Forensic Science International: Digital Investigation 50 (2024) 301816

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

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

DFRWS APAC 2024 - Selected Papers from the 4th Annual Digital Forensics Research Conference APAC

GBKPA and AuxShield: Addressing adversarial robustness and transferability in android malware detection

Kumarakrishna Valeti, Hemant Rathore *

Dept. of CS & IS, Goa Campus, BITS Pilani, India

ARTICLE INFO

Keywords:
Android
Deep learning
Malware
Machine learning
Robustness
Transferability

ABSTRACT

Android stands as the predominant operating system within the mobile ecosystem. Users can download applications from official sources like *Google Play Store* and other third-party platforms. However, malicious actors can attempt to compromise user device integrity through malicious applications. Traditionally, signatures, rules, and other methods have been employed to detect malware attacks and protect device integrity. However, the growing number and complexity of malicious applications have prompted the exploration of newer techniques like machine learning (ML) and deep learning (DL). Many recent studies have demonstrated promising results in detecting malicious applications using ML and DL solutions. However, research in other fields, such as computer vision, has shown that ML and DL solutions are vulnerable to targeted adversarial attacks. Malicious actors can develop malicious adversarial applications that can bypass ML and DL based anti-viruses. The study of adversarial techniques related to malware detection has now captured the security community's attention. In this work, we utilise android permissions and intents to construct 28 distinct malware detection models using 14 classification algorithms. Later, we introduce a novel targeted false-negative evasion attack, *Gradient Based K Perturbation Attack (GBKPA)*, designed for grey-box knowledge scenarios to assess the robustness of these models. The GBKPA attempts to craft malicious adversarial samples by making minimal perturbations without violating the syntactic and functional structure of the application. GBKPA achieved an average fooling rate (FR) of 77 % with only five perturbations across the 28 detection models. Additionally, we identified the most vulnerable android permissions and intents that malicious actors can exploit for evasion attacks. Furthermore, we analyse the transferability of adversarial samples across different classes of models and provide explanations for the same. Finally, we proposed *AuxShield* defence mechanism to develop robust detection models. AuxShield reduced the average FR to 3.25 % against 28 detection models. Our findings underscore the need to understand the causation of adversarial samples, their transferability, and robust defence strategies before deploying ML and DL solutions in the real world.

1. Introduction

Android (2024) stands as the predominant operating system within the mobile ecosystem with a market share of more than 70 %. One of the key attractions is the availability of the numerous and diverse applications in the android ecosystem. Unfortunately, malicious actors create malicious applications to gain financial benefits through data theft, ransomware, etc. Kaspersky (2024) recently reported blocking 33.8 million malware attacks in 2023. Google Play Protect (2024) is designed to secure the android ecosystem from potentially harmful applications. However, another report by Kaspersky (2023) suggests that *Google* recently removed 43 malicious applications with 2.5 + million downloads from its *Play Store*. Many users often employ anti-virus software to safeguard their devices. These anti-virus heavily rely on signatures, rules and other methods to identify potentially harmful applications. Despite these measures, android devices remain vulnerable to modern new-age malware threats.

Recent research suggests that machine learning (ML) and deep learning (DL) techniques in malware detection systems can offer a solution against new-age malware attacks. Arp et al. (2014) utilised static features, such as permissions, APIs and intents extracted from android applications and developed a security system using support vector machines surpassing the performance of several antivirus solutions. Rathore et al. (2020) extracted opcode frequencies from android

* Corresponding author.
E-mail address: hemantr@goa.bits-pilani.ac.in (H. Rathore).

https://doi.org/10.1016/j.fsidi.2024.301816

Available online 18 October 2024
2666-2817/© 2024 The Author(s). Published by Elsevier Ltd on behalf of DFRWS. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

applications using static analysis. They proposed malware detection by combining clustering and classification models. Zhu et al. (2020) proposed the *SEDMDroid* framework, which combines principal component analysis, multi-layer perceptrons, and support vector machines to attain an accuracy of 94.92 %. Zhang et al. (2023) introduced the *Tsdroid* framework, leveraging temporal and spatial metrics within the context of the IoMT for malware detection. ML and DL models have become a viable option with increasing data availability in the security domain. They can be integrated with existing methods to form a superior security system.

Recent studies in fields like computer vision and natural language processing have shown that machine learning and deep learning solutions can be deceived by making small perturbations to the input data. These are called *adversarial attacks* and are designed to induce misclassification in a classification system by introducing *adversarial samples* in the system. Many research works like Goodfellow et al. (2014), Ilyas et al. (2019), Zhang et al. (2020) and Pal et al. (2024) have attempted to explain the existence of adversarial samples due to high-dimensional geometry of models, adversarial spheres, limitations of models, test error, and the presence of noise. Papernot et al. (2016) showed that adversarial samples sometimes exhibit the property of *transferability*. This means that adversarial samples that deceive one classifier often tend to deceive other classifiers and generalize effectively across different classes of models. These adversarial attacks can drastically decrease the performance of classification systems, sometimes making them unusable in the real world.

Adversarial attacks are of two primary forms: *evasion attack* and *poisoning attack*. *Evasion attack* involves the attacker adversarially perturbing the input samples with the aim to increase the rate of *false-negative* and/or *false-positive*. In a *poisoning attack*, the adversary tampers with the training data with the aim to skew the classification model itself. These attacks can be either *targeted* or *indiscriminate*. In a *targeted attack*, the adversary focuses on a specific sample and/or class in a system. On the other hand, in an *indiscriminate attack*, the adversary targets a general set of samples/classes in a system. Depending on the adversary's knowledge and capabilities concerning the target system, adversarial attacks unfold in three scenarios: white-box, grey-box, and black-box. In a white-box scenario, the adversary possesses complete knowledge of the dataset, feature vector, and classifier used by the target system. In a grey-box scenario the adversary has partial knowledge of some of these parameters. Finally, in a black-box scenario, the attacker lacks any information about the target system's parameters. Most of the existing literature concentrates on white-box scenarios, which may not fully mirror real-world conditions.

