applied sciences

MDPI

Article

TrojanDetector: A Multi-Layer Hybrid Approach for Trojan Detection in Android Applications

Subhan Ullah [1], Tahir Ahmad [2], Attaullah Buriro [3,*], Nudrat Zara [1] and Sudipan Saha [4]

[1] Department of Cybersecurity, FAST School of Computing, FAST National University of Computer and Emerging Science, Islamabad 54000, Pakistan
[2] Center for Cybersecurity, Bruno Kessler Foundation, 38123 Trento, Italy
[3] Faculty of Computer Science, Free University Bozen-Bolzano, 39100 Bolzano, Italy
[4] Department of Aerospace and Geodesy, Technical University of Munich, 85521 Ottobrunn, Germany
* Correspondence: attaullah.buriro@unibz.it

Abstract: Trojan Detection—the process of understanding the behaviour of a suspicious file has been the talk of the town these days. Existing approaches, e.g., signature-based, have not been able to classify them accurately as Trojans. This paper proposes TrojanDetector—a simple yet effective multi-layer hybrid approach for Trojan detection. TrojanDetector analyses every downloaded application and extracts and correlates its features on three layers (i.e., application-, user-, and package layer) to identify it as either a benign application or a Trojan. TrojanDetector adopts a hybrid approach, combining static and dynamic analysis characteristics, for feature extraction from any downloaded application. We have evaluated our scheme on three publicly available datasets, namely (i) CCCS-CIC-AndMal-2020, (ii) Cantagio-Mobile, and (iii) Virus share, by using simple yet state-of-the-art classifiers, namely, random forest (RF), decision tree (DT), support vector machine (SVM), and logistic regression (LR) in binary—class settings. SVM outperformed its counterparts and attained the highest accuracy of 96.64%. Extensive experimentation shows the effectiveness of our proposed Trojan detection scheme.

Keywords: android; malware detection; pattern matching; classification

Citation: Ullah, S.; Ahmad, T.; Buriro, A.; Zara, N.; Saha, S. TrojanDetector: A Multi-Layer Hybrid Approach for Trojan Detection in Android Applications. *Appl. Sci.* **2022**, *12*, 10755. https://doi.org/10.3390/app122110755

Academic Editors: Christos Bouras, Safwan El Assad, René Lozi and William Puech

Received: 9 September 2022
Accepted: 21 October 2022
Published: 24 October 2022

1. Introduction

Mobile malware applications are divided into several categories based on their destructive purposes and behaviours (Trojans, Viruses, Worms, Botnets, Spyware, annoying advertising tools, etc.). Trojans are applications that are seemingly useful and risk-free. These applications secretly perform destructive actions such as stealing users' information, damaging their devices, changing system settings, loading malicious applications, etc. [1]. Android is open-source and extremely popular on mobile platforms and is targeted by over 99% of malware attacks [2]. Nearly half of these malware attacks are multi-functional Trojans capable of stealing users' personal and sensitive data, i.e., contacts, email addresses, and phone numbers. Besides stealing data, they are able to download additional modules from the infected servers being maintained by attackers.

Detecting malware, in general, and Trojans, in particular, could be done by performing static or dynamic analysis. Static analysis is performed without executing the program. Examples of the information we can obtain from static analysis include opcode sequences (extracted by disassembling the binary file), control flow graphs, and so on. Such feature sets can be used individually or in combination for malware detection. On the other hand, dynamic analysis requires that we execute the program, often in a virtual environment. Examples of information that can be obtained by dynamic analysis include API calls, system calls, instruction traces, registry changes, memory writes, and so on.

Technically, static analysis finds an app's dangerous traits or faulty code parts without executing it. In comparison, the dynamic analysis collects the app's activity data to detect

malicious behaviour during the app's operations [3]. Static analysis has been shown to be somehow accurate and efficient in detecting known malware; however, it has been found less accurate against apps containing camouflage techniques and encrypted code parts. Additionally, they could not detect intrusions in real-time. In contrast, dynamic analysis can detect zero-day threats; however, at the cost of expensive computations because it could involve multiple resources running in parallel. These schemes are also not shown to be highly accurate [3].

Detecting Trojan attacks by considering a single layer, e.g., the application layer where the users use the application or only the package layer where the files of the apps are saved on the Android, is not enough. Trojans can change their pattern by attacking by SMS or emails or by attacking the system directly by getting permission. For instance, an excessive amount of pop-up ads suddenly changes to user's email service and starts sending spam messages to all addresses in the contacts and then automatically unauthorised application installation, which can make it more difficult to detect if all layers (e.g., application, package, and user layer) are not monitored together. Suppose only one layer with a few features considers SMS quantity and email quality. In that case, the detection of Trojan attacks is a problem on the other layers, such as the user layer, network layer, etc. Due to this problem of detecting the Trojan on a single layer, Trojans still have the door to attack the other layers because they constantly change their behaviours and patterns and cause serious damage to user privacy, security, and data loss.

