Evaluating rule-based global XAI malware detection methods

Li, R.; Gadyatskaya, O.; Li, S.; Manulis, M.; Miyaji, A.

Evaluating Rule-Based Global XAI Malware Detection Methods

Rui Li and Olga Gadyatskaya(✉)

LIACS, Leiden University, Leiden, The Netherlands
{r.li,o.gadyatskaya}@liacs.leidenuniv.nl

Abstract. In recent years explainable artificial intelligence (XAI) methods have been applied for interpreting machine learning-based Android malware detection approaches. XAI methods are capable of providing Android malware analysts with some explanations of why a certain sample has been classified as malicious or benign. However, human analysts also have domain-specific requirements, i.e., expectations of how XAI methods should behave. For example, analysts expect that similar malware samples will be explained in a similar way. Recent works by Warnecke *et al.* [41] and Fan *et al.* [13] have proposed domain-specific properties for *local* XAI methods that provide explanations for a single sample.

In this work, we formulate three domain-specific properties for *global* XAI rule-based malware detection methods: *stability*, *robustness* and *effectiveness*. We evaluate performance of five explanation approaches (SIRUS, deepRED, REM-D, ECLAIRE and inTrees) using these metrics. Our experimental results show that the SIRUS method outperforms the other five state-of-the-art methods, with stability, robustness, and effectiveness values of 96.15%, 95.56%, and 91.65% respectively. Our study provides valuable insights for Android malware analysts seeking reliable explanation approaches.

Keywords: Explainability · Android malware · Rule extraction · Evaluation metrics

1 Introduction

In recent years, smartphones have become one of the most indispensable products. According to the report by StatCounter, the Android operating system dominates the smartphone operating system market with a market share of 67.56%, as of June 2023[1]. The flexibility and openness of the Android system have brought great convenience to developers and users. For example, developers can freely develop and upload applications (apps for short) to an application market, and users can download apps from many markets at will.

[1] StatCounter, Mobile Operating System Market Share Worldwide, https://gs.statcounter.com/os-market-share/mobile/worldwide, accessed on 19/06/2023.

Supported by Chinese Scholarship Council.

S. Li et al. (Eds.): NSS 2023, LNCS 13983, pp. 3–22, 2023.
https://doi.org/10.1007/978-3-031-39828-5_1

However, Android provides no assurance of the trustworthiness of apps installed from sources other than Google Play, the official market owned by Google. This makes users vulnerable to malicious software. As reported in the *Mobile malware evolution 2021* report, in that year Kaspersky detected more than 3 million malicious mobile installation packages. Moreover, attacks on mobile users are becoming more sophisticated in terms of both malware functionality and attack vectors[2]. Driven by the need to protect security of the Android operating system and privacy of Android device users, Android malware detection has become a booming research field in recent years [32]. Several effective techniques have been proposed to counter the sheer volume and sophistication of Android malware, frequently based on machine learning (ML) [9,12,22,33].

Nevertheless, most of the machine learning methods applied to malware detection are *black-box*, which means that these methods do not explain how and why certain classification decisions are made. Due to the size of training data and the complexity of the learned model, malware analysts can find it hard to interpret the detection model and explain the decision reasons [15,43]. Moreover, adversarial analyses have shown that only a few changes to the content of a malicious Android app may suffice for evading detection by a ML-based detector [6,10]. Therefore, the analysts can start distrusting the detection results, and doubt whether the detection model can be deployed in practice [7,30,37,44,46].

To make analysts and users trust the ML-based methods, a variety of interpretable models has been proposed to explain predictions. An interpretable model should be *human-simulatable*, which means that a user can "take in input data together with the parameters of the model and in reasonable time step through every calculation required to produce a prediction" [21]. In the mobile security domain, many XAI methods interpret the detection model by identifying important features or extracting important rules [1–3,14,16,18,24,28,31,34,35,40,42,47].

XAI methods can be generally categorized into *local* and *global* methods depending on *what* they strive to explain [5,8,46]. The *local* explanation approaches provide short, human-accessible explanations why a certain *sample* was classified as malicious or benign, while *global* explanations compute a short representation of important indicators across a set of samples, for example, a malware family or a category.

Moreover, there are some domain-specific requirements for XAI methods applied to cyber security in general and to interpretable malware detection in particular [7,30]. Security analysts, for example, can reasonably expect that the provided explanations will be similar for similar apps, and that they will stay similar across different runs of the model. Thus, there is a need to ensure that the explainable ML techniques proposed for malware detection satisfy these requirements.

Addressing this problem, Warnecke *et al.* [41] and Fan *et al.* [13] have proposed formalizations of several domain-specific requirements for Android mal-

[2] Kaspersky https://securelist.com/mobile-malware-evolution-2021/105876/, accessed on 19/06/2023.

ware detection. They have independently proposed several metrics that can be used to assess the quality of local explainable ML techniques when applied to Android malware, i.e., the explanation approaches applied to the classification of individual samples. For example, Fan *et al.* [13] have shown that explanation results provided by five *local* explanation approaches for the same Android malware sample cannot achieve a consensus in general. However, to the best of our knowledge, there has so far been no investigation of properties for global explainability Android malware detection methods, i.e., methods that provide a single explanation for a set of samples, and there has been no evaluation of existing global XAI techniques with respect to the domain-specific requirements of Android security analysts.

In this paper, we aim to close this gap. Specifically, we make the following contributions:

1. We formulate three metrics – *stability*, *robustness*, and *effectiveness*– to assess the fundamental properties that rule-based global explainable Android malware detection methods should satisfy. These metrics are crucial for evaluating the performance of such methods.
2. We evaluate performance of five state-of-the-art explanation methods, namely, SIRUS [4], DeepRED [48], REM-D [36], ECLAIRE [45], and inTrees [11], for Android malware detection using the CICMalDroid dataset [25,26]. Our experimental results show that SIRUS outperforms the other four state-of-the-art global XAI methods in terms of detection performance, stability, robustness, and effectiveness.

