Cognitively Inspired Three-Way Decision Making and Bi-Level Evolutionary Optimization for Mobile Cybersecurity Threats Detection: A Case Study on Android Malware

Manel Jerbi, Zaineb Chelly Dagdia, Slim Bechikh, Lamjed Ben Said

▶ **To cite this version:**

Manel Jerbi, Zaineb Chelly Dagdia, Slim Bechikh, Lamjed Ben Said. Cognitively Inspired Three-Way Decision Making and Bi-Level Evolutionary Optimization for Mobile Cybersecurity Threats Detection: A Case Study on Android Malware. Cognitive Computation, 2024, 10.1007/s12559-024-10337-6 . hal-04729097

HAL Id: hal-04729097
https://hal.science/hal-04729097v1

Submitted on 20 Oct 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Cognitively-Inspired Three-Way Decision Making and Bi-Level Evolutionary Optimization for Mobile Cybersecurity Threats Detection: A Case Study on Android Malware

Manel Jerbi[1,2*], Zaineb Chelly Dagdia[3,4], Slim Bechikh[1] and Lamjed Ben Said[1]

[1]SMART Lab, CS Department, University of Tunis, ISG, Tunis, Tunisia.
[2]Faculty of Science, Technology and Medicine, University of Luxembourg, Belval Esch-sur-Alzette L-4365, Luxembourg.
[3]LARODEC, University of Tunis, ISG, Tunis, Tunisia.
[4]Université Paris-Saclay, UVSQ, DAVID, France.

*Corresponding author(s). E-mail(s): manel.jerbi@uni.lu;
Contributing authors: zaineb.chelly-dagdia@uvsq.fr; slim.bechikh@fsegn.rnu.tn; lamjed.bensaid@isg.rnu.tn;

Abstract

Background: Malicious apps use a variety of methods to spread infections, take over computers and/or IoT devices, and steal sensitive data. Several detection techniques have been proposed to counter these attacks. Despite the promising results of recent malware detection strategies, particularly those addressing evolving threats, inefficiencies persist due to potential inconsistency in both the generated malicious malware and the pre-specified detection rules, as well as their crisp decision-making process. **Objective and Methods:** In this paper, we propose to address these issues by (i) considering the detection rules generation process as a Bi-Level Optimization Problem, where a competition between two levels (an upper level and a lower one) produces a set of effective detection rules capable of detecting new variants of existing and even unseen malware patterns. This bi-level strategy is subtly inspired by natural evolutionary processes, where organisms adapt and evolve through continuous interaction and competition within their environments. Furthermore, (ii) we leverage the fundamentals of Rough Set Theory, which reflects cognitive decision-making processes, to assess the true nature of artificially generated malicious patterns. This involves retaining only the consistent malicious patterns and detection rules, and categorizing these rules into a three-way decision framework comprising accept, abstain, and reject options. **Results and Conclusion:** Our novel malware detection technique outperforms several state-of-the-art methods on various Android malware datasets, accurately predicting new apps with a 96.76% accuracy rate. Moreover, our approach is versatile and effective in detecting patterns applicable to a variety of cybersecurity threats.

Keywords: Android malware detection, Bi-level optimization, Rough set theory, Artificial malicious patterns, Evolutionary algorithms, Three-way decision making

1 Introduction

Malware programs are highly diverse and can infiltrate any system despite the precautions taken. Malware Detection Techniques (MDT) propose various methodologies for preventing malware intrusions. Researchers have focused on improving the detection techniques [1–4] to be effective against both known and unknown attacks. Among these approaches, some create new malware and variants of known malware, relying mostly on Evolutionary Algorithms (EA) to mimic mobile malware evolution [5–8]. Despite the effectiveness of these methods, several exhibit flaws: some are not fully automated, with tasks such as manually labeling the used patterns, while others are proposed only for a specific attack type. Additionally, most do not verify the consistency of the produced detection rules or the trustworthiness and reliability of the generated attacks, that is, whether they are genuinely malicious. In literature, various theories have been proposed to address data inconsistency issues, such as uncertainty and vagueness, in decision-making. These include Fuzzy Set Theory [9], Dempster-Shafer Theory [9], and Rough Set Theory (RST) [10]. Real-world problems characterized by data inconsistency necessitate multi-way decision-making, implying that the traditional two-way decision system requires refinement to enable effective decision-making.

In this paper, we propose a new efficient and effective malware detection method named "Rough-Set Based Bi-level Malware Detection" (RS-BMD) for mobile cybersecurity threats detection. We focus on Android malware as a case study. Our method enhances the detection rate through the integration of two key components: (i) Bi-Level Optimization [11], and (ii) Rough Set Theory [10]. In RS-BMD, the process of generating malware detection rules is formulated as a Bi-Level Optimization Problem (BLOP), constituting the first component. In fact, BLOP is a special type of optimization problem where one optimization problem embeds another as a constraint. In the RS-BMD BLOP formulation, the upper level employs Genetic Programming (GP) as an evolutionary algorithm to generate an efficient set of detection rules. These rules maximize coverage not only based on real pattern examples but also on artificial patterns generated by the lower level. Conversely, the BLOP lower level employs a Genetic Algorithm (GA) to create a set of artificial malicious patterns that are not detectable by the existing upper-level rules. This competition between the two levels results in an effective set of detection rules capable of identifying new variants of existing and even unseen malware patterns. Regarding the second essential component in RS-BMD, our primary focus lies on the theory of rough sets, which has found widespread application in successful decision-making across various fields, including medical diagnosis, process control, engineering, and computer security. Specifically, RST provides the necessary fundamentals to address data inconsistency and offers a three-way decision-making approach, forming the basis of our malware detection system. RST is integrated into both levels of the RS-BMD BLOP schema: at the lower level, it is coupled with Genetic Algorithm (GA) for evolving malicious patterns, and at the upper level, it is coupled with Genetic Programming (GP) for the induction of malware detection rules. This coupling of RST with EAs represents a promising approach to inducing efficient and reliable decision rules, even from inconsistent information. Utilizing RST fundamentals enables RS-BMD to assess the true nature of artificially generated malicious patterns at the lower level, retaining only the definitively malicious ones. It also allows for evaluating the reliability of generated detection rules at the upper level, keeping only the most efficient and consistent rules. RS-BMD classifies detection rules into a three-way decision-making fashion: "accept" for definitive rules that signify benign apps, "abstain" for provisional rules that indicate indecision or a need for delayed decision, and "reject" for definitive rules identifying malicious apps. This approach not only addresses the challenges posed by artificially generated malicious patterns and inconsistencies in detection rules but also provides a more coherent framework for decision-making. Additionally, our approach exhibits significant versatility in detecting patterns. It also adapts to a range of cybersecurity threats. We can outline the main contributions as follows:

1. We present RS-BMD, a method that integrates Bi-level Optimization with Rough Set

Theory to generate effective malware detection rules. This novel integration results in a robust upper-level process that produces detection rules capable of identifying (i) known malware patterns (extracted from the base of examples) and, (ii) new synthetically malicious patterns generated by the lower-level. RST is employed to evaluate and refine the generated patterns.

2. We detail the competitive dynamic within the RS-BMD framework, showing how each detection rule operates within an extensive search space of possible artificial malicious patterns. We demonstrate how these patterns are effectively filtered to select the most challenging and pertinent ones, thereby significantly improving the detection capabilities of the upper-level rules.
3. We illustrate the combined use of EAs at both levels with RST. This combination is crucial for discarding inconsistent patterns that might emerge at the lower level and unreliable detection rules potentially produced at the upper level.
4. We enhance the accuracy of predicting the nature of new applications and minimize the chances of false decisions through the implementation of RST for three-way decision-making in our model.
5. Our RS-BMD approach stands out by achieving superior performance compared to several state-of-the-art malware detection methods and engines, particularly in terms of maximizing accuracy and minimizing false alarms.

The subsequent sections of the paper are organized as follows: Section 2 reviews previous works on malware detection. Section 3 provides a brief overview of the basic concepts of Bi-level optimization and the fundamentals of rough set theory. Section 4 describes our proposed malware detection technique. The experimental study details are presented in Section 5. Limitations, threats to validity and future research directions are discussed in Section 6. The final section, Section 7, concludes the paper.

2 Related work

In the field of malware detection, previous research has demonstrated that evolving artificial malware is a highly promising approach. This approach can address the issue of limited malicious datasets available for building detection systems. To generate artificial malware, various techniques have been proposed that do not rely on a static base of malware signatures. For example, Wang et al. [12] introduced MDEA, an Adversarial Malware Detection model designed for Windows operating systems. This model employs neural networks and evolutionary optimization to generate attack samples, thereby enhancing the network's resilience against evasion attacks. The evolutionary algorithm in MDEA evolves different action sequences (raw byte data) by selecting actions from a predefined action space and testing these sequences against the detection model. Sequences that successfully bypass the detection model are then used to modify corresponding malware samples, creating a new training set to further refine the detection model. In another study presented in [13], researchers proposed an approach that combines global search and local search heuristics through a memetic evolutionary search process. The tabu-search algorithm, serving as the local search technique, is utilized to enhance the quality and fitness of solutions by thoroughly exploring the neighborhood of each solution for superior individuals. Despite its strengths, the overall method requires further refinement to enhance the search technique's ability to converge towards high-quality solutions. Another study presented in [14] investigated the effectiveness of a genetic algorithm for feature selection in Android malware detection. It focused on selecting permission and API features through machine learning classification algorithms. However, this approach faced two significant limitations: it relied solely on static features and required a relatively lengthy runtime (approximately 15 hours). The state-of-the-art methods discussed earlier suffer from various limitations including a lack of full automation, specialization in handling specific attack types, and an inability to check the reliability and consistency of malicious patterns. Moreover, their two-way decision-making process does not always fit real situations. To address some of these issues, various recent works have emerged. In [15], researchers explored the use of evolutionary computation techniques to develop new variants of mobile malware that can evade static analysis-based anti-malware systems. Also, they investigated methods for automatically developing security solutions to counteract these

threats. The authors proposed a system that uses co-evolutionary algorithms for malware detection, where a first population generates detection rules and a second population generates artificial malware. However, this co-evolutionary approach comes with limitations, such as not validating the reliability of the generated detection rules and the actual nature of the artificial malware produced. Moreover, both populations operate concurrently without hierarchy, potentially resulting in one population converging before the other. Furthermore, both populations are treated independently, which is not the case in our RS-BMD approach. Our previous work, outlined in [16], also utilized evolutionary computation techniques to create a dynamic malware detection method called "Artificial Malware-based Detection" (AMD). AMD makes use not only of extracted malware patterns, i.e., extracted frequent API call sequences (also called behaviors), but also of artificial ones generated using a GA. The latter evolves a population of API call sequences with the aim of finding new malware behaviors following a set of well-defined evolution rules. The artificial fraudulent behaviors are subsequently inserted into the base of examples to enrich it with unseen malware patterns. Despite its interesting detection results, AMD suffers from the following main limitations: (i) AMD uses a static malware detection rule that is defined in a rational way with respect to similarity measures to both benign and malware patterns; (ii) the detection rule is pre-specified independently of the generated artificial malware patterns, and hence both the rule definition and the classification tasks are performed separately without any interaction; (iii) in AMD, the artificial malware patterns are generated in a global ad-hoc manner based on their similarities to real malware and benign patterns from the bases of examples. Such a process may increase the number of false positives as it is possible that some of the generated artificial patterns are not really malicious, hence causing inconsistency in the set of patterns dealt with. Also, Jerbi et al. [17] suggested considering the detection rules generation process as a BLOP, where an inner optimization task is embedded within the outer one. The goal of the outer task is to generate a set of detection rules in the form of trees combining patterns. The inner task aims to generate a set of artificial malicious patterns that evade the rules of the outer task. The main limitation of the BMD (Bi-level Malware Detection) method is its susceptibility to irrelevant malicious patterns, leading to a high rate of false alarms. In [18], Jerbi et al. introduced ProRSDet, which employs the Variable Precision Rough Set technique as a filter to enhance the quality of the generated malicious patterns by retaining only those of "good quality". However, ProRSDet struggles with high false positive rates when predicting the nature of new apps. Our RS-BMD approach overcomes all these limitations. Several studies in the literature have opted not to confine themselves solely to employing EAs. Instead, they delve into utilizing alternative techniques to enhance the efficiency of malware detection. In the following sections, we will discuss some of these techniques, with a particular emphasis on the use of RST.

Despite their structural dissimilarities, the works discussed below share the same goal: enhancing the performance of malware detection systems by utilizing RST for attribute selection and integrating it with other methods, including evolutionary algorithms. One example of such techniques is presented in [19]. The authors proposed a hybrid community-based approach that combines a community detection schema with the QuickReduct algorithm, which is a rough set-based feature selection method. The goal is to select the most promising set of features for Android malware detection. In [20], a novel two-set feature selection approach called RSST (Rough Set and Statistical Test) was proposed to extract relevant system calls for malware detection. The authors aimed to address the problem of high-dimensional attribute sets by deriving a suboptimal system call space that maximizes the separability between malware and benign samples. Similarly, in [21], RST was used in combination with Support Vector Machine (SVM) to detect intrusions. The RST algorithm was first applied to preprocess the data and reduce its dimensionality. Then, the selected features were sent to SVM to learn and test the hybrid model. The work of [22] proposed an efficient real-time intrusion detection system by integrating Q-learning algorithm and RST for feature reduction. The objective of this work was to achieve maximum classification accuracy while detecting intrusions in NSL-KDD network traffic data by classifying it as either "normal" or "anomalous". An

SVM classification method based on RST was proposed in [23] for the detection of malicious codes. The approach involved preprocessing the original sample data with the RST knowledge reduction algorithm to eliminate redundant features and conflicting samples, thereby reducing the dataset's dimension. The preprocessed sample data was then used as the SVM training sample data. In addition to this approach, few works in the literature have combined RST with EA heuristic search techniques, such as Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO), to select the most important subsets of features. It is worth noting that only a limited number of works in this category explored the use of RST for data mining in conjunction with EAs for Android malware detection. One example of such works is [24], where a feature selection method for detecting Android malware based on rough sets and PSO was proposed. The authors introduced a new encoding method called random key encoding to convert the PSO algorithm into a discrete domain, contributing to the feature selection process. Another work worth mentioning is [25] which aimed to identify the minimum set of features for malware detection using a rough set-based feature significance measure combined with ACO as a heuristic search technique. In another context, the authors in [26] proposed a method that investigated a three-way decision-making approach based on the RST method, with decisions of acceptance, rejection, or deferment. Although the proposed method is not based on EA techniques, it is the only work that we are aware of that uses a three-way decision-making model for malware detection. Specifically, the addition of a deferment decision option provides the flexibility to delay a certain decision when there is insufficient information. Additionally, the proposed method aims to mitigate false decisions at the model level by determining a trade-off between different properties of decision-making, such as accuracy, generality, and uncertainty. Authors, in [26], considered three-way decision based on two probabilistic rough set models, namely, game-theoretic rough sets (GTRS) and information-theoretic rough sets (ITRS). It is important to mention that this work suffers from the main limitations of not evolving artificial malware, contrary to the previously cited EA-based techniques, and of being dependent on the base of examples. From another perspective, some recent works have focused on finding techniques to detect obfuscated malware. For instance, in [27], the authors designed three self-supervised attack techniques: (1) the first technique considered the IoT input data and the adversarial sample generation in real-time; (2) the second technique built a generative adversarial network model to generate adversarial samples within the self-supervised structure; and (3) the third technique utilized three well-known perturbation sample techniques to develop adversarial malware and inject it over the self-supervised architecture. Additionally, the authors applied a defense method to mitigate these attacks, namely adversarial self-supervised training, to protect the malware detection architecture against the injection of malicious samples. In [28], the authors introduced a hybrid deep generative model that leverages both global and local features for detecting malware variants. They converted malware into images to capture global features within a predefined latent space and extracted local features from binary code sequences. These features were then concatenated and input into the malware detector. While the model achieved notable detection results, the transformation phase could potentially result in data loss. Moreover, the authors in [29] present MalGAN, a specific GAN model developed for black-box attack methods. This work highlights the potential of GANs in crafting functional and unseen malware examples that challenge and test machine learning-based security systems. While MalGAN demonstrates effectiveness in generating adversarial malware examples that evade detection, this approach assumes that the substitute model accurately reflects the real detection system's behavior. If the substitute model does not closely match the actual system, the generated examples may not be effective in evasion. In [30], the authors investigated using GANs to create adversarial examples that boost classifier training for intrusion detection. By applying transformations such as shifting, rotating, scaling, and skewing data, they achieved improved classifier accuracy. However, this approach can lead to overfitting and performance issues when overly reliant on extensive altered data. Different works have attempted to address obfuscated malware. We can refer to those cited in Table 1, where the main ideas, advantages, and limitations are presented.

