future internet

Article

Using Machine Learning to Detect Vault (Anti-Forensic) Apps

Michael N. Johnstone [1,*], Wencheng Yang [2] and Mohiuddin Ahmed [1]

[1] School of Science, Edith Cowan University, Perth, WA 6027, Australia; mohiuddin.ahmed@ecu.edu.au
[2] School of Mathematics, Physics and Computing, University of Southern Queensland, Toowoomba, QLD 4350, Australia; wencheng.yang@unisq.edu.au
* Correspondence: m.johnstone@ecu.edu.au

Abstract: Content hiding, or vault applications (apps), are designed with a secondary, often concealed purpose, such as encrypting and storing files. While these apps may serve legitimate functions, they unequivocally present significant challenges for law enforcement. Conventional methods for tackling this issue, whether static or dynamic, prove inadequate when devices—typically smartphones—cannot be modified. Additionally, these methods frequently require prior knowledge of which apps are classified as vault apps. This research decisively demonstrates that a non-invasive method of app analysis, combined with machine learning, can effectively identify vault apps. Our findings reveal that it is entirely possible to detect an Android vault app with 98% accuracy using a random forest classifier. This clearly indicates that our approach can be instrumental for law enforcement in their efforts to address this critical issue.

Keywords: software development; vault apps; content hiding; malware detection; machine learning; Android

Academic Editor: Gianluigi Ferrari

Received: 6 February 2025
Revised: 11 April 2025
Accepted: 16 April 2025
Published: 22 April 2025

Citation: Johnstone, M.N.; Yang, W.; Ahmed, M. Using Machine Learning to Detect Vault (Anti-Forensic) Apps. *Future Internet* **2025**, *17*, 186. https://doi.org/10.3390/fi17050186

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Smartphones have become an integral part of daily life, enabling users to communicate, store sensitive information, and perform a wide range of digital activities. As a result, they are also frequently involved in digital investigations, making smartphone forensics a critical field in digital forensic research. The rapid advancements in smartphone technologies, including encryption mechanisms, cloud synchronization, and secure storage, present both opportunities and challenges in forensic investigations [1].

A key challenge in smartphone forensics is the increasing use of applications designed to protect user data and privacy. One such category of applications is vault apps, which allow users to hide files, messages, and even entire applications behind additional layers of encryption [2]. Vault apps are mobile apps that provide extra features that extend beyond their primary functions. Among these, some mobile apps provide extra features that extend beyond their primary functions. For instance, an app that is mainly designed as a calculator might also include advanced features, such as the capability to encrypt files or securely store sensitive data like photographs. These specialized, content-hiding applications are often referred to as "vault" apps. Vault apps are becoming increasingly popular with the growing concern for digital privacy. These apps provide users with secure storage by encrypting files and hiding their presence in the smartphone system [3]. While their main functions are often legitimate, such as securing personal photos, files, or sensitive information, they also pose significant challenges to law enforcement and digital forensic investigations [2].

However, the challenge lies in the fact that these vault apps are not always easily identified by existing detection systems, which can lead to confusion with regular, non-obfuscated applications. This lack of clarity can be problematic because vault apps can carry significant risks. They may facilitate espionage, enable unauthorized spying, and support a wide range of malicious activities that compromise user privacy and security. Recognizing and understanding the potential dangers posed by these apps is essential for maintaining digital safety.

Conventional forensic techniques depend on either static or dynamic analysis methods. Static analysis includes examining an application's code or metadata, while dynamic analysis observes its behavior during execution. Nevertheless, these approaches may not work when the forensic tool lacks root privileges or the vault application utilizes complex obfuscation techniques [4]. Machine learning (ML) has emerged as a prospective approach to address these sorts of challenges. By exploiting classification algorithms, ML models can differentiate vault applications from regular applications by behavioral and structural features [5].

Duncan and Karabiyik [6] introduced an automated method for detecting vault applications on Android devices using a Python-based program. This program automates the identification process by systematically comparing the package names of applications installed on a device, such as a mobile phone, against a curated list of approximately 200 known vault apps. One advantage of this method is its reliance on the internal package names of the applications, which remain constant regardless of any changes made to the app's display name. However, this approach does have a limitation, namely, if a vault app is not included in the established list, it will not be identified as such.

In this paper, we adopt a similar methodology, choosing to conduct a thorough static analysis of APK files. Through the process of reverse-engineering vault applications, we aim to uncover distinctive features that set them apart. A particular focus of our analysis is the Android permissions utilized by these applications, which can provide valuable insights into their functionality and intentions. By examining these permissions, we hope to enhance our understanding of how vault apps operate and improve detection methodologies.

We envisage that other approaches, such as path analysis (static or dynamic), while they have merit, can be problematic. See Figure 1 for an example call graph of a simple program. The density is due to the libraries bound to the app; so much of the call graph shows library calls made during app instantiation, making control flow analysis of the actual app challenging. Our contribution is that we extend the idea of Duncan and Karabiyik by using a machine learning algorithm to classify apps without the need for a fixed list or database that must be constantly updated.

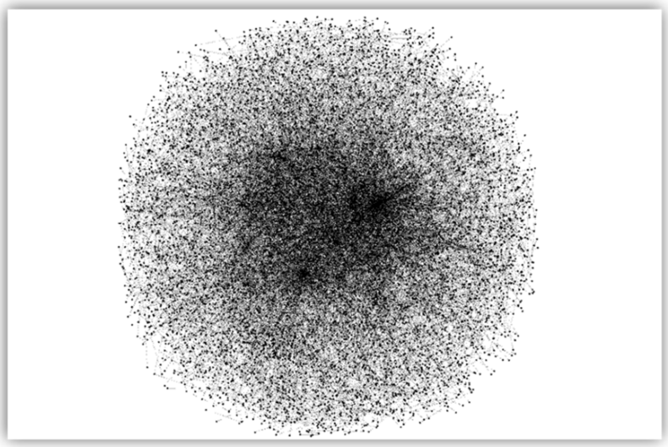


Figure 1. Example call graph of a simple Android app.

Paper Organization

The rest of this paper is organized as follows: Section 2 presents the related work followed; the method used in this paper is presented in Section 3; Section 4 reports the experimental results; and the paper is concluded in Section 5.