2 Related Work

As mentioned, the explainable Android malware detection methods can be divided into *local* explanation methods and *global* explainable methods [5]. Another dimensions to categorize XAI methods refer to the connection of the interpreter to the ML model: some XAI approaches are *intrinsic*, as it is the ML model itself that is interpretable [7,8,46]. All linear classifiers are intrinsically interpretable. In the Android malware research such approaches are, for example, Drebin [2], Traffic AV [40], LUNA [3], and CASANDRA [31], where linear Support Vector Machine, Decision Tree, Bayesian Classifier, and Online CW Classifier are used, respectively. Other XAI approaches are *post-hoc/extrinsic*, as they can be applied to any ML model after training.

The local explanation methods explain why a single Android application was labeled as malware or benign via a detection model. Several research teams have investigated application of local XAI methods for Android malware detection. For example, Fan *et al.* [13] have applied five widely-used *post-hoc* explanation approaches – LIME [34], Anchor [35], LORE [14], SHAP [24] and LEMNA [16] – to Android malware detection. Alani and Awad [1] present the PAIRED lightweight Android malware detection system that integrates SHAP as an interpreter. Morcos *et al.* [29] propose a surrogate-based technique to interpret Random Forest models that integrates SHAP for interpreting the data exfiltrating behavior in Android malware samples.

Martin *et al.* [18] propose a method to explain convolutional neural networks (CNNs) by calculating network activations to identify locations deemed important in an Android app's opcode sequence. Zhu *et al.* [47] develop a backtracking method to infer important suspicious features of apps for explaining classification results.

Melis *et al.* [28] identify the most influential malicious local features by leveraging a gradient-based approach, which enables using nonlinear models to increase accuracy without sacrificing the interpretability of decisions. Lu and Thing [23] develop the PhilaeX method to identify a subset of features for explaining decisions of different AI models, including Support Vector Machines implemented by Drebin [2] and BERT (a transformer-based deep neural network classifier).

The global explanation methods let analysts understand how the whole model makes decisions. Some approaches do this based on a holistic view of model's features and the learned components, such as weights, parameters, and structures. For example, Bozhi *et al.* [42] propose a global XAI approach called Xmal that not only pinpoints the key features most related to the classification result by hingeing the multi-layer perceptron and attention mechanisms, but also automatically produces natural language descriptions to help analysts to interpret malicious behaviours within apps.

Other global XAI methods extract a rule-based representation of the targeted set from the model. For example, AdDroid [27] is based on various combinations of artefacts called rules to analyze and detect malicious behaviour in Android applications. Jerby *et al.* [17] develop the BMD method for malware detection rules generation using a Bi-Level optimization problem. Both AdDroid and BMD are *intrinsic* XAI methods.

Still, explanation methods in security need not only to be accurate but also to satisfy domain-specific requirements, such as complete and robust explanations [41]. This is arguably especially important when *post-hoc* XAI methods are being applied to malware detection, as they are by design not aware of the underlying classification problem. To evaluate how well this problem is addressed by the existing XAI methods in the field, Warnecke *et al.* propose general evaluation criteria, which include descriptive accuracy and sparsity, and security (domain-specific) evaluation criteria, which include stability, robustness, and efficiency. Independently, Fan *et al.* [13] propose stability, robustness, and effectiveness to evaluate the Android malware detection explanation results. We discuss these metrics in more detail in Sect. 5.3. However, they have been developed only for local explanation methods. Our work aims to close this gap and to formulate domain-specific quality metrics to be used in conjunction with rule-based global XAI methods.

3 Candidate XAI Methods and Evaluation Metrics

3.1 Evaluated XAI Methods

In this work we focus on evaluating the *post-hoc* XAI methods, as we were not able to find implementation of the intrinsic XAI approaches AdDroid [27] and BMD [17]. The following global rule-based XAI methods have been selected for experiments:

- The **SIRUS** algorithm [4] is a stable and interpretable rule-based classifier that consists of two main processes: training a black-box model, such as Random Forest (RF), and constructing an *agent* model that extracts rules and generates a rule-based malware detector. The goal of SIRUS is to identify a concise set of non-overlapping detection rules that capture robust and strong patterns in the data [4].
- The **inTrees** method [11] extracts, measures, prunes, selects, and summarizes rules from a tree ensemble (such as RF and Boosted Trees), the rules can be ranked by length, support, error, or a combination of multiple metrics.
- The **deepRED** method [48] extracts rules from deep neural networks by mimicking the internal logic of neural networks at each layer and neurone. This makes hidden logic and features accessible, and also exploits deep structures to improve the efficacy of rule extraction and induction process.
- The **REM-D** (Rule Extraction Methodology-Deep Neural Network) method [36] approximates a deep neural network (DNN) with an interpretable ruleset model and uses that ruleset to explain the results of the DNN. For approximation, REM-D first decomposes the trained DNN into adjacent layers and then uses the C5.0 classification algorithm to extract rules from pairs of layers in the network.
- The **ECLAIRE** (Efficient CLAuse-wIse Rule Extraction) method [45] is a polynomial-time decompositional method applicable to arbitrary DNNs; it exploits intermediate representations in parallel to build an ensemble of classifiers that can then be efficiently combined into a single rule set.

Interpreted Classifiers. Note that the selected interpreters work with two types of classifiers: either the RF or DNNs. These classifiers are quite different from one another, and rules generated with them will be quite distinct. One of the goals of our work is to understand whether the established domain-specific requirements previously considered for local methods can be expected at all from deep neural network interpreters.

The RF algorithm builds multiple independent decision trees following the bagging strategy, using both sample and attribute selection to prevent overfitting. This decision-making tree generation process helps to avoid the problem of under-fitting caused by single tree judgments and significantly improves discrimination. Finally, the model combines the predictions of multiple trees to make category determinations through a voting method, resulting in higher accuracy.

