big data and cognitive computing

MDPI

Article

Behavioral Analysis of Android Riskware Families Using Clustering and Explainable Machine Learning

Mohammed M. Alani [1,*,†] and Moatsum Alawida [2,†]

[1] Department of Electrical Engineering and Computing Sciences, Rochester Institute of Technology (RIT-Dubai), Dubai P.O. Box 341055, United Arab Emirates
[2] Department of Computer Sciences, Abu Dhabi University, Abu Dhabi P.O. Box 59911, United Arab Emirates; moatsum.alawida@adu.ac.ae
* Correspondence: m@alani.me
† These authors contributed equally to this work.

Abstract: The Android operating system has become increasingly popular, not only on mobile phones but also in various other platforms such as Internet-of-Things devices, tablet computers, and wearable devices. Due to its open-source nature and significant market share, Android poses an attractive target for malicious actors. One of the notable security challenges associated with this operating system is riskware. Riskware refers to applications that may pose a security threat due to their vulnerability and potential for misuse. Although riskware constitutes a considerable portion of Android's ecosystem malware, it has not been studied as extensively as other types of malware such as ransomware and trojans. In this study, we employ machine learning techniques to analyze the behavior of different riskware families and identify similarities in their actions. Furthermore, our research identifies specific behaviors that can be used to distinguish these riskware families. To achieve these insights, we utilize various tools such as k-Means clustering, principal component analysis, extreme gradient boost classifiers, and Shapley additive explanation. Our findings can contribute significantly to the detection, identification, and forensic analysis of Android riskware.

Keywords: android; malware; riskware; behavioral analysis; explainable machine learning; XAI

Citation: Alani, M.M.; Alawida, M. Behavioral Analysis of Android Riskware Families Using Clustering and Explainable Machine Learning. *Big Data Cogn. Comput.* **2024**, *8*, 171. https://doi.org/10.3390/bdcc8120171

Academic Editor: Moulay A. Akhloufi

Received: 14 September 2024
Revised: 3 November 2024
Accepted: 22 November 2024
Published: 26 November 2024

1. Introduction

In today's world, we heavily rely on smart devices, especially mobile ones, for a multitude of purposes. These cutting-edge gadgets have been developed to be user-centric, serving functions in communication, data exchange, gaming, entertainment, economics, commerce, and numerous other sectors. At the heart of this digital revolution are mobile applications (apps), which cater to an array of functionalities such as banking, entertainment, and [1]. These applications rely on operating systems (OS) to facilitate seamless user interaction with hardware and manage tasks effectively. Android, a globally popular OS, has garnered significant attention for its open-source nature, making it the OS of choice to many smartphone vendors.

Since its initial release in 2008, Android has experienced substantial growth in adoption. As an open-source operating system, Android has garnered a significant market share due to its "free-to-use" nature. Vendors rushed to adopt this mobile operating system, leading to the impressive growth depicted in [2]. Android has held the top spot in market share for almost a decade now. It has consistently surpassed a 70% market share since 2016. This growth extended beyond mobile phones to Android's adoption as an operating system for tablet computers, Internet-of-Things (IoT) devices, and other types of smart gadgets [3]. Many consumer IoT devices, such as smart cameras, smart health devices, or smart TVs, now feature Android as their primary operating platform.

One of the main reasons of Android's growth is the ecosystem that allows developers to create and upload their apps to the Google Play Store. This app store is easily accessible

to billions of users worldwide. Such accessibility comes with a cost-Android apps often require multiple permissions to gain access to a range of device resources such as sensors, location services, microphone, camera, device storage, and more [4]. These permissions can be easily misused by a malicious actor.

An added dimension in the security challenges of Android apps is that the OS also provides the users with the ability to sideload apps from sources other than the official Google Play Store. While some apps serve valuable purposes such as mobile banking, health tracking, etc., others lurk with malicious intent, aiming to harm users and steal personal data [5]. Permissions pose a particular challenge as discussed in [6–9]. While these permissions provide access to legitimate apps, they can be abused by malicious apps to wreak havoc. Benign and malicious apps often require similar permissions, making it challenging to normal users to recognize harmful intent solely based on access requests. Therefore, relying solely on permission-based detection for malicious apps falls short. While it can aid in risk assessment, it is not a foolproof method for identifying hidden threats [10].

The rapid proliferation of malicious, and malware-infested apps on Android poses a continuous threat to users' security. This could range from lower-risk apps such as Adware [11], to higher-risk ones such as ransomware [12]. While traditional signature-based and machine learning detection methods are still employed, attackers have adapted by developing stealthy apps that can evade both detection methods [13]. This, combined with the rise of unwanted apps (riskware) that can pose security threats even if they are not technically malware, underscores the need for continued research and development of new detection approaches [14]. Further exploration of advanced techniques, such as anomaly detection and behavior analysis, could hold the key to protecting users from these evolving threats.

While malware directly harms systems and devices with malicious code, riskware apps pose security threats through potentially unwanted or suspicious behavior. Section 2.1 discusses riskware in further details.

Machine learning offers three main approaches to detecting malicious apps (including riskware): static, dynamic, and hybrid analysis [15]. Static analysis relies on analyzing the app without actively running it by examining its code, and permissions required [16]. Dynamic analysis relies on running the malicious app and observing its behavior [17]. A hybrid approach employs both static and dynamic analysis [18]. However, identifying riskware presents specific challenges. While unwanted, these apps may not explicitly or immediately harm devices, making them more challenging to recognize as malicious compared to traditional malware. These challenges in the detection and understanding of riskware behavior made it a less likely subject security research. While many studies addressed general malware detection [1,19], riskware detection and analysis did not receive comparable attention.

To address the challenge of identifying riskware, this research explores the use of explainable machine learning, along with clustering techniques to study the behavior of riskware applications and distinct behaviors of riskware families. The research contributions of this research can be summarized as follows:

1. Analyze behavioral similarities of different Android riskware families through clustering.
2. Analyze the specific behavioral traits that distinguish different Android riskware families through the use of explainable machine learning.
3. Utilizing explainable machine learning to identify features, and feature values that can help in distinguishing riskware in Android ecosystem.

The following section in this paper presents preliminary information about riskware, clustering, and Shapley additive explanations. Section 3 provides a review of some related works. Section 4 provides introductory information about the dataset used in our experiments, along with detailed explanation of the experimental steps taken. In Section 5 we present the results, and analyze the findings of the clustering process performed on the

dataset. The creation of cluster detection classifiers is presented in Section 6. The detailed analysis of Shapley additive explanations are presented in Section 7. Discussion of the results is presented in Section 8, while the research conclusions, and directions of future work are presented in Section 9.

2. Preliminaries

In this section, we introduce preliminary knowledge into different research areas within this paper. Three subsections are presented in this section as follows: riskware, clustering scheme, and Shapley additive explanations.

2.1. Riskware

Riskware is a type of software application that can be found on Google Play and exhibits unwanted behaviors. While riskware apps may not directly harm devices or steal sensitive data like traditional malware, they can still pose risks [20,21]. Undesirable consequences of riskware apps include resource draining, displaying unwanted advertisements, a high volume of notifications, unnecessary permissions, unclear functionality, and data sharing [22,23]. Riskware apps serve legitimate purposes but may request access to sensitive data through privacy permissions granted by users on their smart devices. While riskware is generally classified as a category of malware, its behavioral traits make it more challenging to detect in comparison to other types of malware. Typical solutions for detecting riskware apps include using antivirus and anti-malware apps, implementing user interaction permissions, and ensuring regular app updates. Anti-malware apps play a crucial role in identifying and mitigating risks based on their updated databases [24]. Users should be aware of how permissions function to access device features and data, allowing them to classify permissions as either risky or normal. Previous research have shown that many users are negligent when it comes to reading, understanding, and acting upon the apps required permissions. A survey conducted in [6] showed that only 35% of Android users read the required permissions before installing any app. The study also showed that 22% of android users have never refused to install an app due to the intrusive permissions that the app requires.