2. Related Work

Just because an app has a secondary (possibly hidden) function does not automatically make it malware. Are Vault apps malware? Mikalauskas [7] highlights security concerns regarding some vault apps, as reported in an analysis of 30 popular Android vault apps downloaded from the Play Store. The investigation found that while not all vault apps are inherently malicious, some have been flagged for distributing malware, engaging in spyware activities, or collecting sensitive user data without consent. Specifically, in the investigation, the permission data of the apps was examined, and it was found that some apps were malware in the conventional sense. Such apps had no real purpose other than to act as a vector for malware. Other apps requested dangerous permissions which were not necessary for the app to function. Further, some apps requested unnecessary permissions, such as recording audio or obtaining the GPS location of a phone. Interestingly, one app accessed the phone camera without asking for permission; thus, it could take a photo without user interaction.

While some vault apps have been identified as malware, not all fall into this category, it is important to assess where malicious vault apps fit within malware classifications. Malware can be categorized based on detection methods (e.g., static or dynamic analysis) or by type (e.g., adware, trojans, or spyware). However, not all vault apps are inherently harmful. For instance, Keepsafe Photo Vault, a widely used and well-reviewed app, provides genuine encryption-based protection for private photos and videos without engaging in malicious activities. Early work by Grégio et al. [8] classified malware by family using a dynamic approach and a dataset of 12,579 samples. First, their taxonomy uses behaviors (e.g., IRC port connection, information theft), so the behaviors are not directly connected to specific malware (which makes their taxonomy generalizable). Second, some behaviors were exhibited by samples that were determined to be clean (could not be determined to be malware by three different malware detection engines) and those that that were undecided (no consensus across the three malware detection engines). We will return to the problem of behaviors that occur in both benign and malware samples later in our results section.

Soliman et al. [9] developed a taxonomy of malware analysis specifically for IoT devices. They use the conventional static/dynamic mode split and add a hybrid mode. Static analysis, by its very name, analyzes the source code or a binary without executing it, while dynamic analysis involves running a program in a controlled environment, which may include techniques such as emulation, virtualization, or sandboxing to observe its behavior. Soliman et al. contrast the factors limiting static (e.g., fingerprinting, string extraction) and dynamic (e.g., sandboxing, function call monitoring) approaches, but it is unclear how these approaches are specific to IoT devices. Sandboxing or analyzing Wireshark traces are not techniques specific to IoT devices.

Ngo et al. [10] surveyed methods specific to the static analysis of IoT malware. They separated static analysis methods into graph-based and non-graph-based methods. Function call graphs (see Figure 1) are a subset of the former. String analysis (which is the closest to our approach) is a subset of the latter. Curiously, they considered string and opcode analysis to be high-level analysis, while ELF header analysis was considered to be low-level analysis. We would consider file header analysis to be high-level and opcode analysis to be low-level. In their analysis of various static approaches, Ngo et al. mentioned that (embedded) strings were used to generate signatures to classify IoT malware, which might account for its placement in their taxonomy. Our approach is different.

Singh and Singh [11] used various classifiers to detect malware via printable strings across 24,911 binary files. They note that simply counting strings in a binary file does not provide sufficient information about the behavior of the malware. They do claim, however, that malware and benign files have different printable statements (e.g., encryption, decryption) such that the strings in binary files provide enough discriminative information to classify malware. In our experiments, we found that not to be the case as some information was commonly encoded across both vault and non-vault apps. What was interesting was their use of Shannon entropy over the string features to classify files as benign or malware.

Abusitta et al. [12] developed a taxonomy for malware classification and surveyed recent developments. In their taxonomy, depending on whether we examine features or algorithms, our approach would be classified as feature extraction/static or feature type/printable strings or artificial intelligence/supervised learning.

Ruffin et al. [13] conducted a security analysis of 20 popular Android vault apps, focusing on scenarios such as unfair searches during civil unrest and intimate partner violence (IPV). Their study evaluated these apps at three adversary levels, namely, novice, intermediate, and advanced, revealing significant weaknesses in their camouflage and file protection mechanisms. The results of the study show that most of the applications fail to implement strong camouflage and are therefore easily recognizable by simple observation. In addition, many apps store files insecurely using methods such as hiding folders or changing file extensions, vulnerabilities that can be exploited by adversaries with basic Android knowledge. Only a few apps employ encryption to secure user data. Nash and Choo [14] reviewed the security and forensic challenges of password managers and vault applications, emphasizing vulnerabilities in encryption, autofill, and local storage. They also pointed out forensic risks, such as the ease of retrieving sensitive data from memory, and discussed user adoption barriers due to security issues. Their study proposes novel design approaches, such as honey encryption and graphical password hints, to improve security and usability, and they call for further research to address long-standing issues.

Zhang et al. [3] noted that, similar to developers of conventional apps, vault application developers use code obfuscation to protect their intellectual property. They used reverse engineering to acquire evidence hidden by a selection of vault apps. Hutchinson and Karabiyik [15] found that Google's Play Protect service could detect a malware app, but not immediately. Wang et al. [16] used an XGBoost classifier to detect Android malware. They achieved an accuracy of 0.880 using 1000 samples each of malware and benign apps. They also found that 368 features provided the optimal accuracy. Similarly, Mahindru and Singh [17] used a dynamic analysis approach to evaluate the permissions of 11,000 malware and benign Android apps. They achieved an F-measure (presumably F_1, but not stated) of 0.997 for a random forest classifier. They mention that in some cases, the malware and benign apps require the same permissions, making it difficult to differentiate between them—a problem we consider in the next section. Xie et al. [5] used an SVM classifier to detect Android vault apps. They achieved an accuracy of 0.933 using 102 samples across vault and non-vault apps. Table 1 shows a summary of related work. It is important to note that the majority of the work focused on using supervised machine learning models, and it would be interesting to explore other models.

Table 1. Summary of related work.