A DNN consists of a sequence of multiple layers of interconnected neurones. Each neurone in a layer receives input from the previous layer's neurones and performs a computation that typically involves a weighted sum of the inputs followed by a nonlinear activation function. The network's neurones collectively implement a complex nonlinear mapping from the input to the output, which is learned from data by adjusting the weights of each neurone using error back-propagation. This technique involves propagating the prediction error back through the layers of the network to adjust the weights of the connections between neurones. By adjusting the weights in this way, the DNN is able to learn complex patterns and relationships within the data, leading to improved accuracy and robustness in many machine learning tasks.

To explain a *black-box* detection model (an RF or a DNN-based classifier), we apply one of the above explanation methods to extract rules, select rules and generate a new *rule-based detector*, which is now an intrinsically interpretable (*white-box*) classification model. These detectors are then evaluated based on their detection performance and also the proposed domain-specific properties, as we discuss further.

Rules. Rules produced by the considered XAI methods are in the format "`if` $f_1\&f_2\&......f_n$ `then` p_1 `else` p_2", where the conjunction of conditions $f_1\&f_2\&......f_n$ is the detection rule body, p_1 is the model's confidence that the sample is malware under the given condition in the rule body, and p_2 is the confidence that the sample is malware when the condition is not satisfied.

3.2 Evaluation

The literature [13,41] proposes domain-specific metrics for local XAI methods applied to malware detection. However, these metrics are based on feature sets, rather than rules, and are only suitable for local explanation methods. In our study we consider global, rule-based explanation methods. Thus we propose new definitions of the *stability*, *robustness*, and *effectiveness* metrics previously defined in [13,41] to suit rule-based global explanation methods. Table 1 lists the used notations.

Intuition 1. ***Stability*** *requires that the generated explanations result do not vary between multiple runs* [41].

Since the explanation results remain similar on the same pre-trained models, good stability requires that an explanation approach can really capture the actual reason for an individual classification decision. Otherwise, the analyst would be confused and would not trust the explanation results [13]. The stability of an explanation method m, denoted as $stb(m,T)$, is measured on a target testing dataset T as follows.

$$stb(m,T) = \frac{1}{C_n^2} * \sum_{i,j \epsilon n,} sim(e_{n_i}(g), e_{n_j}(g)) \tag{1}$$

Table 1. Notations and definitions

Notation	Definition
C_n^k	the number of combinations for selecting k elements out of n
f	a classifier model constructed on a training dataset
m	an explanation approach
$g = m(f)$	a specific interpreter constructed based on an explanation method m and a trained classifier f
$e(g)$	the explanation results of the samples with interpreter g
$stb(m, T)$	stability of explanation approach m on testing dataset T
$rob(m, T)$	robustness of explanation approach m on testing dataset T
$eff(m, T)$	effectiveness of explanation approach m on testing dataset T

$$g = m(f) \tag{2}$$

$$sim(e_{n_i}(g), e_{n_j}(g)) = 2 * \frac{e_{n_i}(g) \cap e_{n_j}(g))}{|e_{n_i}(g)| + |e_{n_j}(g)|)} \tag{3}$$

where n is the number of times that the experiment is repeated; C_n^2 denotes the number of pre-trained models and is bigger than two. g is a specific interpreter constructed based on an explanation method m and a trained classifier f, and $e_{n_i}(g)$ is the n_i-th explanation results of the samples with the interpreter g. $sim(e_{n_i}(g), e_{n_j}(g))$ is the similarity between $e_{n_i}(g)$ and $e_{n_j}(g)$ based on the Dice coefficient.

The main difference between our stability metric and the one in [13] is that we measure the malware explanation results for all the samples jointly rather than each sample individually. Moreover, we compare all rules in different runs rather than the top-k features.

Intuition 2. ***Robustness*** *is an ability of the explanation method to remain unaffected when slight variations are applied* [13].

Robustness is used to measure how similar the explanation results are for similar instances. Intuitively, the explanation results of similar malware instances should be highly similar. The robustness formula proposed in [13] requires that every sample has an individual explanation, which is not suitable for our work. So according to the intuition of robustness, we propose a new robustness evaluation metric that is based on variations in the whole dataset.

The dataset $T_t(x)$ ranges from $sampleX_{0+(t-1)*u}$ to $sampleX_{r+(t-1)*u}$, $t \in (0, \beta)$, β is equal to total number of samples minus r, and then divided by u. The robustness of an explanation approach p on the dataset T is calculated as below. For example, if $i = 0$, that means we should calculate the similarity of $[X_0, X_1, X_2......X_r]$ and $[X_1, X_2, X_3......X_{r+1}]$.

$$rob(p, T) = \frac{1}{t} * \sum_{i \in t} sim(T_i(g), T_{i+1}(g)), t \in (0, \beta)] \tag{4}$$

Intuition 3. ***Effectiveness*** *measures whether the explanation results are important to the decision-making* [13]*. If the explanation results are really the decision basis for an individual prediction, the classification result would change after mutating rules* [13]*.*

To compute effectiveness, we mutate the produced rules. First, the confidence score of rules above 0.5 will be set 1 (indicates malware), otherwise set to 0 (indicates benign). For instance, a rule "if `android.permission.DISABLE_KEYGUARD` < 1 & `android.permission.SEND_SMS` < 1 `then` *0.0031* `else` *0.95*" will change to "if `android.permission.DISABLE_KEYGUARD` < 1 & `android.permission.SE-ND_SMS` < 1 `then` *0* `else` *1*".

The effectiveness $eff(m,T)$ of an explanation approach m on testing dataset T is then calculated as below.

$$eff(m,T) = \frac{1}{|T|} * \sum_{x_i \epsilon T} eff(m,x_i) \tag{5}$$

$$eff(m,x_i) = \begin{Bmatrix} 1, \hat{y_i^*} \neq \hat{y_i}, \hat{y_i^*} \in \hat{Y^*} \\ 0, \hat{y_i^*} = \hat{y_i}, \hat{y_i^*} \in \hat{Y^*} \end{Bmatrix} \tag{6}$$

$$\hat{Y^*} = f(X^*) \tag{7}$$

$$X^* = mutate(x, e_{(g)}) \tag{8}$$

where $eff(m,x_i)$ denotes the effectiveness of explanation results for the sample x_i with m. The hat sign $\hat{}$ denotes the classification result. X^* is a new ruleset for the samples by mutating the original ruleset X, $\hat{Y^*}$ are classification results of X^*; it is a vector with values 0 or 1. If the *mutate* classification result $\hat{y_i^*}$ is not equal to the original classification result $\hat{y_i}$, $eff(m,x_i)$ is assigned to 1, indicating that rules are important to the current decision-making. Otherwise, $eff(m,x_i)$ is set to 0, indicating that rules are useless to predict malware.