Various security solutions have been proposed recently, which apply static or dynamic analysis of the applications [4] and apply security approaches by implementing data security with run-time enforcement. However, these approaches have limitations: firstly, they are layer-specific, tackle a single layer (e.g., application layer or user layer), and require a custom operating system [5]. Secondly, they are applied individually, resulting in lower detection accuracy. A dire need for a multi-layer hybrid approach (combining both static and dynamic analyses) that could potentially address the limitations of these individual single-layer techniques is observed [6].

Android has its own local security mechanism based on permissions and app isolation. These two features, i.e., permission and app isolation, enforce access control to security resources and operations and keep away from an application that can interfere with the execution of another. However, these schemes are not found too secure for two reasons: they were found ineffective at warning users [7], and Trojan creators can bypass these detection methods [8]. To address these issues, methods based on machine learning have gained significant popularity in recent years.

In this paper, we propose a novel approach, namely, TrojanDetector, to detect android malware or Trojans in a multi-layer (application, package, and user's layer) and hybrid way. TrojanDetector collects execution data of a set of known sample Trojan and benign Android applications to generate patterns of individual system calls, sequential system calls with different calling depths related to file and network access, and so on. TrojanDetector builds up a malicious pattern set and a normal pattern set for malware detection and benign application judgement by comparing the patterns of Trojan and benign applications. When TrojanDetector needs to detect an unknown app, it uses a dynamic method to collect its run-time system call data in terms of individual and sequential system calls with different depths. Then we extract the target patterns of the unknown app from its run-time system calling data. Using our chosen machine learning classifiers trained on both the malicious pattern set and the normal pattern set, TrojanDetector classifies the unknown app as either Trojan or benign. The proposed approach is a generic detection method suitable for several types of Android malware detection. The TrojanDetector uses SVM as the classifier and achieves the highest accuracy of 96.64%. Extensive experimentation shows the effectiveness of our proposed Trojan detection scheme. Specifically, the contribution of this paper can be summarised as below:

- This work presents a multi-layer hybrid approach, i.e., TrojanDetector, to detect the Trojan's misbehaviour in Android applications from three different Android levels based on the selected features and then apply multi-classifiers to train the model.
- This work uses four machine learning classifiers (random forest, support vector machine (SVM), logistic regression and, decision tree) for accurate Trojan detection in Android applications.
- This work performs an extensive experimental evaluation to identify the usefulness of the proposed approach and gauge the accuracy of the machine learning approach using publicly available data sets of CCCS-CIC-AndMal-2020 [9], Contagio-Mobile [10], and Virsushare [11] of Trojan applications.

The remainder of the paper is organised as follows. Section 2 briefly overviews related work. Section 3 presents the proposed approach and defines Trojan characteristics that detect known and anomalous behaviours. Section 4 presents an experimental evaluation and discussion of the proposed approach. Section 5 concludes and presents the direction for future work.

2. Related Work

The analysis of intrusion detection on Android devices is a rich field in the literature. Three main approaches, e.g., static, dynamic, and hybrid, as shown in Table 1, are used in the literature to analyse malware detection in Android applications. Examples of static approaches are DroidSieve [12] and MiMalo [13], which analyse the features on the application layer. Similarly, dynamic approaches [10,14–16] are also available for feature analysis on the applications layer using machine learning and deep learning.

A hybrid analysis approach uses static and dynamic features for detecting malware, which has been found to be more accurate due to the complementary strength of both types of analysis. The hybrid analysis exploits the static and dynamic features to detect malware in specific use cases [17]. A hybrid approach uses the Android application sandbox environment and performs static and dynamic analysis in an offline mode. Both static and dynamic approaches have their limitations, e.g., static analysis is more time-consuming, and dynamic analysis is more resource-consuming for malware detection. Yuan et al. [18] presented an engine called DroidDetector and automatically characterised the application as benign or malicious. The DroidDetector has 96.76% accuracy compared to the traditional machine learning (ML) approaches. Tong and Yan [3] used a hybrid approach that considered the advantages and disadvantages of static and dynamic analysis to detect Android malware. They collected the behaviour data of the application dynamically. They used the static method in offline mode for the analysis of malicious and normal patterns of the application from that collected data. They used a powerful malware detection server to improve the processing speed and acceleration of malware detection. Martin et al. [17] presented a static and dynamic analysis framework. They ran a learning classification model on its OmniDroid dataset, which consists of 22,000 samples of malware and benign Android applications. Hadiprakoso et al. [18] also proposed a hybrid analysis approach and developed a machine learning and deep learning model for detecting malware in the Android operating system. They used the Genome dataset and the Drebin project for static analysis and the CICMalDroiD [11] dataset for dynamic analysis. Then they extracted 261 combined features and applied a hybrid analysis approach. They further used 311 application samples, including 165 benign from the play-store and 146 malicious apps from the VirusShare, and increased the detection rate by about 5%. Another hybrid approach, ShielDriod [19], uses the CICMalDroid dataset and analyses the malware on the application layer. They used random forest (RF) and multilayer perceptron models and achieved 97% accuracy.