Source	Technique	Sample/Dataset	Machine Learning	Model Quality
[2]	Deobfuscation via reverse engineering	Vault apps	No	Not Applicable
[3]	Google's Play Protect service	Android malware	No	Not Applicable
Wang [16]	XGBoost classifier	Android malware	Yes	Supervised
[5]	Random Forest	Android malware	Yes	Supervised
Dorai [2]	SVM	iOS malware	Yes	Supervised
Xie [5]	SVM	Vault apps	Yes	Supervised
Singh [11]	Random Forest (and others)	Generic malware	Yes	Supervised

In the companion area of iOS applications, Dorai et al. [2] used an SVM classifier to detect Apple vault apps. They achieved an F_1 score of 0.900 using 2963 samples across vault and non-vault apps. Peng et al. [18] proposed a deep learning-based system called DECADE for detecting and analyzing content-hiding apps (vault apps) on Android devices. Their study developed an automatic recognition framework using advanced machine learning techniques in combination with text and image features extracted from app descriptions and screenshots. DECADE successfully classifies apps into hidden and non-hidden categories, achieving high accuracy rates and providing a mechanism with which to extract hidden user data for investigative purposes. The system also highlights the challenges in detecting cryptographic and anti-forensic vault applications, providing insights into the broader picture of content-hiding applications in both legal and illegal markets.

To summarize, whilst machine learning is data-driven (therefore, the more data points, the better the results), good results can be obtained with relatively small datasets. As noted, Wang et al. [16] reported an accuracy of 0.880 using 1000 samples each of malware and benign apps. Interestingly, and closer to our problem space, Xie et al. [5] stated an accuracy of 0.933 using only 102 samples across vault and non-vault apps. In contrast, Mahindru and Singh [5] achieved an improved F-measure of 0.997 at a cost of a several-magnitude increase in dataset size (11,000 malware and benign Android apps).

Qiu et al. [19] noted several challenges and open issues in the area of Android malware detection. They highlighted, in particular, the need for (1) accurate ground truth and correct label annotations and (2) effective feature engineering via meaningful feature extraction from apps. We hope that this work contributes to accomplishing both aims, at least in the malware subdomain of vault apps.

3. Method

An Android application contains a manifest file (in XML format) which presents essential information about an application to the Android system, including permissions. We used tools within the Androguard 4.1.2 (a Linux-based tool specifically designed for reverse engineering APK files) package, specifically "androaxml.py", to extract the Android manifest XML file of a given application and "androcg.py" to extract the call graph of the same applications.

The apps were label manually. We started with a list of vault applications as classified by Duncan and Karabyik. Further APK files were sourced from websites that claimed to hold vault apps (www.apkpure.com, www.apkmonk.com, www.apk4now.com, accessed on 13 August 2024) and the Play Store. The vault apps were exercised manually either in an emulator (Android Studio) or on an Android phone to test whether they contained the claimed vault functionality. If so, they were label as vault apps. If not, they were discarded.

Non-vault apps were sourced from the Play Store. We assumed that well-known apps (e.g., Facebook) did not contain any vault (hidden) functionality.

As noted by Xie et al. [5], permissions are often used as features for detection in Android apps. To attempt to identify potential vault activities, various tags were extracted, namely: Uses-permission, Activities, Services, Metadata, Action, Receiver, and Provider. Each of these tags, particularly those related to permissions, are useful for detection purposes.

Uses-permission: This specifies a permission that a user must grant for correct app operation. This can be either when the app is installed or while the app runs, depending on the Android version [20]. The permissions often relate to devices or files that could be accessed by a vault app, such as the camera or the contacts file. The format is "android.permission.<DEVICE_NAME | FILE_NAME>", an example being "android.permission. CAMERA".

Activities: This declares an activity (actually, an Activity subclass) that implements part of an app's user interface [20]. An activity can be launched as the embedded child of another activity, which has implications for vault apps. As noted by Zhang et al. [3], vault apps often employ decoy screens designed to hide activities.

Services: Service subclasses run background tasks, which can potentially be executed before a user unlocks a device, as one of the application's components. Unlike activities, services do not need a user interface [20], which provides a certain utility for vault apps.

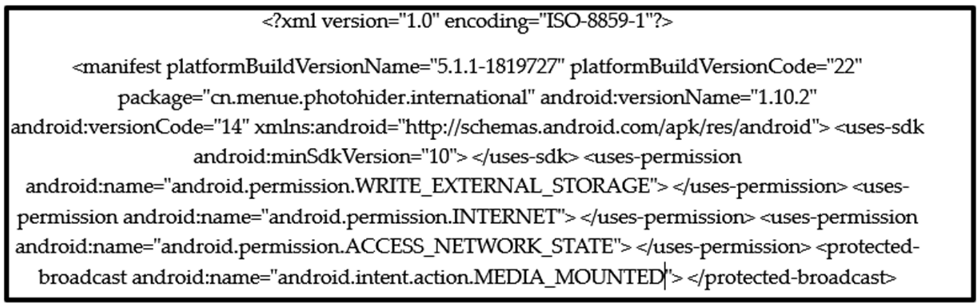
Meta-data: This tag refers to a configurable name–value pair for any item of additional data that can be passed to a parent component (such as an activity or service) [20]. A vault app could use metadata to reference elements of custom data structures (although such references can also be used by non-vault apps to perform legitimate functions).

Action: This adds one or more actions to an intent filter. An intent is a messaging object used to request an action from another app component [20]. Although actions are a common tag, it might be possible to detect vault activities characterized by the declaration of custom messages for actions.

Receiver: This declares a broadcast receiver as an app component. Broadcast receivers enable applications to receive intents (see Action above) that are broadcast by the system or by other applications, even when other components of the app are not running [20] (which could assist in hiding vault activity). Similar to Services, a receiver can run before a user unlocks a device.

Provider: A content provider supplies structured access to data managed by an app [20]. It can contain an intent filter (see Action above). As with receivers and services, a provider can run before a user unlocks a device. Further, a provider could be made available to other applications (although again, such functionality can also be used by legitimate apps).