Several examples of riskware apps on the Android operating system include battery optimizers that claim to enhance battery life but may drain it through background activities. Additionally, certain cleaner apps aim to optimize devices by removing unwanted or temporary files but might unintentionally delete useful files. Fake antivirus apps pose as security solutions but, in reality, install malware on devices. Some apps inject advertisements into other apps, causing frequent on-screen ads. Furthermore, certain games are classified as riskware as they consume device resources such as battery, processing power, memory space, impacting overall performance [25–29].

There are distinctions between other types of malware and riskware apps based on specific criteria. In terms of intention, riskware apps can cause harm to devices and data through unwanted behaviors or privacy risks. Detecting riskware apps is challenging as they often operate as legitimate apps on devices. Table 1 illustrates the primary differences between malware and riskware on the Android operating system. It is evident that detecting riskware requires considerable effort due to users perceiving them as legitimate applications.

Table 1. Comparison of riskware to other malware types.

Feature	Riskware	Other Malware
Intention	Indirectly harmful	Directly harmful
Behavior	Exhibits undesirable behaviors that can be weaponized	Direct attacks
Detection	Traditional methods are not capable to detect riskware apps because they perform behaviors the resemble legitimate apps	Use traditional methods to detect signatures, or behaviors
Example	Battery optimizers, fake antivirus, games	Viruses, Trojans, Spyware, Ransomware

Malicious actors also use riskware as an attack vector to deliver other types of malware. All apps submitted to the Google Play Store receive initial security check before being made available to users. Most riskware apps are capable of bypassing this initial screening because they do not match traditional malware signatures. However, malicious actors can use them at a later stage to deploy other types of malware without raising suspicion. This scenario is especially possible when the riskware app already has a wide range of permissions that the deployed malware would need. This unchecked access to smartphone resources can put the user at high risk of sensitive data leak, ransomware, trojan infections, among many other risks.

2.2. Clustering

Clustering is a powerful technique within unsupervised machine learning designed to group data elements based on shared features [30]. It is frequently employed to unearth similarities and inherent patterns in data. Clustering algorithms are designed to operate on input data lacking labels, aiming to identify distinct groups within the given dataset. Each identified group is referred to as a cluster, characterized by similar features across all data elements within that cluster. Various clustering algorithms exist, each with its unique approach. Prominent examples include K-Means [31], hierarchical clustering [32], Gaussian Mixture Model algorithm [33], and Mean-Shift clustering algorithm [34].

K-means clustering is one of the most widely used clustering algorithms. It utilizes a centroid-based approach to partition input data into a specified number of clusters, denoted as K. Within each cluster, data elements exhibit similar features. The algorithm iteratively refines the clusters to minimize the within-cluster variance. K-means clustering can be mathematically written as shown in Algorithm 1.

Algorithm 1: K-Means clustering algorithm.

Input: X Unlabelled dataset, n datapoints, k clusters
Output: Labelled (Cluster) dataset
Randomly initialize k cluster centroids $\mu_1, \mu_2, ..., \mu_k$
while *convergance = False* **do**
  $c^{(i)} = \arg\min_j \|x^{(i)} - \mu_j\|^2$
  $\mu_j = \frac{1}{|C_j|} \sum_{i \in C_j} x^{(i)}$
end

As shown in the algorithm, the steps start with random initialization of k cluster centroids $\mu_1, \mu_2, \ldots, \mu_k$. In the next step, each data element $x^{(i)}$ is assigned to the cluster whose centroid is closest. In the next step, each cluster centroid μ_j is updated to be the mean of all data elements assigned to cluster j, where C_j is the set of data elements assigned to cluster j. The previous two steps are repeated until a convergence state is reached. The convergence condition is when the assignments of datapoints to clusters no longer change significantly.

2.3. Shapley Additive Explanations

Explainable machine learning can be defined as the identification and understanding of features that have contributed to generating a decision within a specific dataset [35]. There are four main reasons to use explanations for the decision results of a learning model: justification of decisions, control of the model, improvement of the model, and discovery of new knowledge [36]. Explainable machine learning focuses on demystifying the learning model, aiming to produce transparent models and ensure that the decisions are not originating from a black-box model. Various explainability techniques, including decision trees [37], rule lists [38], LIME [39], activation maximization [40], and Shapley Additive Explanations (SHAP) [41], are employed to understand the decisions behind prediction results.

To assess the impact of each attribute value on prediction outcomes, SHAP is employed as a an explanation method used to explain machine learning models [41]. SHAP computes values for the impact of individual features, aiding human comprehension of each feature's significance in the machine-learning model, and each decision made by this model. Moreover, these attribute values enhance our understanding of how the learning model arrives at its final decision results. SHAP was developed based on a game-theoretic concept called Shapley values, which studies the impact of each feature and its contribution to the learning prediction model [42]. The general concept behind it is to calculate the impact of each player in a team by calculating the performance of the team with and without this player.

To calculate Shapley values in the learning model, the Shapley values are initialized for each feature at zero. Then, one Shapley value for one feature is permuted while keeping the other Shapley values of the remaining features constant. The model is reevaluated with the new Shapley values for the permuted feature and constant values for the remaining features. The contribution of each feature is calculated and compared between the original and permuted features for prediction. This process is repeated for all possible permutations of features, and the average contribution across each permuted feature is computed. Average Shapley values for permutations are calculated for all possible orderings of features. Finally, the results yield positive or negative Shapley values, where positive indicates positive contributions, and negative values indicate negative contributions.

3. Related Works

Malware in Android operating systems has seen a surge over the past decade [25]. This rise has spurred researchers to develop detection systems for identifying different types of malware, emphasizing app features. The literature review reveals three main types of detection systems for Android malware: signature-based, specification-based, and behavior-based. In signature-based methods employed by commercial software vendors, semantic patterns of known malware are extracted to generate signatures. When the system matches the signature of an app with those stored in the database, it identifies the app as malware. However, the effectiveness depends on the regularly updated database, and the system may struggle to detect new malware for which signatures are not yet available.

The specification-based method is another malware detection approach that relies on predefined rules to identify malicious software. This method depends on a predefined binding rule-set to classify malware apps. The main difficulty with this method lies in the implementation, where accurately specifying the app features and behaviors can be challenging [43]. The behavior-based method is a widely used approach for detecting malware apps [44], relying on the features and purposes of the apps. The method involves code analysis, log file examination, collected data analysis, shared input data analysis, behavior during runtime, behavior during installation, and permissions extraction from both benign and malicious apps. There are three types of detection systems based on behaviors: static, dynamic, and hybrid [45]. All these methods employ machine learning to classify malware apps.

In related work, numerous researchers have concentrated on detecting malware apps through app permission features. Felt et al. investigated and examined the permissions granted by users to third-party applications during the installation process [46]. The authors observed a strong positive security impact on the system. In a quest to safeguard mobile app privacy, Sendor et al. proposed an information entropy method that leverages Android's permission system for evaluation [47]. Their study zeroed in on security risks posed by permission services, with a particular focus on social apps and potential threats from third-party or unwanted apps. While promising, the method encountered limitations in addressing code and communication vulnerabilities. To hunt down malware apps, Rani and Dhindsa analyzed Android applications using both static and dynamic techniques [27]. Their arsenal included permission-based and behavioral-based filtering, powered by dedicated malware analysis tools. The results were alarming: 80% of

the apps requested potentially dangerous permissions, and a concerning 13% displayed malicious behavior.

Rani and Dhindsa employed permission-based and behavioral foot-printing methods to identify malware in Android apps, analyzing both official and third-party app stores [26]. The results highlighted vulnerabilities in both stores, with infection rates of approximately 12.9% for the official store and 28% for third-party stores. Jianmao et al. introduced a new method utilizing static analysis and collaborative filtering to determine the minimum permissions required for Android apps [48]. The study involved analyzing 16,343 popular apps from Google Play, aiming to identify abnormal permission usage and enhance permission recommendation configurations. Khullar et al. [49] introduced a new approach for analyzing the risk features within a set of mobile apps on the Android platform using a static method. They examined Android manifest files within a dataset comprising approximately 3000 apps. The results revealed 25 hazardous features commonly exploited by attackers to create malicious apps.