Table 1: Recent methods for obfuscated malware detection.

Work	Main idea	Advantages	Limitations
[31]	Two main subsystems that work in parallel: one is trained for benign labeled apps while the second one is trained on malware labeled apps.	Each subsystem uses features from static and dynamic malware analysis. Both are extracted based on an ensemble approach (OC-SVM, LOF, M-iForest classifiers).	The given app needs to be tested based on two different models, then processed through a voting mechanism, which may lead to some anomalous decisions.
[32]	The proposed method utilizes only CNN for discovering common features of API call graphs of malware. For detecting malware, the method employs a lightweight classifier that calculates a similarity between API call graphs used for malicious activities and API call graphs of applications that are going to be classified.	The method uses CNNs to analyze the behaviors of malicious applications based on their API call graphs.	This method requires significant computing resources and time to extract features.
[33]	A security system was built and designed based on the SVM, KNN, LDA, LSTM, CNN-LSTM, and autoencoder algorithms.	A method for signature-based malicious attack detection.	The system was tested with a limited number of features (only 6), and it has not yet been designed for use with a real industrial control system.
[34]	A deep neural network incorporating convolutional methods, with multiple views learning from three raw input feature sets of low-level opcodes, app permissions and proprietary Android API packages.	A CNN-based approach for analyzing API sequence calls, utilizing proprietary Android API packages as raw input features.	The CNN relies only on the relevance of API sequences during training.
[35]	The combination of the printable strings and NLP techniques are used as a filtering method.	The method involves vocabulary selection to build a dictionary of the most frequent words and subwords in training data.	Arbitrary thresholding decisions can create the possibility of missing out on meaningful words.
[16]	AMD generates API call sequences (patterns) using a GA with the aim to find new malware behaviors.	Diversify the base of malware examples in order to maximize the detection rate.	Generates false patterns, leading to a high false alarm rate.

[17]	The detection rules generation process is considered as a BLOP, where a lower-level optimization task is embedded within the upper-level one. The upper-level problem's goal is to generate a set of detection rules, trees of combined patterns, by maximizing the coverage not only on the base of examples of real patterns but also on the artificial patterns generated by the lower-level.	Production of more efficient detection rules capable of revealing new malicious behaviors.	Generation of some inconsistent detection rules.
[18]	Reinforcement of the bi-level detection method, more precisely the lower-level, by using a variable precision rough set analyzer module to filter the false generated patterns.	The improvement of the quality of the generated patterns will help improve the detection performance of the detection rules.	The persistence of some inconsistent detection rules in the upper-level.
[12]	An adversarial malware detection model for Windows operating systems is proposed and combines neural networks with evolutionary optimization to create attack samples to make the network robust against evasion attacks.	Production of new malicious patterns to enrich the base of malware examples.	The malware detection task is independent from the malware generation task. The presence of highly destructive malware variants among the generated ones.
[36]	A GAN model named TrafficGAN designed to generate new malicious network traffic patterns for zero-day attacks.	This method emphasizes the potential of GANs in creating training data that simulates previously unseen attack vectors.	The challenge lies in maintaining a balance between enhancing data diversity and preserving system performance, without significantly compromising it, alongside the risk of model overfitting due to adversarial samples not perfectly representing real-world network traffic scenarios.

Local outlier factor (LOF); one class support vector machine (OC-SVM); a modified version of the Isolation forest (M _iForest). the support vector machine (SVM), k-nearest neighbors (KNN), linear discriminant analysis (LDA), long short-term memory (LSTM), convolution neural network-long short-term memory (CNN-LSTM)

To summarize, most state-of-the-art methods for malware detection employ evolutionary algorithms, where one population generates detection rules and another generates artificial malware. However, these approaches suffer from issues such as premature convergence of one population over the other, and inconsistency in the generated detection rules and malicious patterns. Rough set-based malware detection techniques primarily focus on feature reduction and do not address inconsistency checks or propose a three-way decision-making approach for malware detection. The only work ([26]) that introduced a three-way decision-making approach is not based on EAs and overlooks the benefits of such an approach. This limitation is critical for reducing false alarms and building an efficient malware detection system. To address these challenges, we introduce RS-BMD, our proposed solution, which is discussed in detail in Section 4.

3 Preliminaries

In this section, we introduce the basics of the two key components of our proposed RS-BMD malware detection technique: "Bi-level Optimization" and "Rough Set Theory".

3.1 Bi-level Optimization

A Bi-level Optimization Problem (BLOP) can be described as a combination of two optimization problems, where the lower-level problem (follower) is incorporated as a constraint of the upper-level problem (leader) [11]. There are two classes of variables in a BLOP: the upper-level variables x_u, also referred to as the upper-level decision vector, and the lower-level variables x_l, referred to as the lower-level decision vector, as illustrated in Figure 1. In the lower-level problem, optimization is carried out with respect to x_l, while x_u acts as a parameter. Consequently, for each x_u, a distinct lower-level problem emerges, for which an optimal solution must be identified. Both decision vectors (x_u and x_l) are considered in the upper-level problem, where the optimization aims to account for both decision vectors. The BLOP can then be formulated as follows:

$$\min_{x_u \in X_U, x_l \in X_L} L(x_u, x_l) \text{ subject to } \begin{cases} G_k(x_u, x_l) \leq 0, k = 1, \ldots, K. \\ x_l \in \operatorname{argmin}\{f(x_u, x_l)| \\ g_j(x_u, x_l) \leq 0, j = 1, \ldots, J\} \end{cases} \quad (1)$$

In the given formulation, L represents the upper-level objective function, f represents the lower-level objective function, x_u represents the upper-level decision vector and x_l represents the lower-level decision vector. G_k and g_j represent the constraint functions at the upper and lower levels, respectively. The difficulty in a BLOP consists of the fact that only the optimal solutions of the lower-level optimization task may be acceptable as possible feasible candidates to the upper-level one. For example, a member $x^1 = (x_u^1; x_l^1)$ can be considered feasible at the upper level only if x^1 satisfies the upper-level constraints, and x_l^1 is an optimal solution to the lower-level problem corresponding to x_u^1.

Various malware detection methods have been proposed in literature, yet none have introduced a bi-level model that incorporates a three-way decision process. Our proposed solution combines bi-level optimization with RST for malware detection. To the best of our knowledge, this is the first approach of its kind to use a bi-level model and offer a three-way decision-making framework, showing promising results.

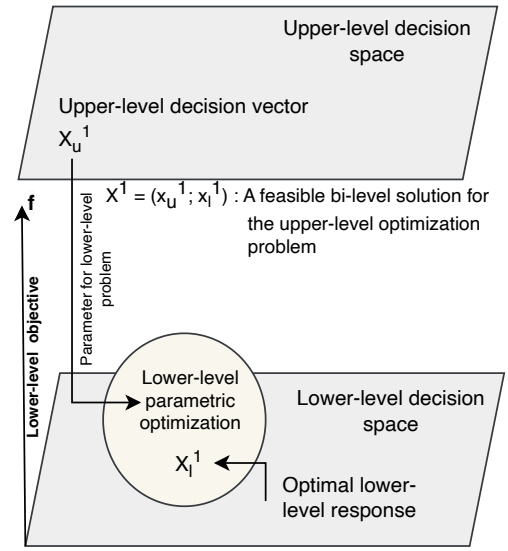

Fig. 1 Bi-level optimization: general overview (inspired by [37]).

3.2 Rough Set Theory

Rough Set Theory (RST) is a key mathematical theory used to handle imprecise, inconsistent, and incomplete information and knowledge [10]. In classifying objects, RST utilizes a set of multi-valued features known as "condition" features and "decisio" features. The information about the objects (data instances) is represented in a format called an "information system", which can be visualized as a table with objects listed in rows and features in different columns. Assume $\mathbf{U}$ is a nonempty finite universe of objects. Imprecise information in an information system causes indiscernibility of objects. The indiscernibility relation is an equivalence relation on $\mathbf{U}$. Technically, an equivalence relation $\mathbf{E} \subseteq \mathbf{U} \times \mathbf{U}$ represents relationships between objects in $\mathbf{U}$. An equivalence relation can be defined based on a set of features in an information system so that two objects are equivalent if and only if they have the same value on every feature. The equivalence relation induces a partition of the universe, denoted by $\mathbf{U}/\mathbf{E}$. The equivalence class containing an object x is given by $[x]_{\mathbf{E}} = y|y \in \mathbf{U}, x\mathbf{E}y$, or simply $[x]$ if $\mathbf{E}$ is understood. RST offers powerful mathematical fundamentals to deal with the inconsistency encountered in data. These are a pair of definable sets known as the "lower-approximatio" and "upper-approximation", used as the main vehicles for problem solving. Both the lower-approximation and the upper-approximation represent the classes of indiscernible objects that possess sharp descriptions of concepts but with no sharp boundaries.

Suppose $\mathbf{C} \subseteq \mathbf{U}$ is a set of objects representing a given concept. The lower-approximation (denoted as $\underline{apr}(\mathbf{C})$) represents a set of objects in $\mathbf{U}$ that can be certainly classified as belonging to concept $\mathbf{C}$ and can be defined as: $\underline{apr}(\mathbf{C}) = x|x \in \mathbf{U}, [x] \subseteq \mathbf{C}$. The upper-approximation (denoted as $\overline{apr}(\mathbf{C})$) represents a set of objects in $\mathbf{U}$ that can possibly be classified as belonging to concept $\mathbf{C}$ and can be defined as: $\overline{apr}(\mathbf{C}) = x|x \in \mathbf{U}, [x] \cap \mathbf{C} \neq \emptyset$. Elements belonging only to the upper-approximation define the "boundary region" (denoted as $\mathbf{BND}(\mathbf{C})$) and can be expressed as: $\mathbf{BND}(\mathbf{C}) = \overline{apr}(\mathbf{C}) - \underline{apr}(\mathbf{C})$. $\mathbf{BND}(\mathbf{C})$ represents the set of objects that cannot be certainly classified as belonging to concept $\mathbf{C}$. Such a concept $\mathbf{C}$ is called a "rough set". In other words, rough sets are sets having non-empty boundary regions. There is another formulation of rough set approximations which is as follows:

- $\mathbf{POS}(\mathbf{C}) = \{x|x \in \mathbf{U}, [x] \subseteq \mathbf{C}\} = \bigcup\{[x]|x \in \mathbf{U}, [x] \subseteq \mathbf{C}\}$
- $\mathbf{NEG}(\mathbf{C}) = \{x|x \in \mathbf{U}, [x] \cap \mathbf{C} = \emptyset\} = \bigcup\{[x]|x \in \mathbf{U}, [x] \cap \mathbf{C} = \emptyset\}$
- $\mathbf{BND}(\mathbf{C}) = \{x|x \in \mathbf{U}, [x] \cap \mathbf{C} \neq \emptyset, [x] \nsubseteq \mathbf{C}\} = \bigcup\{[x]|x \in \mathbf{U}, [x] \cap \mathbf{C} \neq \emptyset, [x] \nsubseteq \mathbf{C}\}$

These three regions are pairwise disjoint and are named, respectively, the positive region, the negative region, and the boundary region of $\mathbf{C}$. That is, the concept $\mathbf{C}$ is approximately described by three disjoint subsets based on equivalence classes. This leads to the definition of the positive region as the lower-approximation of $\mathbf{C}$ and the union of the positive and boundary regions as the upper-approximation:

- $\underline{apr}(\mathbf{C}) = \mathbf{POS}(\mathbf{C})$
- $\overline{apr}(\mathbf{C}) = \mathbf{POS}(\mathbf{C}) \cup \mathbf{BND}(\mathbf{C})$

The RST notion of three-way decision making is introduced based on the above three disjoint regions produced by the lower and upper approximations, which in our case refer to: *accept* (certain rules for acceptance; positive region, i.e., benign apps), *abstain* (possible rules for indecision or delayed decision; boundary region), and *reject* (certain rules for rejection, negative region i.e., malicious apps). Hence, to make three-way decisions, we can either use $\mathbf{POS}(\mathbf{C})$, $\mathbf{NEG}(\mathbf{C})$, and $\mathbf{BND}(\mathbf{C})$, or $\underline{apr}(\mathbf{C})$ with $\overline{apr}(\mathbf{C})$. The *certain rules* ($\underline{apr}(\mathbf{C})$) cover the explicit criteria for accepting benign apps and rejecting malicious ones. The *possible rules* ($\overline{apr}(\mathbf{C})$) pertain to the *abstain* decision, which is used for cases of indecision or delayed decision-making. In this paper, we opted for the second formalization.

By using these concepts, rough set-based models have been successfully applied and used to handle different kinds of uncertain or inconsistent information systems. With respect to this, our RS-BMD malware detection system will be able to deal with the inconsistency of both the generated malicious patterns and the detection rules, and classify apps in a three-way decision fashion.

4 RS-BMD: A Rough Set based Bi-Level Malware Detection Technique

In this section, we first present the motivation behind this work. Then, we give a general description of the proposed model, followed by a detailed description of the different phases of our proposed RS-BMD.