The *mutate* operator in [13] changes the value of a feature that appears in the explanation results. However, in our method, *mutate* applies the logical negation to the rule body by, for instance, changing the rules from the AND condition to the OR condition. Moreover, the conflicting rules are deleted.

As en example, the above rule "if `android.permission.DISABLE_KEYGUARD` < 1 & `android.permission.SEND_SMS` < 1 `then` *0* `else` *1*" will be mutated into a set with two rules: 1) "if `android.permission.DISABLE_KEYGUARD` >= 1 `then` *0* `else` *1*"; and 2)"if `android.permission.SEND_SMS` >= 1 `then` *0* `else` *1*".

4 Methodology

In this section, we introduce our set-up for evaluating the global rule-based XAI Android malware detection methods according to the proposed definitions

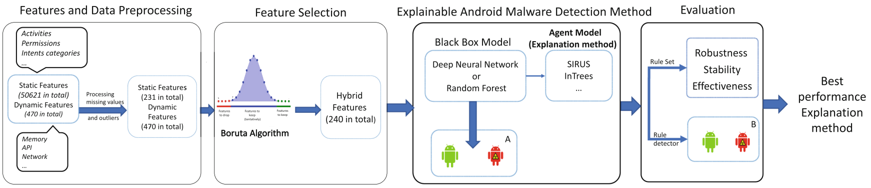

Fig. 1. Our set-up for evaluating global rule-based XAI Android malware detection methods

of stability, robustness and effectiveness, and their performance as white-box classifiers.

Our process is divided into 4 steps: data and feature preprocessing, feature selection, explainable Android malware detection process, and evaluation, as shown in Fig. 1. These steps are further detailed in the remainder of this section. In a nutshell, we first preprocess the data by treating the missing and outlier values. Next, we select the important features using the Boruta algorithm [20]. Then, a black-box malware detection model based on the Random Forest algorithm or Deep Neural Network is trained (the detection task A), and we use the five selected agent models (explanation methods) – SIRUS [4], DeepRED [48], REM-D [36], ECLAIRE [45], and inTrees [11] – to produce rule-based detectors which can detect Android malware (the detection target B). Finally, we evaluate the rules extracted by the considered XAI methods based on their robustness, stability, effectiveness, and also the performance of rule detectors.

4.1 Features and Data Preprocessing

Android apps are software applications running on the Android platform. A typical Android app contains different components: activities, fragments, services, content providers, and broadcast receivers. Most of these app components should be declared in the special Manifest file. This file is used to decide how to integrate the app into the device's overall user experience by the Android OS. The apks (Android application packages) are used to install Android apps onto device. Different properties of apks are used to detect malicious apps as features. These features are usually extracted using some program analysis techniques [39].

For static analysis of Android apps, apks should be unzipped and decompiled. The Manifest file (`AndroidManifest.xml`) and the code file (`classes.dex`) are usually used in static analysis. *Static features* like sensitive permissions, names of activities, and intents are extracted from the Manifest file, while sensitive API calls are extracted from the code file [2].

For dynamic analysis of the Android apps, apps should be executed in a dedicated analysis environment, like, e.g., CopperDroid [38], to automatically reconstruct low-level OS-specific and high-level Android-specific behaviours of Android apps [26]. *Dynamic features* like system calls and Binder calls could be extracted in dynamic analysis. We refer the interested reader to a survey by Tam *et al.* [39] on Android malware detection techniques for more details on static and dynamic analysis techniques and features.

Due to the challenges with the automatic processing of third-party Android apps [19], there might be outliers or missing values in the collected data. We therefore apply the usual data preprocessing step to remove the features with missing values, transform categorical values into numeric values, etc.

4.2 Feature Selection

After completing the data collection and preprocessing steps, we apply the Boruta algorithm [20] to eliminate redundant and irrelevant features from our dataset. The Boruta algorithm is effective in minimizing the impact of random fluctuations and correlations during feature selection [20]. The approach involves augmenting the original features with a set of shadow features, which are randomized copies of the original features. To identify the most important features in the dataset, the Boruta algorithm trains a classifier using the extended feature set. It then compares the importance of each original feature with that of its corresponding shadow feature. If a feature has higher importance than its corresponding shadow feature, it is considered important. This process is repeated until all features are either confirmed as important, rejected as unimportant, or remain uncertain.

5 Experimental Evaluation

5.1 Dataset

A generic Android malware detection pipeline requires automated app analysis tooling to extract static and dynamic features. In our work we use the CICMalDroid [25,26] dataset, which already contains static and dynamic features extracted from 1795 benign and 9803 malware samples. The samples were collected from December 2017 to December 2018. The dataset includes five distinct categories: Adware, Banking, SMS malware, Riskware and Benign, as shown in Table 2.

The dataset includes 470 dynamic features, such as frequencies of system calls, Binder calls, and composite behaviours; and 50,621 static features, such as intent actions, permissions, sensitive APIs, services, etc. To balance the dataset for experiments, we randomly subselected 1795 benign and 1795 malware samples.