Bhat et al. introduced a new approach for detecting malicious attacks on Android through dynamic behavioral analysis of malware [50]. They reconstructed the behavior features of Android malware apps by incorporating system calls, binders, and complex Android objects into the detection and classification system. The proposed system employed feature selection and homogeneous and heterogeneous ensemble machine learning algorithms. The results demonstrated a high accuracy rate of approximately 98.08% in the classification process. Yang et al. proposed an innovative method for Android malware detection by extracting API semantics in three phases [51]. The initial phase involved API clustering to represent API functions, generating API sentences to summarize API features. In the second phase, call graphs were extracted from each app, and in the third phase, feature call pairs were collected from each cluster in API clustering. Machine learning classifiers were then employed with feature vectors to detect malware apps. The results showed high accuracy and slower speed in the detection method. Data from 2012 to 2023 was utilized, achieving an average F1-measure of 82.6%

Liu et al. introduced a new method for the classification of Android malware by integrating a convolutional neural network (CNN) with batch normalization [52]. They collected 347-dimensional network traffic features using inception-residual network modules. The results demonstrated an accuracy of 99.73% in binary classification and the ability to classify malware into four categories and malicious families into 35 categories. The accuracy for both classifiers was approximately 99.53% and 94.38%, respectively. To mitigate the need for modifications or costly re-training, a new method to detect Android malware based on app behavior was proposed [53]. API calls were abstracted, and a Markov chain was generated to build a sequence of app behaviors. Features were extracted as families and packages of API calls. The dataset consisted of 8.5 K benign and 35.5 K malicious apps. The classifier successfully classified malware with up to a 99% F-measure.

In the literature review, we observed a lack of studies specifically focusing on riskware apps in the Android operating system. We summarized papers related to detecting malware based on permissions, behaviors, signatures, and static methods. While some studies explored the risk analysis of certain app features, they did not classify apps as riskware. Consequently, our study concentrates on clustering riskware families and analyzing app behaviors using explainable machine learning.

4. Dataset and Analysis Plan

In this section we discuss the dataset used in our experiments, the implementation environment, and explain the experimentation steps.

4.1. Dataset

In our experiments, we use CIC-AndMal-2020 dataset. This dataset was first introduced in [54] in 2020. The dataset includes data extracted from 400,000 Android applications (200,000 benign, and 200,000 malicious). The dataset included two subsets; one

that includes static features, extracted from the application files directly, and another that includes dynamic features extracted by running the malicious and benign applications and capturing relevant dynamic features. The data included covered 14 different categories of malware divided into 191 different families.

As our research is focused on analyzing Riskware behavior, we focus on the dynamic features extracted from Riskware samples. The dynamic features in the dataset were a total of 141 features. The complete list of features can be found on the dataset's webpage [55]. These features were categorized into the following categories:

- Memory: Features capturing the malware interactions with the device's memory.
- API: Application Programming Interface (API) features capture the interactions of the applications with other applications, and the OS.
- Network: Network features capture metadata about the foreground, and background network activity of the application.
- Battery: Battery features capturing requests that impact the battery, such as waking up the screen.
- Logcat: Logcat features create log entries relevant to a malware function.
- Process: Process features count the interaction of malware with total number of processes.

Upon examining the riskware dataset, the only preprocessing that was found to be necessary, was to remove the sample hash signatures, and the malware category (riskware) from the data. At the end of the preprocessing, the dataset consisted of 14,053 samples. Each sample carrying 141 dynamic features extracted from Android devices upon running the riskware apps, and after restarting the infected device. These 14,053 samples were unevenly distributed on 18 different known riskware families, and 1 class for samples collected from unknown riskware families. The number of samples per family varied. The highest number of samples was from `smsreg` riskware with 8112 samples, and the lowest was one sample from `badpac` family. The number of samples with unknown riskware family was 597.

4.2. Implementation Environment

All experiments, training and testing was performed on a computer with the following specifications:

- Processor: AMD Ryzen 9 5950x
- RAM: 128 GB
- GPU: Nvidia RTX 4070Ti
- Operating System: Linux Manjaro 6.5.5
- Python 3.9.18
- Sci-Kit Learn 1.2.2
- SHAP 0.42.1

4.3. Experiment and Analysis Plan

The analysis plan is comprised of the following steps:

1. Isolate the riskware samples from other types of malware within the selected dataset.
2. Clustering riskware families using k-Means algorithm, after finding the best number of clusters.
3. Build detection classifiers designed to detect each individual cluster of riskware families.
4. Perform machine learning explainability calculations to have a deeper understanding of the behavior of the riskware families within each cluster, and what differentiates them from other families falling within the other clusters.

5. Clustering Riskware Families

At the starting stage, we used k-Means algorithm, as described in Section 5, to put different riskware families is clusters based on their behavioral features captured in the dataset. In this stage, we need to identify a suitable number of clusters before the clustering process starts.

The method used in detecting the optimal number of clusters is the Elbow Method [56]. In this method, we iterate through the number of clusters (k) from 1 to 10. For each cluster, we calculate the sum of squared sitance between each point in the cluster, and the cluster's centroid. This is called Within-Cluster Sum of Squares (WCSS), or inertia. As the obtained WCSS values are ploted with the value of k, the plot looks like an elbow. As k increases, the WCSS value decreases. However, within the plot, there usually is a value of k where the value of WCSS will rapidly change to create the elbow shape. Beyond this point, the graph moves in a shape that is almost parallel to the x-axis. The point of change between the high-rate change in WCSS to the low-rate change represents the optimum k value.

Figure 1 shows the result of applying the elbow method to identify the optimum number of clusters.

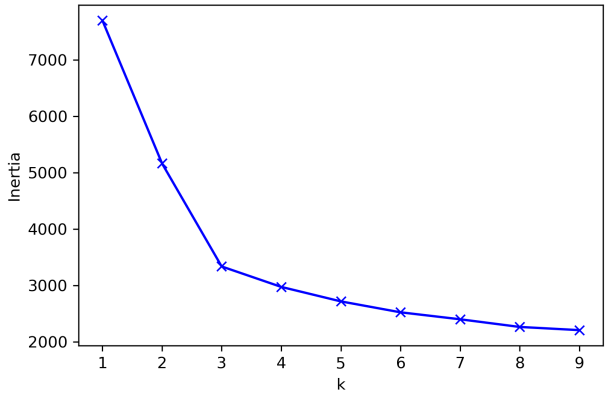


Figure 1. Selecting the best number of clusters using elbow graph.

As shown in the figure, $k = 3$ is the point where the drop in inertia starts stabilizing, and the rate of change of the WCSS becomes low. Thus, the selected number of clusters for our experiment is 3.

Upon choosing $k = 3$, we proceed with the k-Means clustering process as stated in Algorithm 1. Upon the completion of clustering the 14,053 samples, we cross-match the clusters with the original riskware family classes of these samples. Table 2 shows the number of samples from each riskware family, in each cluster. This table was demonstrated in Figure 2 as a heatmap of riskware families in each cluster.

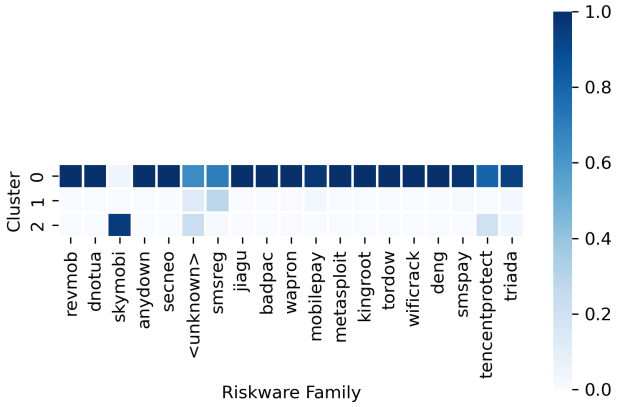


Figure 2. Heatmap of clusters and Riskware families.

Table 2. The number of samples of each family in the three clusters.

