3.5 was installed on one of the iPad device followed by the installation of several software package managers. ZJailbreak (Jailbreak11, 2018), Cydia (Freeman, 2018a) and various file system tools (AppleBetas, 2018; Freeman, 2018b) were used for rooting a device. A list of tools and their uses is shown in Table 10. The jail-broken device had limitations on the version of apps that could be installed, as many newer applications require iOS 10 or above.

Table 9
Estimated Number of Vault Apps from International App Stores using Trained Classifiers.

Classifier	US	Russia	India	China
Gaussian Naive Bayes	1091	768	874	767
Support Vector Machine	1680	1272	1413	1263
Decision Tree	1412	1037	1171	1038

In general, content hiding applications are protected by either a bio-metric, PIN, or alphanumeric password. Access to the content within the application is only granted if the correct authentication information is provided. Content hiding applications are used for various purposes such as: storing multimedia, storing passwords, private browsing, decoy mode of operation, spying and storing encrypted contents such as images and text.

Multimedia vault apps typically allow users to capture and store photos, videos and audio recordings. This is the most popular type of content hiding app, as users often seek a safe method to hide personal media from prying eyes. Another type of vault app is primarily meant for hiding passwords (banking, email accounts, etc.) and uses textual content storage. The template-like structures used in these apps allow users to customize their data organization by adding and renaming fields. Another popular category of vault apps are those used for private browsing. Using the underlying web-browsing mechanism by Apple, users are able to “privately” browse the internet without leaving any trace of their history. These apps contain a very minimalist browser as a primary interface and some also have the capability for saving bookmarks. A very interesting category of vault apps are those that have a *Decoy* mode. The decoy/ghost/fake mode is an interface within a content hiding app that can perform one of these two functions:

- Store data (photos, videos, etc.) using a passcode different from the passcode used for the main mode.
- Behave as a different app (other than content hiding) such as a calculator or clock, to appear as an innocent non-vault application.

This decoy capability poses an extra layer of difficulty for investigators attempting to look up data on a suspect's iPhone device. Based on our manual verification and categorization of vault applications, it seems that decoy applications that either appeared to have a different function (using a decoy interface) or contain a separate data set (in the decoy mode) are primarily used for multimedia storage.

The goal of our automated analysis described in section 6 is to iterate through the application directory (obtained using logical acquisition) for each identified vault app and extract relevant artifacts. To understand how to do this, we analyzed a set of known vault applications to determine functionality, features and data storage structures. This involved the following steps:

Table 10
List of software tools and devices used for experiments.

Software/Device	Source	Purpose
ZJailbreak	Jailbreak11	For jailbreaking
Cydia	Jay Freeman	File system software
Impactor	Jay Freeman	To install IPA files
OpenSSH	OpenBSD Project	To use a Secure shell
MTerminal	AppleBetas	For iOS terminal commands
idevicebackup2	libimobiledevice	Backup utility tool
Google Translate	Google	Keyword translations
iPad (iOS 9.3.5)	Apple Inc.	Experimental Device-I
iPad (iOS 10.3.3)	Apple Inc.	Experimental Device-II
iPhone (iOS 12.3.1)	Apple Inc.	Experimental Device-III

- Download the app, add some user content, and identify features and functionality by interacting with the app.
- Using logical acquisition selectively extract the relevant vault app folders.
- Finally, explore stored artifacts and storage structures, and extract the data.

The logical acquisition process results in a file structure with hashed file/folder names (Piccinelli and Gubian, 2011). The "Manifest.plist" file contains a list of hashes that can be matched with each individual application-specific folder. Applications were installed on both the iOS devices (jailbroken and non-jailbroken). As previously mentioned, the jailbroken device (with iOS 9.3.5) was not compatible with some applications due to the minimum OS requirement of iOS 10. Hence, on the jail-broken device, we were able to use only a restricted number of applications for our analysis. Apps listed in Table 11 were specifically chosen based on different purposes, such as private browsing, photo storage, potential encryption, etc. User data from the selected content hiding applications was recovered through extraction and analysis. Typically, most third-party applications obtained using logical acquisition have very similar directory structures at the top level. This structure includes a "Documents" folder and a "Library" folder. Fig. 4 (com.red.msf) shows the directory structure for the application called "My Secret Folder," which has the capability of storing images, audio, and notes. This directory structure was found to be more or less similar for several content hiding applications. A summary of all recovered artifacts is shown in Table 11.