We proposed a hybrid technique and compared it to the state-of-the-art techniques as baselines. We found that the TrojanDetector did not outperform state-of-the-art peer approaches to malware detection on Android in general, such as DroidSieve [12] and DroidCat [14]. Both DroidSieve and DroidCat achieved higher accuracy than the proposed

detection approach. DroidSieve is an Android malware classifier based on static analysis with an accuracy of 99.82%, whereas DroidCat is based on dynamic analysis with an accuracy of 97%. However, considering the intrinsic limitations of both static and dynamic approaches (refer to Section 1), the results are not comparable to TrojanDetector (with an accuracy of 96.64%) based on combining both approaches and features extracted from multiple layers, unlike DroidSieve and DroidCat. TrojanDetector improves the state-of-the-art by implementing a hybrid analysis approach on three different layers, called the multi-layer hybrid analysis approach.

Table 1. Comparison of TrojanDetector with state-of-the-art approaches.

Trojan Detectors	Dataset Used	Intrusion Detection and Classification Techniques	Feature Selection on Layers for Intrusion Detection	Analysis Approach	Accuracy of Malware Detection
MiMaLo [13]	MiMaLo technique	Neural Network Algorithm	Application	Static	88.5%
DroidSieve [12]	exploited obfuscation-invariant features	Obfuscation			99.82%
Bokolo et al. [15]	Tested on emulators	STAMINA approach based on DL	Application	Dynamic	Nil
Yuan et al. [18]	Apps: Google play store and Malware: from Contagio community and Gnome project	DBN-based DL			96.76%
Gan et al. [16]	CICAndMal2017	ML-based random forest model			98%
DroidCat [14]	Drebin and Gnome along with virus total and virus share databases	Supervised ML			97%
Aminuddin et al. [20]	Trojan dataset from Drebin and google play store	Random forest, Naive Bayes, and J48			81.2%
Martin et al. [17]	OmniDroid	AndroPyTool	Application	Hybrid	89.7%
Hadiprakoso et al. [18]	Genome & CICMalDroid	ML & DL			
Tong and Yan [3]	Own data set created by comparing the patterns of malware and benign apps	Offline comparisons of dynamic patterns set to judge the unknown app.			88%
ShielDroid [19]	CICMalDroid	Random forest, and Multilayer perceptron			97%
TrojanDetector	CCCS-CIC-AndMal-2020, Cantagio-Mobile and Virusshare with 13,559 Trojan samples	Random forest, decision tree, Linear Regression, and SVM	(MultiLayer) Application, user and package layers	Hybrid	86.68% with an EER of 0.1346 and 96.64%, with an EER of 0.036

Further, compared to the state-of-the-art approaches, this work considers the most notable features only at various levels of Android phones (called the multi-layer hybrid approach) to detect Trojans. Aminuddin et al. [20] focused on only a single layer of Android. They took only sys call features. However, they used a set of a total of 61 sys calls. In addition to that, they used random forest, Naive Bayes, and J48. Their experimental results show that random forest achieved the highest detection accuracy of 81.2%, with the lowest false positive rate of 0.188.

On the other hand, TrojanDetector targeted three main layers of Android: the application-level layer, user-level layer, and package-level layer. Compared to the state-of-the-art, we selected features from all these layers and targeted almost the ten most crucial Trojan families. All of the features were selected by taking into account the Trojan families. Similarly, the proposed approach used four classifiers for predicting Trojan viruses in Android devices: random forest, decision tree, linear regression, and SVM. By comparing

our random forest results with the random forest results from [20], our classifier reported an accuracy of 86.68% with an EER of 0.1346. In contrast, according to the selection of features concerning our approach, the SVM classifier achieved the highest accuracy of 96.64% with an EER of 0.036.

Further, the state-of-the-art comparison with our proposed approach in Table 1 shows that TrojanDetector is a multilayer simple and accurate Trojan detection scheme. It detects intrusions in general, while specific intrusions are called malware in particular. Common examples of malware categories include worms and ransomware, which can compromise similar scenarios and require the selection of more specific features for their detection. The proposed approach can be integrated with deep learning- and blockchain-enabled frameworks [21,22] to enhance data privacy and to provide secure communication in such specific Industrial IoT scenarios [23,24].

3. TrojanDetector Approach

3.1. Threat Model

To identify the attack patterns and threat agents, we hereby present the typical threat scenarios of Android applications.

- Trojans can come automatically through the ad popups.
- Trojans can come by installing unauthorised applications.
- Trojans can enter the user space by fraud-clicking.
- Trojans can masquerade as legitimate programmes.

3.2. Attacks

In this section, we discuss the possible attacks based on the aforementioned threat scenarios.

- Trojans can exploit the granted permissions to perform actions not intentionally authorised by the OS.
- Trojans can harm user data. It can leak user credentials by exploiting compromised user data.
- Trojans can also steal user data by sending SMS and emails.

3.3. Trojan Detector

To mitigate the threats found in the threat model, we propose TrojanDetector. Our approach to malware analysis starts with the training process at three levels in parallel; as shown in Figure 1, TrojanDetector installs an Android phone application and monitors its analysis features at the application, package, and user levels in parallel. The TrojanDetector uses four phases to detect a Trojan.