5.2 The Experiment Procedure

To ensure the integrity of the dataset and minimize the impact of outliers and missing values, we undertook the preprocessing steps before analysis. The dataset initially included 50,621 static features and 470 dynamic features. We removed features with missing values exceeding 90%, and converted object-type features

Table 2. The details of CICMalDroid dataset categories

Category	Description	# of samples
Adware	Adware can infect and root-infect a device, forcing it to download specific Adware types and allowing attackers to steal personal information	1253
Banking	Mobile Banking malware is a specialized malware designed to gain access to the user's online banking accounts by mimicking the original banking applications or banking web interface	2100
SMS malware	SMS malware exploits the SMS service as its medium of operation to intercept SMS payload for conducting attacks. They control attack instructions by sending malicious SMS, intercepting SMS, and stealing data	3940
Riskware	Riskware refers to legitimate programs that can cause damage if malicious users exploit them. Consequently, it can turn into any other form of malware such as Adware or Ransomware, which extends functionalities by installing newly infected applications	2546

such as `incognito.is valid APK` (with values of `True` or `False`) to integer-type values. After these preprocessing steps, we reduced the number of static features to 231, while retaining all 470 dynamic features.

With the resulting 701 features, we have applied the Boruta algorithm for feature selection that has identified 240 important features[3].

Then, the 10-fold cross-validation method is used to do the experiment. We use the training set to train a Random Forest detection model and a Keras DNN model[4].

Next, we extract rules from the trained black-box detection models using the studied SIRUS, deepRED, REM-D, ECLAIRE and inTrees methods. The maximum rule number of the SIRUS algorithm was set to 200; the hyperparameter $p_0 = \langle p_0.pred, p_0.stab \rangle$ is used to select rules, where $p_0.pred$ minimizes the error and $p_0.stab$ finds a tradeoff between error and stability. In the SIRUS algorithm, the error means 1-AUC for classification and the unexplained variance for regression, and stability refers to the average proportion of rules shared by two SIRUS

[3] The settings of the Boruta algorithm were: the Random Forest classifier, auto estimators, `verbose` is set to 2, `random state` is set to 1, number of trees is set to 200.

[4] The parameters of Keras included: last-layer activation – softmax, loss function – softmax_xentr, and learning rate – 0.001.

models fit on two distinct folds in the cross-validation. We choose $p_0.pred$ as an optimal hyperparameter.

In the REM-D method, *trials* (the number of sampling trials to use when using bagging for C5.0 rule extraction) is set to 1, *min_cases* ((the minimum number of samples we must have to perform a split in a decision tree)) is set to 30. In the ECLAIRE method, *min_cases* is set to 30, *block_size* (the hidden layer sampling frequency) is set 1, *ccp_prune* (whether or not we perform the post-hoc cost complexity pruning in the trees we extract with CART before rule induction) is set to `True`. In the deepRED method, *min_cases* is set to 20, *ccp_prune* is set to `True`, *trials* is set to 1.

5.3 Evaluation Metrics

We will evaluate the performance of black-box models, white-box models and rules using detection performance metrics and domain-specific metrics. Specifically, we will use detection performance metrics, such as accuracy, precision, recall, and F-measure, to evaluate the performance of both black-box models (i.e., RF, DNNs) and white-box models (i.e., rule detectors). To do so, we will use the standard confusion matrix, which summarizes the number of true positives, false positives, true negatives, and false negatives (see Table 3). Using this matrix, Table 4 provides the definitions of accuracy, precision, recall, and F-measure, which are commonly used to evaluate the performance of classification models and rule detectors. In addition to these metrics, we will use domain-specific metrics that we proposed (see Sect. 5.3) to evaluate the stability, robustness, and effectiveness of rules extracted from the black-box models using five explanation methods.

5.4 Experimental Results

The performance of black-box models (RF and DNNs) in the 10-fold validation scheme on the pre-processed CICMalDroid dataset is presented in Table 5. The performance of agent models (the considered explanation methods) on the same dataset are shown in Table 6, where stability, robustness, effectiveness are evaluated on the produced rulesets, while accuracy, precision, recall, and F-measure are evaluated on the generated rule detectors.

Table 3. Confusion matrix

Truth	Prediction	
	Malware	Benign
Malware	True Positive (TP)	False Negative (FN)
Benign	False Positive (FP)	True Negative (TN)

Table 4. Definitions of detection performance evaluation metrics

Term	Description
Precision	$\frac{TP}{TP+FP}$
Recall (Detection Rate)	$\frac{TP}{TP+FN}$
Accuracy	$\frac{TP+FN}{TP+TN+FP+FN}$
F-measure	$\frac{2*Recall*Precision}{Recall+Precision}$

Table 5. Performance of black-box models

Metric	RF	DNN
Accuracy	98.97%	95.54%
Precision	99.22%	96.73%
Recall	98.72%	93.94%
F-measure	98.74%	95.32%

Table 6. Performance of the chosen explanation methods

Metric	SIRUS (RF)	inTrees (RF)	deepRED (DNN)	REM-D (DNN)	ECLAIRE (DNN)
# of rules	55	12	3	2	2
Stability	96.15%	0%	0%	0%	0%
Robustness	95.56%	0%	0%	0%	0%
Effectiveness	91.65%	86.64%	–	–	–
Accuracy	92.47%	88.19%	88.99%	86.35%	92.34%
Precision	87.20%	91.70%	88.29%	87.85%	86.87%
Recall	99.82%	87.11%	89.05%	76.08%	93.95%
F-measure	93.09%	87.75%	88.67%	81.54%	91.16%

To compute the robustness score according to the Eq. 4 in our experiments, the variation parameter u was set 10, the r was set to 2000, the total number of samples is 3580 (1790 malware and 1790 benign samples).

The deepRED, REM-D, and ECLAIRE methods output less than 5 rules; the inTrees method produces 12 rules; and SIRUS produces 55 rules. Although the ECLAIRE method only has 2 rules, they contain above 100 features per rule. In contrast, SIRUS contains less than 3 features per rule.

The stability and robustness of SIRUS are above 95%. The other considered XAI methods all have very low stability and robustness (0%). These results suggest that SIRUS has higher potential as explainable Android malware detection method, as it shows high stability and robustness. It is known that the stable and robust XAI methods will improve the human trust and will not confuse the analyst [13]. At the same time, the state-of-the-art methods inTrees, deepRED, REM-D and ECLAIRE seem to have much higher variability of the rule conditions. It will be interesting to investigate how to improve stability and robustness of such methods. Otherwise, explanation results provided by these methods can be regarded by human analysts as meaningless as they would not understand how the detection model works [13].