4.1 Motivation

When installing an app, the user generally faces the two-way crisp decision-making of the malware detection system, which gives him/her a narrow choice: either to accept or to reject the installation process, preventing him/her from acquiring the desired app even in cases where there is doubt about the real nature of the app (i.e., benign or malicious). An important number of previous works have proposed to extract frequent API call sequences from already encountered harmful apps using pattern mining techniques. These sequences build a base of *fraudulent behaviors*. Afterwards, API call sequences can be extracted from any program, and based on these, the considered program behavior can be judged to be more similar to malware behaviors or to benign-ware ones, which forms the basis for the crisp two-way decision-making process of the state-of-the-art malware detection techniques, which, as noticed, remain highly dependent on the base of examples.

Also, in Section 2, we have discussed EA-based detection techniques – with an emphasis on the previous works of Jerbi et al., AMD [16], BMD [17], and ProRSDet [18], as well as the work of Sen et al. [15] – which can overcome the lack of diversity of the base of examples of malware behaviors by generating new variants of malware. However, these suffer from some limitations. These limitations were discussed in Section 1 and Section 2, and are mainly related to the generation of inconsistent malicious patterns, which negatively affect the false alarm rate. Based on these limitations, there is clearly a need to evaluate the new artificial patterns, as well as the generated detection rules, and perform a selection among them before their re-injection into their final datasets, namely the malicious patterns dataset and the detection rules dataset. This task is accomplished by using RST's approximations (i.e., the upper and the lower approximations) in our proposed RS-BMD bi-level detection approach.

RS-BMD aims to improve the detection process targeting essentially obfuscated malware. In addition to its ability to overcome the problem of lack of diversity in the base of examples of malware behaviors in an automatic way, RS-BMD is also capable of detecting new variants of malware. This is achieved via the development of a bi-level optimization technique that integrates RST in both levels. The leader (upper-level) utilizes (i) patterns extracted from both the base of examples (input), i.e., set of malicious patterns, and (ii) the artificially generated malicious patterns to produce efficient and consistent detection rules. The reliability of these is checked by the RST coupled component. The detection rules generation process consists of creating a combination of patterns used to detect malicious patterns from new files. For example, for a new file P having a set of patterns, we can decide its fate based on its extracted patterns by comparing them to our base of detection rules, resulting in three possibilities: first, if the extracted patterns match a certain rule in the malicious set of rules, then P is rejected. Second, if the extracted patterns match a certain rule in the benign set of rules, then P is accepted. And third, if they match a possible rule, then P is abstained, and hence the final decision is forwarded to the user. Such a decision process illustrates the three-way decision-making fashion of our RS-BMD technique, which is made by the RST coupled component.

The upper-level keeps exchanging solutions with the lower-level, i.e., the upper-level sends detection rules to the lower-level and the lower-level sends the generated artificial malicious patterns to the upper-level, until a stopping criterion is met (i.e., number of iterations). Within these exchanges, the detection rules are improved from one iteration to another as they are capable of detecting the new generated malicious patterns. On the other side, at the lower level, the generated malicious artificial patterns are continuously improved. Verified by the RST-coupled component for reliability, these patterns evolve to evade detection by the rules sent from the upper level, with each iteration enhancing their ability to remain undetected. At the end of these exchanges, the most effective detection rules present the final output produced by our RS-BMD approach. The

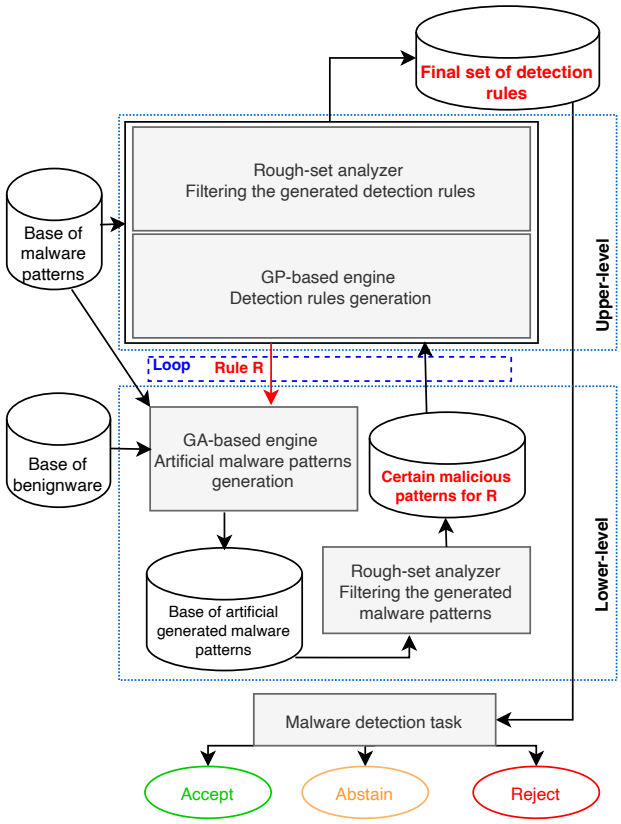

Fig. 2 The general methodological flowchart of RS-BMD.

general methodological flowchart of RS-BMD is depicted in Figure 2. In what follows, the entire RS-BMD operating process is described in detail.

4.2 RS-BMD phases

4.2.1 General overview

RS-BMD is based on two main phases as illustrated in Figure 3. The initial phase, known as the *"detection rules generation"* (upper-level problem), involves generating three types of detection rules: *certain rules* which are further divided into those for acceptance (i.e., benign apps) and rejection (i.e., malicious apps), and *possible rules*, which include the abstain decision for situations where a decision cannot be made immediately or is delayed. The second phase is responsible for the generation of *certain* artificial malicious patterns and is referred to as the *"malware generation phase"* (lower-level problem).

The first phase invokes a detection model that uses an enriched collection of malicious patterns, i.e., the malware patterns from the base of examples, which are stored in the database of malicious API call sequences ($MPDB$), and the *certain* artificially verified generated ones (the output of the second phase) which are stored in the certain artificial malicious patterns database ($CAMDB$). Both of them are used in the generation process of three types of detection rules, i.e., the certain malicious/benign rules stored in the final set of certain detection rules ($FCDR$), and the possible rules stored in the final set of possible detection rules ($FPDR$). Throughout this phase, either the certain malicious, the certain benign, or the abstained (suspected) programs will be detected among the new apps by using the three types of generated detection rules, which is made possible by the use of the Rough-Set analyzer component. The evaluation of the generated detection rules (upper-level) is based on the coverage of the base of examples (input) and also the coverage of the *certain* artificial malicious patterns generated by the lower-level. The internal technical details tied to this first phase are given in Section 4.2.2.

The second phase, assured by the lower-level, includes the steps responsible for the generation of artificial malicious patterns and their check. This process is defined via two main steps: The first step's aim is to generate a set of artificial malware patterns using the GA-based engine with two input datasets composed of API call sequences: the $BPDB$ for the benign sequences and $MPDB$ for the malicious ones. On one hand, the GA-based engine finds the best solutions (i.e., most undetectable malicious patterns) that will be used by the upper-level's GP-engine to evaluate the associated detection rules. On the other hand, the GP engine at the upper level identifies optimal solutions, namely the set of detection rules SDR, representing the most effective detection rules. These rules are then utilized by the GA engine at the lower level to assess the generated malicious patterns. The second step consists of filtering the artificially generated patterns using the Rough-Set analyzer component in order to keep only the certain malicious ones. Those ones, stored in the certain artificial malicious patterns database ($CAMDB$), will be fed to the upper-level in order to try escaping the generated detection rules. The internal technical details tied to this second phase are given in Section 4.2.3.

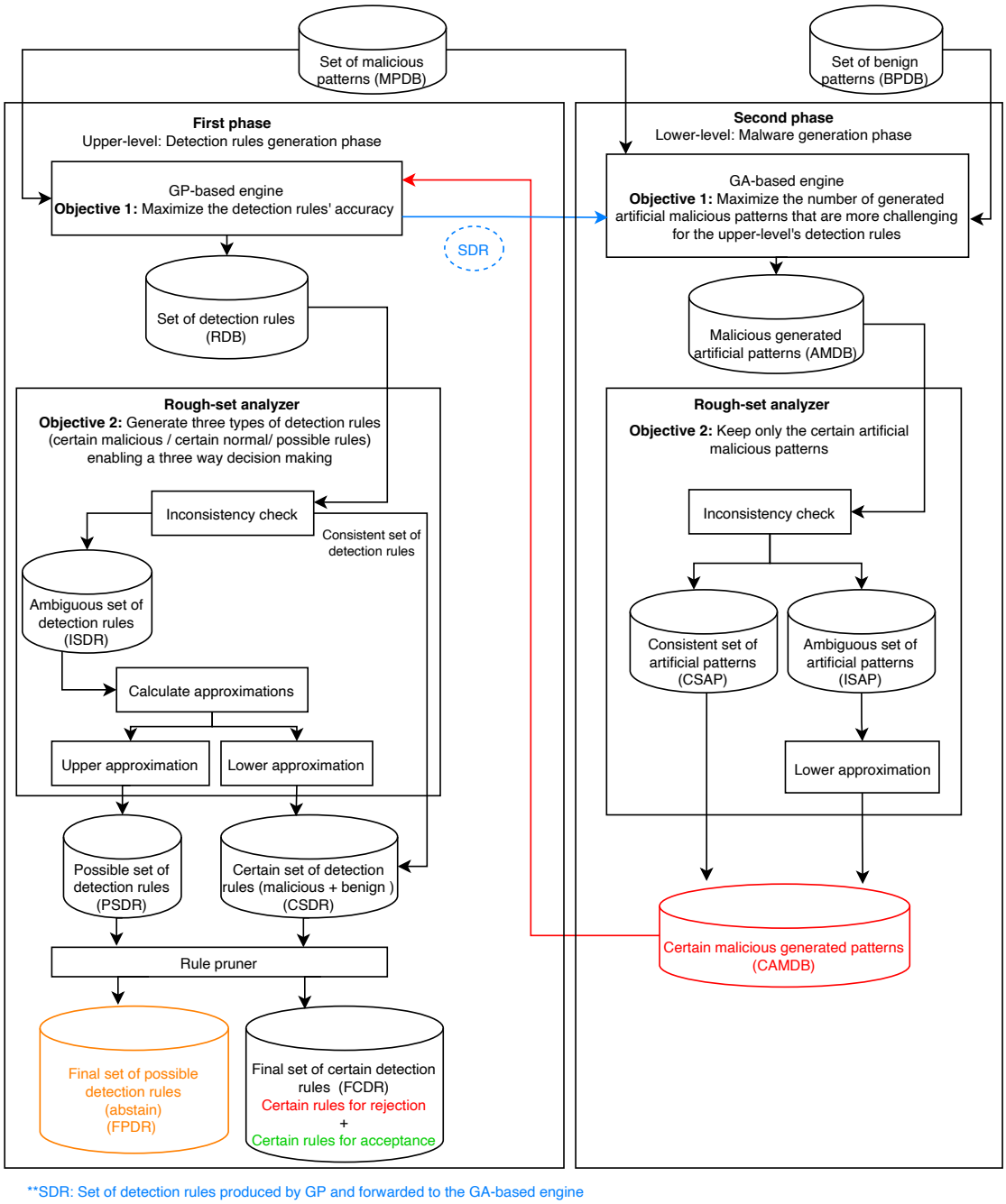

Fig. 3 The RS-BMD technical workflow.

To summarize, in the RS-BMD schema, the upper-level is executed for a number of iterations, then the lower-level for another number of iterations. After that, a selection is made among the generated lower-level solutions using a Rough-set analyzer to keep only the *certain* ones. Afterwards, the best solution found in the lower-level will be used by the upper-level to evaluate the associated solution (among the detection rules), and then this process is repeated a number of times until reaching a termination criterion (e.g., number of iterations). Both levels are dependent. As presented,

the evaluation of every detection rule solution (upper-level) requires running a search algorithm to find the best undetectable artificial malicious patterns by the upper-level solution. The final outcome of our RS-BMD approach consists of the ultimate set of detection rules, comprising a collection of certain rules for *rejection* or *acceptance*, and a set of *abstained*, also referred to as possible, rules. An example of a certain *rejection* rule is given as follows:

DR1: IF (MF301 **AND** MF35 **AND** MF405) **OR** (MF21 **AND** MF211) **OR** (MF301 **AND** MF311 **AND** MF78) **THEN** *App* is malicious.

In this sample, the ($DR1$) *rejection* rule, in which the antecedent corresponds to a succession of patterns (i.e., $MF301$, $MF35$, etc.) with a set of logical operators, shows that an app *App* is considered as malicious. The consequent of a detection rule determines its label (malicious). As for an *abstained* rule, the decision is left to the user to either continue or abort the app's installation process.

4.2.2 First phase: Detection rules generation

Description of the upper-level functioning

Figure 3 depicts the entire upper-level technical framework. Its algorithmic details are given in Algorithm 1. The upper-level is composed of three modules, namely: (i) A GP-based engine, (ii) an upper-level Rough-Set analyzer, and (iii) a Rule Pruner.

The GP-based engine is responsible for generating the detection rules and their evaluation. First, the GP proceeds with the generation of the initial set of detection rules SDR_0 (Algorithm 1, lines 1-3) based on the set of malicious patterns extracted from the base of examples ($MPDB$), the set of benign patterns extracted from the base of examples ($BPDB$), and the set of certain generated artificial malicious patterns ($CAMDB$). The description of the GP mechanism which covers the solution encoding, variation, and evaluation is given in the section below. After that, a cycle of production and evaluation of the detection rules is accomplished (Algorithm 1, lines 4-6). The evaluation of the produced rules is performed via the use of an objective function. This function helps maximize the coverage of patterns from the base of examples (input), and to maximize the coverage of the generated artificial patterns at the lower-level. The generation/evaluation processes stop when the already set number of generations is met. The fitness function used for the evaluation of a detection rule (DR), at the upper-level, is defined in Equation 3.

When the generation task of the detection rules is complete (Algorithm 1, line 7), the obtained set of detection rules (RDB) is given as an input to the Rough-set analyzer module. The latter performs two tasks: the consistency check on the detection rules dataset (RDB) (Algorithm 1, line 8), and their classification based on the rough-set approximations basics, i.e., the lower approximation and the upper approximation (Algorithm 1, lines 9-10). For any ambiguous set of detection rules, three steps, namely the (i) identification of attributes, (ii) the concept forming, and (iii) the approximation, are performed. An example of an ambiguous set of detection rules is given in Table 3.