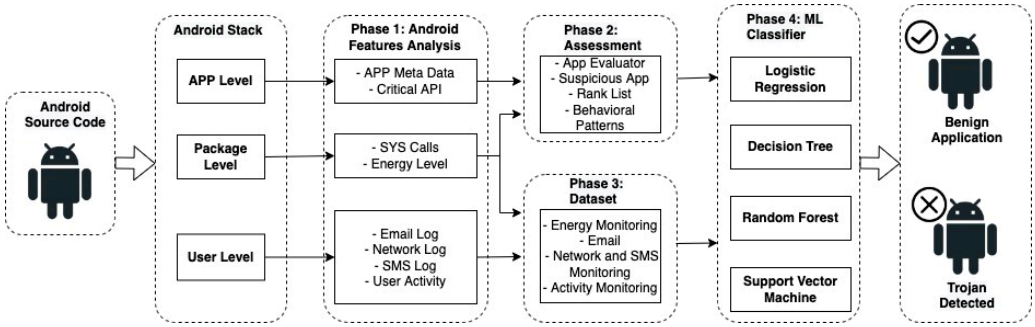

Figure 1. TrojanDetector: Multi-Layer Hybrid Trojan Detection.

- **Phase 1:** TrojanDetector performs static analysis at the application level features, e.g., meta-data and critical API, and package level features, e.g., system calls and the energy level.

- **Phase 2:** TrojanDetector applies dynamic analysis at the user level features (activity/network, emails, SMS logs). Once triggered, it performs app evaluation, ranks lists, and extracts behavioural patterns.
- **Phase 3:** TrojanDetector extracts energy, email, SMS, and activity features.
- **Phase 4:** TrojanDetector feeds all the features extracted from all the stages to the chosen classifiers to apply classification.

Algorithm 1 shows the underlying TrojanDetector Algorithm. It elaborates how TrojanDetector collects meta-data from the source code package, collects the features, evaluates the meta-data and then ranks them. Afterwards, the model observes the behavioural patterns of the Trojan and extracts the user-level features. After getting all the features, it is given to the Prediction ML models, which then predict the Trojan and give the results.

Algorithm 1 Detect Trojan

1. **procedure** Detect Trojan(DATA)
2. *Meta Data-set* ← *application is installed permissions, ratings*
3. *CollectFeatures* ← GetFeaturesFromMata *(Meta Data-set[])*
4. *PackageLevelAppEvaluaion* ← EvaluateMeta *(CollectFeatures[])*
5. *PackageLevelRank* ← RANK *(PackageLevelAppEvaluaion)*
6.
7. ***Check Known Behavioral Patterns***:
8. *BPrediction* ← PredictB *(BehavioralPatterns[])*
9.
10. ***Check User Level Features***:
11. *Activity Data-set* ← *MonitorActivity (app)*
12. *SMS Data-set* ← *MonitorSMS (app)*
13. *Energy Data-set* ← *MonitorEnergy (app)*
14. *Network Data-set* ← *MonitorNetwork (app)*
15. *Email Data-set* ← *MonitorEmail (app)*
16.
17. ***Predict Trojan Using AI Models Based on Features***:
18. *Predictions* ← *AIModel (PackegeLevelFeatures+ BehavioralPatterns+UserLevelFeatures)*
19. *IsTrojan* ← Classifier *(Predictions)*

4. Experimental Validation

4.1. Dataset

- CCCS-CIC-AndMal-2020: The dataset includes 200 K benign and 200 K malware samples of Android apps with 14 prominent and 191 eminent malware families. A dataset is generated in collaboration with the Canadian Centre for Cyber Security (CCCS) to capture 200 K android malware apps labelled and characterised into the corresponding family. The benign Android apps (200 K) are collected from the Androzoo dataset to balance the huge dataset. They collected 14 categories of malware, including adware, backdoor, file infector, Potentially Unwanted Apps (PUA), ransomware, scareware, riskware, trojan, trojan-banker, trojan-dropper, trojan-SMS, trojan-spy, and zero-day [25].
- Contagio Mobile: It is a blog-like website that collects malware for several mobile OS. Contagio presents more than 1000 samples for each Trojan family. Contagio has collected Trojans since 2012, including malicious apps found on Google Play, which may affect any Android device regardless of nationality [26].
- VirusShare: It is a website hosting a database of malware for several operating systems, which also includes a good share of Android malware. The database is continuously updated, and it is also possible to retrieve some of the latest threats discovered [27].

4.2. Features Extraction

The Android Profiler is used by the proposed approach to detect how much CPU is used by the Trojan, how much battery is consumed, and how much the Trojan uses RAM and network. Based on these features, our approach trained the proposed machine learning

model to detect Trojan applications efficiently. This features extraction phase dynamically examines the collected Trojan.apk samples and extracts the requisite features. Next, it will analyse the source code of an application by identifying system calls, user behaviours, and the data at the package level for each application. Relevant features from every application have been extracted to apply any machine learning technique. Android applications are delivered as .apk files. This is expected since permissions allow applications to perform actions that can potentially harm the user. Trojan applications also tend to request unusual permissions compared to legitimate ones. For each sample, if the requested permission matches the Android permission, it is assigned a 1 to indicate its presence in those samples, while a 0 indicates the absence of the permission. Feature selection is done with the help of information gain. We need to complete the features from the literature review and compare them for information gain.