In general, the high-level components of most content hiding applications typically contain some or all of the following items:

- Database files (.db or.sqlite) with links to images (such as on Amazon servers).
- Sub-folders with images/thumbnails/audio.
- *plist* files containing user/application metadata.

In the following we discuss interesting forensic artifacts retrieved related to some of the apps we manually analyzed. We note that many of the apps used some type of database file to store application metadata. From these files, content such as contacts, banking information, passwords and other sensitive data were easily recovered.

Fig. 5 shows artifacts extracted from the sqlite database file of the My Wallet Lite app, located within the Documents folder. The columns from left to right are: description, sub-description, value, record-deleted and timestamp. The example shows credit card information (for one card) that we entered using the app. The values highlighted are elements that were manually input by the authors to be stored in the My Wallet Lite vault app. The expiration date is empty as we did not enter this information. The last column shows the time when we entered the data. The database table above was viewed using an open source tool called sqlite browser and it is trivial to dump user information from desired columns with an SQL command. Note that no knowledge of the passcode for the app was needed to access this information.

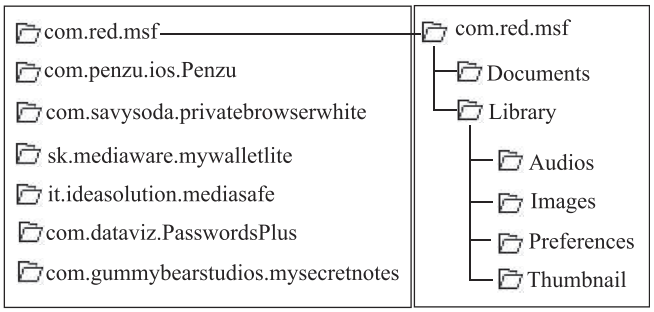

Fig. 4. Example of a Typical File Structure of Vault Apps (figure expands "My Secret Folder").

Name	Bank, card na...	Test1	0	1280544501
Owner	Card holder	john appleseed	0	1280544501
Category	Card type	Testcard	0	1280544501
Kind	Card name		0	1280544501
Number		1234 5678 12...	0	1280544501
Expiration			0	1280544501
CCV	Security code	123	0	1280544501
PIN		1234	0	1280544501

Fig. 5. Recovered user data from my wallet lite.

Penzu is a free application that is designed to let users create journals with multimedia features. While this app does not use encryption of content, it does allow for journals to be locked with an alphanumeric password. Following an edit to a journal, all changes are saved to the device as well as to Penzu's servers which allows for broader accessibility. This feature is only possible since the application requires an account registration before inputting any content. From the Penzu application folder, it was relatively trivial to find the user's data, regardless of whether or not the journal was locked with a passcode. Several artifacts including thumbnail of images and files were recovered from the "Documents" directory of this app. Of particular interest is the database file labeled "penzu.db". This file is the key component to gaining access to the content stored on Penzu's server running in Amazon AWS (see Fig. 6 - value of 'remoteFullURL'). The remote URL is contained within the database and contains links to all images that are enclosed within journals on the device. Access to this image did not require logging into the Penzu account. Fig. 7 shows metadata that can be extracted: password hint for the journal, first name of the user ("Mobile") and an associated password hash from the account. This is a prime example of also being able to access sensitive data from the Cloud since we can access the stored URL

Table 11
List of apps analyzed.