The Rough-set analyzer first proceeds with identifying the attributes, which represent in our case the condition attributes (API call sequences identified as MFX_i in Table 3). After the attribute's identification, the Rough-set analyzer looks for the concepts, in the concept forming step, which are the decision attributes. The decision attributes indicate the nature of an app (either malicious or normal). The approximation step involves the use of the RST lower- and upper-approximations concepts. Within this step, the ambiguous set of detection rules ($ISDR$) is verified and then split into *certain* set of detection rules ($CSDR$) and *possible* set of detection rules ($PSDR$) based on the lower approximation and the upper approximation, respectively. Subsequently, these sets, i.e., the $CSDR$ and the $PSDR$, are forwarded to the rule pruner (Algorithm 1, line 11). The rule pruner examines the rules, both *certain* and *possible* rules, extracted by the GP-based search engine and uses Boolean operators such as union and intersection to prune and hence simplify the rules. During the pruning operation, redundant rules are removed. The retained rules are classified as the final set of certain detection rules ($FCDR$) and final set of possible detection rules ($FPDR$).

Algorithm 1 Upper-level algorithm

Require: $MPDB$: set of malicious patterns, $BPDB$: set of benign patterns, $CAMDB$: Certain generated artificial malicious patterns, NDR: number of generated rules, NAP: number of generated certain artificial malicious patterns in $CAMDB$, NU: number of iterations in the upper-level, NL: number of iterations in the lower-level.

Ensure: Final certain detection rules $FCDR$, Final possible detection rules $FPDR$.

$SDR_0 \leftarrow$ Initialization(NDR,$MPDB$,$BPDB$) /*First generation of detection rules*/
for each DR_0 in SDR_0 **do** /*DR means detection rule*/
  $SAP_0 \leftarrow$ APGeneration(DR_0,$CAMDB$,NAP, NL) /*call lower-level*/
  $DR_0 \leftarrow$ Evaluation(DR_0,$CAMDB$,SAP_0)
end for
$t \leftarrow 1$
while $t < NU$ **do**
  $Q_t \leftarrow$ Variation(SDR_{t-1})
  for each DR_t in Q_t **do** /*Evaluate each rule based on upper fitness function*/
    $DR_t \leftarrow$ UpperEvaluation(DR_t,$CAMDB$)
    $SAP_t \leftarrow$APGeneration(DR_t,$CAMDB$,NAP, NL)
    $DR_t \leftarrow$ EvaluationUpdate(DR_t,SAP_t)
  end for
  $U_t \leftarrow Q_t \cup SDR_t$
  $SDR_{t+1} \leftarrow$ Selection(NDR,U_t)
  $t \leftarrow t+1$
end while
$RDB \leftarrow$ FittestSelection(SDR_t)
$(ISDR, CSDR) \leftarrow$ InconsistencyCheck(RDB) /*Inconsistent set of DR and consistent set of DR*/
$SCDR \leftarrow$ LowerApproximation($ISDR$) $\cup$ $CSDR$ /*Set of certain detection rules*/
$SPDR \leftarrow$ UpperApproximation($ISDR$) /*Set of possible detection rules*/
$(FCDR, FPDR) \leftarrow$ Pruning($SCDR$,$SPDR$)

As *possible* rules cannot provide a final firm decision, they need further evaluation that reflects their quality and measures their reliability. For every *possible* rule, RS-BMD estimates its reliability using an index (named $Safety\_index$), which is defined as the ratio of the number of instances that are correctly classified by a *possible* rule, and the number of instances whose condition attributes are covered by the same rule in the input dataset. This is expressed as follows:

$$Safety\_index = \frac{Instance\_Possible\_Rule}{Instance\_Possible\_Original\_Data} \quad (2)$$

where $Instance\_Possible\_Rule$ refers to the number of instances that are correctly classified by a **possible** rule and $Instance\_Possible\_Original\_Data$ refers to the total number of instances within the training dataset. This index can, therefore, be viewed as the probability of classifying the ambiguous input dataset correctly. It shows the effectiveness and usefulness of such rules and will guide the user to make the ultimate decision about the "fate" of an app. An illustrative example of this index is given as follows: Suppose that when classifying a new app, its extracted patterns matched a *possible* rule DRX_{80}. The $Safety\_index$ of DRX_{80} is (2340/3000). This means that the user can trust this rule as there are 78% chances that DRX_{80} correctly classifies the app. This means that DRX_{80} succeeded in correctly classifying a total of 2340 apps among the training set. This $Safety\_index$ will be given as an output to the user together with the abstain decision. It will help the user decide whether to pursue the app's installation process or to abort it based on the trust degree assigned to the applied possible rule.

The GP evolutionary mechanism

The evolutionary operators used in our bi-level approach need to be defined to seek the highest performance. An adaptation is required concerning the different aspects of the manipulated solutions (detection rules) at the upper-level algorithm, which are respectively: the solution representation, the solution variation, and the solution evaluation. Let us mention that the same evolutionary mechanism will be performed in the lower-level's algorithm, but for the GA.

Solution representation A GP algorithm [38] is used for the upper-level optimization problem in which a solution is composed of terminals and functions. After evaluating many parameters related to the malware detection problem, the terminals and the functions are decided to meet the current problem's requirements. The terminals correspond to different patterns (frequent API call sequences). The functions that can be

used between these patterns are Intersection (AND) and Union (OR). Formally, each candidate solution S in this problem is a sequence of detection rules, where each rule is represented by a tree:

- Each leaf-node (Terminal) L belongs to the set of patterns.
- Each internal-node (Functions) I belongs to the Connective (logic operators) set $C = \{AND, OR\}$.

Let us recall that the set of candidate solutions (rules) is nothing other than a logic program represented as a forest of AND-OR trees. An individual in the upper-level, representing a detection rule, is a tree consisting of all API call sequences as terminal nodes and some operators as non-terminal nodes. Logical operators (AND/OR) are employed to form a GP tree. Each individual produces an if/then rule to determine the maliciousness of an application being analyzed. An example of a detection rule ($DR1$) was previously given in Section 4.2.

To generate an initial population for GP, we begin by defining the maximum tree length (maximum number of API call sequences per solution). The tree's length is proportional to the number of API call sequences used for malware detection. A high tree length does not necessarily mean that the results are more precise. These parameters can be either randomly chosen or specified by the user.

Solution variation The crossover and mutation operators used when combining information from individuals (parents) should be adapted to our solution representation. The GP mutation operator can be applied to a function node or to a terminal node. It starts by randomly selecting a node in the tree. Then, if the selected node is a terminal (pattern), it is replaced by another terminal; if it is a function (AND-OR), it is replaced by a new function; and if tree mutation is to be applied, the node and its subtree are replaced by a new randomly generated subtree. As for the GP crossover operator, two parent individuals are selected, and a subtree is picked from each selected parent. The crossover swaps the nodes and their related subtrees from one parent to the other. This operator must ensure the respect of the depth limits. The crossover operator can be applied only with parents having the same rule aim (malicious or benign pattern to detect). Each child thus combines information from both parents. In any given generation, a variant will be the parent in at most one crossover operation.

Solution evaluation The encoding of an individual should be formalized as a mathematical function called the "fitness function". The fitness function quantifies the quality of the proposed detection rules and the generated artificial malicious patterns. The goal is to define efficient and simple fitness functions to reduce computational cost. For the GP adaptation, we used the fitness function f_{upper} defined in Equation 3 to evaluate detection rule solutions (DR).

$$f_{upper}(DR) = Max(\frac{\frac{Precision(DR)+Recall(DR)}{2} + \frac{\#damp}{\#amp}}{2}) \quad (3)$$

where $\#damp$ refers to the number of detected artificial malicious patterns and $\#amp$ refers to the number of artificial malicious patterns and

$$Precision(DR) = \frac{\sum_{i=1}^{p} DR_i}{t} \quad (4)$$

$$Recall(DR) = \frac{\sum_{i=1}^{p} DR_i}{p} \quad (5)$$

p is the number of detected malicious patterns after executing the solution, i.e., the detection rule, on the base of malicious patterns examples ($MPDB$), t is the total number of malicious patterns within $MPDB$, and DR_i is the i^{th} component of a detection rule DR such that:

$$DR_i = \begin{cases} 1 \text{ if the } i^{th} \text{ detected malicious pattern} \\ \text{exists in the malicious base of examples} \\ 0 \text{ otherwise} \end{cases} \quad (6)$$

It is to be noted that all the specific variation operators used (the same goes for the lower-level), may cause the manipulated solutions to be distorted in different ways and with different degrees. This issue may have negative effects on the detection rates as well as on the false positive rates. This issue is taken into account at both levels and justifies the use of RST at both levels.

4.2.3 Second phase: Malware generation phase

Description of the lower-level functioning

Figure 3 depicts the whole lower-level technical framework. Its algorithmic details are given in Algorithm 2. The proposed lower-level is composed of two main steps: (i) the GA-based engine (responsible for the malware generation task) and (ii) the Rough-Set analyzer that deals with the inconsistency problem. The lower-level should seek to optimize the following two objectives:

1. Maximize the number of uncovered artificial malicious patterns by the solutions of the first population (i.e., detection rules SDR).
2. Keep only the *certain* malicious patterns checked by the Rough-set analyzer ($CAMDB$), i.e., remove the inconsistent ones.

These two objectives define the quality of the solutions: the set of generated malicious patterns. The quality of an artificial malicious pattern (AP) is evaluated based on the GA's fitness function (defined in Equation 7). The quality of the patterns will affect the system's detection rate.

The process of artificially generating malicious patterns using GA goes through different steps, as shown in Algorithm 2. First, a set of malicious patterns is generated. These patterns mimic the compositional characteristics of real patterns stored in the malicious patterns database ($MPDB$), which is separate from the benign patterns database ($BPDB$). These generated patterns are evaluated according to the upper-level's produced detection rules (SDR) (Algorithm 2, lines 1-2). Second, a cycle of generation and evaluation of malicious patterns is performed (Algorithm 2, lines 4-4.5) until a stopping criterion is reached (i.e., the number of generations is reached). Each generated pattern is evaluated according to a fitness function; this is done to keep only the best-fitting patterns (Algorithm 2, line 5). Once the artificial malicious patterns are generated, they will be stored in the malicious generated artificial patterns database ($AMDB$). The description of the GA mechanism, which covers the solution encoding, variation, and evaluation, is given in the section below. The artificial patterns (output of the GA, stored in $AMDB$) are given as input to the lower-level Rough-Set analyzer, which performs two tasks: checking inconsistency in the input dataset (i.e., the artificial malicious patterns $AMDB$), and filtering objects based on the RST lower approximation (Algorithm 2, line 6). For an ambiguous malicious patterns dataset ($ISAP$), three steps are performed: identification of attributes, concept forming, and approximation. An example of an ambiguous set of patterns is given in Table 2.

The lower-level's Rough-set analyzer, identifies the attributes, which represent in this part the condition attributes (API calls identified as MLX_i in Table 2). Then, the Rough-set analyzer looks for the concepts (the decision attributes) that indicate the nature of a pattern (either malicious or benign). The approximation step involves the use of the RST lower approximation. Throughout this step, only the certain dataset (i.e., the certain generated artificial malicious patterns $CAMDB$) given by the lower approximation is kept (Algorithm 2, line 7). Subsequently, this same *certain* dataset is sent to the RS-BMD's upper-level for rule extraction.

Algorithm 2 Lower-level algorithm

Require: $MPDB$: set of malicious patterns, $BPDB$: set of benign patterns, SDR: set of generated detection rules, G: number of generations, N: population size.

Ensure: Set of certain generated artificial malicious patterns $CAMDB$

1: $SAP_0 \leftarrow$ Initialization($BPDB$,$MPDB$,N,G)
2: $SAP_0 \leftarrow$ Evaluation(SAP_0,$BPDB$,$MPDB$, SDR) /*Evaluation depends on SDR*/
3: $t \leftarrow 1$
4: **while** $t < G$ **do**
5: $Q_t \leftarrow$ Variation(SAP_{t-1})
6: $Q_t \leftarrow$ Evaluation(Q_t,$BPDB$,$MPDB$,SDR)
7: $U_t \leftarrow Q_t \cup SAP_t$
8: $SAP_{t+1} \leftarrow$ Selection(N,U_t)
9: $AMDB \leftarrow$ FittestSelection(SAP_t)
10: $t \leftarrow t+1$
11: ($CSAP$, $ISAP$) $\leftarrow$InconsistencyCheck($AMDB$) /*Set of consistent AP and a set of inconsistent AP*/
12: $CAMDB \leftarrow$ LowerApproximation($ISAP$) $\cup$ $CSAP$
13: **end while**

The GA evolutionary mechanism

As already mentioned in Section 4.2.2, an adaptation of the used GA is also needed concerning

the specific aspects of the manipulated solutions (patterns) at the lower-level's algorithm which are respectively: the solution representation, the solution variation, and the solution evaluation.

Solution representation A GA is used to generate artificial patterns (chromosomes) which are composed of API call sequences (genes, also represented as item vectors in [16]). The API call sequences are identified via their identifiers (IDs). They are also described by their class labels indicating their nature (malicious or benign). They also have different calling depths. And finally they are composed of a set of binary values indicating if an API call shows or not in the whole API call sequence. Figure 4 represents an API call sequence (gene).

To generate an initial population (patterns), we start by defining some parameters (i.e., maximum number of API calls in a sequence (solution), the number of generations, etc.). These parameters can be either randomly chosen or specified by the user. More precisely, our used GA begins with a set of suitable solutions: the set of selected malicious patterns; namely $MPDB$. Aiming at finding a better population, the GA selects solutions from one population and uses them to generate a new one. The process is based on the solutions' fitness. The best fitting ones have more chances to reproduce new solutions. The process stops when a specific condition is met (i.e., the fixed number of generations is reached).

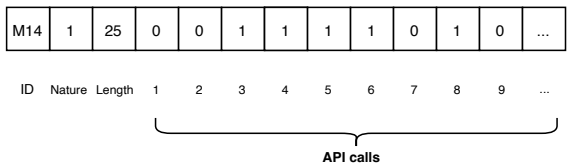

Fig. 4 A gene representation: a gene encodes an item vector corresponding to a particular behavior defined by a sequence of API calls [16].

Please, note that this second phase, i.e., the generation of malicious and benign patterns phase, was formerly proposed and detailed in [16].

Solution variation The crossover and mutation operators used when combining information from individuals (parents) should be adapted to our solution representation. For the GA (*crossover operator*), two parent chromosomes are selected, and a specific gene is chosen from each. The *crossover* then swaps these selected genes between the two parents. The crossover operator can be applied with only parents having the same nature (malicious or benign). Each child thus combines information from both parents. In any given generation, a variant will be the parent in at most one crossover operation. As for the mutation operator, it is applied to the selected chromosome to maintain genetic diversity from one generation of individuals to the next one. We start by randomly selecting a gene in the chromosome. Then, if the selected gene has a certain class (same nature), it is replaced by another gene from the same class.

Solution evaluation The quality of an artificial malicious pattern (AP) is evaluated based on the following GA's fitness function:

$$f_{lower}(AP) = Max(z + \sum_{i=1}^{N} f_{Qual}(AP_i)) \quad (7)$$

where $i \in [1, n]$; n indicates the total number of artificially generated patterns, and

$$z = \#gamp - \#dagmp \quad (8)$$

$\#gamp$ refers to the number of generated artificial malicious patterns and $\#dagmp$ refers to the number of detected artificial generated malicious patterns.