Our proposed approach extracts 23 features related to various processes, such as CPU usage, App-services, etc. We briefly explain these processes and extract features. For the sake of brevity, we only provide an algorithm for calculating network usage.

Network Usage (user-level feature: dynamic).This feature describes whether or not this code uses the network. The value of this feature is in Boolean form, either 1 or 0. Algorithm 2 shows the algorithm to capture network usage after the installation of the app over the first 30 s (selected as a reference point) and compare it with bandwidth usage at a particular instance for the same period.

Algorithm 2 Calculating Network Stats

Input Trojan apps with the APPs
Output Bandwidth value

```
getting_BandwidthBroadcastReceivers = 0
BandwidthBroadcastReceivers[] ← GetBandwidthBatteryFromBroadcasts
for each BatteryBroadCast in BroadCastsReceivers do
CurrentBandwidthValue ← GetCurrentBandwidthValue(NetworkLog)
BandwidthValueConsumed_After30Sec ← MonitorBandwidthValue(BatteryBroadCast)
Value_change = CurrentBandwidthValue - BandwidthValueConsumed_After30Sec return value change
```

- **CPU Usage (package level feature: static)** This feature describes the use of CPU cycles by the Trojan, as there are three levels of CPU utilization. (i) a high describes the highest level of CPU usage (2) (ii) a Medium means the medium amount of CPU usage (1), and (3) a low means the code is using minimum CPU cycles (1).
- **App-Services (application-level feature: static)** This feature describes whether the Trojan uses the dangerous services which start in the background or not. The value of this feature is also in Boolean.
- **App-Permission (user-level feature: dynamic)** This feature describes how the Trojan automatically uses dangerous permission. The value of this feature is also in Boolean.
- **App-APIs (application feature: static)** This feature describes how the Trojan uses third-party APIs designed to get users' information without knowing them. It also describes whether these APIs are running in the background or not. The value of this feature is also in Boolean.
- **Bandwidth-Consumed (user-level feature: dynamic)** This feature describes the bandwidth consumed by a Trojan application for 30 s. It also uses third-party APIs designed to get the users' information without knowing the use of the bandwidth. These APIs start in the background or not. The value of this feature is in Boolean.
- **Is-SMS-Sent (user level feature: dynamic)** This feature shows that the Trojan sends the SMS in the background, gets the user's information without their knowledge, and monitors the sending numbers. The value of this feature is in Boolean.
- **SMS-Quantity (user level feature: dynamic)** This feature describes the number of sending SMS by the Trojan and its injected applications. It checks whether the Trojan is sending messages or not in the background, which gets the user's information without their knowledge.

- **Email-sent (user-level feature: dynamic)** This feature describes that the Trojan sends the email in the background and gets the users' information without their knowledge. It also monitors the number of sent Emails. The value of this feature is in Boolean.
- **Email-Quantity: (user level feature: dynamic)** This feature describes the number of sent emails and the injected applications of Trojans. It also checks that the email is sent in the background and gets information without the user's knowledge. The value of this feature is also in numbers.
- **Sys-Call-Invoked (package level feature: dynamic)** This feature checks the system and whether the application invokes the system calls, e.g., turning off the ringtone, putting the phone into airplane mode, turning on the user location, etc. It checks whether or not the event in the background is getting the users' information without their knowledge. The value of this feature is in Boolean.
- **Sys-Call (application-level feature: static)** This feature describes system calls invoked by the application.

4.3. Classifier Selection

Our classification toolbox includes a decision tree (DT), random forest (RF), logistic regression (LR), and support vector machine (SVM). A decision tree is a tree-structured classifier where branches represent the decision rules. One step further, random forest is an ensemble method that operates by constructing a multitude of decision trees in the form of a forest. Logistic regression is a simple yet powerful classification method that combines features and processes them using the sigmoid function. The support vector machine is a maximum-margin classifier and is the most complex among the four. Altogether, these four methods cover a range of ML classifiers working on different principles [28]. Our toolbox does not include a deep neural classifier because such networks require a large dataset and are computationally expensive.

4.4. Experimental Settings

The proposed work performed extensive tests to verify the effectiveness of the proposed approach for detecting Trojans coming from a comprehensive Android malware dataset. TrojanDetector uses the train/test split method: 80% of the dataset is used for training and 20% for testing.

4.5. Feature Selection

Feature or variable subset selection selects the best predictive feature subset from the original feature set. We performed feature selection using a well-explored Feature Importance method to obtain the top-10 features based on their scores.

Algorithm 1 shows the importance of every feature used in this work for different classifiers, i.e., 2a for DT, 2b for RF, 2c for LR, and 2d for SVM. The x-axis mentions all the features, while the y-axis shows the scores from 0 to 100. We select the top 10 features based on the score and use the reduced feature vectors (10-features long) to train and test our chosen classifiers.