This work aims to develop adversarially robust ML and DL models for effectively identifying malicious android applications. We first created a balanced dataset of malicious applications (*AMD dataset*) and benign applications (*Google Play Store* verified by *Virus-total*). We then created 28 distinct models for malware detection using four classes of algorithms (machine learning, deep neural network (DNN), bagging, and boosting) and two kinds of features. We performed exhaustive feature analysis to identify relevant features for detection and identified hierarchical relationships between malware families. Subsequently, we assumed an adversary's role to investigate the robustness of the above models and formulated a novel *targeted false-negative evasion attack* titled *Gradient Based K Perturbation Attack (GBKPA)*. We design this attack for a grey-box knowledge scenario, assuming knowledge of the dataset and feature vector but not of the model architecture. GBKPA attempts to craft malicious adversarial samples by making minimal perturbations without violating the syntactic and functional structure of the application. We assess the robustness of various classification algorithms against GBKPA and introduce the transferability score metric (ranging from 0 to 1) to measure the transferability of adversarial samples. We analyse their transferability and provide an explanation using the similarities and differences in feature usage across different model classes. Finally, we implement our defence mechanism *AuxShield* to create robust malware detection models that can effectively guard against adversarial attacks. The major contributions of our study are as follows:

- **Feature Usage and Malware Family Analysis**: We build 28 distinct malware detection models using four classes of classification algorithms and two features. We identified the top features (android permissions and intents) utilised by different detection models. Additionally, we provide hierarchical relationships of 10 malware families based on feature space similarities.
- **GBKPA Adversarial Attack**: We introduce *Gradient Based K Perturbation Attack (GBKPA)*, a targeted false-negative evasion attack that generates adversarial malware applications for a grey-box knowledge scenario. It accomplished an average fooling rate of 77 %, making at most five perturbations against 28 malware detection models. We also provide a list of the android permissions and intents that are most susceptible to exploitation by malicious actors in the design of adversarial malware applications.
- **Transferability**: We investigated the transferability of adversarial samples by examining the similarity in feature usage among different malware detection models. Our results indicate that the higher transferability score of 1.0 between DNNs, compared to an average score of 0.88 between DNNs and tree-based models, can be attributed to the greater similarity in feature usage among DNNs.
- **Adversarial Defence**: We proposed a defence strategy *AuxShield*, a two-level hierarchical security mechanism based on an auxiliary model and adversarial retraining. It reduced the average fooling rate from 77 % to 3.25 % in 28 malware detection models.

The subsequent sections of this paper are as follows: Section 2 presents the framework, providing insights into its background, problem definition, GBKPA evasion attack, transferability, and AuxShield defence strategy. Section 3 outlines the experimental setup, while Section 4 delves into the experimental findings. Section 5 discusses existing literature, offering a comparative analysis with existing works. Finally, Section 6 offers concluding remarks and outlines the future scope of this study.

2. Overview and framework

This section covers the framework for creating robust malware detection models, the problem definition, the proposed evasion attack GBKPA, transferability, and the defence mechanism AuxShield.

2.1. Framework

Fig. 1 outlines the framework to build robust malware detection models with GBKPA, transferability analysis, and AuxShield. Step-1 involved crafting balanced datasets with benign and malicious android applications. In Step-2, we extracted two static features (permissions and intents) for each sample in the dataset and built 28 classification models. In Step-3, we analysed the important features of each of the 28 models and performed malware family analysis. In Step-4, we performed the GBKPA attack using a deep neural network as the surrogate model. After crafting the adversarial samples, we evaluated the fooling rate of each classification model. In Step-5, we analysed the transferability of adversarial samples against different detection models. Finally, in Step-6, we deploy the adversarial defence mechanism AuxShield to fortify the malware detection model against adversarial attacks.

2.2. Problem definition and background

Consider a balanced dataset containing both benign and malicious android applications. The dataset D having n android applications X, will be represented using m features and class label Y as follows:

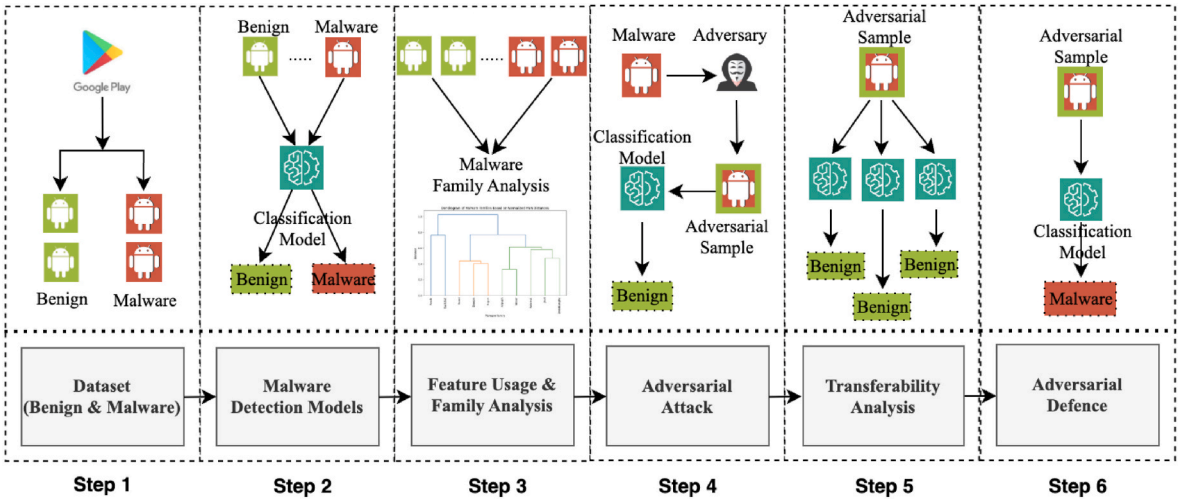


Fig. 1. Framework for building robust malware detection models with GBKPA, transferability analysis and AuxShield

$$D = \{(X_i, Y_i) | i \in 1 \ldots n\}$$
$$\text{where } X_i = \{x_i^1, \ldots, x_i^m\} \quad and \quad Y_i \in \{0, 1\} \tag{1}$$

A classification model aims to learn a function F: $X \rightarrow Y$ such that $F(X_{\text{benign}}) = 0$ and $F(X_{\text{mal}}) = 1$. A classification model searches for optimal hyperparameters to find the best function to accurately predict the true class label.

The adversary (malware designer) can design an adversarial attack with the aim to cause misclassifications in the classification model (F) by making perturbations (δ) in the applications.

$$X_{i_{\text{adv}}} = X_i + \delta \quad \text{such that} \quad F(X_i) \neq F(X_{i_{\text{adv}}}) \tag{2}$$

An adversarial defence strategy aims to create a robust model F' such that $F'(X_i) = F'(X_{i_{\text{adv}}})$ effectively safeguarding against attackers with malicious intent.

2.3. Gradient Based K Perturbation Attack

We introduce a novel *targeted false-negative evasion attack* titled *Gradient Based K Perturbation Attack (GBKPA)* for a grey box knowledge scenario. We assume the attacker has knowledge/access to the dataset and feature vectors but lacks knowledge of the model architecture, and aims to perturb a correctly classified malware sample m to create an adversarial sample m' that can evade detection by the ML/DL model. GBKPA is designed to minimize perturbations without violating the syntactic and functional structure of the application.