The effectiveness of SIRUS is 91.65%, which is higher than the inTrees method. We note that we could not compute effectiveness of deepRED, REM-D and ECLAIRE as they use transformed features to generate rules, which do not correspond to the app features from the original dataset. Thus, we could not define a meaningful mutation procedure for them. As an example, the explanation results produced by these DNN-based methods can look like the following: “ `if (0.4975 | 1.0000)[(`h_0_0 $\leq$ `9805) AND (`h_0_263 $\leq$ `25)] ... OR (0.9746 | 1.0000)[(`h_0_143 `> 40002)] then 1`”. Therefore, we can conclude our formulation of the effectiveness metric needs to be improved in the future to cover this case.

We examined the accuracy, precision, recall and precision of black-box models (RF and DNN) and the produced rule-based classifiers (agent models). Compared to black-box models, detection performance of agent models has decreased. This is understandable, as the agent models are based on rules extracted from black-box models. For the rules to be readable and less complex, the neural network or trees should be pruned, which leads to loss of detection performance compared to the black-box models.

We note that accuracy of SIRUS is 92.47%, which is higher than the other considered methods. The precision of inTrees is 91.70%, which is the highest in all methods. The value of recall and F-measure of SIRUS is 99.82% and 93.09%; higher than the other comparison methods. Overall, these results suggest that SIRUS has acceptable stability, robustness, effectiveness, and detection performance (as measured by accuracy, recall, and F-measure). The inTrees method has better precision in malware detection.

6 SIRUS Rules

We have demonstrated in our experiments that the SIRUS method could be considered a viable XAI solution for Android malware detection. We now give examples of some detection rules produced by SIRUS.

The detection rule body produced by SIRUS is a conjunction of logic conditions f_i in the path from the root node to the current node in the tree. We note that SIRUS takes care of removing overlapping rules. Therefore the generated 55 rules are not redundant. Five examples of the extracted rules are shown in Table 7. To help understand how to read the rules, they are explained below.

Rule 1: `If TelephonyManager.getLine1Number` $<$ 2 & `TelephonyManager.getSubscriberId` $<$ 1 `then` *0.04* `else` *0.87*.

Explanation: `TelephonyManager.getLine1Number` is an API that obtains a phone number, `TelephonyManager.getSubscriberId` is an sensitive API that gets device information. The value of these features represents the count of API calls in the code. These are all sensitive behaviors, that might lead to private user data leakage. So this rule means: if an application tries to access the phone

Table 7. Example rules extracted by the SIRUS method

ID	Rules
1	`if TelephonyManager.getLine1Number` < 2 & `TelephonyManager.getSubscriberId` < 1 `then` *0.04* `else` *0.87*
2	`if Android.permission.SEND_SMS` < 1 & `removeAccessibilityInteractionCon nection` < 3 `then` *0.012* `else` *0.97*
3	`If TelephonyManager.getCellLocation` < 3 & `TelephonyManager.getSubscriberId` < 1 `then` *0.05* `else` *0.88*
4	`if Android.intent.action.PACKAGE_ADDED` < 1 & `getInstallerPackageName` ≥ 1 `then` *0.0089* `else` *0.76*
5	`if Android.permission.READ_PHONE_STATE` < 1 & `target_sdk` < 19 `then` *0.24* `else` *0.52*

number at least 2 times or calls for device information, then there is a 87% possibility that it belongs to malware.

Rule 2: `If Android.permission.SEND_SMS` < 1 & `removeAccessibilityInteractionConnection` < 3 `then` *0.012* `else` *0.97*.

Explanation: `Android.permission.SEND_SMS` is a permission that is required to send SMS messages, the value of this feature is 0 means the app without requesting this permission, otherwise means the app has been granted the corresponding permissions. `RemoveAccessibilityInteractionConnection` is a dynamic behavior to consume lots of system memory, which can reduce the app's speed or lead to crashes. So this rule means: if the application request the `SEND_SMS` permission or makes the system unstable by removing the accessibility interaction connection more than 3 times, there is a 97% possibility that it belongs to malware. It indicates that this is a strong rule to identify malware.

Rule 3: `If TelephonyManager.getCellLocation` < 3 & `TelephonyManager.getSubscriberId` < 1 `then` *0.05* `else` *0.88*.

Explanation: `TelephonyManager.getCellLocation` is an API that obtains the location information; `TelephonyManager.getSubscriberId` is an API that obtains device information. The value of these features represents the count of API calls in the code. This rule means that if an application calls for the user's location more than 3 times or tries to access the phone number, then there is a 88% possibility that it is malware.

Rule 4: `If Android.intent.action.PACKAGE_ADDED` < 1 & `getInstallerPackageName` ≥ 1 `then` *0.0089* `else` *0.76*.

Explanation: `Android.intent.action.PACKAGE_ADDED` is an action that notifies of an apk package added to the system; `GetInstallerPackageName` is an API that obtains the source of the package, it could be from Google Play or other third-party markets. The large number of these features might be indicate there are abnormal frequent application installation behaviors or a large number of installations of untrusted application packages.

This rule means that if an application is notified about added more than one packages or the app does not show the source of the apk package, then there is a 76% possibility that it is malware.

Rule 5: `If Android.permission.READ_PHONE_STATE` < 1 & `target_sdk` < 19 `then` *0.24* `else` *0.52*.

Explanation: `Android.permission.READ_PHONE_STATE` is a permission that allows read-only access to phone states, such as phone numbers, network information, and device identifiers. The value of 0 for this feature indicates that the app has not requested the `READ_PHONE_STATE` permission, while a non-zero value indicates that the app has been granted this permission. `Target_sdk` is the app Android SDK target version. This rule means that if an application tries to access the phone state or has the target SDK version above 19, there it is a 52% possibility that it is malware. So it is not a very strong rule for the analysts to distinguish the malware.