We obtained 521 vault and 519 non-vault apps from various web sites and the Google Play Store. The list of application names and dataset is available on request. The vault apps were tested for the presence of malware using Virus Total. If clean, the vault apps were exercised manually to ensure that they contained the claimed vault-like functionality. The SHA256 hash of each app was computed and stored. Each app was processed with Androguard to extract the app properties in XML format (see Figure 2). The hashes were then re-computed to check that the tool did not modify the apps in any way. Each XML file containing the properties was then post-processed to extract the properties into a CSV file. Long property names were shortened for convenience in the CSV output. The short names were derived by selecting the first letter of the property name and concatenating every first letter of every subsequent word, separated by full stop (".") characters (see Table 2 for examples).



```
<?xml version="1.0" encoding="ISO-8859-1"?>

<manifest platformBuildVersionName="5.1.1-1819727" platformBuildVersionCode="22"
package="cn.menue.photohider.international" android:versionName="1.10.2"
android:versionCode="14" xmlns:android="http://schemas.android.com/apk/res/android"> <uses-sdk
android:minSdkVersion="10"> </uses-sdk> <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"> </uses-permission> <uses-
permission android:name="android.permission.INTERNET"> </uses-permission> <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"> </uses-permission> <protected-
broadcast android:name="android.intent.action.MEDIA_MOUNTED"> </protected-broadcast>
```

Figure 2. Example manifest XML file generated by Androguard.

Table 2. Example properties and their abbreviations.

Property	Abbreviation
android.permission.CAMERA	apC
android.permission.RECORD_AUDIO	apR
android.permission.ACCESS_NETWORK_STATE	apA
it.airgap.vault.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSION	iavD
com.android.vending.BILLING	cavB
android.permission.WRITE_EXTERNAL_STORAGE	apW
android.intent.action.MEDIA_MOUNTED	aiaM
android.permission.INTERNET	apI

The process workflow of the proposed method is demonstrated in Figure 3.

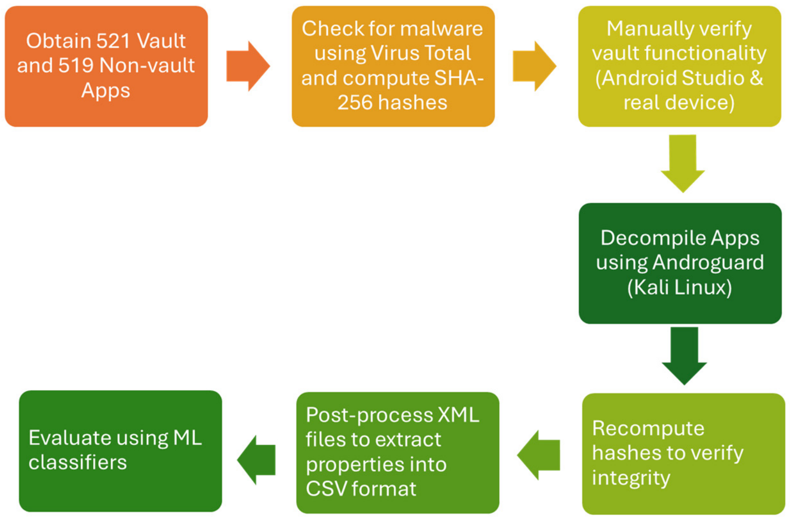

Obtain 521 Vault and 519 Non-vault Apps
Check for malware using Virus Total and compute SHA-256 hashes
Manually verify vault functionality (Android Studio & real device)
Decompile Apps using Androguard (Kali Linux)
Recompute hashes to verify integrity
Post-process XML files to extract properties into CSV format
Evaluate using ML classifiers

Figure 3. Process workflow of the proposed method.

This CSV file containing the matched properties for the 1040 apps was then used as input to several machine learning algorithms. As there were over 500 unique properties detected, resulting in a large sparse matrix, a chi-square test was used to determine the properties that provided the highest contribution to the detection of vault apps. Feature selection was one of the key steps undertaken to increase the model's predictive capability. The SelectKBest method was used, making use of the chi-square test to select the features that contributed most. Further, principal component analysis (PCA) was used to further focus the detection capability.

PCA, invented by Pearson [21], is a technique designed to reduce high-dimension datasets to make them more manageable and provides an intuitive graphic representation of the relationships between the variables. Jolliffe [22] (p1) describes the purpose of PCA succinctly as "to reduce the dimensionality of a dataset consisting of a large number of interrelated variables, while retaining as much as possible of the variation present in the dataset. while retaining as much as possible of the variation present in the dataset by transforming to a new set of variables, the principal components (PCs), which are uncorrelated, and which are ordered so that the first few retain most of the variation present in all of the original variables.".

We used the sklearn package for the classifiers, with the default train/test split of 75%/25%. Conventional metrics, viz., accuracy, F_1 score, and Matthews correlation coefficient (MCC), were used to evaluate the classifiers. Default parameters were used for the PCA.

To evaluate the effectiveness of the proposed method, we used the TP, TN, FP, and FN standard values to calculate the metrics, these are as follows:

True positive (TP): indicates that a vault app was correctly classified and labeled as a vault app.

True negative (TN): indicates that a non-vault app was correctly classified and labeled as a non-vault app.

False positive (FP): indicates that a non-vault app was incorrectly classified and labeled as a vault app.

False negative (FN): indicates a vault app sample was incorrectly classified and labeled as a non-vault app.

Based on the above, we define the metrics as follows.

$$\text{Accuracy} = (\text{TP} + \text{TN})/(\text{TP} + \text{FP} + \text{TN} + \text{FN})$$

$$F_1 \text{ score} = \text{TP}/(\text{TP} + \tfrac{1}{2}\,(\text{FP} + \text{FN}))$$

$$\text{MCC} = (\text{TP} \times \text{TN - FP} \times \text{FN})/\text{sqrt}((\text{TP} + \text{FP}) \times (\text{TP} + \text{FN}) \times (\text{TN} + \text{FP}) \times (\text{TN} + \text{FN}))$$

4. Results

As previously mentioned, across the 1040 apps, there were over 500 unique properties detected, resulting in a large, sparse matrix of apps and their properties. Many of these properties were redundant as they appeared in both vault and non-vault apps, rendering them useless for detection purposes. Further, Android apps can be complex (cf. Figure 1), and the manifest files contain a plethora of properties. Not every property is, however, directly related to vault behavior. Just because an app uses a WRITE_EXTERNAL_STORAGE property (see Figure 2) does not mean that it can be identified immediately as a vault app or is engaged in malware-like behavior. For example, an app could legitimately write a settings file to maintain state across app invocations. As mentioned previously, we focus on seven specific property tags which are likely to be useful for forensic analysis, viz., Uses-permission, Activities, Services, Metadata, Action, Receiver, and Provider. These properties help identify the behavioral patterns and functionalities of applications, particularly in distinguishing between vault (anti-forensic) apps and non-vault apps. Each property plays a crucial role in forensic analysis by revealing how an application interacts with system resources and user data. To illustrate the forensic significance of these properties, we provide an example focusing on the "uses-permission" property, which governs an app's access to sensitive resources as shown below:

- Vault App: A vault application may request READ_EXTERNAL_STORAGE and WRITE_EXTERNAL_STORAGE permissions to access and conceal user files. These permissions allow the app to hide sensitive data from standard file managers or gallery applications, making it difficult to detect stored content.
- Non-Vault App: A gallery app might also request these permissions to display and manage user images, but without any concealment. The requested permissions, in this case, are intended for legitimate media management rather than data obfuscation.