The adversary begins by constructing a *surrogate deep neural network* with a binary cross-entropy loss function using the dataset D. The DNN adjusts its parameters to minimise the loss function as defined by:

$$Loss = -y^* log(p) - (1 - y)^* log(1 - p) \tag{3}$$

In this equation, y denotes the class label, while p represents the predicted probability of the sample belonging to the *malware* class.

Given a malicious sample m, the attacker aims to maximize the loss function $L(x, y)$ by iteratively perturbing the features with the highest impact on the loss. A larger loss value indicates a reduction in the efficacy of the classification model. Algorithm 1 provides the GBKPA pseudocode for generating adversarial samples. The steps involved are as follows:

1. **Compute Partial Derivatives**: Compute the partial derivatives of the loss function $L(x, y)$ for features where $x_i = 0$:

$$g_i = \frac{\partial L}{\partial x_i} \tag{4}$$

2. **Select and Perturb Features**: Among the features with $x_i = 0$, select the feature with the largest g_i. Set this feature to 1. Perturbing the feature with the largest partial derivative will result in the most substantial increase in the loss function. By only adding to the pre-existing permissions and intents, we ensure the permissions and intents required for the functioning and malicious nature of the android application are unaffected, thereby ensuring application integrity.
3. **Iterate until Misclassification or Limit Reached**: Repeat this process K times or until the sample is misclassified by the grey-box model.

We used a DNN with 5 hidden layers of 512, 256, 128, 64, and 32 neurons as the surrogate model.

2.4. Transferability

Adversarial samples designed to evade one model may evade others due to shared vulnerabilities in learned features. This phenomenon is known as *transferability*.

For a given sample $\mathbf{m}$, if we have a grey-box model F_1, we can build a surrogate model F_2 and use that to craft an adversarial sample $\mathbf{m}'$. Szegedy et al. (2013) showed that due to transferability, it is likely that:

$$F_1(\mathbf{m}) \neq F_1(\mathbf{m}') \quad \text{since} \quad F_2(\mathbf{m}) \neq F_2(\mathbf{m}')$$

Thus, the adversarial sample $\mathbf{m}'$, designed to fool the surrogate model F_2, will also successfully fool the original model F_1. This effectively transforms a grey-box scenario, where the architecture details of model F_1 are unknown, into a semi white-box scenario, enabling us to generate adversarial samples for F_1 using the knowledge of F_2. Similarities in adversarial direction (Kurakin et al. (2018)) and feature usage (Ilyas et al. (2019)) leads to F_1 and F_2 being alike resulting in transferability:

$$F_1 \approx F_2 \tag{5}$$

Algorithm 1. Gradient Based K Perturbation Attack

```
Input:
M - Malicious samples
GM - Grey-box Malware Classifier
F - Feature List
K - Maximum Number of Perturbations
SM - The surrogate DNN built by the attacker
Output:
M' - Adversarial samples returned by GBKPA
Auxilliary Functions:
sort_descending - sorts the list in descending order
predict - returns the prediction of input model
loss_func - returns the binary cross entropy loss function
p_derive - returns the partial derivative
procedure GBKPA(M, GM, F, K)
    M' ← {}
    for m in M do
        m' ← m
        for j in range(K) do
            pred ← predict(GM, m')
            if pred == 1 then
                pd_list ← {}
                for i in range(len(F)) do
                    if m'[i] == 0 then
                        pd_list ← pd_list ∪
                            p_deriv(loss(SM, m), m[i], i)
                    end if
                end for
                sort_descending(pd_list)
                index ← pd_list[0].feature_index
                m'[index] ← 1
                M'.append(m')
            end if
        end for
    end for
    return M'
end procedure
```

2.5. Defence strategy - AuxShield

We propose a novel defence strategy *AuxShield* that employs a two-level hierarchical malware detection system to enhance security against adversarial and malicious inputs. The first level of the defence strategy is a classification model C_1 which is responsible for determining if the input sample x is adversarial. The second level C_2 is responsible for distinguishing between malicious and benign samples. Utilising two layers allows for the flexibility of selecting different model architectures for each level. Alternatively, a single multiclass classification model with an additional class for adversarial samples could be used as a defense strategy. If C_1 deems the input sample x to be adversarial ($C_1(x) =$ adversarial), the sample is then promptly marked as malicious:

$$C_1(x) = \begin{cases} \text{adversarial} & \rightarrow \text{malicious} \\ \text{non} - \text{adversarial} & \rightarrow \text{proceed to } C_2 \end{cases} \tag{6}$$

If C_1 determines the input to be non-adversarial, the second-level classification system C_2 is invoked. In this framework, C_1 is trained on a dataset D_1 consisting of adversarial samples (x_{adv}, y_{adv}), non-adversarial malicious samples (x_{mal}, $y_{\text{non-adv}}$), and benign samples (x_{ben}, $y_{\text{non-adv}}$).

$$C_2(x) = \begin{cases} \text{malicious} \\ \text{benign} \end{cases} \tag{7}$$

The classifier at the second level, C_2, undergoes *adversarial retraining*. This means it is trained on a balanced dataset D_2 containing adversarial samples x_{adv} labelled as malicious (y_{mal}), along with traditional benign (x_{ben}) and malicious (x_{mal}) applications:

$$\mathscr{D}_2 = \{(x_{\text{ben}}, y_{\text{ben}}), (x_{\text{mal}}, y_{\text{mal}}), (x_{\text{adv}}, y_{\text{mal}})\} \tag{8}$$

By exposing C_2 to adversarial samples during training, it becomes more robust to such inputs during real-world operations. AuxShield thus provides a dual-layered security mechanism: an auxiliary model (C_1) to detect and handle adversarial samples, and an adversarially retrained robust classifier (C_2) to accurately determine whether the input is malicious or benign.

3. Experimental setup

This section details the process of dataset creation, feature extraction, classification algorithms and evaluation metrics employed in the paper.

3.1. Data collection

Wei et al. (2017) from *Argus Lab* curated the *Android Malware Dataset (AMD)*, comprising 24, 650 malicious android applications from 135 distinct types spanning 71 malware families. These malicious applications threaten users' privacy and device security. We collected android applications from the *Google Play Store* and confirmed their benign status using *VirusTotal* to compile the benign dataset. Applications were labeled as *benign* only if they received a non-malicious classification from all 50 + antivirus scans provided by VirusTotal. The benign samples, acquired from the Google Play Store and verified through VirusTotal, were merged with the malware samples from the AMD dataset to create an extensive dataset for our experiments. This consolidated dataset consists of 24, 482 malicious android applications representing 71 malware families (after removing corrupted malicious applications), along with 25, 407 benign android applications.

3.2. Feature extraction

We extracted two kinds of features, namely *android permission* and *android intent*, using static analysis for each sample in the above dataset (refer to Rathore et al. (2021a, 2023a)). *Android Permissions* are access rights that an application must obtain before accessing certain device features in the android ecosystem. Similarly, *Android Intents* are messaging objects used to request action from another component of the android ecosystem. Using APKTool, we decompiled the android applications and parsed the *AndroidManifest.xml* file to extract *android permissions* and *android intents* utilised by the applications. Finally, we created two feature vectors: *android permission feature vector* with 195 features and *android intent feature vector* with 273 features.