7 Conclusion

In this study, we aimed to evaluate the quality of rule-based global XAI methods in the context of Android malware detection and to provide useful insights for malware analysts regarding the existing post-hoc XAI approaches. To achieve this goal, we formulated three domain-specific properties to measure the quality of the detection methods: *stability*, *robustness*, and *effectiveness*. Using these metrics, we evaluated five state-of-the-art explanation approaches using the CICMalDroid dataset. Our work investigating domain-specific evaluation metrics for global rule-based explanation methods extends the elegant works by Fan *et al.* [13] and Warnecke *et al.* [41]. They proposed domain-specific properties for *local* XAI methods that provide explanations for a single sample. However, we now focus on rule-based *global* XAI methods.

Our experimental results demonstrate that these evaluation metrics can assess the rule-based global XAI approaches, providing valuable insights for researchers and practitioners. Specifically, we found that the SIRUS method can generate stable, robust, and effective rules with high detection performance, outperforming other state-of-the-art methods that were evaluated in our study. Indeed, in our experiments, the deepRED, REM-D, and ECLAIRE methods show zero stability and robustness: this means that with every run the produced rules are different and any small change will change the explanation results. These methods can still provide valuable malware-related information to human analysts, but the analysts might become confused receiving constantly changing explanations.

Our findings highlight the importance of evaluation metrics in assessing the quality of rule-based global XAI Android malware detection methods. The proposed metrics can provide useful guidance for researchers and practitioners working in this field, helping them to select the most effective and reliable detection methods.

In the future work, we intend to focus on improving the proposed metrics in discussion with practitioners, to be able to propose new domain-specific metrics definitions that will capture important properties while being computable for the vast majority of available XAI methods. In addition, we are interested in exploring the impact of the number of rules on the performance of explanation methods. This is an important consideration, as it can help us to better understand the trade-offs involved in using larger rulesets versus smaller ones, and to identify the optimal ruleset size. Finally, we are interested in extending our metrics definition to cover global XAI methods relying on interpretations in terms of significant features rather than rules.

Acknowledgements. This research was partially supported by the Chinese Scholarship Council (CSC).

References

1. Alani, M., Awad, A.: PAIRED: an explainable lightweight Android malware detection system. IEEE Access **10**, 73214–73228 (2022)
2. Arp, D., Spreitzenbarth, M., Hübner, M., Gascon, H., Rieck, K.: DREBIN: effective and explainable detection of android malware in your pocket. In: Symposium on Network and Distributed System Security (NDSS) (2014)
3. Backes, M., Nauman, M.: LUNA: quantifying and leveraging uncertainty in Android malware analysis through Bayesian machine learning. In: 2017 IEEE European Symposium on Security and Privacy, Los Alamitos, CA, USA, pp. 204–217. IEEE (2017)
4. Bénard, C., Biau, G., Da Veiga, S., Scornet, E.: SIRUS: stable and interpretable rule set for classification. Electron. J. Stat. **15**(1), 427–505 (2021)
5. Burkart, N., Huber, M.F.: A survey on the explainability of supervised machine learning. J. Artif. Intell. Res. **70**, 245–317 (2021)
6. Calleja, A., Martín, A., Menéndez, H.D., Tapiador, J., Clark, D.: Picking on the family: disrupting Android malware triage by forcing misclassification. Expert Syst. Appl. **95**, 113–126 (2018)
7. Capuano, N., Fenza, G., Loia, V., Stanzione, C.: Explainable artificial intelligence in cybersecurity: a survey. IEEE Access **10**, 93575–93600 (2022)
8. Charmet, F., et al.: Explainable artificial intelligence for cybersecurity: a literature survey. Ann. Telecommun. **77**, 1–24 (2022)
9. Dashevskyi, S., Zhauniarovich, Y., Gadyatskaya, O., Pilgun, A., Ouhssain, H.: Dissecting Android cryptocurrency miners. In: Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy, pp. 191–202 (2020)
10. Demontis, A., et al.: Yes, machine learning can be more secure! a case study on android malware detection. IEEE Trans. Dependable Secure Comput. **16**(4), 711–724 (2017)

11. Deng, H.: Interpreting tree ensembles with intrees. Int. J. Data Sci. Anal. **7**(4), 277–287 (2019)
12. Dhalaria, M., Gandotra, E.: Android malware detection techniques: a literature review. Recent Patents Eng. **15**(2), 225–245 (2021)
13. Fan, M., Wei, W., Xie, X., Liu, Y., Guan, X., Liu, T.: Can we trust your explanations? Sanity checks for interpreters in Android malware analysis. IEEE Trans. Inf. Forensics Secur. **16**, 838–853 (2020)
14. Guidotti, R., Monreale, A., Ruggieri, S., Pedreschi, D., Turini, F., Giannotti, F.: Local rule-based explanations of black box decision systems. arXiv preprint arXiv:1805.10820 (2018)
15. Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., Pedreschi, D.: A survey of methods for explaining black box models. ACM Comput. Surv. (CSUR) **51**(5), 1–42 (2018)
16. Guo, W., Mu, D., Xu, J., Su, P., Wang, G., Xing, X.: LEMNA: explaining deep learning based security applications. In: proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS), pp. 364–379 (2018)
17. Jerbi, M., Chelly Dagdia, Z., Bechikh, S., Ben Said, L.: Android malware detection as a bi-level problem. Comput. Secur. **121**, 102825 (2022)
18. Kinkead, M., Millar, S., McLaughlin, N., O'Kane, P.: Towards explainable CNNs for Android malware detection. Procedia Comput. Sci. **184**, 959–965 (2021)
19. Kong, P., Li, L., Gao, J., Liu, K., Bissyandé, T.F., Klein, J.: Automated testing of Android apps: a systematic literature review. IEEE Trans. Reliab. **68**(1), 45–66 (2018)
20. Kursa, M.B., Jankowski, A., Rudnicki, W.R.: Boruta - a system for feature selection. Fund. Inform. **101**, 271–285 (2010)
21. Lipton, Z.C.: The mythos of model interpretability: in machine learning, the concept of interpretability is both important and slippery. Queue **16**(3), 31–57 (2018)
22. 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 **8**, 124579–124607 (2020)
23. Lu, Z., Thing, V.L.: PhilaeX: explaining the failure and success of AI models in malware detection. arXiv preprint arXiv:2207.00740 (2022)
24. Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. In: Advances in Neural Information Processing Systems, vol. 30 (2017)
25. Mahdavifar, S., Alhadidi, D., Ghorbani, A.A.: Effective and efficient hybrid Android malware classification using pseudo-label stacked auto-encoder. J. Netw. Syst. Manage. **30**, 1–34 (2022)
26. Mahdavifar, S., Kadir, A.F.A., Fatemi, R., Alhadidi, D., Ghorbani, A.A.: Dynamic Android malware category classification using semi-supervised deep learning. In: 2020 IEEE International Conference on Dependable, Autonomic and Secure Computing(DASC/PiCom/CBDCom/CyberSciTech), pp. 515–522. IEEE (2020)
27. Mehtab, A., et al.: AdDroid: rule-based machine learning framework for Android malware analysis. Mob. Netw. Appl. **25**(1), 180–192 (2020)
28. Melis, M., Maiorca, D., Biggio, B., Giacinto, G., Roli, F.: Explaining black-box Android malware detection. In: 2018 26th European Signal Processing Conference (EUSIPCO), pp. 524–528 (2018). https://doi.org/10.23919/EUSIPCO.2018.8553598
29. Morcos, M., Al Hamadi, H., Damiani, E., Nandyala, S., McGillion, B.: A surrogate-based technique for Android malware detectors' explainability. In: 2022 18th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), pp. 112–117. IEEE (2022)