The function $f_{Qual}()$, defined in Equation 9, guarantees the diversity of the generated malicious patterns.

$$f_{Qual}(AP_i) = \frac{Sim_1 + Sim_2 + Overlap(AP_i)}{3} \quad (9)$$

Based on $f_{Qual}()$, the quality of a solution which refers to an artificially generated pattern (AP_i) is evaluated using the following three criteria:

1. $Sim_1 = Sim(MS, AP_i)$ refers to the similarity between the generated pattern AP_i and the malicious patterns (MS).

This measure of similarity needs to be maximized.

$$Sim(MS, AP_i) = \frac{\sum MS_{j \in MS} Sim(AP_i, MS_j)}{\mathsf{MS}} \quad (10)$$

where $j \in [1, m]$; m indicates the total number of malicious patterns.

2. $Sim_2 = Sim(BS, AP_i)$ refers to the similarity between the generated pattern AP_i and the benign patterns (BS) which has to be the lowest.

$$Sim(BS, AP_i) = \frac{\sum BS_{k \in BS} Sim(AP_i, BS_k)}{\mathsf{BS}} \quad (11)$$

where $k \in [1, p]$; p indicates the total number of benign patterns.

3. Overlap(AP_i) is measured as the average value of the individual $Sim(AP_i, AP_l)$ between the generated pattern AP_i and all the other generated patterns AP_l in the generated data set $AMDB$. l refers to the total number of the generated artificial patterns.

$$Overlap(AP_i) = 1 - \frac{\sum AP_{l, i \neq l} Sim(AP_i, AP_l)}{\mathsf{AP}} \quad (12)$$

To calculate the similarity $Sim()$, used in the above equations but with different parameters, between two patterns, we adapted the Needleman-Wunsch [39] alignment algorithm to our context. A detailed description of the similarity function $Sim()$ can be found in [16].

The generated malicious patterns ($AMDB$) need to be inspected (i.e., structure, nature, etc.) before injecting them in the final base of certain generated artificial malicious patterns ($CAMDB$). In fact, during this process, the generated malicious patterns can have some inconsistency issues as shown in Table 2. A set of patterns is declared inconsistent when the patterns share the same values of the condition attributes but do have different decision attributes' values. Consequently, these malicious patterns will be the subject of a selection using RST; more precisely by the Rough-set analyzer. By using the lower-approximation concept, only the **certain** malicious patterns (the consistent ones) will be kept. The detection rate is expected to be improved and the false alarms' rate is expected to decrease.

Illustration of inconsistencies In this illustration, RS-BMD is applied to the Drebin dataset [40]. The manipulated patterns by the lower-level are API call sequences. Each API call sequence is named MFX_i (as shown in Table 2) and is composed of different API calls named MLX_j. A conflict (or inconsistency) may exist between objects (artificial patterns). It is the case of the objects MFX_7 and MFX_9 because they are indiscernible by condition attributes $MLX_1, \ldots, MLX_n$ and have different decision attributes (Nature) (we assume that all attribute values MLX_j are the same). Similarly, another inconsistency exists between objects MFX_3 and MFX_8.

Table 2 Examples of ambiguous patterns.

Malicious artificial patterns	Condition attributes (API call) MLX_1	MLX_2	...	MLX_n	Decision (Nature)
MFX_1	1	1	...	1	M
MFX_2	0	0	...	0	M
MFX_3	1	0	...	0	M
MFX_4	1	0	...	1	M
MFX_5	1	1	...	0	M
MFX_6	1	0	...	1	M
MFX_7	1	0	...	1	B
MFX_8	1	0	...	0	B
MFX_9	1	0	...	1	M
MFX_{10}	1	1	...	0	B

After removing the inconsistent patterns, only the certain ones will be kept. In the same way, in the upper-level (Table 3), a conflict (or inconsistency) exists between objects (detection rules) DRX_5 and DRX_6 because they are indiscernible by condition attributes MFX_1, ..., and MFX_m and have different decision attributes (Label) (we assume that all attribute values MFX_j are the same). Similarly, another conflict exists between objects DRX_3 and DRX_9. In both levels, the origin of the inconsistency may be due to the crossover operator used by the EAs when generating respectively the malicious patterns (lower-level) and the detection rules (upper-level).

Table 3 Examples of ambiguous detection rules.

Detection rule	Condition attributes (pattern) MFX_1	MFX_2	...	MFX_m	Decision (Label)
DRX_1	0	0	...	0	N
DRX_2	0	0	...	1	M
DRX_3	0	1	...	0	N
DRX_4	0	0	...	0	N
DRX_5	1	1	...	0	M
DRX_6	1	1	...	0	N
DRX_7	0	0	...	0	M
DRX_8	1	1	...	0	N
DRX_9	0	1	...	0	M
DRX_{10}	0	0	...	0	M

4.2.4 Detection process based on detection rules: the three-way decision making module

Our model performs its classification task (upper-level problem) as illustrated in Figure 5, where a new app APP, the executable, will be classified into one of three possibilities: malware, benignware, or the decision is deferred to the user when the app does not clearly correspond to one of the previous possibilities (i.e., an uncertain decision). This is achieved by using the set of certain

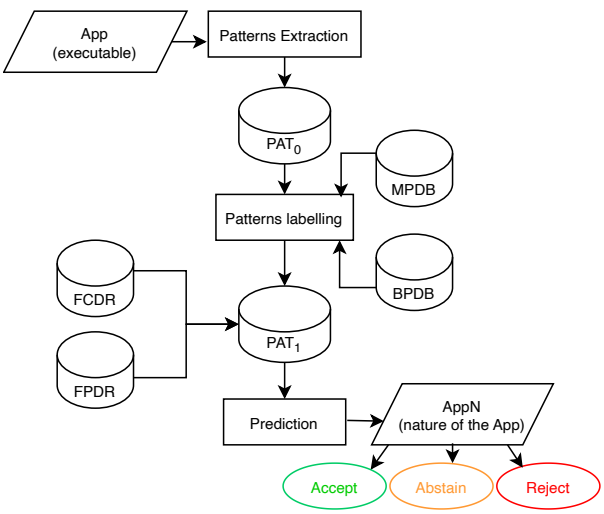

Fig. 5 The RS-BMD classification task.

detection rules ($FCDR$) which covers two types of detection rules: malicious ones and benign ones, and the set of *possible* detection rules ($FPDR$). Let us recall that $FCDR$ and $FPDR$ are the outputs of the upper-level phase. The algorithm for classifying a new app, the executable, is given in Algorithm 3.

Algorithm 3 Classification algorithm

Require: App: The executable, $MPDB$: malicious patterns database, $BPDB$: benign patterns database, $FCDR$: set of certain detection rules (malicious/benign), $FPDR$: set of possible detection rules (abstention).

Ensure: $AppN$: Label indicating the nature of the executable.

1: $PAT_0 \leftarrow$ Extraction(App)
2: $PAT_1 \leftarrow$ Labelling(PAT_0,$BPDB$,$MPDB$)
3: $AppN \leftarrow$ Prediction(PAT_1,$FCDR$,$FPDR$)
4: Return($AppN$)

The first step involves extracting the patterns (PAT_0) of the executable (Algorithm 3, line 1). Then, we proceed with pattern labeling: each pattern is labeled as benign or malicious by comparing it to the patterns in the $MPDB$ (malicious patterns) and $BPDB$ (benign patterns) databases. The labeled patterns form the set PAT_1 (Algorithm 3, line 2). Consequently, to predict the nature of a new app ($APPN$), the extracted labeled patterns are compared to the antecedents of all the final detection rules. If they match the antecedent of a possible rule among the set of final possible detection rules ($FPDR$), then the user is given the opportunity to decide about the app's nature: this represents an abstain decision, and the prediction is accompanied by a $Safety\_index$. The $Safety\_index$ could assist the user in making a decision by providing an indication of the trust level associated with the used possible rule. Otherwise, if the extracted app's patterns match the antecedent of one of the certain rules ($FCDR$), then the app is classified as malicious or normal based on the rule's label (Algorithm 3, lines 3-4) (accept or reject decision).

5 Experimental study

In this section, we present and analyze the experimental results of RS-BMD when applied to real-world settings. We conduct a comparative study with a set of state-of-the-art malware detection approaches and engines.

5.1 Research questions and benchmark datasets

The aim of this research study is to answer a set of research questions (RQs) which are presented as follows:

- **RQ1**: How accurately does RS-BMD detect malicious patterns?
- **RQ2**: How does RS-BMD perform when compared to the state-of-the-art methods?
- **RQ3**: How does the BLOP component benefit the RS-BMD approach?
- **RQ4**: What advantages does RS-BMD offer in addressing inconsistency issues at both levels?
- **RQ5**: Does a three-way decision making fashion help RS-BMD improve its detection rates?

The evaluation of RS-BMD's performance will help us provide an answer to **RQ1**, via the use of the following evaluation metrics: true positives (TP), false positives (FP), true negatives (TN), false negatives (FN), recall, specificity, accuracy, precision, F1_score, and the Area Under the Receiver Operating Characteristics (ROC) Curve (AUC). All of these metrics are discussed in Section 5.3. To answer **RQ2**, we compare our obtained results to those accomplished by recent state-of-the-art methods using the same evaluation metrics previously stated. This is to show the outperformance of RS-BMD in comparison to other anti-malware systems when confronted to a set of unknown malware variants. Also, we evaluate the required run time by our proposed approach via the *R_ Time* measure based on different parameters settings, in order to highlight the ability of our approach in detecting efficiently malicious apps within a reasonable time-frame when compared to previous works. All these evaluations are discussed in Section 5.4 and all the details about the methods used for the sake of these comparisons are given in Section 5.2. The values of false positive rate and false negative rate will bring answers to **RQ3** and they will be discussed in Section 5.5. To answer **RQ4** and starting with the upper-level, a comparison between the set of obtained detection rules before and after the inconsistency check accomplished by the upper-level Rough-set analyzer will highlight the usefulness of such a task (Section 5.6.1). The same goes for the lower-level by comparing the number of generated malicious patterns before and after applying the RST inconsistency check (Section 5.6.2). As an answer to **RQ5**, we discuss the number of the obtained possible and certain (accept and reject) detection rules along with the registered accuracy values of RS-BMD in comparison to different other detection approaches and engines. This is presented in Section 5.7.

The considered datasets in our experimental study for the RS-BMD's evaluation are obtained from the Android Malware Data set (AMD set) [41], from the DROIDCat dataset [42], and from various portable benign tools such as Google play. We have gathered 3 000 Android apps where 2 000 are malicious and 1 000 apps are benign files. The Drebin dataset [40], which contains 123 453 benign applications and 5 560 malware samples, is used for the evaluation of RS-BMD against the new variants of malware and 0-day attacks. The choice of testing our detection method using a dataset that is different from the one used for building is based on the need of a proof that RS-BMD is not fitting the base of examples.

5.2 Peer algorithms and parameters settings

To respond to the different previously highlighted research questions, we have used different state-of-the-art methods for the sake of comparison. These are categorized into three main comparison sets: the first set of comparison aims to compare RS-BMD to non-EA-based classifiers and two GAN-based methods. The GAN-based methods are those of [29, 30] and the set of non-EA-based classifiers are the following: Logistic Regression (LR), Linear Discriminant Analysis (LDA), Random Forest (RF), Decision Tree (J48), Naive Bayes (NB), Reinforcement Learning (RL), k-Nearest Neighbours (k-NN) and Tabu Search and Decision Tree (TDST). The second set of comparison aims to compare RS-BMD to four recent and specific EA-based state-of-the-art approaches which are similar in some aspects to RS-BMD. These are AMD [16], Sen et al. [15], [43] and [17] which were previously discussed in Section 2. The selection of these detection techniques is justified by the fact that we took into consideration all common traits between our developed approach RS-BMD, the work of Jerbi et al. (AMD [16], BMD [17] and [18]) and the work of Sen et al. [15].

The third set of comparison aims to compare RS-BMD to a set of commercial anti-malware engines which are Cyren, Ikarus, VIPRE, McAfee, AVG, AVware, ESET NOD32, CAT QuickHeal, Aegis-Lab and NANO. All of them were compared in terms of accuracy using the same dataset (Drebin [40] dataset).

Concerning the parameters' settings, for the first set of comparison, we used Weka[1] with the proposed default parameter settings. For the second set of comparison, and to ensure the fairness of comparisons between evolutionary approaches, we utilized the parameter settings described in Table 4. In this way, all of the evolutionary approaches performed 810,000 function evaluations in each run. We conducted several experiments using different values of the population size and the number of generations by applying the trial and error method at both levels. We succeeded in generating 476,000 malicious patterns with Eclipse[2] (approximately half the number of the generated malicious patterns in our experiment), with a total number of 4,407 API calls (items). Both RS-BMD levels were run with a population of 30 individuals and 30 generations. The following reasons can explain the use of a reduced population size at both levels: according to our formulation, detection rules are evaluated at the upper level based not only on its performance with respect to the upper fitness function but also on its performance in detecting associated generated malicious patterns by the lower level. In this way, the lower level assists the upper level in *(i)* identifying uninteresting upper search directions that should be disregarded and *(ii)* favoring interesting ones, thereby reducing the number of required evaluations at the upper level.

For the lower level, we also maintained the same population size (30 individuals), as our objective is to gain insight into the performance of the detection rule at the lower level. Consequently, the algorithm will perform 810,000 fitness evaluations for each level. Regarding the variation operators, we used a crossover rate of 0.9 and a mutation rate of 0.5. During the experiments, we observed that when using a population size of 30 for both levels, the fitness functions stabilized around the 40^{th} generation. Consequently, the algorithms did not suffer from premature convergence. Thus, the comparison is fair not only from the standpoint of the stopping criterion but also from the parameter setting perspective.

As for the third set of comparison, we relied on a free online service, VirusTotal[3], a subsidiary of Google, which analyzes files and URLs using various antivirus engines and website scanners. All experiments were conducted using a 10-fold cross-validation test and were run on an *Intel Xeon* Processor CPU E5-2620 v3 with 16 GB of RAM.

5.3 Analysis of the RS-BMD performance

To demonstrate the performance of RS-BMD, we primarily focus on the results presented in Table 5. This will allow us to answer specifically **RQ1**, and partly **RQ3**, **RQ4**, and **RQ5**. In our experiments, we collected a set of apps, including 2,000 malicious executables and 1,000 benign executables. From this dataset, we extracted a total of 29,483,201 distinct malicious item sets (i.e., API calls) and 11,302,447 distinct benign item sets. These item sets were used to form a final set of 27,534,880 malicious patterns ($MPDB$) and 10,172,203 benign patterns ($BPDB$), which will be used as input for RS-BMD. The various sets obtained are summarized in Table 6. Note that the inputs and results related to the non-EA methods (the classifiers and the GAN-based methods mentioned in Section 5.2), as well as the EA-based methods (AMD, BMD, ProRSDet, and Sen et al.), will be discussed later in Section 5.4.1 and Section 5.4.2, respectively.