3.3. Classification algorithms

Table 1 presents a list of 14 classification algorithms that have been used in this work. These algorithms are drawn from four distinct classes: machine learning, bagging-based methods, boosting-based methods,

Table 1
Classification algorithms.

Category	Classification Algorithm
Machine Learning	Logistic Regression (LR) Bernoulli Naive Bayes (NB) Decision Trees (DT)
Bagging	Decision Tree based Bagging (BG) Random Forest (RF) Extra Trees (EX)
Boosting	Gradient Boosting (GB) Adaptive Boosting (AB) eXtreme Gradient Boosting (XG)
Deep Neural Network (DNN)	DNN with 1 hidden layers (DNN1) DNN with 2 hidden layers (DNN2) DNN with 3 hidden layers (DNN3) DNN with 4 hidden layers (DNN4) DNN with 5 hidden layers (DNN5)

and deep neural networks.

3.4. Evaluation metrics

In our experiments, we have employed the following evaluation metrics. We designated malware samples as the *positive class* and benign samples as the *negative class*.

- *True Positive (TP)*: Malware samples correctly classified as the positive class.
- *True Negative (TN)*: Benign samples correctly classified as the negative class.
- *False Positive (FP)*: Benign samples falsely classified as the positive class.
- *False Negative (FN)*: Malware samples falsely classified as the negative class.
- *True Positive Rate (TPR)*: ratio of true positives to the no. of positive samples.
- *False Positive Rate (FPR)*: ratio of false positives to the no. of negative samples.
- *Accuracy:* percentage of samples correctly classified by the model.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \times 100 \tag{9}$$

- *AUROC*: Area under the ROC curve (plot of TPR vs FPR at various thresholds).
- *Fooling Rate (FR)*: This metric represents the percentage of successful misclassifications among the adversarial samples (M'), accounting for false negatives that naturally occur due to limitations of the model. FR is a measure of effectiveness of an adversarial attack.

$$\text{FR} = \max\left(\frac{\text{FN}_{M'} - \text{FN}_{\text{val}}}{\text{Num}(M') - \text{FN}_{\text{val}}} \times 100\%, 0\%\right) \tag{10}$$

$$\text{TS} = \frac{\text{FR}_{\text{grey-box}}}{\text{FR}_{\text{surrogate}}} \tag{11}$$

- Transferability Score (TS): It is the ratio of FR on the grey-box model to the FR on the surrogate model used to generate the adversarial samples. It is a measure of the transferability of adversarial samples from the surrogate model to the grey-box model.
- Evasion Cost: Evasion cost defined by Chen et al. (2017) measures the expense incurred by attackers when implementing perturbations during an attack. It costs 0.5 to include permissions and intents, 0.4 to introduce API calls and new instances, and 0.95 to eliminate permissions and intents. Removing API calls and new instances incurs a cost of 0.7.

4. Experimental results

This section begins by examining the performance and feature utilisation of the base classification models. Subsequently, we analyse the hierarchical relationships between malware families, the performance and transferability of GBKPA, and the effectiveness of AuxShield.

4.1. Baseline malware detection models

We created 28 distinct models utilising 14 classification algorithms and 2 feature vectors. Fig. 2 illustrates the accuracy of the 14 classification algorithms using android permissions (L) and android intents (R). The y-axis denotes the model accuracy, while the x-axis indicates the no. of features utilised by the model.

Using Permissions: On average, utilising 195 android permissions, the models accomplished an accuracy of 94.48 % and an AUROC score of 0.98. The baseline random forest model using 195 android permissions accomplished the maximum accuracy of 95.95 %, while the naive bayes model attained the least accuracy of 88.27 %. The random forest model based on 195 android permissions also obtained the highest AUROC score of 0.99, while naive bayes attained the lowest score of 0.96.

Using Intents: On average, utilising 273 android intents, the models accomplished an accuracy of 84.82 % and an AUROC score of 0.92. The baseline random forest model using 273 android intent accomplished the maximum accuracy of 86.08 % while naive bayes attained the least accuracy of 80.51 %. Similarly, the random forest model based on 273 android intents achieved the highest AUROC score of 0.94 while naive bayes attained the lowest score of 0.88.

Overall, the baseline permissions-based malware detection models outperformed the baseline intent-based models at encoding information and helped build more accurate malware detection models. Most baseline permissions-based models achieved 90–95 % accuracy, while intents-based models attained 80–85 % accuracy.

4.2. Accuracy vs number of features

The baseline malware detection models based on permissions and intents were constructed using 195 and 273 features, respectively. For each of the 28 baseline models, we estimated the importance of each feature using various methods. For example, In the logistic regression models, we used weights to estimate the importance of features. Similarly, in naive bayes models, we used mutual information with the class labels to determine important features. For tree-based models, we measured feature importance using the normalised reduction of the impurity criterion. In deep neural network models, we estimated important features by calculating the mean of the absolute values of the partial derivatives of the loss function for each feature, over all samples in the dataset.

Fig. 2 shows the accuracy of the 28 baseline models on the y-axis using K top android permissions (L) and intents (R). The no. of features used by the models is shown on the x-axis. Considering only the top 1, 5,

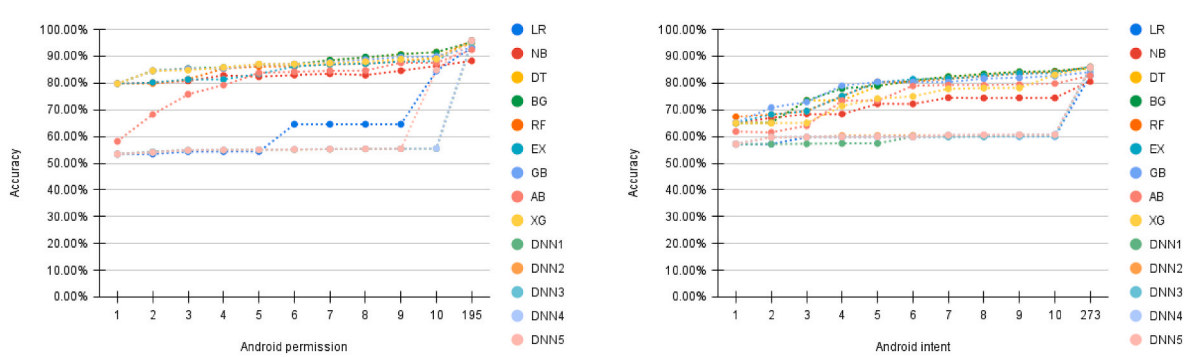


Fig. 2. The accuracy of 14 classification algorithms utilising the top K android permissions (L) and android intents (R).