Riskware Family	Cluster 0	Cluster 1	Cluster 2
revmob	937	3	0
dnotua	36	0	0
skymobi	123	0	2734
anydown	61	0	0
secneo	5	0	0
unknown	385	76	136
smsreg	5676	2328	108
jiagu	326	0	0
badpac	1	0	0
wapron	16	0	0
mobilepay	102	3	0
metasploit	28	0	0
kingroot	3	0	0
tordow	3	0	0
wificrack	16	0	0
deng	9	0	0
smspay	756	14	0
tencentprotect	4	0	1
triada	153	4	6
Total	8640	2428	2985

As shown in both the table and the heatmap figure, Cluster 0 included the highest number of samples, totaling to 8640. Clusters 1 and 2 included 2428 and 2985 samples, respectively.

Upon examining the distribution of the samples from different families across the clusters, we found the following observations:

1. Most samples generated by 16 out of the 18 known families exhibit similar behaviors that resulted in them falling withing Cluster 0. This indicates that these families exhibit similar behaviors.
2. The only family that has significant portion of its samples falling within Cluster 2, is `skymobi`. Similar behavior was demonstrated by a small proportion of the unknown-family samples.
3. Some riskware families, such as `smspay`, `skymobi` and `revmob` have a relatively small number of outlier samples that have fallen outside of the cluster where most of their samples exist. Such phenomenon is expected an complex clustering problems.
4. The samples with unknown-family were distributed across all clusters, with 64% of these samples in cluster 0, and 13% and 23% in clusters 1 and 2, respectively. This is expected as the families of these samples are not particularly known. Therefore, they are expected to exhibit behaviors that can fall within any cluster.
5. The riskware family named `smsreg`, which is the family with the highest number of samples, demonstrated an interesting set of results. About 70% of its samples fell within cluster 0, 28.7% within cluster 1, and 1.3% within cluster 2. While the samples falling within cluster 2 can be considered outliers, and can be ignored, the 28.7% within cluster 1 cannot be ignored. Upon further analysis, we found that this riskware family is one of the largest families of riskware, with many variants that were developed in the past decade. Many of these variants were designed to perform differnet actions, and hence its behavior can fall within more than one cluster [57].

As the data is multi-dimensional (141 features), it is not easy to produce proper visualization showing the clusters. However, we use PCA to perform dimensionality reduction down to two features only, solely for the purpose of visualization. Once the data is reduced to two features, it is used to produce the two-dimensional representation of the three clusters. This visualization is presented in Figure 3.

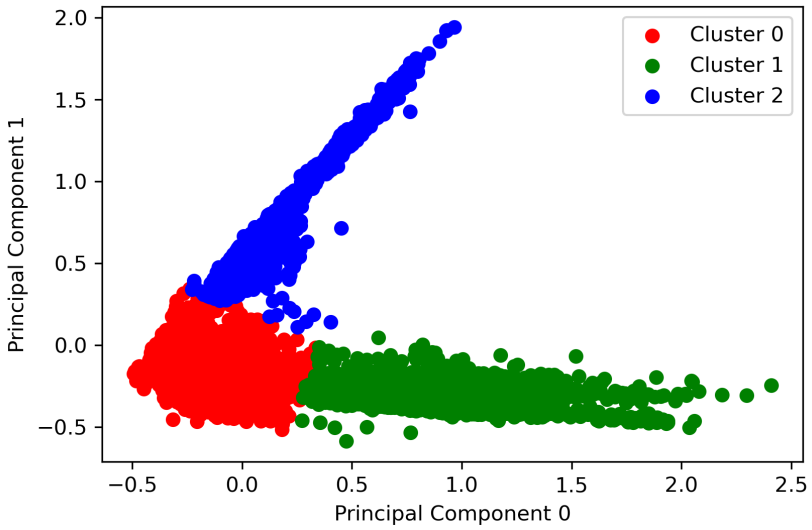


Figure 3. Clusters 2-dimensional visualization.

While the 2-dimensional visualization shown in Figure 3 is not a perfect representation of the data, it does show reasonable distinction between the clusters. It also shows some overlap between the clusters, which is a reasonable phenomenon to expect because all of these families fall within the same category of malware, and are expected to exhibit somewhat similar actions.

6. Cluster-Detection Classifiers

After classifying the data into three clusters, we proceed to create three distinct detection classifiers where each classifier is trained to detect one cluster of the three only. This helps in providing a better understanding of the specific features that distinguish the behavior of riskware families within each cluster. Therefore, we adopt a one-vs-all technique to create these three different classifiers; one classifier to detect cluster 0, one to detect cluster 1, and one to detect cluster 2.

To facilitate the creation of these three classifiers, we create three different datasets by re-labelling the original dataset. The steps followed to create these three datasets are shown in Algorithm 2.

The steps shown in Algorithm 2 iterates through all data samples one bye one. If the current sample belongs to Cluster 0, it would be labelled 1, and appended to Cluster0-dataset, and re-labelled 0 and appended to Cluster1-dataset, and cluster2-dataset. This way, Cluster-0 dataset will have all samples from Cluster0 labelled as 1, and all other samples labelled 0. Similarly, a sample labelled Cluster 1, would be re-labelled to 1, and appended to Cluster1-dataset, and relabelled 0, and added to the Cluster0-dataset, and Cluster2-dataset, and so on with Cluster 2 samples. At the end of the algorithm, we produced three datasets. In the dataset Cluster0-dataset, all cluster 0 samples are labelled 1, and all other samples are labelled 0. Similarly, in the dataset Cluster1-dataset, all cluster 1 samples are labelled 1, and all other samples are labelled 0. In the dataset Cluster2-dataset, all cluster 2 samples are labelled 1, and all other samples are labelled 0. The purpose of building such datasets is to build a group of one-vs-all classifiers that can be trained to detect one specific cluster. Table 3 shows the statistics of each of the three datasets.

At this stage, we utilize an XGB classifier for the detection of each cluster. Each dataset was randomly split to 66.6% samples for training, and 33.3% samples for testing. The random splits were performed with stratification to maintain the percentage of representation of each class in the train and test subsets. Instead of using a single multi-class classifier to detect the three clusters, we chose to use a one-vs-all method to create one classifier to detect each individual cluster. The reason behind that is that these individual binary

classifiers will be used to determine the most effective features in detecting each individual cluster using SHAP, rather than one classifier with one set of features for all clusters.

Algorithm 2: Creating three datasets by re-labelling the original dataset.

```
Input: RiskwareDataset
Output: Three cluster datasets
forall sample in RiskwareDataset do
    if sample.label="Cluster0" then
        sample.label = 1
        Cluster0_dataset ← sample
        sample.label = 0
        Cluster1_dataset ← sample
        Cluster2_dataset ← sample
    end
    else if sample.label="Cluster1" then
        sample.label = 1
        Cluster1_dataset ← sample
        sample.label = 0
        Cluster0_dataset ← sample
        Cluster2_dataset ← sample
    end
    else
        sample.label = 1
        Cluster2_dataset ← sample
        sample.label = 0
        Cluster0_dataset ← sample
        Cluster1_dataset ← sample
    end
end
```

Table 3. Statistics of the three cluster-specific datasets.

Datasets	0	1	Total Samples
Cluster0-dataset	5413	8640	14,053
Cluster1-dataset	11,625	2428	14,053
Cluster2-dataset	11,068	2985	14,053

Figure 4 shows the confusion matrix plots of the three classifiers. The detailed performance metrics obtained by testing are shown in Table 4.

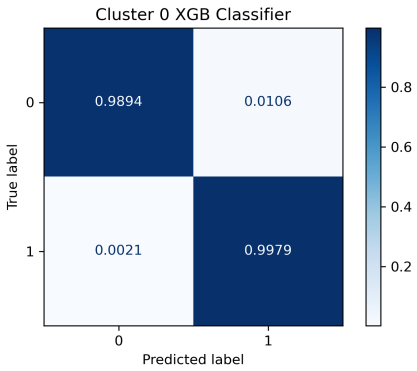
Table 4. XGB classifiers' testing results.