False negatives and false positives analysis An increase in false positives, or type 1 errors, is less alarming since they are considered less harmful than false negatives (type 2 errors). Our aim is to keep both the false positive and false negative rates as low as possible. When analyzing 27,534,880 malicious patterns and 10,172,203 benign patterns, the scored values of FP and FN for RS-BMD are respectively 01.61% for FP and 01.35% for FN, as shown in Table 5. The scored values of FP and FN can be considered as interesting ones, and this is mainly due to the bi-level component and the RST modules, both of which helped

[1] https://www.cs.waikato.ac.nz/ml/weka/
[2] https://www.eclipse.org/
[3] https://www.virustotal.com

Table 4 EAs' parameters used by each approach.

Approach	Parameters					
	Population size		Generation size	Crossover rate	Mutation rate	Number of evaluations
RS-BMD	Upper- level	30	30	0.9	0.5	810 000
	Lower-level	30	30	0.9	0.5	810 000
AMD	180		4 500	0.9	0.5	810 000
BMD	Upper- level	30	30	0.9	0.5	810 000
	Lower-level	30	30	0.9	0.5	810 000
ProRSDet	First layer	30	30	0.9	0.5	810 000
	Second layer	30	30	0.9	0.5	810 000
Sen et al.	Malware generation	500	1 000	0.1	0.9	500 000
	Anti-malware generation	310	1 000	0.1	0.9	310 000

Table 5 Ten-fold cross validation results.

Classifier	TP	FP	TN	FN	Rec	Spe	Acc	Pre	FS	AUC
LR	93.81	06.19	96.75	03.25	96.65	93.98	95.28	93.17	95.60	63.69
NB	92.30	07.70	28.41	71.59	56.31	78.67	60.35	92,37	93,62	65.06
RF	97.41	02.59	95.90	04.10	96.00	98.37	97.16	97.36	97.17	73.04
J48	97.18	02.82	93.98	06.02	94.27	97.13	96.58	97.73	97.96	83.90
k-NN	89.52	10.48	95.21	04.79	94.92	90.08	92.37	85.74	90.56	57.69
LDA	97.29	02.71	98.36	01.64	98.34	97.31	97.82	98.36	97.32	75.96
TDST	97.69	02.31	96.65	03.35	96.68	97.65	97.17	97.69	97.18	75.23
RS-BMD	**98.39**	**01.61**	**98.45**	**01.35**	**98.40**	**98.40**	**98.51**	**98.39**	**98.12**	**87.13**

Rec: recall; Spe: specificity; Acc: accuracy; Pre: precision; FS: F1_score

in maintaining efficient and consistent detection rules. Also, it is still possible to increase our base of examples with more benign and malicious patterns to reduce the number of FPs and FNs. However, we have to be aware that making the detection model overfitting may cause a degradation of detection performance in real-life settings.

Precision interpretation When the number of false positives is relatively high, it is recommended to determine the precision value because it indicates the number of predicted positive instances that are actually positive. In our detection model, a false positive means that a pattern that is benign (actual negative) has been identified as malicious. Consequently, the detection model might overlook important apps if the precision is not high. From Table 5, we can see that the achieved precision value is 98.39% for RS-BMD. Therefore, we can affirm that our RS-BMD approach is able to classify new instances with high precision. In fact, these results can be explained by the fact that our base of examples is kept fairly varied thanks to the inclusion of the *certain* generated malicious patterns which are guaranteed by the RST component.

Accuracy, recall and specificity interpretation Accuracy is simply the ratio of correctly predicted observations to the total observations. However, having a high accuracy does not necessarily mean that the model is perfect. Therefore, we have to look at other metrics (i.e., specificity and recall) to evaluate the performance of our model. In the conducted experiments, we achieved an accuracy of 98.51% for RS-BMD (as shown in Table 5), indicating that our model has high chances of being considered accurate. This can be attributed to the considerable number of observations that are correctly predicted. These good results demonstrate the benefits gained from not selecting a static base of

examples but rather opting for a varied base of examples enriched by the artificially generated patterns, output of the lower-level, when constructing our RS-BMD bil-level detection model.

Recall is the recommended metric for selecting the best model when there is a significant cost associated with false negatives. In a detection model, the recall metric calculates how many of the actual positives the model correctly identifies as positive (true positives). In a malware detection model, predicting a fraudulent behavior (actual positive) as non-fraudulent (predicted negative) can cause damage to the operating system and consequently result in losses for the user. Therefore, a recall value of 98.40% for RS-BMD is positively interpreted. This satisfactory value can be attributed to the high number of true positives accurately detected at 98.39% by RS-BMD.

Specificity measures a test's ability to correctly identify negative instances that do not have the condition being tested for. A high specificity test will accurately predict nearly every negative instance and therefore will not generate a high number of false positives. For example, the specificity value of 98.40% obtained by RS-BMD is reassuring. This result is explained by the high number of true negatives accurately detected. The *certain* generated malicious patterns resulting from the RST component, which enriched the base of examples, are of high quality and hence contributed to better detection of malicious patterns.

F1_score and AUC interpretation When measuring how well our detection approach is doing, it is useful to have the F1_score to describe its performance. In Table 5, we can see that for RS-BMD we reached 98.12% of F1_score and this could be explained by the high values of precision (98.39%) and recall (98.40%) achieved by our detection model. AUC provides an aggregate measure of performance for all possible classification thresholds. One can interpret the AUC as a measure of the probability that the model will rank a random positive example above a random negative example. The AUC value obtained is 87.13%, which means that the achieved detection rules could be considered as efficient in separating malicious and benign instances. We can deduce that when we assure a continuous variability to our base of examples, guaranteed by the BLOP component, and by injecting the instances from the set of the *certain* generated malicious patterns, guaranteed by the RST component, we guarantee a better detection of malware.

5.4 Comparison with the state-of-the-art methods

In this section, we mainly bring answers to **RQ1** and **RQ2**, and partial answers to the rest of the research questions. We will start the comparison with the non-EA-based techniques and bring answers to **RQ1** and **RQ2**. After that, we will compare RS-BMD to the EA-based techniques, where we will specifically analyze the scored values of false positive and false negative rates, to show the benefits of using a bi-level approach. This will bring answers to **RQ3**. In this set of comparison, we will also highlight the benefits of using RST for both inconsistency check and three-way-decision making. This will partly bring answers to **RQ4** and **RQ5**. It is to be noted that the later two research questions will be separately investigated in Section 5.6 and Section 5.7, respectively. At the end, we will proceed with a comparison with the commercial anti-viruses and again bring answers to **RQ1** and **RQ2**; mainly.

5.4.1 Comparison with non-EA-based techniques

Comparison with non-EA-based classifiers

The section provides comparisons between RS-BMD and various state-of-the-art non-EA-based classifiers, with results presented in Table 5. The input data for these experiments comprises sets of 27,534,880 malicious patterns (i.e., sequences of API calls) ($MPDB$) and 10,172,203 benign patterns ($BPDB$), as summarized in Table 6.

From Table 5, we can see that, on the one hand, the LDA classifier reached interesting values of accuracy and precision with 97.82% and 98.36%, respectively, thanks to the reached values of TP (97.29%) and TN (98.36%). Regarding the accuracy, NB came last with 60.35%. This rate was affected by the poor reached FP rate

Table 6 The number of obtained artificial malicious patterns in evolutionary approaches RS-BMD, AMD, and Sen et al.

Approach	Number of apps	Number of API calls (item sets)	Number of patterns	Number of the generated malicious patterns
RS-BMD				256 000**
AMD	1 000 benign	11 302 447 benign	10 172 203 benign	476 000*
BMD	2 000 malicious	29 483 201 malicious	27 534 880 malicious	476 000*
ProRSDet				288 000**
Sen et al.	2 000 malicious	29 483 201 malicious	27 534 880 malicious	772 000*

: **certain malicious patterns
*: **not checked** set which may include inconsistent patterns

(02.71%). Concerning the recall and the specificity metrics, both the LDA and $TDST$ classifiers had close values with a recall of 98.34% for LDA and 96.68% for $TDST$, and a specificity of 97.31% for LDA and 97.65% for $TDST$. The $J48$ classifier outperformed the rest of the classifiers in terms of F1_score and AUC with the respective values of 97.18% of TP and 02.82% of FP. Among all other classifiers, and for the same metrics, the k-NN classifier is ranked last with an F1_score of 90.56% and an AUC of 57.69%, which corresponded to 89.52% of TP and 10.48% of FP. Also, from Table 5, we can see that RS-BMD outperformed all classifiers with respect to all of the used evaluation metrics. To further highlight the performance of our RS-BMD technique, we have performed a graphical analysis of the ROC curve for the different used classifiers.

The ROC curve offers a classical method to validate the obtained AUC value by plotting a ROC curve and estimating the AUC. Figure 6 depicts the ROC curves obtained: The first curve (top) illustrates the False Positive Rate (FPR) on the X-axis and the accuracy on the Y-axis, while the second curve (bottom) displays the False Positive Rate (FPR) on the X-axis and the True Positive Rate (TPR) on the Y-axis for all possible thresholds (or cutoff values). These curves aid in selecting the most suitable thresholds required for our experiments. From Figure 6, we observe the ROC curves of the LDA, $TSDT$, and RF classifiers, which stand out among all obtained classifier's ROC curves, characterized by very similar shapes and strong support. Conversely, the ROC curves for the $J48$, k-NN, NB, and LR classifiers overlap. In comparison to all these classifiers, RS-BMD's ROC curves perform the best. For instance, the optimal cutoff for RS-BMD exhibits the highest accuracy of 98.51%, the highest true positives of 98.39%, and the lowest false positives of 01.61%. Visualizing accuracy, sensitivity, and specificity as a function of the cutoff points in a plot enhances the informativeness and interpretability of the results. The ROC curves obtained for RS-BMD closely align with both the left-hand and top borders of the ROC space, indicating high accuracy of the results. However, while the shapes of the ROC curves are promising, they alone may not provide a comprehensive interpretation of the results. Therefore, we calculated and discussed the AUC value previously, serving as a quantitative summary to assess the effectiveness of the RS-BMD retained detection rules in classifying positive and negative instances.

Comparison with GAN-based state-of-art methods

In this section, we will compare our RS-BMD method to two state-of-the-art GAN-basd methods (discussed in Section 2) which are MalGAN

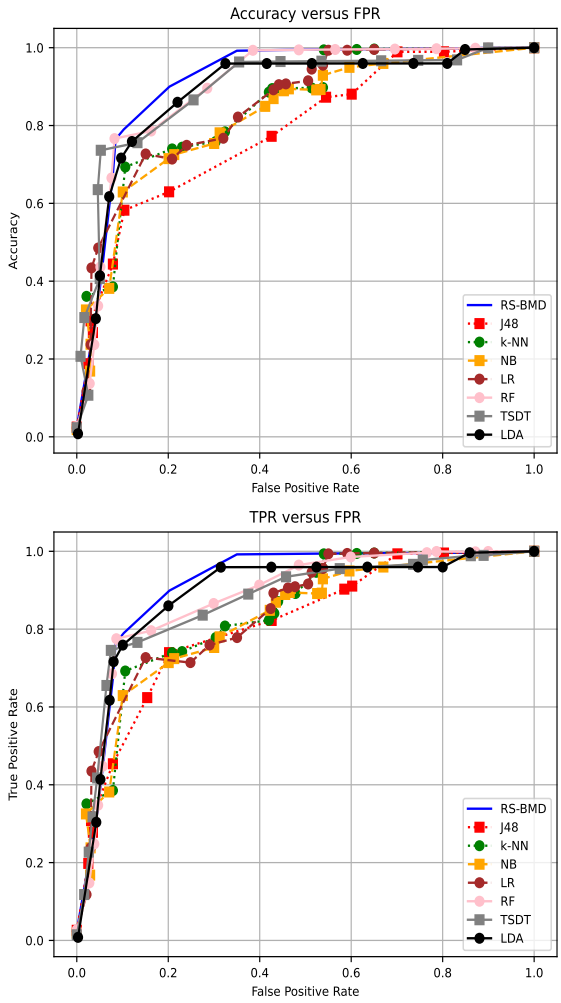

Fig. 6 The obtained ROC curves with RS-BMD and the different classifiers.

[29] and Kang et al.'s method [30]. The comparison is performed based on the accuracy metric registered values reported in Table 7.

Table 7 Detection rates of GAN-based methods and RS-BMD.

Method	Accuracy (%)
MalGAN	85.00
Kang et al.	70.16
RS-BMD	**98.51**

From Table 7, we can say that our RS-BMD outperformed MalGAN and Kang et al.'s method in terms of accuracy. In fact, RS-BMD registered 98.51% of accuracy while MalGAN and Kang et al.'s registered 85% and 70.16% respectively. These GANs are powerful for generating new data instances. They typically require a fixed structure and rely on the interplay between the generator and discriminator to refine output quality. However, this makes them less flexible when it comes to discovering entirely new solutions (i.e., malware variants) which is not the case of our RS-BMD method. RS-BMD is in fact less affected by the quality of the input data as it is a self-learning algorithm.

5.4.2 Comparison with EA-based techniques

In this section, a number of comparisons between RS-BMD and four main state-of-the-art EA-based techniques was performed in terms of different evaluation metrics and execution time. The input data for these experiments is summarized in Table 6 (27 534 880 malicious patterns and 10 172 203 benign patterns; for AMD, BMD, ProRSDet and RS-BMD, and 27 534 880 malicious patterns for Sen et al.). It is worth noting that Sen et al. [15] used only malicious patterns in their malware generation task.

Evaluation of RS-BMD based on the used evaluation metrics

Table 8 demonstrates that RS-BMD outperforms Sen et al., AMD, BMD, and ProRSDet across all metrics. For example, RS-BMD achieved an accuracy of 96.76%, while ProRSDet, BMD, Sen et al., and AMD attained 96.66%, 95.16%, 95.15%, and 92.28%, respectively. Regarding precision, the values are as follows: RS-BMD 97.23%, BMD 96.76%, ProRSDet 96.31%, Sen et al. 97.13%, and AMD 93.60%. From the same table, we can observe the values of recall and specificity for all approaches, ranked from highest to lowest. For RS-BMD, the values are (98.35%, 97.97%), for BMD (98.27%, 95.37%), for Sen et al. (98.24%, 95.37%), for ProRSDet (96.99%, 96.32%), and for AMD (96.20%, 92.70%). These values are primarily influenced by the true positive rates of each approach. Furthermore, the precision and recall values provide insight into the F1_score performance of each approach. RS-BMD achieved the highest F1_score at 96.71%. Similarly, RS-BMD excelled in AUC with a value of 87.30%. These promising results underscore the advantages of RS-BMD's bi-level architecture, particularly the

integration of RST techniques at both levels. This architecture facilitated the generation of robust detection rules, leveraging the high quality of the generated malicious patterns ensured by the RST modules.