Table 2
Top ten most significant android permissions (T) and intents (B) for detecting malware.

Android Permission
android.permission.REQUEST_INSTALL_PACKAGES
android.permission.READ_PHONE_STATE
android.permission.SEND_SMS
android.permission.READ_EXTERNAL_STORAGE
com.android.launcher.permission.INSTALL_SHORTCUT
android.permission.USE_FINGERPRINT
android.permission.ACCESS_COARSE_LOCATION
android.permission.FOREGROUND_SERVICE
android.permission.GET_TOP_ACTIVITY_INFO
android.permission.MANAGE_DOCUMENTS
Android Intent
android.intent.category.LEANBACK_LAUNCHER
android.intent.action.MY_PACKAGE_REPLACED
android.intent.category.HOME
android.intent.action.USER_PRESENT
android.intent.action.BOOT_COMPLETED
android.intent.action.PACKAGE_ADDED
android.intent.action.VIEW
android.intent.action.APPLICATION_PREFERENCES
android.intent.action.ASSIST
android.intent.category.DEFAULT

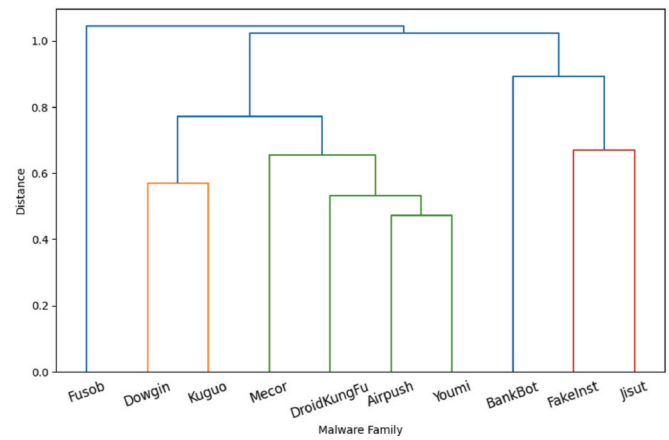
and 10 features, we accomplished an average accuracy of 67 %, 72 %, and 79 % for the permission-based models and 62 %, 70 %, and 73 % for the intent-based models, respectively. The ML-based models attained an average accuracy of 71 %, 74 %, and 88 % using the top 1, 5, and 10 permissions and 62 %, 70 %, and 73 % with the top 1, 5, and 10 intents, respectively. Bagging-based models outperformed the other model classes and attained an average accuracy of 80 %, 84 %, and 90 % using 1, 5, and 10 permissions and 66 %, 80 %, and 88 % with intents. Boosting-based models attained an average accuracy of 73 %, 86 %, and 89 % using 1, 5, and 10 permissions and 53 %, 55 %, and 61 % with intents. DNNs attained an average accuracy of 71 %, 74 %, and 88 % using 1, 5, and 10 permissions and 57 %, 60 %, and 61 % with intents. The above results indicate that the top 5–10 features are sufficient to detect malware accurately. Additionally, these feature can be utilised to create rule-based security systems to complement the ML/DL-based security system.

4.3. Top features for detecting malware

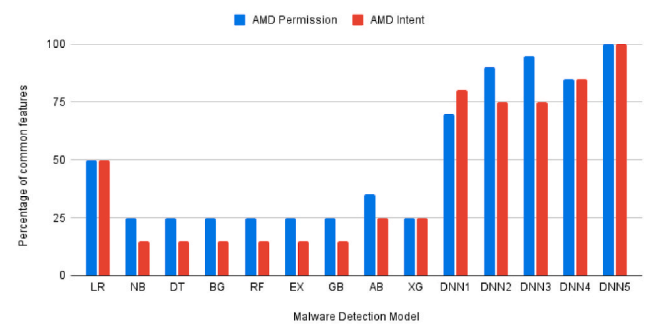
With 10 features, models achieve significant accuracies. Table 2 shows the top 10 most important and frequently used android permissions and intents by the models. The permission *android.permission.REQUEST_INSTALL_PACKAGES* and the intent *android.intent.category.LEANBACK_LAUNCHER* were considered important by 13 out of the 14 malware detection algorithms. Each of the top 10 android permissions and intents were used by at least 5 and 6 classification algorithms respectively, contributing to transferability.

4.4. Malware family analysis

We analyse the malware families from the AMD dataset containing more than 500 samples: *Airpush*, *Youmi*, *DroidKungfu*, *Mecor*, *FakeInst*, *Jisut*, *Bankbot*, *Dowgin*, *Kuguo*, and *Fusob*. Fig. 3a shows a dendrogram illustrating the hierarchical relationships among these malware families based on similarities in permissions and intents. The y-axis indicates distances between clusters, while the x-axis denotes families. A K-Nearest Neighbors model was used to calculate the mean distance between a pair of families. A linkage matrix was calculated using the *Ward method*, based on the normalised distance between each pair of families, to generate the dendrogram. The dendrogram notably highlights three prominent clusters: *(Dowgin, Kuguo)*, *(Mecor, DroidKungFu, Airpush, Youmi)*, and *(FakeInst, Jisut)*. Here, *Fusob* and *Bankbot* stand out as



(a) Hierarchical clustering of malware families based on similarity of Android permissions and intents



(b) Feature usage similarity of malware detection models with surrogate model

Fig. 3. Analysis of malware families and feature usage similarity in detection models.

outliers and are the furthest from these three clusters. Following this exploratory family analysis, we plan to investigate family-specific attacks that integrate traditional obfuscation attacks with adversarial attacks in the future.

4.5. Similarity in feature usage with surrogate model

Most model accuracies reach saturation with the 20 most important features. Fig. 3b demonstrates the percentage of top 20 features common with the surrogate model along the y-axis, with each model represented along the x-axis. DNNs exhibit an average similarity of 88 % for permissions and 83 % for intents with the surrogate model. The machine learning methods show an average similarity of 33 % for permissions and 27 % for intents, the bagging-based methods show an average similarity of 25 % for permissions and 15 % for intents and the boosting-based methods show an average similarity of 28 % for permissions and 22 % for intents. Tree-based models show an average similarity of 26 % with permissions and 18 % with intents. The difference in similarity of important features across classes occurs due to the difference in learning methods of these classes. DNNs demonstrate the highest similarity, trailed by boosting methods, while logistic regression predominantly contributes to the higher averages observed in ML methods. Conversely, traditional decision trees and bagging-based methods display the lowest similarity.