Similar analyses can be applied to other properties—such as Activities, Services, Metadata, Action, Receiver, and Provider—to further differentiate between vault and non-vault applications.

To address the "curse of dimensionality", we use a chi-square test to determine the properties that provided the highest contribution to the detection of vault apps. We also apply PCA to further improve the detection capability.

Initial experiments (Figure 4) with a random forest classifier, using a subset of the data and without fine-tuning (i.e., without the chi-square selection and PCA dimensionality reduction), were far from perfect but encouraging. Approximately 1/3 of vault apps and 1/3 of non-vault apps were misclassified.

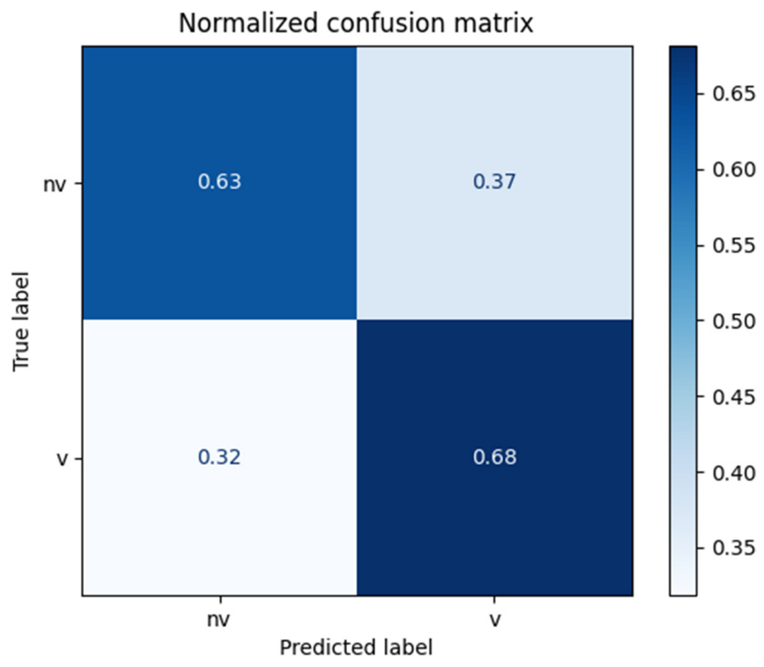


Figure 4. Initial confusion matrix for RF classifier.

Figure 5 shows the confusion matrix for the random forest classifier, using chi-square and PCA. Note that no non-vault apps are misclassified as vault apps, and very few vault apps are misclassified as non-vault apps.

Principal component analysis was employed to reduce the dimensionality of the feature space, as shown in Figure 6. The data points are projected across one or two principal components to test the variance between classes (vault and non-vault apps). The PCA visualization depicts how the data is distributed in two-dimensional space. On the plot, the separation into clusters of points shows that feature selection and dimensionality reduction effectively captures the difference between vault and non-vault applications. The clusters clearly distinguish between the classes, which suggests that our model has high discriminative power.

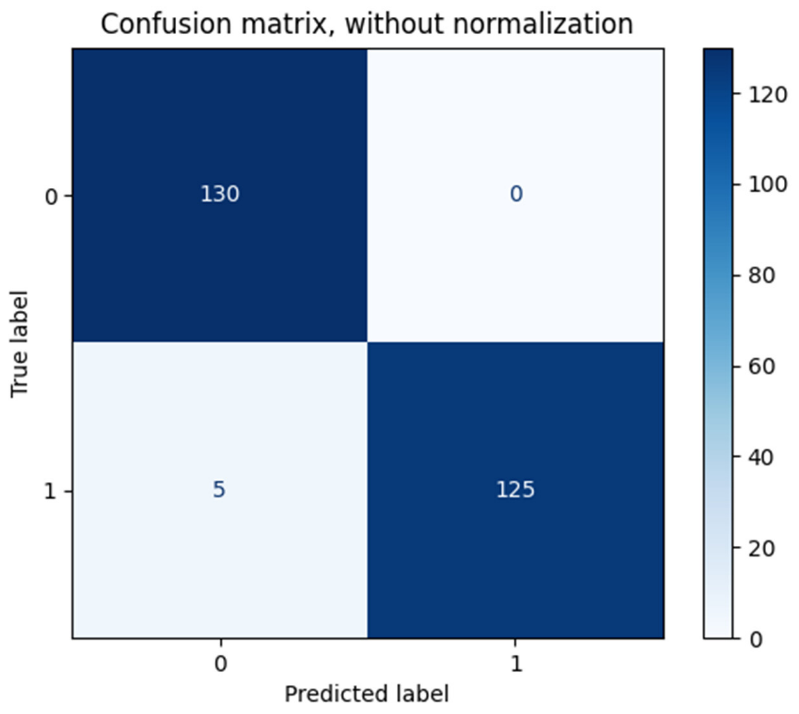


Figure 5. Confusion matrix for RF classifier.