Evaluation of RS-BMD based on its execution time

In this section, we evaluate RS-BMD in terms of execution time (R_time) – the time needed by our RS-BMD approach to produce the detection rules. The discussion of the results is based on Figure 7. It is expected that RS-BMD will consume higher execution time than all other approaches. This can be explained by the fact that RS-BMD plugs two EAs both enriched with RST modules. All of them need to be executed in an embedded way to optimize both the upper and the lower levels.

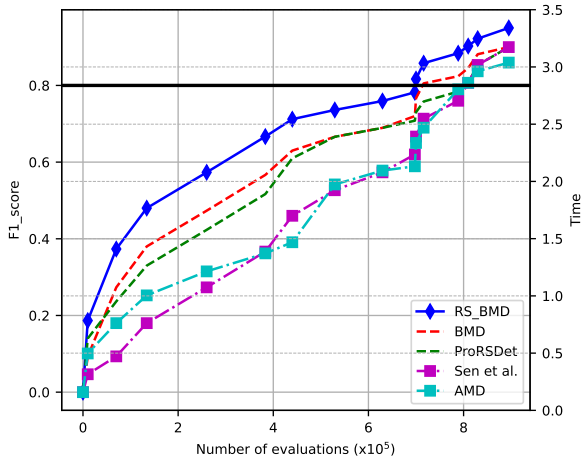

Fig. 7 The EA-based approaches' evolution in time to reach suitable results. (F1_score = 0.8).

Based on the obtained results, from Figure 7, RS-BMD took about 4.5 hours execution time, whereas, AMD took 3.35 hours and Sen et al. took 2.75 hours. This can be explained by the fact that both AMD and Sen et al. use only one EA for the malware generation task.

Despite RS-BMD requiring more execution time than both AMD and Sen et al., its execution time can be justified as the algorithm is executed only once. It is important to remember that the primary goal of the algorithm's execution is to generate the rules used for detecting malicious patterns. A new execution of the entire algorithm is recommended only when significant updates are made to the base of examples used by the upper-level. Furthermore, this relatively high runtime is not problematic because we are not operating in a real-time setting. The algorithms can run continuously. When we need new detection rules to enhance the base of examples, we select the best rules from RS-BMD in a delayed mode. We used the F1_score metric to evaluate the quality of the best solution at each iteration for our RS-BMD approach. An F1_score value higher than 0.8 is considered an indication of acceptable detection rules based on our dataset. We selected a threshold value of 0.8 since it represents a good balance between precision and recall that can lead to acceptable detection solutions. As seen in Figure 7, after around 685,000 evaluations, RS-BMD generated detection rules that have an F1_score value of 0.8. So, RS-BMD needed fewer evaluations than both AMD [16] and Sen et al. [15] (Figure 7) to generate efficient detection rules. In fact, Sen et al. and AMD required approximately 772,000 and 800,000 evaluations, respectively, as seen in Figure 7, to achieve a similar outcome. Hence, we can conclude that the improvement in the quality of the generated malicious patterns fed to the upper-level helped it rapidly generate effective detection rules. Although RS-BMD needs important execution time, it is clear that the good solutions provided by a single-level approach can be reached quickly by our bi-level adaptation as described in Figure 7. So, we can conclude that the lower-level helped the upper-level to quickly generate efficient detection rules.

To sum up, we believe that an execution time of about 4 hours remains acceptable, since the developers will not use our tool in their daily activities, they just need to execute it once to extract the rules.

5.4.3 Comparison with top ten commercial anti-malware

This section focuses on comparing RS-BMD against a set of 10 antivirus engines, which are considered among the best antiviruses in the malware detection field. This is to give answers to **RQ1** and **RQ2**. As previously mentioned, we used new variants of malware and 0-day attacks stored in the Drebin [40] dataset. The obtained results in

Table 8 The different obtained measures for all compared evolutionary approaches in terms of TP, FP, TN, FN, precision, recall, specificity, accuracy,F1_score and AUC on Drebin data set [40].

Approach	Obtained values									
	TP	FP	TN	FN	Rec	Spe	Acc	Pre	FS	AUC
RS-BMD	**97.23**	**02.77**	**98.29**	**01.71**	**98.35**	**97.97**	**96.76**	**97.23**	**96.71**	**87.30**
AMD	93.80	06.19	90.90	09.10	96.20	92.70	92.28	93.60	92.37	57.69
BMD	95.23	04.77	98.29	01.71	98.27	95.37	95.16	96.76	95.23	87.23
ProRSDet	96.31	03.69	97.01	02.99	96.99	96.32	96.66	96.31	96.65	86.00
Sen et al.	97.10	02.80	93.25	06.75	98.24	95.37	95.15	97.13	95.88	82.10

Rec: recall; Spe: specificity; Acc: accuracy; Pre: precision; FS: F1_score

terms of the accuracy metric are recorded in Table 9 for all the engines.

Table 9 Accuracy results of RS-BMD and top ten commercial engines by Virus-Total on Drebin data set [40].

Anti-malware	Reference	Acc(%)
RS-BMD	**Our current approach**	**96.76**
ESET NOD32	https://www.eset.com	66.68
AegisLab	www.aegislab.com	66.23
NANO antivirus	http://www.nanoav.ru	66.23
VIPRE	https://www.vipre.com	62.53
McAfee	https://www.mcafee.com	56.21
Ikarus	https://www.ikarussecurity.com	55.65
AVG	https://www.avg.com	55.56
CAT QuickHeal	www.quickheal.com	54.23
AVware	http://www.avware.com.br/comprar.php	45.56
Cyren	https://www.cyren.com	45.23

Table 9 shows that all of the 10 anti-malware engines reached close results regarding the accuracy metric as all of the registered values lied between 45.23% (Cyren) and 66.68% (ESET NOD32). RS-BMD is ranked first when compared to the 10 anti-malware engines with 96.76% of accuracy. Specifically, we can split these engines into three main groups based on their accuracy rate: a first group (ESET NOD32, AegisLab and NANO antivirus) achieved a bit over 66%, a second group (VIPRE, McAfee, Ikarus, AVG, and CAT QuickHeal) achieved an accuracy rates that lied approximately between 54% and 63%, and a third group (AVware and Cyren) which registered about 45% of accuracy. This experiment brings to light the efficiency of RS-BMD in detecting unknown variants of malware, and hence shows (i) how accurate RS-BMD is in detecting malicious patterns (**RQ1**), and (ii) how RS-BMD performs when compared to the state-of-the-art methods (**RQ2**); i.e., the commercial anti-malware engines in this case.

5.5 Performance analysis of RS-BMD in terms of generating efficient detection rules and high quality malicious patterns (the BLOP component)

Analyzing the False Positive Rate (FPR) and the False Negative Rate (FNR) is necessary to know the reliability that may be accorded to a given malware detection system. Those rates will give us answers to **RQ3**. Relying on the values recorded in Table 10, RS-BMD reached interesting values of both FPR (02.74%) and FNR (01.72%). Actually, these values undeniably show the merit of the bi-level architecture adopted by RS-BMD in producing high quality detection rules and high quality malicious patterns which kept FNR and FPR as low as possible. Note that FPR and FNR were deduced from Table 8 and stored separately in Table 10.

Table 10 Effect of the use of a bi-level approach (deduced from Table 8).

	Results				
Measure	Sen et al.	ProRSDet	BMD	AMD	**RS-BMD**
FPR %	02.91%	04.77%	03.69%	06.37%	**02.74%**
FNR %	06.49%	01.71%	02.99%	08.84%	**01.72%**

In order to situate our reached values of FPR and FNR regarding the EA-based state of-the-art approaches, we will report those two metrics for all EA-based approaches. The obtained results in Table 10, show that Sen et al. recorded 02.91% of FPR and 06.49% of FNR, ProRSDet registered 04.77% of FPR and 01.71% of FNR whereas BMD and AMD recorded respectively 03.69% and 06.37% of FPR, and 02.99% and 08.84% of FNR. The FPR and FNR values clearly show the important gain obtained from a robust upper-level in boosting the number of correctly predicted instances and how this clearly allows reducing both the FPR and the FNR. Also, the high quality of the artificial patterns in the lower-level helped the upper-level reaching those satisfactory results.

5.6 Performance analysis of RS-BMD in terms of dealing with data inconsistency (the RST component)

In this section, we will discuss the removed number of detection rules by the upper-level's Rough-set analyzer and the removed number of malicious generated patterns by the lower-level's Rough-set analyzer. Knowing how much instances were removed from each level of RS-BMD will give us an idea about the role played by RST in keeping the set of consistent and meaningful instances that will help improving the malware detection task. Also, we will compare RS-BMD to Sen et al.'s approach and Jerbi et al.'s previous approaches (AMD, BMD and ProRSDet) in terms of the number of the generated detection rules and the generated malicious patterns. Comparing the registered numbers of detection rules and the generated malicious patterns can explain the benefits gained from the choice of including RST in our developed approach. This whole analysis, detailed in Section 5.6.1 and Section 5.6.2, will undoubtedly bring answers to **RQ4**.

5.6.1 Upper-level inconsistency check

To show the important role played by RST in the upper-level, more precisely the Rough-set analyzer module, a study of the number of the removed and kept detection rules is necessary. These numbers are reported in Table 11.

Table 11 Numbers of consistent and inconsistent generated rules in RS-BMD.

Number of generated detection rules	
Inconsistent	Consistent
2 182 336	9 286 892

Table 11 clearly shows that there is a considerable amount of inconsistent rules that has been removed (2 182 336 detection rules). These inconsistent rules were detected by the upper-level's Rough-set analyzer. Specifically, the detection was achieved via the Lower approximation task (Figure 3). So, each detection rule that does not meet the selection criteria fixed by the Lower approximation is removed. We can conclude that the Rough-set analyzer module has not only alleviated the amount of the produced detection rules but also participated greatly in producing powerful detection rules.

Also, understanding the number of detection rules produced by other EA-based state-of-the-art approaches can help explain why RS-BMD outperforms them. This advantage is partly due to its RST module, which effectively removes inconsistent instances. In fact, Table 12 shows that AMD generated a total of 7 787 379 detection rules, BMD generated 6 801 000, ProRSDet produced 6 298 322 and that Sen et al. generated 14 054 405 detection rules against 5 896 652 *certain* rules generated by RS-BMD.

Table 12 Number of generated rules in AMD, Sen et al. and RS-BMD approaches.

Approach	Number of generated rules
RS-BMD	certain malicious **3 506 412** certain benign **2 390 240**
BMD	malicious 4 092 015 benign 2 708 985
ProRSDet	malicious 3 890 510 benign 2 407 812
AMD	malicious 4 432 825 benign 3 354 554
Sen et al.	14 054 405

We can conclude that RST helped RS-BMD maintain a better set of detection rules by removing the inconsistent ones. This improvement contributed to the overall efficiency of RS-BMD, as demonstrated in the evaluations discussed in Section 5.4.2.

5.6.2 Lower-level inconsistency check

In this part, we focus specially on the number of the removed generated malicious patterns to confirm if the lower-level's Rough-set analyzer helped in filtering the GA generated artificial patterns and in keeping only the **certain** malicious patterns.

According to Table 13, the removed number of inconsistent generated patterns is 212 000. Based on this number, we can confirm that the Rough-set analyzer helped keeping the malicious patterns of better quality. Those generated malicious patterns will be fed to the upper-level and consequently improve the quality of the generated detection rules produced by this level.

Table 13 Numbers of consistent and inconsistent generated malicious patterns in RS-BMD.

Number of generated patterns	
Inconsistent	Consistent
212 000	256 000

The worth of the lower-level's RST module can be also highlighted by comparing the amount of the produced malicious patterns of all three EA-based approaches. With respect to Table 6, we can note that the number of generated malicious patterns decreased to 256 000 *certain* malicious patterns against 476 000 generated malicious patterns for AMD and for BMD, 288 000 for ProRSDet and 772 000 malicious patterns for Sen et al.

Once again, we can conclude that RST helped RS-BMD in removing inconsistent generated artificial malicious patterns, thereby achieving superior detection results, as discussed in Section 5.4.2.

5.7 Performance analysis of RS-BMD in terms of three-way decision making (the RST component)

In this section, we provide answers to **RQ5** to demonstrate the advantages of employing a three-way decision-making approach in our developed RS-BMD method. To understand the practical use of a three-way decision fashion in a malware detection environment, the analysis of the generated number of *possible* rules and their "usefulness" for the end user seems necessary. We will hence refer to the obtained numbers of detection rules, specially the *possible* ones (Table 14). Also, we highlight the benefits gained from using a three-way decision fashion, granted by the upper-level's RST module, in improving the malware detection task to ensure the system's security. For this purpose, we evaluate RS-BMD approach in terms of accuracy and the false positive rate, as shown in Table 8, using the dataset provided by [40].

Table 14 summarizes the obtained numbers of all types of detection rules: the *certain* rules (5 896 148) and the *possible* rules (3 390 744) using all gathered 3 000 apps and also the generated certain malicious patterns; output of the lower-level. The obtained set of *possible* rules will grant the user to make an "extra choice" about the "fate" of an app which is not possible with other methods. In fact, in other approaches, this app would be more likely to be denied from accessing the system and consequently deprive the user from using it. The number of possible rules, representing approximately 37% of the total set of detection rules (across all types), also indicates the potential for an app to be denied access in systems that do not provide a third option to the end user.

Table 14 Number of final generated rules by RS-BMD's upper-level.

Generated rules		Values
Certain rules	Malicious rules	3 506 412
	Benign rules	2 390 240
Possible rules		3 390 744

Contrariwise, RS-BMD provides the user with the possibility to permit/refuse an app's installation thanks to the set of possible rules. When dealing with this set of *possible* rules, a $Safety\_index$ (Equation 2) that estimates the reliability degree of each *possible* rule is also provided to the user (as discussed in Section 4.2.4). This metric, specific to the evaluation of each *possible* rule, will help the user when choosing to pursue installing an app or not. To be more specific, Figure 8 represents the number of the obtained *possible rules* with regards to their $Safety\_index$.