4.6. Adversarial attack and fooling rate

The underlying classification model can incorrectly classify malicious samples as benign, leading to False Negatives. GBKPA aims to efficiently cause minimal perturbations to the malicious samples in the

dataset to create adversarial samples that can evade detection by the underlying classification models. We conducted the attack on all 28 classification models using a varying number of perturbations, ranging from 1 to 10, observing saturation at 10 perturbations. Fig. 4 depicts the fooling rate attained by GBKPA on the y-axis with the no. of perturbations depicted on the x-axis for each model.

GBKPA with 1 Perturbation: With 1 perturbation and an evasion cost of 0.5, GBKPA achieved an average fooling rate of 35 % and 46 % against models using permissions and intents respectively. Against ML-based models using permissions and intents, the average fooling rate was 33 % and 43 % respectively. Against bagging-based models using permissions and intents, the average fooling rate was 15 % and 40 % respectively. Against boosting-based models using permissions and intents, the average fooling rate was 30 % and 47 % respectively. Against DNNs using permissions and intents, the average fooling rate was 50 % and 53 % respectively.

GBKPA with 5 Perturbations: With 5 perturbations and an evasion cost of 2.5 perturbations, GBKPA achieved an average fooling rate of 70 % and 84 % against models using permissions and intents, respectively. Against machine learning (ML)-based models using permissions and intents, the average fooling rate was 68 % and 83 %, respectively. Against bagging-based models using permissions and intents, the average fooling rate was 59 % and 80 %, respectively. Against boosting-based models using permissions and intents, the average fooling rate was 70 % and 83 %, respectively. Against deep neural network (DNN) models using permissions and intents, the average fooling rate was 78 % and 86 %, respectively. Against tree-based models using permissions and intents, the average fooling rate was 61 % and 82 %, respectively. Notably, decision trees were more robust compared to other models in the machine learning class with a fooling rate of 55 % and 80 % against permissions and intents, respectively. Models using permissions demonstrated greater adversarial robustness compared to models using intents.

Comparison with Surrogate: GBKPA achieves 78 % and 86 % fooling rates against the surrogate model after 10 perturbations using android permissions and intents, respectively. Its performance against base models yields an average fooling rate of 70 % and 84 %, showing comparable efficacy to a white-box attack in a grey-box scenario.

4.7. Perturbation Lists

To identify the most frequently perturbed permissions and intents, we generate samples with 5 perturbations. Table 3 presents the top five permissions and intents that modified most frequently by GBKPA. The top permission and intent were perturbed in 82 % and 61 % of the samples, respectively, while the top three permissions and intents were perturbed in 45 % and 24 % of the samples on average, respectively. These perturbed permissions and intents contribute to imparting a benign appearance to the malicious samples, aiding in masking their true malicious nature. Potential attackers can exploit these features to evade ML and DL classifiers.

4.8. Transferability scores

Transferability scores were computed for different models after conducting the attack with 10 perturbations. These scores help gauge the transferability of adversarial samples from the surrogate model to the grey-box model. Fig. 6 presents the transferability scores for each model, with the y-axis representing the scores and the x-axis denoting the models.

Android Permissions: Using permissions, the average transferability scores for ML-based methods, bagging-based methods, boosting-based methods and DNN's were 0.87, 0.76, 0.90 and 1 respectively.

Android Intents: Using intents, the average transferability scores for Ml-based methods, bagging-based methods, boosting-based methods and DNN's were 0.97, 0.94, 0.97, and 1 respectively.

Against tree-based models the average transferability score was 0.81 and 0.95 using permissions and intents, respectively. Noticeably, in the class of ML-models decision trees scored significantly less than other models, with scores of 0.71 and 0.94 with permissions and intents respectively. GBKPA aims to efficiently craft adversarial samples that evade the classifier by perturbing the features considered most important by the surrogate model. The transferability of adversarial samples can be attributed to the similarity in feature usage, as demonstrated in Sections 4.2 and 4.3. DNNs exhibit the highest average transferability scores, followed by boosting-based methods. This can be explained by the higher similarity of feature usage with the surrogate model, as discussed in Section 4.5.

4.9. AuxShield

We reinforce our base models using the AuxShield defence strategy. Employing random forest, gradient boosting, and a DNN as grey-box models, we generate adversarial samples with 1–10 max perturbations using GBKPA. To detect if the input is adversarial, we build a Random Forest model using these samples along with additional malicious and benign samples from our training dataset, constituting the first level of AuxShield. The second level involves adversarially retraining the aforementioned base models, enhancing their robustness. In Fig. 5, GBKPA's fooling rate is plotted on the y-axis against the number of perturbations on the x-axis for each model after implementing AuxShield. In permissions (L), values are $\leq 1\%$, resulting in lines close to the x-axis.

The models show a difference of $\pm$ 3 % in accuracy and AUROC scores after implementing AuxShield. After 10 perturbations, the models show an average fooling rate (FR) of 0.11 % and 6.4 % with permissions and intents, respectively. This signifies a substantial improvement from the previous FR of 70 % and 84 %, respectively. ML-based models exhibit an average FR of 0.09 % and 9 % with permissions and intents, respectively. Bagging-based models demonstrate an average FR of 0.08 % and 8 % with permissions and intents, respectively. Boosting-based models exhibit an average FR of 0.18 % and 8 % with permissions and intents, respectively. DNNs display an average FR of 0.09 % and 3 %

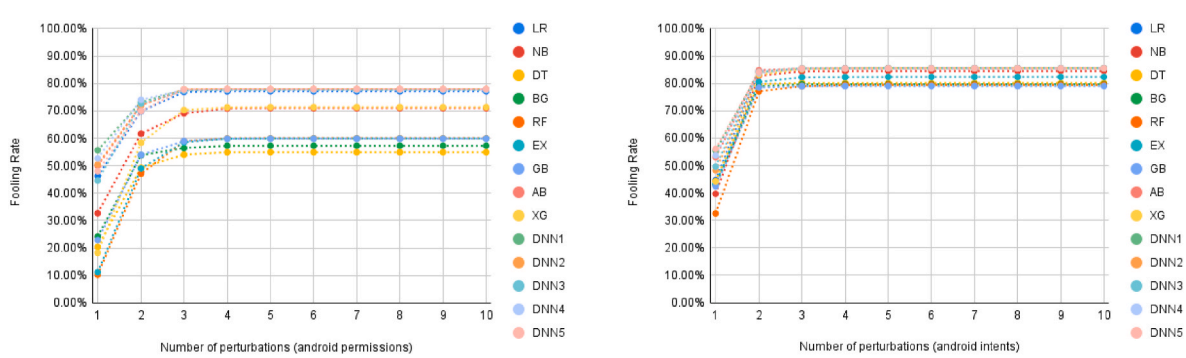


Fig. 4. Fooling rate of GBKPA against different classification algorithms using android permissions (L) and intents (R).

Table 3
Top five frequently perturbed permissions (L) and intents (R) by GBKPA.