Classifier	Accuracy	Precision	Recall	F_1 Score
Cluster0	0.9946	0.9949	0.9936	0.9943
Cluster1	0.9978	0.9972	0.9952	0.9962
Cluster2	0.9971	0.9967	0.9948	0.9958

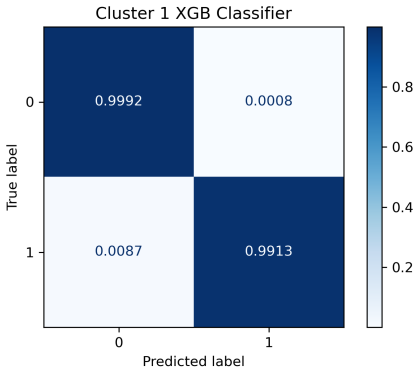
The high accuracy, and F_1 scores, shown in Table 4 shows that these trained classifiers were capable, with high accuracy, to capture the distinct behavioral differences between the different clusters. The riskware families in cluster 0, demonstrate dynamic features that are adequately distinct to enable the classifier to detect it with such high accuracy. The same can be said about the other two classifiers.

Figure 4a shows a false-positive rate (FPR) of 1.06%, and a false-negative rate (FNR) of 0.21% in detecting Cluster 0 samples in a one-vs-all manner. Similarly, Figure 4b shows a

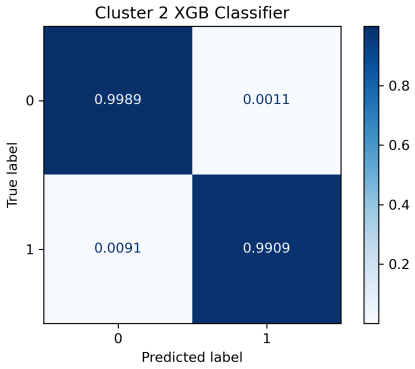
FPR of 0.08%, and a FNR of 0.87% in detecting Cluster 1 samples, while Figure 4c shows 0.11% and 0.91%, in FPR, and FNR, respectively, in detecting Cluster 2 samples.



(**a**) Cluster 0 detection



(**b**) Cluster 1 detection



(**c**) Cluster 2 detection

Figure 4. Confusion matrix plots for the XGB classifiers.

7. SHAP Analysis of Clusters

SHAP values were calculated for each of the one-vs-all classifiers to identify the most effective features that leads the classifier to detecting each cluster. At this stage, each of the three classifiers was trained to detect one cluster only, in a one-vs-all manner. The inference output will be 1 if the sample is a member in that cluster, or 0 if the sample is not a member of that particular cluster. Explainability calculations helps unveil insights into the distinct behavioral features that differentiate each cluster of riskware families from other clusters. Such analysis help us understand the specific distinctive behaviors of families within these clusters.

Figures 5–7 show the summary SHAP plots for each of the clusters' detection classifiers. In each summary plot, the five features with the highest impact are shown. These are the top features that help a classifier detect this specific cluster. Each dot resembles a sample. When the dot is clue, the feature's value is on the lower side, while the red dot shows a higher value of the feature. Dots on the right side of the axis show the values of the feature that push the prediction towards "1" (i.e. being a member of that particular cluster). The dots on the left side of the axis show the values of the feature that push the prediction to "0" (i.e. not a member of this cluster). The features names are mentioned as numbers in these plots because the names of these features are very long and was not possible to include in the figures. Therefore, we show the names of these features, in addition to their SHAP feature importance in Table 5. It is worth mentioning that in cluster 0 SHAP feature importance there were 66 features with 0 importance, and 94, and 82 features with 0 importance in clusters 1, and 2, respectively. Features with 0 importance indicate that these features did not have a significant impact on the classifiers' decisions. These relatively high numbers of features with zero importance is an important

indicator of riskware family behavior. It indicates that only a small number of features actually capture the distinctive behaviors of each cluster, while a large number of features capture the commonalities between different families behaviors.

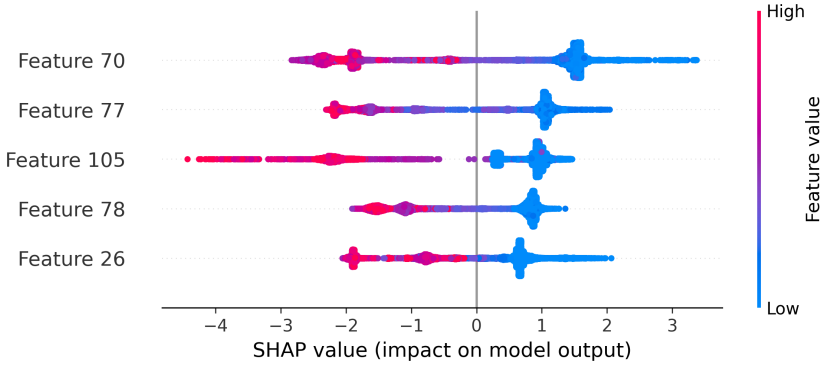


Figure 5. SHAP summary plot for Cluster 0.

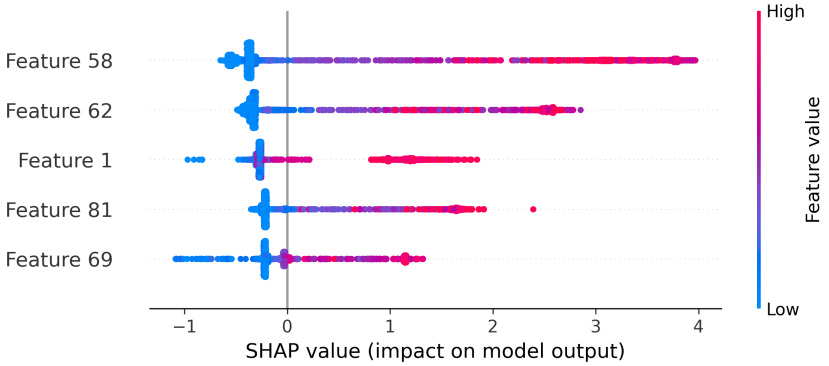


Figure 6. SHAP summary plot for Cluster 1.

Table 5. Features with highest SHAP feature importance in each cluster.

	Feature Number	Feature Name	Feature Importance
Cluster 0	70	API_Binder_android.app.ContextImpl_registerReceiver	1.660
	77	API_DeviceInfo_android.telephony.TelephonyManager_getDeviceId	1.303
	105	API_DexClassLoader_dalvik.system.DexClassLoader_$init	1.125
	78	API_DeviceInfo_android.telephony.TelephonyManager_getSubscriberId	0.972
	26	API_Command_java.lang.Runtime_exec	0.886
Cluster 1	58	API_Database_android.database.sqlite.SQLiteDatabase_rawQuery	0.924
	62	API_Database_android.database.sqlite.SQLiteDatabase_compileStatement	0.733
	1	Memory_PssClean	0.478
	81	API_DeviceInfo_android.telephony.TelephonyManager _getNetworkOperatorName	0.464
	69	API_IPC_android.content.ContextWrapper_registerReceiver	0.323
Cluster 2	105	API_DexClassLoader_dalvik.system.DexClassLoader_$init	2.464
	72	API_Binder_android.app.Activity_startActivity	1.075
	106	API_Base64_android.util.Base64_decode	1.050
	8	Memory_HeapAlloc	0.661
	77	API_DeviceInfo_android.telephony.TelephonyManager_getDeviceId	0.635

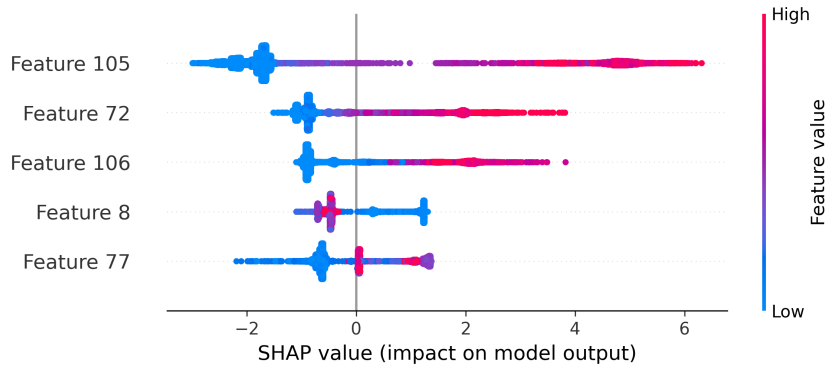


Figure 7. SHAP summary plot for Cluster 2.