App Name	Bundle-ID	Passwords	Notes	Multimedia	Contacts	Encryption	Recovered Artifacts
My Wallet Lite (Kanzelsberger, 2014)	sk.mediaware.mywalletlite	✓				✗	Passwords
My Browser White (SavySoda, 2017)	com.savysoda.privatebrowserwhite					✗	Browsing history
My Secret Notes (GummyBearStudios, 2018)	com.gummybearstudios.mysecretnotes		✓			✗	Saved notes
Private Journal (PenzuInc, 2018)	com.penzu.ios.Penzu		✓	✓		✗	Notes
Secret Photos (IdeaSolutions, 2018)	it.ideasolution.mediasafe		✓	✓		✓	Audio, Encrypted images
My Secret Folder (RedKnight, 2018)	com.red.msf		✓	✓	✓		Photos, Audio, Notes
Passwords + (DataVizInc, 2017)	com.dataviz.PasswordsPlus	✓	✓		✓	✓	None

remoteFullURL	https://s3.amazonaws.com/1silo.penzu.com/16861517/photos/huge/penzu_iOS_15415...
type	Photo
uploadRequired	0
localFullFileNa...	54593517-B6B6-495F-BEEF-029A64E198DD-0-full.jpg
localThumbFil...	54593517-B6B6-495F-BEEF-029A64E198DD-0-thumb.jpg
downloadReq...	0
remoteThumb...	https://s3.amazonaws.com/1silo.penzu.com/16861517/photos/tray/penzu_iOS_154152...
syncedAt	2018-11-06T12:24:35-0500

Fig. 6. Remote cloud storage url from penzu.

type	Journal
syncedAt	2018-12-01T18:06:47-0500
isDefault	1
passwordHint	lockedd
syncRequired	0
rememberPas...	0
journalEmail	setup@post.penzu.com
name	Mobile's Journal
passwordHash	$2a$10$6GHoD/AlpPVRzUOl3GufFuCUBBenclSrIAwGi5nNS908mR6IlxbMm

Fig. 7. Journal metadata from penzu.

information.

My Secret Folder is a vault app that stores photos, videos and notes. My Secret Folder provides the user with a way to store multimedia data where photos are stored into respective albums in the application. It turns out that all data is stored (unencrypted) in the Library folder. The naming convention for this application follows the pattern of the album name followed by the image number. Audio files (.m4a format) are stored in a corresponding directory called "Audios" and named with a timestamp at the time of recording. Another feature of this application includes "break-in reports." These typically consist of a log based documentation of any invalid login attempts. In the case of My Secret Folder, a photo is taken automatically using the front-camera upon every invalid login. The location coordinates are also stored while the invalid login takes place. This data was recovered by accessing the "secretfolder.sqlite" database file, from within the application folder.

Secret Photos - KYMS Free (IdeaSolutions, 2018) is the only application within the study that actively advertised "military grade AES-encryption." This free version of a paid app lets users store photos, videos, recordings and documents in an encrypted manner. The directory structure within the backup is similar to the other applications, however there is no unencrypted media locally stored within the application folder. Likewise, the database file requires a key to open. An investigator can identify that content has been added to the application by noticing "[filename].jpg.encrypted," or something similar. The key to extracting user data from this application was identifying unencrypted strings from the database files. While some strings of the database were in plain text, the majority of other files were encrypted. KYMS stores the decoy mode contents in the "._collections.fake" directory, and maintains separate encryption keys as indicated in Fig. 9 Part-3. An overview of recovered files and folders from the Secret Photos app by KYMS is shown in three different parts of Fig. 9.

Our manual analysis gave us sufficient information about the typical structures of vault files, folders, databases, etc. that allowed us to develop an automatic data extraction component for vault apps. Using logical acquisition we obtained similar information from both jailbroken and non-jailbroken devices. Hence we can surmise that vault apps were not preventing iOS from backing up

{'iRateEventCount': 0, 'num-albums': 1, 'correct': True, 'background': 'white', 'passcode': 'abcd', 'num-passwords': 0, 'signedUp': True, 'browser-reset': datetime.datetime(2018, 2, 17, 2, 33, 51, 838522), 'WebKitShrinksStandaloneImagesToFit': True, 'fullversion': False, 'iRateUseCount': 5, 'breakin': True, 'WebKitLocalStorageDatabasePathPreferenceKey': '/var/mobile/Containers/Data/Application/AB377F43-1347-4975-8C32-F6E008F16E42/Library/Caches', 'pending': 'abcd', 'gotbrowser': False, 'iRateFirstUsed': datetime.datetime(2018, 2, 17, 2, 32, 3, 962735), 'num-files': 0, 'limit': 5, 'cloudsync': False, 'passtype': 'alpha',

Fig. 8. A Snapshot of com.appostrophe.privatecalc.plist File Showing Passcode, Cloud Status, Dual Mode and Album Information.