Android Permission	% of Samples	Android Intent	% of Samples
android.permission.REQUEST_INSTALL_PACKAGES	82.37	android.intent.category.LEANBACK_LAUNCHER	60.87
android.permission.FOREGROUND_SERVICE	35.66	android.intent.action.MY_PACKAGE_REPLACED	5.75
android.permission.GET_TOP_ACTIVITY_INFO	15.70	android.intent.action.SCREEN_OFF	4.88
android.permission.USE_FINGERPRINT	11.05	android.intent.action.ACTION_SHUTDOWN	4.42
android.permission.SYSTEM_ALERT_WINDOW	6.21	android.intent.action.PACKAGE_REPLACED	2.59

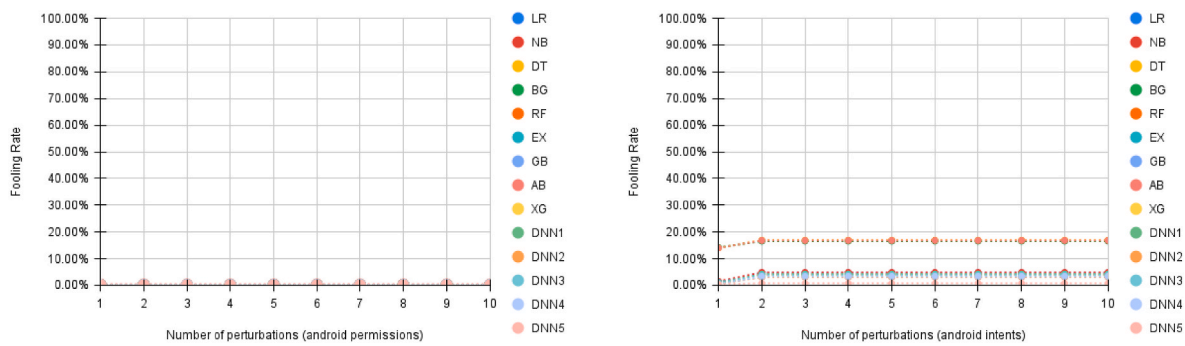


Fig. 5. Fooling rate of GBKPA against different classfication algorithms using android permissions (L) and intents (R) after implementing AuxShield defence.

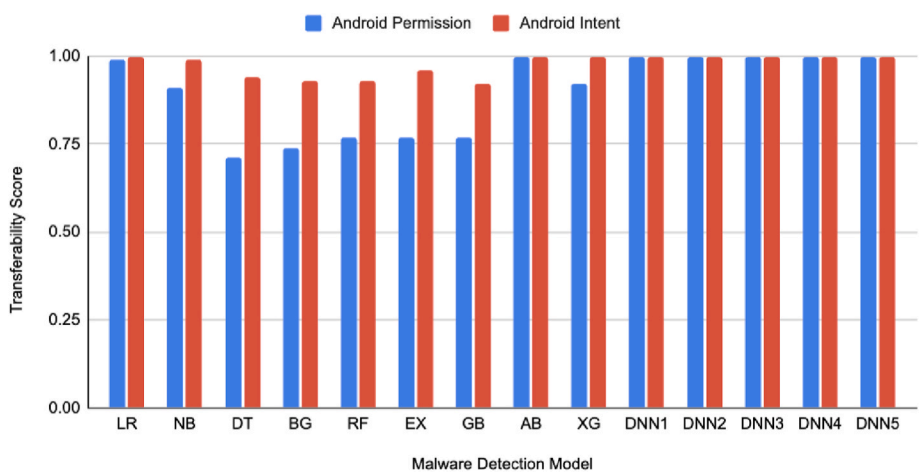


Fig. 6. Transferability scores of baseline classification algortihms using android permissions and intents against GBKPA.

with permissions and intents, respectively. Tree-based models show an average FR of 0.1 % and 8 % with permissions and intents, respectively. Against permissions and intents-based models, the highest FR was 0.35 % (against AdaBoost) and 17 % (against DT), respectively.

5. Related work

Table 4 presents the current literature on adversarial robustness and transferability of adversarial malicious android applications. Grosse et al. (2017) presented a gradient-based white box approach to craft adversarial samples against neural networks and obtained a fooling rate of 65.48 %. Chen et al. (2017) also devised an attack for a white-box scenario utilising information gain to rank the features, achieving an average fooling rate of 84 %. However, these attacks did not represent the real-world scenario, as the attacker typically lacks access to the classifier's internal mechanisms. Nevertheless, these studies laid the foundation for subsequent research focusing on grey-box scenarios that represent a more realistic scenario in the real world. Several works like Taheri et al. (2020), Sewak et al. (2020), Cara et al. (2020), Rafiq et al. (2022) and Rathore et al. (2023b) tried to address these research gaps and obtained average fooling rates of 27 %, 90 %, 44.23 %, and 59 % respectively in grey-box scenarios. However, many of these studies did not assess the attack's effectiveness across various model classes or discuss the transferability of adversarial samples. Additionally, they did not provide a comprehensive list of perturbations that could serve as potential vulnerabilities for evading models. Moreover, defence strategies in the existing literature did not significantly reduce the fooling rate to below 15 %.

We address the limitations of the existing literature in our work. Our work attains a relatively high average fooling rate of 77 % for a grey-box scenario while making minimal perturbation to ensure the least evasion cost. Additionally, we construct a varied set of models to assess our attack and explain the transferability of adversarial samples. We also provide a list of potential perturbations that effectively conceal the malicious properties of malware samples. We also introduce a dual-layered security mechanism called AuxShield to counter these adversarial attacks, which significantly reduced the average fooling rate to 0.11 % for permissions and 6.4 % for intents.

Table 4
Proposed work versus state-of-the-art in existing literature.

Existing Literature	Attack Scenario	Max % of Features Perturbed	Evasion Cost	# of Model classes	Feature Usage Analysis	Average FR	Pertub List[a]	Transfer Analysis[b]	Adversarial Defence
Grosse et al. (2017)	White Box	36 %	9	1	No	65.48	No	No	Adversarial Retraining and Defensive Distillation
Chen et al. (2017)	White Box	5.39 %	25	1	No	84	No	No	SecureDroid
Taheri et al. (2020)	Grey Box	20 %	30	2	No	27	No	No	Adversarial Retraining and GAN
Cara et al. (2020)	Grey Box	100 %	80	1	No	90	Yes	No	No
Rathore et al. (2021b)	Grey Box	2.56 %	2.5	4	No	44.23	No	No	Adversarial Retraining
Rafiq et al. (2022)	Grey Box	3.96 %	N.A.	1	No	59	No	No	No
Proposed Work	Grey Box	1.83 %–2.56 %	2.5	4	Yes	77	Yes	Yes	AuxShield

[a] Perturbation List.
[b] Transferability Analysis.