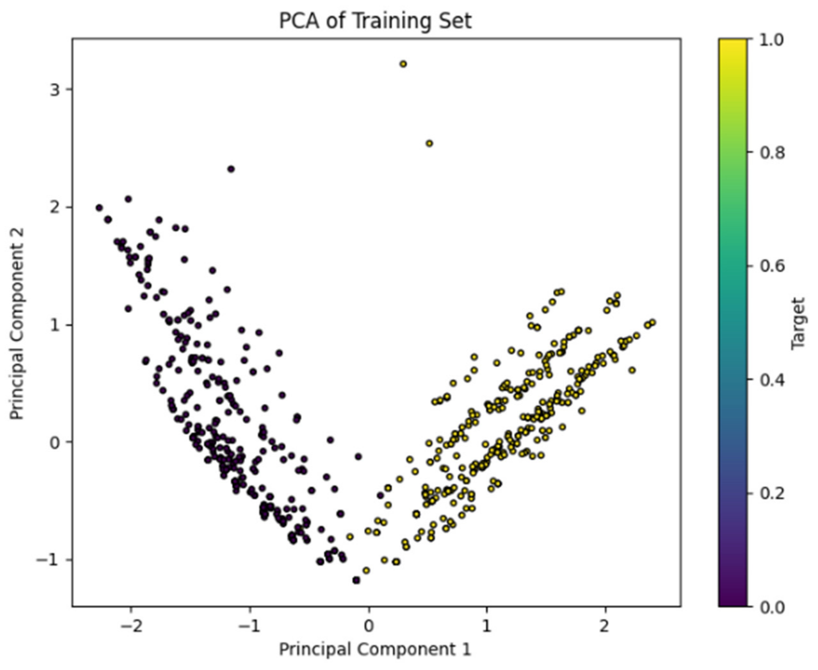


Figure 6. PCA plot for RF classifier.

The loading plot identifies the properties that have the largest effect on each principal component. The coefficient range is ±1. Large coefficients indicate that a property strongly influences a component. Coefficients close to 0 indicate that a property has little influence on the component. The coefficients for the apR.1 property are 0.171 and 0.406 for the PC1 and PC2 components, respectively, indicating that this property influences PC2 relatively strongly (and PC1 much less so). Conversely, the coefficients for the apW property are 0.336 and 0.156 for PC1 and PC2, respectively, indicating that this property influences PC1 relatively strongly compared to PC2. Further, the angle between the properties indicates their influence on one another; 90 degrees (orthogonal) would mean that the properties are independent. In Figure 7, the apR and apW properties are orthogonal, meaning that they have no influence on one another (which is desirable).

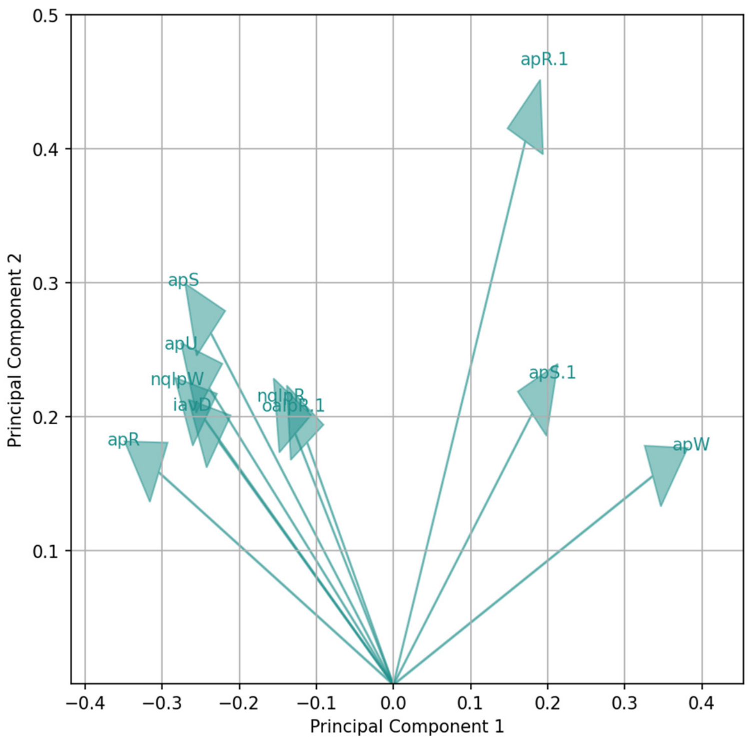


Figure 7. PCA Loadings of top ten properties.

Figure 7 projects the loadings of the top ten properties to examine their relationships. As previously mentioned, the angles between the vectors illustrate how the components correlate with one another and contribute to the respective principal components.

Figure 8 shows that the first ten properties explain 75% of the variance in the model. Using 30 properties explains almost all of the variance, which suggests that using more properties (out of the ~500 available) would not improve the model. Figure 9 provides a final check of the model by illustrating the RoC curve of one of the ML classifiers.

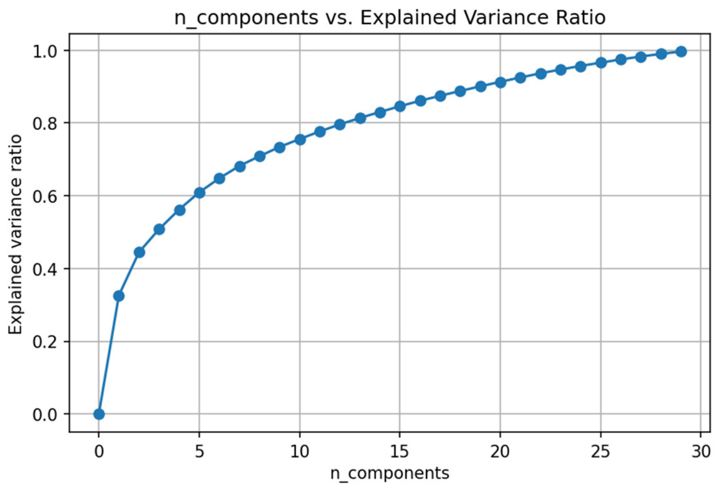


Figure 8. Ranked model contribution of components.