The importance of the features varies according to the chosen ML classifier and is given by the feature importance graphs of the corresponding classifier. Technically speaking, API (F1), system call (F2), CPU usage (F3), and dangerous permission used (F8) are consistently important irrespective of the chosen classifier. On the other hand, energy level (F4) and is-email-sent (F6) are of limited significance for any classifier.

4.6. Experimental Results

The outcome of the chosen binary classifiers is either True Positive Rate (TPR), False Reject Rate (FRR), False Positive Rate (FPR), or True Reject Rate (TRR). TPR is the total number of Trojan apps correctly classified as Trojans, FRR is the total number of Trojan apps incorrectly classified as benign, FPR is the total number of benign apps incorrectly classified as Trojans, and TRR is the total number of benign apps correctly classified as

benign. Equal Error Rate (EER) describes situations where FPR and FRR are equal. We show the evaluation results regarding TPR, EER, and accuracy. We do not report FPR, FRR, and TRR because they can be easily derived as FRR = 1-TPR, and TRR = 1-FPR.

Figure 2 summarises the results of the considered classifiers over the selected features. The DT classifier on chosen features (shown in Figure 3a) is the lowest accurate classifier with 82.25% accuracy, followed by RF with an accuracy of 86.68%. SVM is the most effective algorithm in this scenario. It attains an accuracy of 96.64%, followed by LR with 91.50% accuracy.

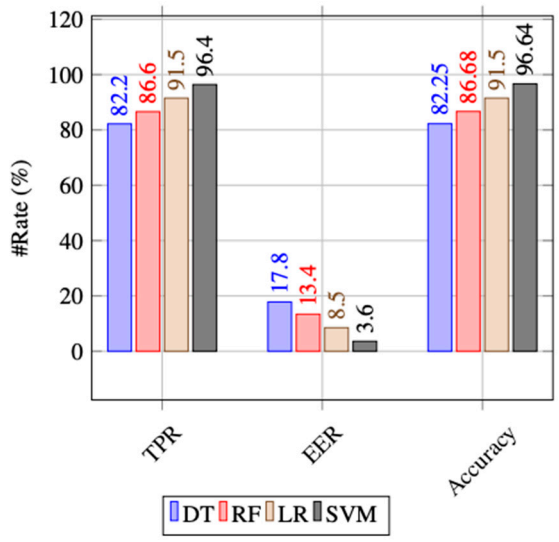

Figure 2. Results of the chosen classifier on selected features.

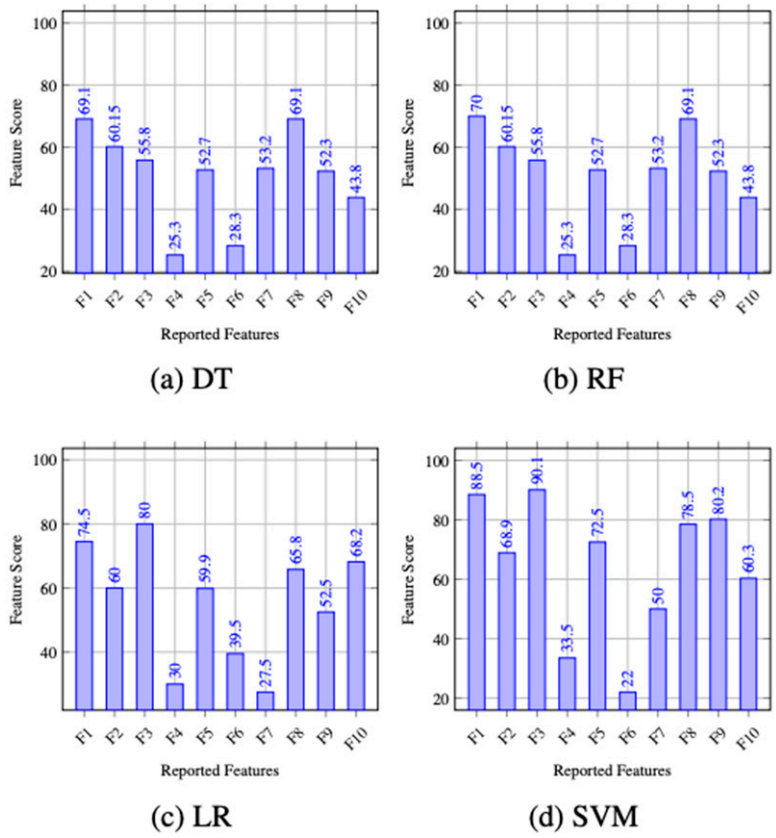

Figure 3. Features importance graph for (**a**) DT, (**b**) RF, (**c**) LR, and (**d**) SVM classifier.

The results of the chosen classifier on selected features indicate that TrojanDetector could be effectively used to differentiate between Trojans and benign applications.

4.7. Discussion on Results

Experimental results on three publicly available datasets, i.e., CCCS-CIC-AndMal-2020, Contagio Mobile, and VirusShare, are promising. The analysis suggests that the TrojanDetector using SVM as the classifier attains the highest accuracy of 96.64%.