6. Conclusion and future work

Researchers have begun studying the efficacy of ML/DL techniques in detecting malware. However, these models face significant challenges, including vulnerability to adversarial attacks and concerns regarding their robustness.

This study evaluated the adversarial robustness of 28 distinct ML/DL models, constructed using classification algorithms spanning four distinct classes, alongside two separate features (permissions and intents) extracted from android applications. These models demonstrated high average accuracies of 94.5 % and 84.5 % for permission-based and intent-based models, respectively, highlighting their effectiveness in detecting malicious android applications. Following that, we introduced *Gradient Based K Perturbation Attack* (GBKPA) to evaluate the adversarial robustness of these models. GBKPA demonstrated significant fooling rates, achieving 70 % against models using permissions and 84 % against models using intents. This highlights the susceptibility of these ML/DL models to adversarial attacks. Furthermore, we identified the vulnerable permissions and intents. DNNs showed higher transferability with the surrogate model (average score of 1) than other models (average score of 0.9), likely due to differing feature reliance across classes and similarities within the same class. Finally, we implemented *AuxShield* and reduced the average fooling rate to 0.11 % and 6.4 % against permission and intent based models, respectively.

Our study highlights the need for further analysis of transferability's causation and implications, alongside developing innovative adversarial defences. However, we focused on a grey-box scenario and did not explore pure black-box scenarios, alternative feature vectors beyond permissions and intents, or family-specific attacks. Future research could comprehensively investigate these aspects and explore transferability across different datasets, proposing effective defences. This research underscores the importance of addressing vulnerabilities and limitations in ML/DL models within the evolving security landscape.

References

Android, 2024. Market share of mobile operating systems worldwide from 2009 to 2024, by quarter. https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/. (Accessed June 2024).

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., Siemens, C., 2014. Drebin: effective and explainable detection of android malware in your pocket. In: Network and Distributed System Security Symposium (NDSS 2014), pp. 23–26.

Cara, F., Scalas, M., Giacinto, G., Maiorca, D., 2020. On the feasibility of adversarial sample creation using the android system api. Information. MDPI 11, 433.

Chen, L., Hou, S., Ye, Y., 2017. Securedroid: enhancing security of machine learning-based detection against adversarial android malware attacks. In: 33rd Annual Computer Security Applications Conference (ACSAC 2017), pp. 362–372.

Goodfellow, I.J., Shlens, J., Szegedy, C., 2014. Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572.

Google Play Protect, 2024. Google play protect. https://developers.google.com/android/play-protect. (Accessed June 2024).

Grosse, K., Papernot, N., Manoharan, P., Backes, M., McDaniel, P., 2017. Adversarial examples for malware detection. In: European Symposium on Research in Computer Security (ESORICS 2017). Springer, pp. 62–79.

Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., Madry, A., 2019. Adversarial examples are not bugs, they are features. Adv. Neural Inf. Process. Syst. 32 (NIPS 2019.

Kaspersky, 2023. Google play malware clocks up more than 600 million downloads in 2023. https://www.kaspersky.co.in/blog/malware-in-google-play-2023/26621/. (Accessed June 2024).

Kaspersky, 2024. The mobile malware threat landscape in 2023. https://securelist.com/mobile-malware-report-2023/111964/. (Accessed June 2024).

Kurakin, A., Goodfellow, I.J., Bengio, S., 2018. Adversarial examples in the physical world. In: Artificial Intelligence Safety and Security. Chapman and Hall/CRC, pp. 99–112.

Pal, A., Sulam, J., Vidal, R., 2024. Adversarial examples might be avoidable: the role of data concentration in adversarial robustness. Adv. Neural Inf. Process. Syst. 36 (NIPS 2024.

Papernot, N., McDaniel, P., Goodfellow, I., 2016. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. arXiv preprint arXiv:1605.07277.

Rafiq, H., Aslam, N., Issac, B., Randhawa, R.H., 2022. On impact of adversarial evasion attacks on ml-based android malware classifier trained on hybrid features. In: 14th International Conference on Software, Knowledge, Information Management and Applications (SKIMA). IEEE, pp. 216–221.

Rathore, H., Nandanwar, A., Sahay, S.K., Sewak, M., 2023a. Adversarial superiority in android malware detection: Lessons from reinforcement learning based evasion attacks and defenses. Forensic Sci. Int.: Digit. Invest. 44, 301511.

Rathore, H., Nikam, P., Sahay, S.K., Sewak, M., 2021a. Identification of adversarial android intents using reinforcement learning. In: 2021 International Joint Conference on Neural Networks (IJCNN), IEEE, pp. 1–8.

Rathore, H., Sahay, S.K., Nikam, P., Sewak, M., 2021b. Robust android malware detection system against adversarial attacks using q-learning. Inf. Syst. Front 23, 867–882. Springer.

Rathore, H., Sahay, S.K., Thukral, S., Sewak, M., 2020. Detection of malicious android applications: classical machine learning vs. deep neural network integrated with clustering. In: International Conference on Broadband Communications, Networks and Systems (BROADNETS). Springer, pp. 109–128.

Rathore, H., Samavedhi, A., Sahay, S.K., Sewak, M., 2023b. Towards adversarially superior malware detection models: an adversary aware proactive approach using adversarial attacks and defenses. Inf. Syst. Front 25, 567–587.

Sewak, M., Sahay, S.K., Rathore, H., 2020. Assessment of the relative importance of different hyper-parameters of lstm for an ids. In: IEEE Region 10 Conference (TENCON). IEEE, pp. 414–419.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R., 2013. Intriguing Properties of Neural Networks arXiv preprint arXiv:1312.6199.

Taheri, R., Javidan, R., Shojafar, M., Vinod, P., Conti, M., 2020. Can Machine Learning Model with Static Features Be Fooled: an Adversarial Machine Learning Approach, vol. 23. Cluster Computing, Springer, pp. 3233–3253.

Wei, F., Li, Y., Roy, S., Ou, X., Zhou, W., 2017. Deep ground truth analysis of current android malware. In: 14th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA). Springer, pp. 252–276.

Zhang, C., Benz, P., Imtiaz, T., Kweon, I.S., 2020. Understanding adversarial examples from the mutual influence of images and perturbations. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 14521–14530.

Zhang, G., Li, Y., Bao, X., Chakarborty, C., Rodrigues, J.J., Zheng, L., Zhang, X., Qi, L., Khosravi, M.R., 2023. Tsdroid: a novel android malware detection framework based on temporal & spatial metrics in iomt. ACM Trans. Sens. Netw. 19, 1–23.

Zhu, H., Li, Y., Li, R., Li, J., You, Z., Song, H., 2020. Sedmdroid: an enhanced stacking ensemble framework for android malware detection. IEEE Transactions on Network Science and Engineering (IEEE TNSE) 8, 984–994.