In Figure 5, the top five features in terms of impact on the classifier's decision are shown. These features are the ones with the highest impact in distinguishing whether a sample belongs to cluster 0 or not. The feature with the highest impact on the classifier trained to detect cluster 0 is `API_Binder_android.app.ContextImpl_registerReceiver`. As shown in the figure, lower values of this feature are associated with the sample being identified as cluster 0 sample. This feature records the number of API calls to register an app as a receiver. Such API calls are considered normal behavior in many applications. However, not performing this API call is a common behavior of cluster 0 riskware. Figure 8 shows the SHAP values of a single sample that contains the highest SHAP value for feature 70. The waterfall plot shows that a value of 0 has a significant impact in pushing the prediction decision of the classifier to 1. This means that the families of riskware belonging to Cluster 0 do not employ such API calls to register as a receiver. The other four features in the top five display similar impact to feature 70, as shown in Figure 5. Lower values of these features cause the classifier to produce a prediction of 1, which means that the sample is a member of cluster 0, with the exception of Feature 77. As shown, Feature 77, which indicates the number of API calls to get the unique device ID, impacts the decision by pulling the compass towards 0 class (not belonging to cluster 0). It is noticeable that all the top five features in cluster 0 classification are API calls related. Features 77, in addition to feature 78 are particularly interesting because they refer to API calls that are used to uniquely identify the device or subscriber. These API calls are quite common in normal behaving apps that provide advertisements, such as free games monetized by advertisements. Such apps, will call for the unique identifiers every time an advertisement is shown in an attempt to personalize these advertisements. On the contrary, riskware apps in cluster 0 access this information less frequently because they might either store the information, or pass it to the malicious attacker quickly so, it doesn't need this pattern of repeated access. The fifth feature in this class, feature 26, is a feature that captures the number of times the app runs a shell command at run time. Figure 5 shows that cluster 0 samples tend to use such call at a lower number than other clusters, or normal apps. The reason behind that is that such a call would require for the app to be using a signed-APK file, that exists within the system folder. Therefore, higher number of such calls are usually made by legitimate, or system apps only. For that reason, we see that in Figure 8, feature 62 is pushing the prediction to a lower value for trying to make such calls 36 times.

In SHAP summary plot for Cluster 1 detection classifier shown in Figure 6, the feature with the highest impact in detecting cluster 1 is `API_Database_android.database.sqlite.SQLiteDatabase_rawQuery`. The figure shows that higher values of this feature are associated with predicting the sample as a member of cluster 1. This feature captures the number of API calls to make raw queries in an SQLite database. This generally means that the most distinctive feature of the behavior of a Cluster 1 riskware family is that it makes a significant number of calls to the SQLite database. Figure 9 shows the SHAP values of the sample where this feature had the highest impact on the classification decision. As shown in the figure, 100 raw queries

were performed by the app captured in this sample. While normal behaving apps can make such high number of queries, it is often associated with malicious behavior, and data exfiltration. As shown in Figure 9, the high value of this feature alone has an impact that is almost equivalent to half of the total SHAP value of that sample. While the absolute SHAP feature importance value, from Table 5, shows only 0.924, it is considered a high value out of 48 features that had SHAP importance more than 0. The second feature in terms of importance was also a database-related one. This indicates that riskware families in cluster 1 are more adamant on accessing, and possibly exfiltrating data.

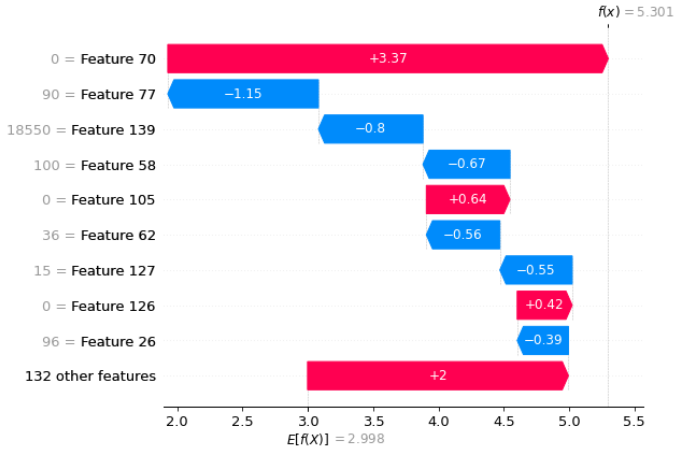


Figure 8. SHAP values for a single sample with the highest SHAP value in feature 70 in Cluster 0.

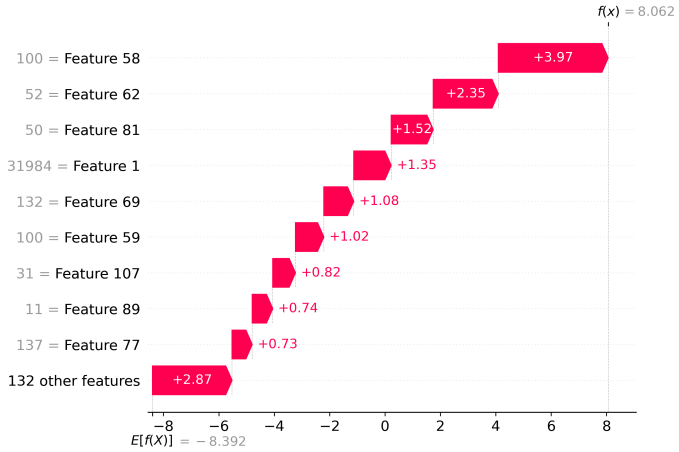


Figure 9. SHAP values for a single sample with the highest SHAP value in feature 58 in Cluster 1.

Figure 7 shows that the feature with the highest impact on the classification of cluster 2 is `API_DexClassLoader_dalvik.system.DexClassLoader_$init`. The figure shows that higher values of this feature are associated with the sample being classified as a cluster 2 sample. This feature captures the number of times an API call is made to dynamically load code from other apps or locations. This use enables malicious actors to decrypt malicious code that was previously encrypted either as part of the riskware app or downloaded later. While this type of API calls has legitimate use cases such as enabling apps to trigger events in a specific place in the code of another app (such as sharing a link from your browser to be posted on a social media app), it can easily be misused by malicious actors. This behavior was recognized by [58] as malicious behavior associated with different malware types. Figure 10 shows the cluster 2 sample where this feature had the highest impact. As shown, this sample makes 30 such calls in one run, which is a good indicator of malicious activity. The feature with second highest SHAP importance for cluster 2, as shown in Table 5 was feature 72. In terms of behavior this feature is relevant to the first one as it

measures the number of "startActivity" API calls made to start an activity within the app, or call an activity from another app. Such action is not common in benign applications. Figure 10 shows that this sample made 42 such API calls. The combination of the explanation of these features points to the fact that riskware families in cluster 2 are more likely to behave as "droppers" that can download and run other types of malware. This can be exploited by a malicious actor to download and run ransomware, and remote access trojans. Such behavior is quite distinct from the behavior of the previous two clusters.

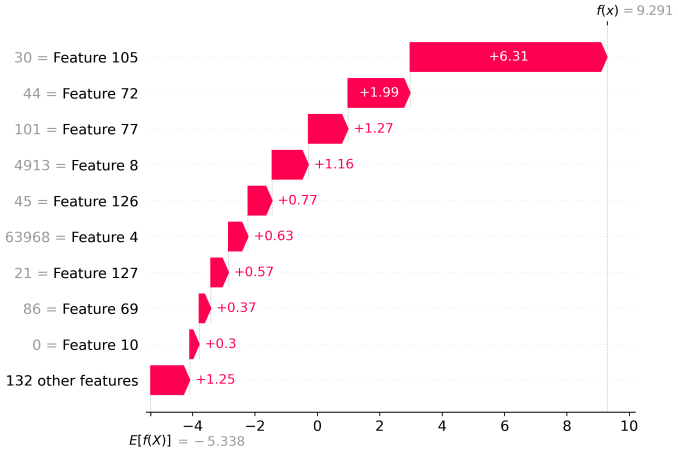


Figure 10. SHAP values for a single sample with the highest SHAP value in feature 105 in Cluster 2.