This research exploits the decision tree classifier with a depth of 10—the value of min-weight-fraction-leaf to 3 and max-leaf-nodes to 5. DT is normally considered useful on a smaller dataset, which is not the case here. In a large dataset, this classifier normally overfits, resulting in over-prediction. Similarly, we applied RF with just five trees and used the default criterion of (mse, mae). Here, we set the max depth to 5, with min-samples-split to 3 and min-weight-fraction-leaf to 10.

Though the accuracy increases with respect to the DT classifier, it is still lower than the top-performing SVM classifier. This work uses SVM with a Radial Basis Function (RBF) as its kernel function. This algorithm is commonly known as a maximum margin classifier and is utilised to tackle classification problems for a large dataset. There are several kernel selections available for the SVM method. We applied all the available kernels, and the RBF kernel yielded the highest accuracy. There are two tuning parameters for SVM-RBF: kernel parameter sigma (σ) and cost parameter (C) adjusted for repeated training. The sigma value plays a role in getting a good fit model to the data. The cost parameter is the penalty limit if the data point is miss-classified or oversteps the maximum margin. The SVM works better on the margins, and due to the clear margins, the SVM gives the highest accuracy.

4.8. Limitations

TrojanDetector has some limitations: Firstly, it is evaluated on just three publicly available datasets (which we believe are sufficient to draw an initial conclusion). Secondly, it exploited simple classifiers; it would be interesting to check the performance of advanced machine learning classifiers, e.g., CNN, Transformers, etc., on these datasets. Finally, TrojanDetector is analysed in offline settings; it would be interesting to extend TrojanDetector to work in online mode to perform this classification in real-time.

5. Conclusions and Future Works

In this paper, we presented an effective trojan detection technique, namely, TrojanDetector, for detecting Trojans using a multi-layer hybrid approach, i.e., by checking its features at various levels of an Android device. We used four different ML classifiers to detect the Trojan in conjunction with the extracted features. Among them, SVM obtains the best performance. We further identified four features that play a crucial role, irrespective of the classifier. Our work is a step toward providing a safe platform for Android users by effectively detecting Trojans.

In the future, we plan to further improve the proposed method by considering features from the kernel layer of the Android device. We also have a plan for extensive experimental evaluation of TrojanDetector. Like performing ablation tests to measure the impact of each feature (when dropping it) on the classifiers and performing experiments with a particular focus on zero-day exploits. Moreover, we would like to address the limitations of this work, as identified in Section 4.8.

Author Contributions: Conceptualization, S.U., N.Z. and T.A.; methodology, S.U., N.Z. and T.A.; software, N.Z. and S.S.; validation, N.Z., A.B. and S.S.; formal analysis, A.B. and S.S.; investigation, S.U., N.Z. and T.A.; resources, S.U., N.Z. and T.A.; data curation, N.Z., A.B. and S.S.; writing—original draft preparation, S.U., N.Z. and T.A.; writing—review and editing, T.A., A.B. and S.S.; visualization, A.B. and S.S.; supervision, A.B. and S.S.; project administration, S.U.; funding acquisition, A.B. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by the Open Access Publishing Fund of the Free University of Bozen-Bolzano.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Available upon request.