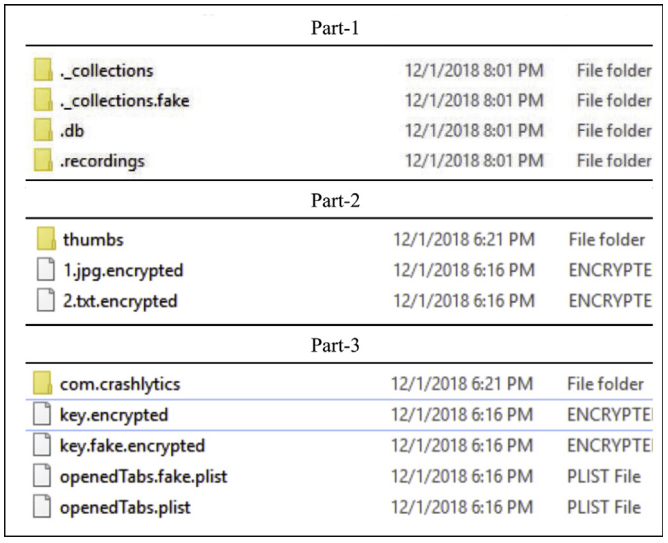

Fig. 9. KYMS app artifacts: Part-1. Decoy contents, Part-2. Encrypted images, Part-3. Potential keys.

certain files and law enforcement would not need to jailbreak a phone to extract content from such apps. We discuss the automated component of VIDE in the next section. Note that our extraction engine only extracts the needed selective data from the vault apps and does not need to touch data in other apps.

6. Automated extraction engine

The goal of the extraction engine is to extract all information possible from each vault app on a target iOS device. The vault detection system component of our extraction engine compares the Bundle IDs stored in the Vault Apps DB with the Bundle IDs of all applications installed in the phone in order to determine which are the vault apps in the iOS device. The artifact extraction system will then extract only the artifacts from these vault apps. In order to get this data, the possible storage structures of the data stored by (third-party) vault apps needed to be identified, as described in the previous section.

We were able to do this because fundamentally iOS developers have to follow certain developer guidelines laid out by Apple in order to organize data storage within iOS applications. Hence, most applications tend to use popular file types such as plist files, json files, and sqlite database files, and store them within *Preferences*, *Library* and *Document* containers (iOS app directories). When we connect an unlocked iOS device to a computer followed by launching our VIDE tool, the computer and the iOS device are paired and logical acquisition is done. VIDE then extracts the relevant files and folders from the detected vault apps and parses them appropriately. The extraction engine first organizes the

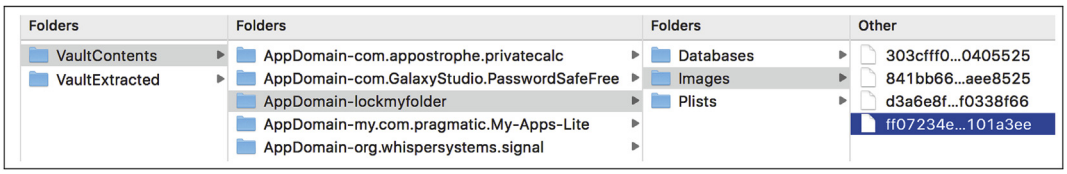

Fig. 10. Organization of extracted vault contents.

extracted artifacts by the App's Bundle-ID and artifact type as a high-level overview. An example of this high-level view is shown in Fig. 11, outlining various artifacts identified, all files which were extracted from the vault apps and other useful information. More detailed information about each plist file, sqlite file, image files, etc. is then identified for each vault app.