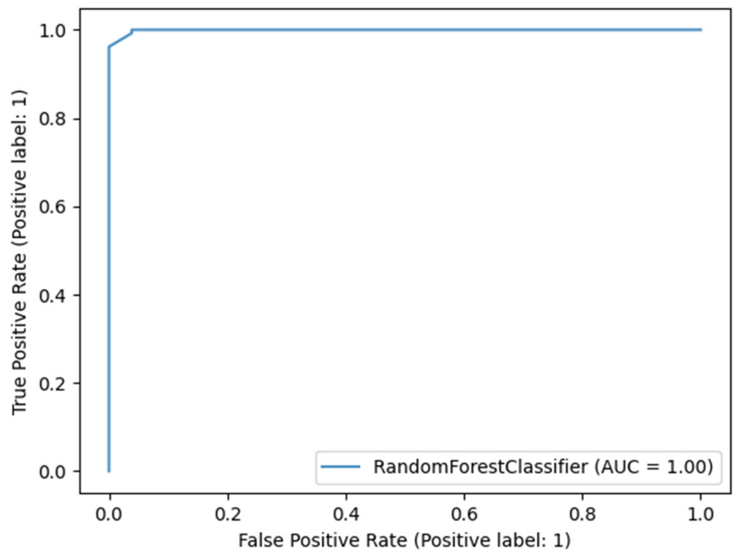


Figure 9. RoC Curve for RF classifier.

Table 3 describes our evaluation of six machine learning classifiers across our dataset. We note that the SVM, MLP, and random forest algorithms perform best, which is consistent with other work in the domain.

Table 3. Comparison of various classifiers.

Classifier	Accuracy	F_1 Score	MCC
MultinomialNB	0.9385	0.9382	0.8836
GaussianNB	0.9077	0.9070	0.8273
KNeighbors	0.9346	0.9345	0.8736
SVM	0.9808	0.9808	0.9623
MLP	0.9808	0.9808	0.9623
RandomForest	0.9808	0.9808	0.9623

We used the default hyperparameters for the classifiers in Table 3, except for KNeighbors, where we used k = 3; SVM, where we used kernel = "linear" and C = 1.0; MLP, where we set alpha = 1 and max_iter = 1000; and RandomForest, where we set max_depth = 5, n_estimators = 10, and max_features = 1.

With such a relatively small dataset, there is a danger of overfitting. To test this, we used k-fold cross-validation with k = 10 and accuracy as the metric. We found our model to be relatively stable, as shown in Figure 10, with a mean of 0.9615 and a standard deviation of 0.0243 (which is in agreement with Table 3).

Table 4 compares our approach against existing work. A direct comparison is challenging due to differing metrics reported (Accuracy, F_1 score), sample sizes (102-24911) and specific domains (vault vs. generic malware). Nonetheless, our approach performs as well as—or in some cases, slightly better than—those in [4,5,11,17], even with a (mostly) smaller sample size.

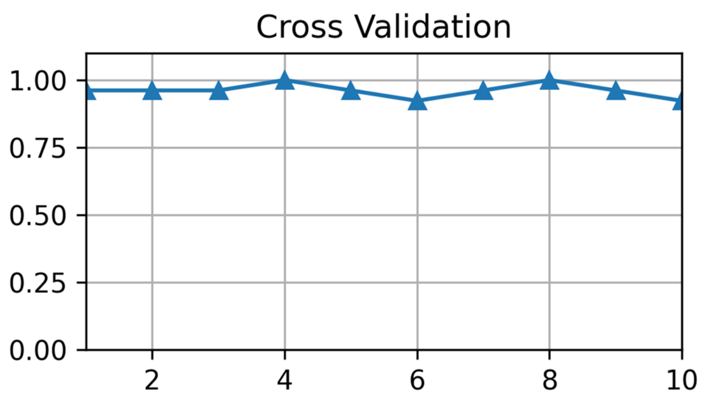


Figure 10. Cross-validation plot for RF classifier.

Table 4. Comparison of Our RF classifier with related work.

Reference	Accuracy	F_1 Score	Sample Size	Vault Apps
Wang [4]	0.880	-	2000	No
Mahindru [17]	-	0.997	11,000	No
Xie [5]	0.933	-	102	Yes
Singh [11]	0.988	0.990	24,911	No
Our Approach	0.981	0.981	1040	Yes

5. Conclusions and Future Work

Detecting and classifying vault applications on mobile devices remains a challenge due to various forensic and legal considerations. While some vault apps may exhibit characteristics similar to malware, many serve legitimate privacy functions. Given the popularity of phones, any approach that is non-invasive and accurate would be of use to law enforcement bodies. This research undertook the reverse-engineering of Android vault and non-vault applications to uncover and extract their distinct properties. We then applied various machine learning algorithms to classify these properties, aiming to validate the effectiveness of a machine learning-driven approach. Given that machine learning thrives on data, we successfully demonstrated that good results can be obtained even with a modest dataset of around 1000 observations. Notably, our analysis revealed that the support vector machine (SVM), multi-layer perceptron (MLP), and random forest classifiers yielded the most promising outcomes. Looking ahead, we plan to test additional algorithms, expand our sample size, and further investigate other Android permissions, paving the way for more robust findings.

In particular, using unsupervised machine learning algorithms can divulge meaningful insights, and a robust comparison can be made between supervised, semi-supervised, and unsupervised machine learning algorithms. In addition, the usage of vault apps by cyber criminals such as ransomware groups will be investigated to see whether a correlation exists between the successful ransomware attacks and the vault apps. It would also be worthwhile to check the usage of vault apps by online scammers. These future investigations will enhance our research contributions in this area of digital forensics, which is a growing area, as noted by Baig et al. [23]. We will endeavor to integrate our research findings into industrial practices such as law enforcement and criminal justice systems, leading to more impactful and multidisciplinary research.

We highlight that false positives in machine learning-based detection of vault (anti-forensic) apps pose significant legal and ethical concerns, as wrongfully flagging a legitimate privacy-focused application as malicious could lead to privacy violations, reputational damage, and legal liability. As a future research direction, mitigating false positives requires enhancing dataset diversity, incorporating explainable AI techniques, implementing adaptive thresholding, and involving human supervision in decision-making. Ensuring a balanced detection approach minimizes wrongful classifications while maintaining the integrity and reliability of forensic investigations.

Author Contributions: Conceptualisation, M.N.J.; methodology, M.N.J.; implementation, M.N.J.; writing—original draft preparation, M.N.J.; writing—review and editing, M.N.J., W.Y. and M.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: All data are available on request.