Another noticeable observation is feature number 77 appearing in cluster 2's top five, in addition to cluster 0's. However, the behavior is different between the two. Higher values of feature 77 makes the behavior closer to cluster 0, while lower values of it makes the behavior closer to cluster 2. This is an indicator that the different intentions of different riskware families lead to different behaviors, which can be captured by analyzing the explainability of the detection models. A similar noticeable observation is the existence of feature 105 in both clusters 0 and 2, with contradicting values as well. In cluster 0, feature 105 held lower values, while it held higher values in cluster 2, making it the most impactful feature in detecting cluster 2.

8. Discussions

In this study, a wide range of machine learning techniques were applied to analyze the behavior of riskware apps in the Android ecosystem. K-Means clustering technique was first applied to group families of riskware into three clusters based on dynamic features capturing their behavior. Then, these clusters were used to build one-vs-all XGB-based classifiers to detect samples from each cluster individually. These classifiers were used to calculate SHAP values to provide detailed explainability of the detection decisions. The explainability was used to analyze the distinguishing features that help in identifying the behaviors of these riskware families.

By examining Table 5, we can see that each cluster of riskware families has distinctive behaviors captured by a varying range of features. As we examine the table, we finds that the most distinctive characteristic is that apps within these families do not make API calls to register the app as a receiver. On the other hand, families falling within Cluster 1 performs a significantly higher number of class to SQLite databases, which makes them a good candidate for data exfiltration operations, and unauthorized access to private information. Families falling within Cluster 2 make significant number of API calls to load code from other apps. This makes it a suitable candidate to download other types of malware, such as ransomware and trojans.

This type of analysis enables researchers to have a deeper understanding of how different riskware apps operate, and what their intents might be. Explainability, in this context, provides deep insights into behaviors that make these riskware families so distinct,

and what these malicious applications are trying to achieve. SHAP values generated helped us identify the most prevalent dynamic features that can help in identifying these clusters of families. In addition, it helps in understanding exactly how these features impact the classification decision. It also helps in identifying features that are not effective, or do not represent distinct behaviors of these riskware families.

The results obtained showed that different groups of riskware families have distinct behaviors based on the intent of the malicious actor behind them. Families such as `skymobi`, and some samples of `tencentprotect` showed behaviors that are closer to dropper malware behaviors that is usually utilized by malicious actors to drop other types of malware. On the other hand, families such as `revmob`, and `kingroot` showed behaviors that are comparably different from normal applications' behavior. This includes lower access to device and user identities, combined with lower class loading within the app.

To the best of our knowledge, no other study has analyzed riskware apps behavior in the way that was presented in our study. Thus, a comparative analysis was not possible.

9. Conclusions and Future Works

In this research we explored the distinct behavioral characteristics of different Android riskware families. We utilized a variety of machine learning techniques such as K-Means clustering, PCA, and SHAP to analyze the behavior of different riskware families. This analysis provided in-depth insights into the behaviors that distinguish different family groups. It can also help in improving the detection capabilities in capturing this often overlooked type of malicious software that can cause significant harm to users.

The findings of this research can benefit the research community in having a deeper understanding of riskware and its behavior in the Android ecosystem.

Future directions in this research can include the following:

1. Corroborate the findings of this study using a different dataset to validate the conclusions obtained.
2. Applying the analysis methodology to other types of malware to provide a deeper understanding of their behavior.
3. Apply feature selection methods to reduce the number of studied features to keep the focus on the most effective ones.
4. Exploring other explainable machine learning techniques and compare their outcome with the ones obtained by SHAP.

Author Contributions: Methodology, M.M.A. and M.A.; Software, M.M.A.; Formal analysis, M.M.A.; Writing—original draft, M.M.A. and M.A.; Writing—review & editing, M.A.; Project administration, M.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: Data is available upon request.

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

References

1. 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. (CSUR)* **2020**, *53*, 1–36. [CrossRef]
2. Mobile OS Market Share Worldwide 2009–2023|Statista. 2023. Available online: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009 (accessed on 26 December 2023).
3. Khan, M.A.; Ahmad, I.; Nordin, A.N.; Ahmed, A.E.S.; Mewada, H.; Daradkeh, Y.I.; Rasheed, S.; Eldin, E.T.; Shafiq, M. Smart android based home automation system using internet of things (IoT). *Sustainability* **2022**, *14*, 10717. [CrossRef]
4. Karbab, E.B.; Debbabi, M.; Derhab, A.; Mouheb, D. MalDozer: Automatic framework for android malware detection using deep learning. *Digit. Investig.* **2018**, *24*, S48–S59. [CrossRef]
5. Perwej, Y.; Abbas, S.Q.; Dixit, J.P.; Akhtar, N.; Jaiswal, A.K. A systematic literature review on the cyber security. *Int. J. Sci. Res. Manag.* **2021**, *9*, 669–710. [CrossRef]
6. Alani, M.M. Android Users Privacy Awareness Survey. *Int. J. Interact. Mob. Technol.* **2017**, *11*, 130–144. [CrossRef]

7. Sokolova, K.; Perez, C.; Lemercier, M. Android application classification and anomaly detection with graph-based permission patterns. *Decis. Support Syst.* **2017**, *93*, 62–76. [CrossRef]
8. Yang, M.; Wang, S.; Ling, Z.; Liu, Y.; Ni, Z. Detection of malicious behavior in android apps through API calls and permission uses analysis. *Concurr. Comput. Pract. Exp.* **2017**, *29*, e4172. [CrossRef]
9. Cai, H.; Ryder, B. A longitudinal study of application structure and behaviors in android. *IEEE Trans. Softw. Eng.* **2020**, *47*, 2934–2955. [CrossRef]
10. Yuan, Z.; Lu, Y.; Xue, Y. Droiddetector: Android malware characterization and detection using deep learning. *Tsinghua Sci. Technol.* **2016**, *21*, 114–123. [CrossRef]
11. Alani, M.M.; Awad, A.I. AdStop: Efficient flow-based mobile adware detection using machine learning. *Comput. Secur.* **2022**, *117*, 102718. [CrossRef]
12. Chen, J.; Wang, C.; Zhao, Z.; Chen, K.; Du, R.; Ahn, G.J. Uncovering the face of android ransomware: Characterization and real-time detection. *IEEE Trans. Inf. Forensics Secur.* **2017**, *13*, 1286–1300. [CrossRef]
13. Faruki, P.; Bhan, R.; Jain, V.; Bhatia, S.; El Madhoun, N.; Pamula, R. A Survey and Evaluation of Android-Based Malware Evasion Techniques and Detection Frameworks. *Information* **2023**, *14*, 374. [CrossRef]
14. Sk, H.K. A literature review on android mobile malware detection using machine learning techniques. In Proceedings of the 2022 6th International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 29–31 March 2022; pp. 986–991.
15. Kwon, H.Y.; Kim, T.; Lee, M.K. Advanced intrusion detection combining signature-based and behavior-based detection methods. *Electronics* **2022**, *11*, 867. [CrossRef]
16. Alani, M.M.; Awad, A.I. Paired: An explainable lightweight android malware detection system. *IEEE Access* **2022**, *10*, 73214–73228. [CrossRef]
17. Feng, P.; Ma, J.; Sun, C.; Xu, X.; Ma, Y. A novel dynamic android malware detection system with ensemble learning. *IEEE Access* **2018**, *6*, 30996–31011. [CrossRef]
18. Choudhary, M.; Kishore, B. Haamd: Hybrid analysis for android malware detection. In Proceedings of the 2018 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 4–6 January 2018; pp. 1–4.
19. Razgallah, A.; Khoury, R.; Hallé, S.; Khanmohammadi, K. A survey of malware detection in Android apps: Recommendations and perspectives for future research. *Comput. Sci. Rev.* **2021**, *39*, 100358. [CrossRef]
20. Bayazit, E.C.; Sahingoz, O.K.; Dogan, B. Protecting Android Devices from Malware Attacks: A State-of-the-Art Report of Concepts, Modern Learning Models and Challenges. *IEEE Access* **2023**, *11*, 123314–123334. [CrossRef]
21. Jiang, C.; Xia, C.; Liu, Z.; Wang, T. FedDroidMeter: A Privacy Risk Evaluator for FL-Based Android Malware Classification Systems. *Entropy* **2023**, *25*, 1053. [CrossRef]
22. Dimitrios, T. Privacy and Data Protection in Mobile Applications. Master's Thesis, International Hellenic University, Thermi, Greece, 2022.
23. Jennings, A. Surveillance by Software: A Code for Employee Monitoring. Ph.D. Thesis, ResearchSpace, Auckland, New Zealand, 2023.
24. He, D.; Chan, S.; Guizani, M. Mobile application security: Malware threats and defenses. *IEEE Wirel. Commun.* **2015**, *22*, 138–144. [CrossRef]
25. Dahri, K.A.; Vighio, M.S.; Zardari, B.A. Detection and prevention of malware in android operating system. *Mehran Univ. Res. J. Eng. Technol.* **2021**, *40*, 847–859. [CrossRef]
26. Rani, S.; Dhindsa, K.S. Android application security: Detecting Android malware and evaluating anti-malware software. *Int. J. Internet Technol. Secur. Trans.* **2020**, *10*, 491–506. [CrossRef]
27. Rani, S.; Dhindsa, K.S. Android Malware Detection in Official and Third Party Application Stores. *Int. J. Adv. Netw. Appl.* **2018**, *9*, 3506–3509.
28. Apvrille, A.; Strazzere, T. Reducing the window of opportunity for Android malware Gotta catch'em all. *J. Comput. Virol.* **2012**, *8*, 61–71. [CrossRef]
29. Arshad, S.; Shah, M.A.; Khan, A.; Ahmed, M. Android malware detection & protection: A survey. *Int. J. Adv. Comput. Sci. Appl.* **2016**, *7*, 463–475.
30. Usama, M.; Qadir, J.; Raza, A.; Arif, H.; Yau, K.L.A.; Elkhatib, Y.; Hussain, A.; Al-Fuqaha, A. Unsupervised machine learning for networking: Techniques, applications and research challenges. *IEEE Access* **2019**, *7*, 65579–65615. [CrossRef]
31. Ding, C.; He, X. K-means clustering via principal component analysis. In Proceedings of the Twenty-First International Conference on Machine Learning, Banff, AB, Canada, 4–8 July 2004; p. 29.
32. Murtagh, F.; Contreras, P. Algorithms for hierarchical clustering: An overview. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **2012**, *2*, 86–97. [CrossRef]
33. Bouman, C.A.; Shapiro, M.; Cook, G.; Atkins, C.B.; Cheng, H. *Cluster: An Unsupervised Algorithm for Modeling Gaussian Mixtures*; The Board of Trustees of Purdue University: West Lafayette, IN, USA, 1997.
34. Cariou, C.; Le Moan, S.; Chehdi, K. A novel mean-shift algorithm for data clustering. *IEEE Access* **2022**, *10*, 14575–14585. [CrossRef]
35. Roscher, R.; Bohn, B.; Duarte, M.F.; Garcke, J. Explainable Machine Learning for Scientific Insights and Discoveries. *IEEE Access* **2020**, *8*, 42200–42216. [CrossRef]