An example of a plist file which we extracted from one of the vault apps is shown in Fig. 8. We can see various critical details such as the passcode information, whether the cloud was used to backup the artifacts or not, whether the cloud syncing feature was turned ON and whether the app can function in dual mode or not. Such information is identified by searching for various keywords such as *pass*, *passcode*, *cloud*, *sync*, mode, alternate, etc. in the plist/json files. We additionally list the schema, tables and columns of any database files and related metadata. This information thus identifies aspects such as break-in, date and time at which a file/content was stored, whether any attempt was made to delete a particular content, etc., in addition to the data entered by the user and stored by the vault app. Once a vault app has been detected by VIDE, the app group location is identified using the *Manifest.db* file. Based on the file type/content, it will be placed in one of four folders inside a folder named after the apps's bundle identifier: Plists, Databases, Media and Others. For example, files with extensions jpeg, jpg, png, mov, mp3, and wav are classified as Media files. An example of the organization of extracted vault content from our experiment is shown in Fig. 10.

In order to test VIDE, we ran several experiments by downloading and installing various vault applications on iOS devices which were identified using our initial scan. In one of our experiments (the overview shown in Fig. 11), we loaded an iOS device with 5 different vault applications randomly chosen from the App Store. Their Bundle-IDs are: com.GalaxyStudio.PasswordSafeFree, com.appostrophe.privatecalc, my.com.pragmatic.My-Apps-Lite, org-.whispersystems.signal and lockmyfolder. Our iOS device already had about 65 apps installed in it. All of the vault apps were populated with some random data that we input. We found that VIDE was able to extract most of the artifacts from these vault apps. Irrespective of whether a vault app's album was protected with a passcode or not, VIDE was able to extract all images from that album. There are certain types of data extraction however that VIDE is not able to support currently. Consider the example of "my.com.pragmatic.My-Apps-Lite.plist" which represents the Bundle-ID of a vault app that is backed up in the cloud. It contained a *PICODB* file which VIDE extracted and placed in the *Others* folder. However we did not have the information needed to parse the content of this file. In another instance, VIDE extracted the contents of a private messaging (content hiding) vault application. However, there was no database file found to be associated with this application. And possible hints about any chat communications were not present in the plist file.

VAULT IDENTIFICATION, DETECTION AND EXTRACTION REPORT

Details about the Device:
Device Name: XXXXXX iPhone
Phone Number:+1 (XXX) XXX-XXXX
IMEI Number: 35XXXXXXXXX1202
Product Version: 12.3.1
Logical Acquisition Status: Finished
Logical Acquisition Date Time: 2019-05-10 17:04:46.278286

Total numbers of Apps Found on the Device: 70
Numbers of Vault Apps Found on the Device: 5
List of Vault Apps Detected:
1. lockmyfolder
2. com.appostrophe.privatecalc
3. my.com.pragmatic.My-Apps-Lite
4. org.whispersystems.signal
5. com.GalaxyStudio.PasswordSafeFree

Files and Artifacts Extracted:
1. **App Bundle ID: lockmyfolder**
PLIST FILES:
Library/Preferences/lockmyfolder.plist
Library/Application Support/com.crashlytics/CLSUserDefaults.plist
IMAGE FILES:
Documents/PHOTOS/A6A5B438-3028-4B49-9E8F-B467356F5DB1/IMG1.jpg
Documents/PHOTOS/8632C492-5014-434B-9F83-4DB9661CB2C5/IMG3.jpg
Documents/PHOTOS/8632C492-5014-434B-9F83-4DB9661CB2C5/IMG2.jpg
Documents/PHOTOS/8632C492-5014-434B-9F83-4DB9661CB2C5/IMG1.jpg
DATABASE FILES:
Documents/LockMyFolder.sqlite
Any media files found? Yes
2. **App Bundle ID: com.appostrophe.privatecalc**
PLIST FILES:
Library/Preferences/com.appostrophe.privatecalc.plist

Fig. 11. A Sample Overview of VIDE Report.