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

References

1. Wotring, B.; Potter, B. *Host Integrity Monitoring: Using Osiris and Samhain*; Books24x7.com: Norwood, MA, USA, 2005.
2. Sharma, H.; Govindan, K.; Poonia, R.C.; Kumar, S.; Wael, M. Advances in Computing and Intelligent Systems. *Springer Nat.* **2022**. [CrossRef]
3. Tong, F.; Yan, Z. A hybrid approach of mobile malware detection in Android. *J. Parallel Distrib. Comput.* **2017**, *103*, 22–31. [CrossRef]
4. Cayron, C. *ARPGE*: A computer program to automatically reconstruct the parent grains from electron backscatter diffraction data. *J. Appl. Crystallogr.* **2007**, *40*, 1183–1188. [CrossRef] [PubMed]
5. Edraki, M.; Karim, N.; Rahnavard, N.; Mian, A.; Shah, M. Odyssey: Creation, Analysis and Detection of Trojan Models. *IEEE Trans. Inf. Forensics Secur.* **2021**, *16*, 4521–4533. [CrossRef]
6. Liu, K.; Xu, S.; Xu, G.; Zhang, M.; Sun, D.; Liu, H. A Review of Android Malware Detection Approaches Based on Machine Learning. *IEEE Access* **2020**, *8*, 124579–124607. [CrossRef]
7. Felt, A.P.; Ha, E.; Egelman, S.; Haney, A.; Chin, E.; Wagner, D. Android permissions: User attention, comprehension, and behavior. In Proceedings of the Eighth Symposium on Usable Privacy and Security—SOUPS '12, Washington, DC, USA, 11–13 July 2012; p. 1. [CrossRef]
8. Baghirov, E. Techniques of Malware Detection: Research Review. In Proceedings of the 2021 IEEE 15th International Conference on Application of Information and Communication Technologies (AICT), Baku, Azerbaijan, 13–15 October 2021; pp. 1–6. [CrossRef]
9. Martín, A.; Lara-Cabrera, R.; Camacho, D. Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the OmniDroid dataset. *Inf. Fusion* **2018**, *52*, 128–142. [CrossRef]
10. Hadiprakoso, R.B.; Kabetta, H.; Buana, I.K.S. Hybrid-Based Malware Analysis for Effective and Efficiency Android Malware Detection. In Proceedings of the 2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS), Jakarta, Indonesia, 19–20 November 2020; pp. 8–12. [CrossRef]
11. Zhao, Y.-l.; Qian, Q. Android malware identification through visual ex-ploration of disassembly files. *Int. J. Netw. Secur.* **2018**, *20*, 1061–1073.
12. Suarez-Tangil, G.; Dash, S.K.; Ahmadi, M.; Kinder, J.; Giacinto, G.; Cavallaro, L. DroidSieve: Fast and Accurate Classification of Obfuscated Android Malware. In Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, Scottsdale, AZ, USA, 22–24 March 2017; pp. 309–320. [CrossRef]
13. Sriyanto; Sahrin, S.B.; Faizal, A.M.; Suryana, N.; Suhendra, A. MiMaLo: Advanced Normalization Method for Mobile Malware Detection. *Int. J. Mod. Educ. Comput. Sci. (IJMECS)* **2022**, *14*, 24–33. [CrossRef]
14. Cai, H.; Meng, N.; Ryder, B.; Yao, D. DroidCat: Effective Android Malware Detection and Categorization via App-Level Profiling. *IEEE Trans. Inf. Forensics Secur.* **2018**, *14*, 1455–1470. [CrossRef]
15. Bokolo, B.; Sur, G.; Liu, Q.; Yuan, F.; Liang, F. Hybrid Analysis Based Cross Inspection Framework for Android Malware Detection. In Proceedings of the 2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA), Las Vegas, NV, USA, 25–27 May 2022; pp. 99–105.
16. Gan, Y.; Qian, H.; Gao, Y. Combining traditional machine learning and anomaly detection for several imbalanced Android malware dataset's classification. In Proceedings of the 2022 7th International Conference on Machine Learning Technologies (ICMLT), Rome, Italy, 11–13 March 2022; pp. 74–80.
17. Qamar, A.; Karim, A.; Chang, V. Mobile malware attacks: Review, taxonomy & future directions. *Futur. Gener. Comput. Syst.* **2019**, *97*, 887–909. [CrossRef]
18. Yuan, Z.; Lu, Y.; Xue, Y. Droiddetector: Android malware characterization and detection using deep learning. *Tsinghua Sci. Technol.* **2016**, *21*, 114–123. [CrossRef]
19. Ahmed, M.F.; Biash, Z.T.; Shakil, A.R.; Ryen, A.A.N.; Hossain, A.; Ashraf, F.B.; Hossain, M.I. ShielDroid: A Hybrid Approach Integrating Machine and Deep Learning for Android Malware Detection. In Proceedings of the 2022 International Conference on Decision Aid Sciences and Applications (DASA), Chiangrai, Thailand, 23–25 March 2022; pp. 911–916.
20. Aminuddin, N.I.; Abdullah, Z. Android Trojan Detection Based on Dynamic Analysis. 2019. Available online: https://fazpublishing.com/acis/index.php/acis/article/view/4 (accessed on 21 August 2022).
21. Kumar, R.; Kumar, P.; Tripathi, R.; Gupta, G.P.; Islam, A.K.M.N.; Shorfuzzaman, M. Permissioned Blockchain and Deep Learning for Secure and Efficient Data Sharing in Industrial Healthcare Systems. *IEEE Trans. Ind. Inform.* **2022**, *18*, 8065–8073. [CrossRef]
22. Kumar, P.; Kumar, R.; Gupta, G.P.; Tripathi, R.; Srivastava, G. P2TIF: A Blockchain and Deep Learning Framework for Privacy-Preserved Threat Intelligence in Industrial IoT. *Trans. Ind. Inform.* **2022**, *18*, 6358–6367. [CrossRef]

23. Kumar, P.; Kumar, R.; Gupta, G.P.; Tripathi, R. BDEdge: Blockchain and Deep-Learning for Secure Edge-Envisioned Green CAVs. *IEEE Trans. Green Commun. Netw.* **2022**, *6*, 1330–1339. [CrossRef]
24. Kumar, R.; Kumar, P.; Tripathi, R.; Gupta, G.P.; Garg, S.; Hassan, M.M. BDTwin: An Integrated Framework for Enhancing Security and Privacy in Cybertwin-Driven Automotive Industrial Internet of Things. *IEEE Internet Things J.* **2022**, *9*, 17110–17119. [CrossRef]
25. 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, Tokyo, Japan, 27–29 November 2020; pp. 70–82. [CrossRef]
26. Mila, P. Contagiodump. Available online: http://contagiodump.blogspot.com/ (accessed on 21 August 2022).
27. Virusshare. Available online: https://virusshare.com/ (accessed on 21 August 2022).
28. Alpaydin, E. *Machine Learning*; MIT Press: Cambridge, UK, 2021.