30. Nadeem, A., et al.: SoK: explainable machine learning for computer security applications. arXiv preprint arXiv:2208.10605 (2022)
31. Narayanan, A., Chandramohan, M., Chen, L., Liu, Y.: Context-aware, adaptive, and scalable Android malware detection through online learning. IEEE Trans. Emerg. Top. Comput. Intell. **1**(3), 157–175 (2017)
32. Odusami, M., Abayomi-Alli, O., Misra, S., Shobayo, O., Damasevicius, R., Maskeliunas, R.: Android malware detection: a survey. In: Florez, H., Diaz, C., Chavarriaga, J. (eds.) ICAI 2018. CCIS, vol. 942, pp. 255–266. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01535-0_19
33. Qiu, J., Zhang, J., Luo, W., Pan, L., Nepal, S., Xiang, Y.: A survey of Android malware detection with deep neural models. ACM Comput. Surv. (CSUR) **53**(6), 1–36 (2020)
34. Ribeiro, M.T., Singh, S., Guestrin, C.: "Why should I trust you?" Explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1135–1144 (2016)
35. Ribeiro, M.T., Singh, S., Guestrin, C.: Anchors: high-precision model-agnostic explanations. In: Proceedings of the AAAI Conference on Artificial Intelligence (2018)
36. Shams, Z., et al.: REM: an integrative rule extraction methodology for explainable data analysis in healthcare. medRxiv (2021)
37. Srivastava, G., et al.: XAI for cybersecurity: state of the art, challenges, open issues and future directions. arXiv preprint arXiv:2206.03585 (2022)
38. Tam, K., Fattori, A., Khan, S., Cavallaro, L.: CopperDroid: automatic reconstruction of android malware behaviors. In: NDSS Symposium 2015, pp. 1–15 (2015)
39. Tam, K., Feizollah, A., Anuar, N.B., Salleh, R., Cavallaro, L.: The evolution of Android malware and Android analysis techniques. ACM Comput. Surv. (CSUR) **49**(4), 1–41 (2017)
40. Wang, S., et al.: TrafficAV: an effective and explainable detection of mobile malware behavior using network traffic. In: Proceedings of 24th International Symposium on Quality of Service (IWQoS) (2016)
41. Warnecke, A., Arp, D., Wressnegger, C., Rieck, K.: Evaluating explanation methods for deep learning in security. In: Proceedings of European Symposium on Security and Privacy (EuroS&P), pp. 158–174. IEEE (2020)
42. Wu, B., Chen, S., Gao, C., Fan, L., Liu, Y., Wen, W., Lyu, M.R.: Why an Android app is classified as malware: toward malware classification interpretation. ACM Trans. Softw. Eng. Methodol. (TOSEM) **30**(2), 1–29 (2021)
43. Xu, F., Uszkoreit, H., Du, Y., Fan, W., Zhao, D., Zhu, J.: Explainable AI: a brief survey on history, research areas, approaches and challenges. In: Tang, J., Kan, M.-Y., Zhao, D., Li, S., Zan, H. (eds.) NLPCC 2019. LNCS (LNAI), vol. 11839, pp. 563–574. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32236-6_51
44. Yan, F., Wen, S., Nepal, S., Paris, C., Xiang, Y.: Explainable machine learning in cybersecurity: a survey. Int. J. Intell. Syst. **37**(12), 12305–12334 (2022)
45. Zarlenga, M.E., Shams, Z., Jamnik, M.: Efficient decompositional rule extraction for deep neural networks. arXiv preprint arXiv:2111.12628 (2021)

46. Zhang, Z., Hamadi, H.A., Damiani, E., Yeun, C.Y., Taher, F.: Explainable artificial intelligence applications in cyber security: state-of-the-art in research. arXiv preprint arXiv:2208.14937 (2022)
47. Zhu, D., Xi, T., Jing, P., Wu, D., Xia, Q., Zhang, Y.: A transparent and multimodal malware detection method for Android apps. In: Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM), New York, NY, USA, pp. 51–60. ACM (2019). https://doi.org/10.1145/3345768.3355915
48. Zilke, J.R., Loza Mencía, E., Janssen, F.: DeepRED – rule extraction from deep neural networks. In: Calders, T., Ceci, M., Malerba, D. (eds.) DS 2016. LNCS (LNAI), vol. 9956, pp. 457–473. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46307-0_29