6.1. Limitations

We did not explore decrypting data for vault apps that encrypt the stored data. Although we can retrieve any encrypted data, VIDE cannot automatically decrypt it. It is possible that the passcode used to derive the encryption key could be in the raw data and possibly used to decrypt. We also did not explore vault apps that store data in the cloud. Additionally, VIDE assumes that the phone is unlocked.

7. Conclusions and future work

Current state-of-the-art data extraction tools typically have the ability to extract everything from a smartphone but often require require the investigator to further sort and filter the data by date/time/etc. Tools that can more precisely support the ability to extract various types of selective information from targeted sets of applications can be useful and significantly reduce the analysis time for investigators. In this paper we discussed VIDE which can automatically identify and extract data from the class of vault applications. These apps are particularly problematic as a great deal of addition information or exploration of the app would typically be required even to identify such an application.

In future work, we would like to explore issues related to reasons for the frequent removal of vault apps as it may relate to the iOS vetting process. The prevalence of possible adversarial hiding of vault apps on the App Store is also intriguing. We would also like to explore various issues related to live analysis, including getting data from the cloud.

References

Apple, 2019a. App store improvements [online]. https://developer.apple.com/

support/app-store-improvements/.
Apple, 2019b. Categories and Discoverability - App Store - Apple Developer. https://developer.apple.com/app-store/categories/.
Apple, 2019c. Change your Apple ID country or region [Online]. https://support.apple.com/en-us/HT201389.
Apple, 2019d. More malicious apps found in mac app store that are stealing user data [online]. https://appleinsider.com/articles/18/09/07/more-malicious-apps-found-in-mac-app-store-that-are-stealing-user-data.
AppleBetas, 2018. MTerminal [Online]. https://github.com/AppleBetas/MTerminal-Jailed/.
Azadegan, S., Yu, W., Liu, H., Sistani, M., Acharya, S., 2012. Novel anti-forensics approaches for smart phones. In: 2012 45th Hawaii International Conference on System Sciences. IEEE, pp. 5424–5431.
Bader, M., Baggili, I., 2010. iPhone 3GS forensics: logical analysis using apple iTunes backup utility. In: Small scale digital device forensics journal, 4, pp. 1–15, 1.
Cellebrite, 2018. Cellebrite. http://www.cellebrite.com. (Accessed 1 January 2019).
DataVizInc, 2017. Passwords Plus - Free Secure Vault on the App Store (Version 4.001). https://itunes.apple.com/us/app/passwords-plus-free-secure-vault/id486941825?mt=8. (Accessed 21 May 2019).
Davey Winder, CyberSecurity, F., 2019. Apple app store security bypassed by government ios surveillance malware [online]. https://www.forbes.com/sites/daveywinder/2019/04/09/apple-app-store-security-bypassed-by-government-ios-surveillance-malware-what-you-need-to-know/#3132b6b91b82.
De Raedt, L., Passerini, A., Teso, S., 2018. Learning constraints from examples. In: Thirty-Second AAAI Conference on Artificial Intelligence.
Distefano, A., Me, G., Pace, F., 2010. Android anti-forensics through a local paradigm. In: Digital Investigation. Proceedings of the 10th Annual DFRWS Conference, vol. 7. Elsevier, pp. S83–S94.
Duncan, M., Karabiyik, U., 2018. Detection and recovery of anti-forensic (vault) applications on android devices. In: Proceedings of the Annual ADFSL Conference on Digital Forensics, vol. 6. Security and Law.
Freeman, J., 2018a. Cydia [online]. https://cydia.saurik.com.
Freeman, J., 2018b. Impactor [online]. http://www.cydiaimpactor.com.