Acknowledgments: We thank E. Mobegi, K.P. Chiu, K. Sodha, N. Areba, and O. Adeniyi for their assistance with some of the data collection, data curation, pre-processing, and aspects of the algorithms.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Khubrani, M.M. Mobile device forensics, challenges and blockchain-based solution. In Proceedings of the 2023 Second International Conference on Smart Technologies for Smart Nation (SmartTechCon), Singapore, 18–19 August 2023; pp. 1504–1509.
2. Dorai, G.; Aggarwal, S.; Patel, N.; Powell, C. VIDE—Vault App Identification and Extraction System for iOS Devices. *Forensic Sci. Int. Digit. Investig.* **2020**, *33*, 301007. [CrossRef]
3. Zhang, X.; Baggili, I.; Breitinger, F. Breaking into the Vault: Privacy, Security and Forensic Analysis of Android Vault Applications. *Comput. Secur.* **2017**, *70*, 516–531. [CrossRef]
4. Rasul, T.; Latif, R.; Jamail, N.S.M. A computational forensic framework for detection of hidden applications on Android. *Indones. J. Electr. Eng. Comput. Sci.* **2020**, *20*, 353–360. [CrossRef]
5. Xie, N.; Bai, H.; Sun, R.; Di, X. Android Vault Application Behavior Analysis and Detection. In *Data Science. ICPCSEE 2020. Communications in Computer and Information Science*; Zeng, J., Jing, W., Song, X., Lu, Z., Eds.; Springer: Singapore, 2020; Volume 1257. [CrossRef]
6. Duncan, M.; Karabiyik, U. Detection and Recovery of Anti-Forensic (VAULT) Applications on Android Devices. In Proceedings of the ADFSL Conference on Digital Forensics, Security and Law, San Antonio, TX, USA, 17–18 May 2018; University of Texas at San Antonio: San Antonio, TX, USA, 2018. Available online: https://commons.erau.edu/adfsl/2018/presentations/6 (accessed on 13 August 2024).
7. Mikalauskas, E. These Android Privacy Apps Might Be Distributing Malware and Spying on You. 2020. Available online: https://cybernews.com/security/these-android-privacy-apps-might-be-distributing-malware-and-spying-on-you/ (accessed on 29 January 2025).
8. Grégio, A.R.A.; Afonso, V.M.; Filho, D.S.F.; de Geus, P.L.; Jino, M. Toward a Taxonomy of Malware Behaviors. *Comput. J.* **2015**, *58*, 2758–2777. [CrossRef]
9. Soliman, S.W.; Sobh, M.A.; Bahaa-Eldin, A.M. Taxonomy of malware analysis in the IoT. In Proceedings of the 2017 12th International Conference on Computer Engineering and Systems (ICCES), Cairo, Egypt, 19–20 December 2017; pp. 519–529. [CrossRef]
10. Ngo, Q.D.; Nguyen, H.T.; Le, V.H.; Nguyen, D.H. A survey of IoT malware and detection methods based on static features. *ICT Express* **2020**, *6*, 280–286. [CrossRef]
11. Singh, J.; Singh, J. Detection of malicious software by analyzing the behavioral artifacts using machine learning algorithms. *Inf. Softw. Technol.* **2020**, *121*, 106273. [CrossRef]
12. Abusitta, A.; Li, M.Q.; Fung, B.C.M. Malware classification and composition analysis: A survey of recent developments. *J. Inf. Secur. Appl.* **2021**, *59*, 102828. [CrossRef]
13. Ruffin, M.; Lopez-Toldeo, I.; Levchenko, K.; Wang, G. Casing the Vault: Security Analysis of Vault Applications. In Proceedings of the 21st Workshop on Privacy in the Electronic Society, Los Angeles, CA, USA, 7 November 2022; pp. 175–180. [CrossRef]
14. Nash, A.; Choo, K.K.R. Password Managers and Vault Application Security and Forensics: Research Challenges and Future Opportunities. In *Digital Forensics and Cyber Crime*; Goel, S., Nunes De Souza, P.R., Eds.; Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering; Springer Nature: Cham, Switzerland, 2024; pp. 31–53. [CrossRef]

15. Hutchinson, S.; Karabiyik, U. Forensic Analysis of Spy Applications in Android Devices. In Proceedings of the ADFSL Conference on Digital Forensics, Security and Law, Daytona Beach, FL, USA, 15–16 May 2019; 3. Available online: https://commons.erau.edu/adfsl/2019/paper-presentation/3 (accessed on 29 January 2025).
16. Wang, J.; Li, B.; Zeng, Y. XGBoost-Based Android Malware Detection. In Proceedings of the 13th International Conference on Computational Intelligence and Security (CIS), Hong Kong, China, 15–18 December 2017; pp. 268–272.
17. Mahindru, A.; Singh, P. Dynamic Permissions Based Android Malware Detection using Machine Learning Techniques. In Proceedings of the 10th Innovations in Software Engineering Conference (ISEC '17), Jaipur, India, 5–7 February 2017; Association for Computing Machinery: New York, NY, USA, 2017; pp. 202–210. [CrossRef]
18. Peng, M.; Khanov, M.; Madireddy, S.R.; Chi, H.; Akbas, E.; Dorai, G. DECADE-deep learning based content-hiding application detection system for Android. In Proceedings of the 2021 IEEE International Conference on Big Data (Big Data), Orlando, FL, USA, 15–18 December 2021; pp. 5430–5440. [CrossRef]
19. Qiu, J.; Zhang, J.; Luo, W.; Pan, L.; Nepal, S.; Xiang, Y. A Survey of Android Malware Detection with Deep Neural Models. *ACM Comput. Surv.* **2020**, *53*, 1–36. [CrossRef]
20. Android Developer Guide, App Manifest Overview. 2025. Available online: https://developer.android.com/guide/topics/manifest/manifest-intro/ (accessed on 11 April 2025).
21. Pearson, K. On lines and planes of closest fit to systems of points in space. *Lond. Edinb. Dublin Philos. Mag. J. Sci.* **1901**, *2*, 559–572. [CrossRef]
22. Jolliffe, I.T. *Principal Component Analysis*; Springer: New York, NY, USA, 2002.
23. Baig, Z.; Szewczyk, P.; Valli, C.; Rabadia, P.; Hannay, P.; Chernyshev, M.; Johnstone, M.; Kerai, P.; Ibrahim, A.; Sansurooah, K.; et al. Future Challenges for Smart Cities: Cyber-Security and Digital Forensics. *Digit. Investig.* **2017**, *22*, 3–13. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.