36. Adadi, A.; Berrada, M. Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). *IEEE Access* **2018**, *6*, 52138–52160. [CrossRef]
37. Huysmans, J.; Dejaeger, K.; Mues, C.; Vanthienen, J.; Baesens, B. An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decis. Support Syst.* **2011**, *51*, 141–154. [CrossRef]
38. Letham, B.; Rudin, C.; McCormick, T.H.; Madigan, D. Interpretable classifiers using rules and Bayesian analysis: Building a better stroke prediction model. *Ann. Appl. Stat.* **2015**, *9*, 1350–1371. [CrossRef]
39. Ribeiro, M.T.; Singh, S.; Guestrin, C. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *arXiv* **2016**, arXiv:1602.04938.
40. Nguyen, A.; Dosovitskiy, A.; Yosinski, J.; Brox, T.; Clune, J. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. *Adv. Neural Inf. Process. Syst.* **2016**, *29*, 3395–3403.
41. Lundberg, S.M.; Lee, S.I. A unified approach to interpreting model predictions. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 4765–4774.
42. Štrumbelj, E.; Kononenko, I. Explaining prediction models and individual predictions with feature contributions. *Knowl. Inf. Syst.* **2014**, *41*, 647–665. [CrossRef]
43. Kouliaridis, V.; Kambourakis, G. A comprehensive survey on machine learning techniques for android malware detection. *Information* **2021**, *12*, 185. [CrossRef]
44. Galal, H.S.; Mahdy, Y.B.; Atiea, M.A. Behavior-based features model for malware detection. *J. Comput. Virol. Hacking Tech.* **2016**, *12*, 59–67. [CrossRef]
45. Manzil, H.H.R.; Naik, S.M. Detection approaches for android malware: Taxonomy and review analysis. *Expert Syst. Appl.* **2023**, *238*, 122255. [CrossRef]
46. Felt, A.P.; Greenwood, K.; Wagner, D. *The Effectiveness of Install-Time Permission Systems for Third-Party Applications*; Technical Report for University of California at Berkely, Electrical Engineering and Computer Sciences; University of California at Berkely: Berkeley, CA, USA, 2010.
47. Sandor, A.; Simion, E. An Entropy-based Method for Social Apps Privacy Assessment Using the Android Permissions Architecture. *Adv. Electr. Comput. Eng.* **2022**, *22*, 79–86. [CrossRef]
48. Xiao, J.; Chen, S.; He, Q.; Feng, Z.; Xue, X. An Android application risk evaluation framework based on minimum permission set identification. *J. Syst. Softw.* **2020**, *163*, 110533. [CrossRef]
49. Khullar, V.; Gera, T.; Mehta, T. Static Method to Locate Risky Features in Android Applications. In Proceedings of the 2023 2nd Edition of IEEE Delhi Section Flagship Conference (DELCON), Rajpura, India, 24–26 February 2023; pp. 1–6.
50. Bhat, P.; Behal, S.; Dutta, K. A system call-based android malware detection approach with homogeneous & heterogeneous ensemble machine learning. *Comput. Secur.* **2023**, *130*, 103277.
51. Yang, H.; Wang, Y.; Zhang, L.; Cheng, X.; Hu, Z. A Novel Android Malware Detection Method with API Semantics Extraction. *Comput. Secur.* **2023**, *137*, 103651. [CrossRef]
52. Liu, T.; Zhang, H.; Long, H.; Shi, J.; Yao, Y. Convolution neural network with batch normalization and inception-residual modules for Android malware classification. *Sci. Rep.* **2022**, *12*, 13996. [CrossRef] [PubMed]
53. Mariconti, E.; Onwuzurike, L.; Andriotis, P.; De Cristofaro, E.; Ross, G.; Stringhini, G. Mamadroid: Detecting android malware by building markov chains of behavioral models. *arXiv* **2016**, arXiv:1612.04433.
54. Rahali, A.; Lashkari, A.H.; Kaur, G.; Taheri, L.; Gagnon, F.; Massicotte, F. Didroid: Android malware classification and characterization using deep image learning. In Proceedings of the 2020 The 10th International Conference on Communication and Network Security, Xi'an, China, 21–23 August 2020; pp. 70–82.
55. AndMal 2020|Datasets|Research|Canadian Institute for Cybersecurity|UNB. 2023. Available online: https://www.unb.ca/cic/datasets/andmal2020.html (accessed on 26 December 2023).
56. Gan, G.; Ma, C.; Wu, J. *Data Clustering: Theory, Algorithms, and Applications*; SIAM: Philadelphia, PA, USA, 2020.
57. Kaur, G.; Lashkari, A.H. Understanding Android Malware Families: Riskware—Is It Worth It? (Article 4)—IT World Canada. 2021. Available online: https://www.itworldcanada.com/blog/understanding-android-malware-families-riskware-is-it-worth-it-article-4/446692 (accessed on 28 December 2023).
58. Schütte, J.; Fedler, R.; Titze, D. Condroid: Targeted dynamic analysis of android applications. In Proceedings of the 2015 IEEE 29th International Conference on Advanced Information Networking and Applications, Gwangiu, Republic of Korea, 24–27 March 2015; pp. 571–578.