Gorla, A., Tavecchia, I., Gross, F., Zeller, A., 2014. Checking app behavior against app descriptions. In: Proceedings of the 36th International Conference on Software Engineering. ACM, pp. 1025–1035.
GummyBearStudios, 2018. My Secret Notes on the App Store, version 2.0.2. https://itunes.apple.com/us/app/my-secret-notes/id437215704?mt=8. (Accessed 21 May 2019).
Harris, R., 2006. Arriving at an anti-forensics consensus: examining how to define and control the anti-forensics problem. Digit. Invest. 3, 44–49.
Hoog, A., Strzempka, K., 2011. iPhone and iOS Forensics: Investigation, Analysis and Mobile Security for Apple iPhone, iPad and iOS Devices. Elsevier.
IdeaSolutions, 2018. Secret Photos - Kyms on the App Store, version 3.4.7. https://itunes.apple.com/us/app/secret-photos-kyms/id471303390?mt=8. (Accessed 21 May 2019).
Jailbreak11, 2018. ZJailbreak [Online]. http://zjailbreak.com.
Kanzelsberger, 2014. Mywallet Lite - Secure Password Manager, version 1.5.6. https://itunes.apple.com/us/app/mywallet-lite-secure-password-manager/id423959784?mt=8. (Accessed 21 May 2019).
Kolb, S., Paramonov, S., Guns, T., De Raedt, L., 2017. Learning constraints in spreadsheets and tabular data. Mach. Learn. 106 (9-10), 1441–1468.
Morrissey, S., Campbell, T., 2011. iOS Forensic Analysis: for iPhone, iPad, and iPod Touch (Apress).
Murphy, C., 2016. 'What You're Missing in Hidden Apps (And 5 Data Forensic Tools!)'. https://www.officer.com/command-hq/technology/computers-software/data-forensics/article/12203542/hidden-mobile-apps-mobile-investigations-forensic-data-tools.
Oxygen, 2018. Oxygen. https://www.oxygen-forensic.com/en/. (Accessed 3 November 2018).
Pandita, R., Xiao, X., Yang, W., Enck, W., Xie, T., 2013. WHYPER: towards automating risk assessment of mobile applications. In: Presented as Part of the 22nd USENIX Security Symposium (USENIX Security 13), pp. 527–542.
Paraben, 2018. Paraben. https://shop.paraben.com. (Accessed 6 December 2018).
PenzuInc, 2018. Penzu on the App Store, version 3.5. https://itunes.apple.com/us/app/penzu-free-diary-private-journal/id452674732?mt=8. (Accessed 21 May 2019).
Piccinelli, M., Gubian, P., 2011. Exploring the iphone backup made by itunes. J. Digit. Forensics Secur. Law 6 (3), 4.
Qu, Z., Rastogi, V., Zhang, X., Chen, Y., Zhu, T., Chen, Z., 2014. Autocog: measuring the description-to-permission fidelity in android applications. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. ACM, pp. 1354–1365.
RedKnight, 2018. My Secret Folder on the App Store, version 2.0.3. https://itunes.apple.com/us/app/my-secret-folder/id489803044?mt=8. (Accessed 21 May 2019).
Rughani, D.P., 2017. Forensic analysis of content hiding android applications. Int. J. Adv. Res. Comput. Sci. Software Eng. 7, 404–408.
SavySoda, 2017. Private Browsing White (Version 11.1). https://itunes.apple.com/us/app/private-browsing-white/id428855226?mt=8. (Accessed 18 January 2020).
Tan, P.-N., 2018. Introduction to Data Mining. Pearson Education.
Tso, Y.-C., Wang, S.-J., Huang, C.-T., Wang, W.-J., 2012. Iphone social networking for evidence investigations using itunes forensics. In: Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication. ACM, p. 62.
Watanabe, T., Akiyama, M., Sakai, T., Washizaki, H., Mori, T., 2018. Understanding the inconsistency between behaviors and descriptions of mobile apps. IEICE Trans. Info Syst. 101 (11), 2584–2599.
Zhang, X., Baggili, I., Breitinger, F., 2017. Breaking into the vault: privacy, security and forensic analysis of android vault applications. Comput. Secur. 70, 516–531.
Zhou, J., 2016. Automated app categorization using API analysis [Online]. http://www0.cs.ucl.ac.uk/staff/Y.Jia/resources/studentprojects/Kelly_Zhou_Automated_App_Categorization_usingAPI_analysis.pdf.