Figure 8 shows that 31.07% of possible detection rules have a $Safety\_index$ that exceeds 50%. Also, 29.27% of possible rules succeeded to correctly classify apps with rates that lie between 41% and 50%. 16.67% of those rules ranked just below with a $Safety\_index$ comprised between

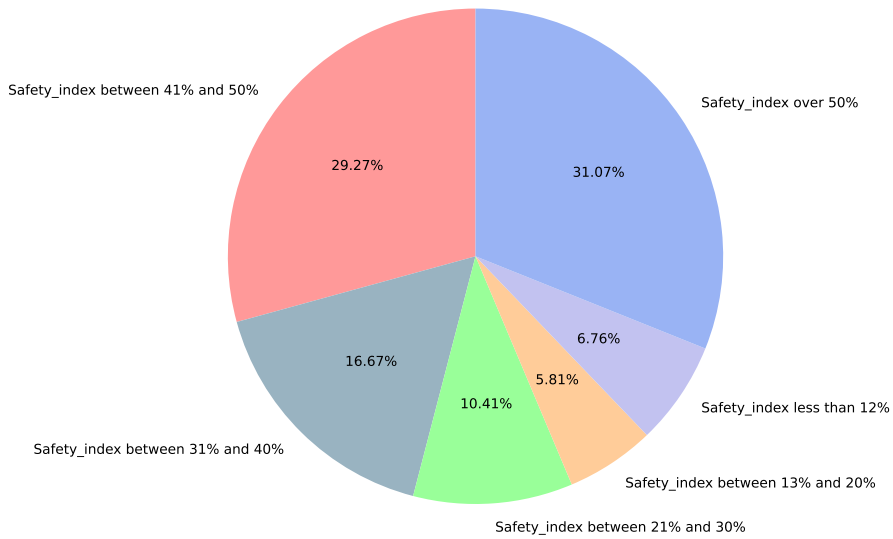

Fig. 8 Number of *possible* rules with regards to the Safety_index.

31% and 40%. Approximately only 23% of the rules failed to have a $Safety\_index$ above 30%. An example of the use of this index was previously given in Section 4.2.2.

Also, the additional set of detection rules, specifically the generated possible rules, significantly boosts the effectiveness of our RS-BMD approach. With an accuracy rate of 96.76% (Table 8) and a false positive rate of 01.72% (Table 10), our results are notably satisfactory compared to two-way decision-making EA techniques. For comparison, AMD reports an accuracy of 92.28% and a false positive rate of 06.37%, BMD achieves an accuracy of 95.16% and a false positive rate of 03.69%, ProRSDet has an accuracy of 96.66% and a false positive rate of 4.77%, and Sen et al. record an accuracy of 95.15% and a false positive rate of 02.91%. This highlights the importance of developing a detection system that not only relies on high-quality datasets but also maintains flexibility in app acceptance or denial decisions.

6 Limitations, threats to validity and future directions

Our RS-BMD approach has demonstrated potential in malware detection. However, we recognize limitations that align with typical threats to research validity. Addressing these will be our focus in future work to enhance the algorithm's robustness and applicability.

Internal validity threats: These threats question the causal relationship between the algorithm and the observed outcome in our study. Potential internal threats in our work include possible selection bias, potentially affecting the validity of our results. It arises when the dataset used to train and test the algorithm does not accurately represent the true diversity of malware. To address this, future work will include assembling a more comprehensive and varied dataset and expand the diversity of Android sample features beyond API calls to include additional features like permissions. This approach will help improve the detection ability significantly by providing a richer dataset for analysis.

External validity threats: External threats pertain to the generalizability of our findings to other settings or datasets. Our current results may not transfer to different types of malware or operating systems. We plan to test the RS-BMD approach using diverse datasets from various sources and to apply the model to different operating systems, enhancing its generalizability.

Construct validity threats: Construct validity threats emerge if there is a concern that the study's assessments may not accurately represent the study under examination. There is a risk that our current evaluation metrics do not fully capture the practical efficacy of malware detection. To address this and in light of our plan to integrate GAN techniques for malware generation and classification, we will reassess and potentially expand our evaluation framework to include a broader range of performance metrics. This expansion will incorporate metrics that assess the effectiveness of the synthetic instances generated by GANs, thereby ensuring our evaluations accurately reflect the capabilities of our hybrid EA and GAN approach in enhancing malware detection.

To sum up, by addressing these limitations and exploring these directions, we aim not only to refine the RS-BMD approach but also to meet the evolving challenges in malware detection. Hence, ensuring robust, scalable, and efficient detection capabilities.

7 Conclusion

In order to detect Android malware efficiently and effectively, we have developed a bi-level Android malware detection system based on both Bi-level Optimization and Rough Set Theory. BLOPs have proven their efficiency in various research domains but have yet to be investigated in the malware detection field. The competition that takes place between both levels in BLOPs provides high-quality solutions to the targeted problem. RST, on the other hand, was widely applied for malware detection for feature reduction as a main task and has fully proven its merits in such field. But combining both strategies to evolve and to detect malware was not proposed in literature.

In our RS-BMD approach, we used two levels that are in competition: an upper one or the leader which is responsible for the detection rules generation, and a lower one which aims to generate a set of certain malicious patterns. Each of these levels is expanded with RST modules. These modules are in charge of boosting the quality of the output of each level. In fact, the leader's Rough-set analyzer not only checks the consistency of the detection rules but also offers three types of them which are: rules for acceptance, rules for rejection and rules for abstinence. The follower's Rough-set analyzer is in charge of removing all inconsistent generated malicious patterns. In our bi-level approach, we have shown the usefulness of each component based on different performance metrics. On the one hand, the lower-level composed of a GA coupled with an RST module helped generate 257,000 *certain* malicious patterns using 10,172,203 benign and 27,534,880 malicious API call sequences. According to the different comparisons made in the overall experimental section, the bi-level technique enriched by RST modules helped improve the detection rate and reached a value of 97.31% with RS-BMD. This also proves the importance of the *quality* of the generated malicious patterns in altering the detection results. On the other hand, the upper-level was enriched with a third possibility for classifying the apps. In fact, most of the malware detection systems offer a two-way decision labeling: an app is either malicious or benign. In RS-BMD, thanks to the generation of a third set of rules "*possible* rules" granted by the upper-level Rough-set analyzer (more precisely the upper approximation), the labeling task can be given to the user. In fact, for the uncertain apps, which do not match the certain benign rules or the certain malicious ones, the user can make the choice to either continue the installation process of the app or to abort it. This may help reduce depriving users from installing their apps. The conducted experiments and the extensive evaluation of our RS-BMD prove the good results reached when compared to different state-of-the-art approaches and detection engines. The registered accuracy is nearly 97%. Also RS-BMD successfully dropped the FPR to 02.74%. Finally, our approach demonstrates considerable adaptability, effectively detecting patterns and countering a wide array of cybersecurity threats.

Data availibilty statement

Data sharing is not applicable to this article as no new data were created in this study. All the data used in the experimental section come from : (1) the Android Malware Data set (AMD set) [41], (2) the DROIDCat dataset [42] and (3) the Drebin dataset [40].

Compliance with Ethical Standards

Funding

This study did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Conflict of Interest

The authors declare that they have no conflict of interest.

Ethical Approval

This article does not contain any studies with human participants or animals performed by any of the authors.

Informed Consent

Not applicable, as this study did not involve human participants.

References

[1] Wang, S., Chen, Z., Yan, Q., Ji, K., Peng, L., Yang, B., Conti, M.: Deep and broad url feature mining for android malware detection. Information Sciences **513**, 600–613 (2020)

[2] Wang, Y., Wang, Q., Qin, X., Chen, X., Xin, B., Yang, R.: Dockerwatch: a two-phase hybrid detection of malware using various static features in container cloud. Soft Computing, 1–17 (2022)

[3] Masood, Z., Majeed, K., Samar, R., Raja, M.A.Z.: Design of epidemic computer virus model with effect of quarantine in the presence of immunity. Fundamenta Informaticae **161**(3), 249–273 (2018)

[4] Salvakkam, D.B., Saravanan, V., Jain, P.K., Pamula, R.: Enhanced quantum-secure ensemble intrusion detection techniques for cloud based on deep learning. Cognitive Computation, 1–20 (2023)

[5] Tong, F., Yan, Z.: A hybrid approach of mobile malware detection in android. Journal of Parallel and Distributed Computing **103**, 22–31 (2017)

[6] Martín, A., Menéndez, H.D., Camacho, D.: Mocdroid: multi-objective evolutionary classifier for android malware detection. Soft Computing **21**(24), 7405–7415 (2017)

[7] Xiong, P., Wang, X., Niu, W., Zhu, T., Li, G.: Android malware detection with contrasting permission patterns. China Communications **11**(8), 1–14 (2014)

[8] Chen, C.-M., Lai, G.-H., Lin, J.-M.: Identifying threat patterns of android applications. 2017 12th Asia Joint Conference on Information Security (AsiaJCIS), 69–74 (2017). IEEE

[9] Denźux, T.: 40 years of dempster-shafer theory. International Journal of Approximate Reasoning **79**(C), 1–6 (2016)

[10] Zhang, Q., Xie, Q., Wang, G.: A survey on rough set theory and its applications. CAAI Transactions on Intelligence Technology **1**(4), 323–333 (2016)

[11] Colson, B., Marcotte, P., Savard, G.: An overview of bilevel optimization. Annals of operations research **153**(1), 235–256 (2007)

[12] Wang, X., Miikkulainen, R.: Mdea: Malware detection with evolutionary adversarial learning. 2020 IEEE Congress on Evolutionary Computation (CEC), 1–8 (2020). IEEE

[13] Akandwanaho, S.M., Kooblal, M.: Intelligent malware detection using a neural network ensemble based on a hybrid search mechanism. The African Journal of Information and Communication **24**, 1–21 (2019)

[14] Lee, J., Jang, H., Ha, S., Yoon, Y.: Android

malware detection using machine learning with feature selection based on the genetic algorithm. Mathematics **9**(21), 2813 (2021)

[15] Sen, S., Aydogan, E., Aysan, A.I.: Coevolution of mobile malware and anti-malware. IEEE Transactions on Information Forensics and Security **13**(10), 2563–2574 (2018)

[16] Jerbi, M., Dagdia, Z.C., Bechikh, S., Said, L.B.: On the use of artificial malicious patterns for android malware detection. Computers & Security **92**, 101743 (2020)

[17] Jerbi, M., Dagdia, Z.C., Bechikh, S., Said, L.B.: Android malware detection as a bi-level problem. Computers & Security **121**, 102825 (2022)

[18] Jerbi, M., Dagdia, Z.C., Bechikh, S., Said, L.B.: Malware evolution and detection based on the variable precision rough set model. In: 2022 17th Conference on Computer Science and Intelligence Systems (FedCSIS), pp. 253–262 (2022). IEEE

[19] Bhattacharya, A., Goswami, R.T.: A hybrid community based rough set feature selection technique in android malware detection, 249–258 (2018)

[20] K, D., G, R., P, V., Shojafar, M., Kumar, N., Conti, M.: Featureanalytics: An approach to derive relevant attributes for analyzing android malware. CoRR **abs/1809.09035** (2018) arXiv:1809.09035

[21] Chen, R.-C., Cheng, K.-F., Chen, Y.-H., Hsieh, C.-F.: Using rough set and support vector machine for network intrusion detection system. 2009 First Asian Conference on Intelligent Information and Database Systems, 465–470 (2009). https://doi.org/10.1109/ACIIDS.2009.59

[22] Sengupta, N., Sen, J., Sil, J., Saha, M.: Designing of on line intrusion detection system using rough set theory and q-learning algorithm. Neurocomputing **111**, 161–168 (2013)

[23] Zhang, B., Yin, J., Tang, W., Hao, J., Zhang, D.: Unknown malicious codes detection based on rough set theory and support vector machine. The 2006 IEEE International Joint Conference on Neural Network Proceedings, 2583–2587 (2006). IEEE

[24] Bhattacharya, A., Goswami, R.T., Mukherjee, K.: A feature selection technique based on rough set and improvised pso algorithm (psors-fs) for permission based detection of android malwares. International journal of machine learning and cybernetics **10**(7), 1893–1907 (2019)

[25] Penmatsa, R.K.V., Vatsavayi, V.K., Samayamantula, S.K.: Ant colony optimization-based firewall anomaly mitigation engine. SpringerPlus **5**(1), 1–32 (2016)

[26] Nauman, M., Azam, N., Yao, J.: A three-way decision making approach to malware analysis using probabilistic rough sets. Information Sciences **374**, 193–209 (2016)

[27] Golmaryami, M., Taheri, R., Pooranian, Z., Shojafar, M., Xiao, P.: Setti: As elf-supervised adv e rsarial malware de t ection archi t ecture in an i ot environment. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) **18**(2s), 1–21 (2022)

[28] Kim, J.-Y., Cho, S.-B.: Obfuscated malware detection using deep generative model based on global/local features. Computers & Security **112**, 102501 (2022)

[29] Hu, W., Tan, Y.: Generating adversarial malware examples for black-box attacks based on gan. ArXiv **abs/1702.05983** (2017)

[30] Kang, M., Kim, H., Lee, S., Han, S.: Resilience against adversarial examples: Data-augmentation exploiting generative adversarial networks. KSII Transactions on Internet & Information Systems **15**(11) (2021)

[31] AbuAlghanam, O., Alazzam, H., Qatawneh, M., Aladwan, O., Alsharaiah, M.A., Almaiah, M.A.: Android malware detection system based on ensemble learning (2023)

[32] Kim, J., Ban, Y., Ko, E., Cho, H., Yi, J.H.: Mapas: a practical deep learning-based android malware detection system. International Journal of Information Security **21**(4), 725–738 (2022)

[33] Alkahtani, H., Aldhyani, T.H.: Developing cybersecurity systems based on machine learning and deep learning algorithms for protecting food security systems: industrial control systems. Electronics **11**(11), 1717 (2022)

[34] Millar, S., McLaughlin, N., del Rincon, J.M., Miller, P.: Multi-view deep learning for zero-day android malware detection. Journal of Information Security and Applications **58**, 102718 (2021)

[35] Mimura, M., Ito, R.: Applying nlp techniques to malware detection in a practical environment. International Journal of Information Security **21**(2), 279–291 (2022)

[36] Liu, Z., Li, S., Zhang, Y., Yun, X., Cheng, Z.: Efficient malware originated traffic classification by using generative adversarial networks. In: 2020 IEEE Symposium on Computers and Communications (ISCC), pp. 1–7 (2020). https://doi.org/10.1109/ISCC50000.2020.9219561

[37] Sinha, A., Malo, P., Deb, K.: A review on bilevel optimization: from classical to evolutionary approaches and applications. IEEE Transactions on Evolutionary Computation **22**(2), 276–295 (2017)

[38] Willis, M.-J., Hiden, H.G., Marenbach, P., McKay, B., Montague, G.A.: Genetic programming: An introduction and survey of applications. Second international conference on genetic algorithms in engineering systems: innovations and applications, 314–319 (1997). IET

[39] Nanni, L., Lumini, A.: Generalized needleman–wunsch algorithm for the recognition of t-cell epitopes. Expert Systems with Applications **35**(3), 1463–1467 (2008)

[40] Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., Siemens, C.: Drebin: Effective and explainable detection of android malware in your pocket. Ndss **14**, 23–26 (2014)

[41] Wei, F., Li, Y., Roy, S., Ou, X., Zhou, W.: Deep ground truth analysis of current android malware. International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, 252–276 (2017). Springer

[42] Rashidi, B., Fung, C.: Xdroid: An android permission control using hidden markov chain and online learning. Communications and Network Security (CNS), 2016 IEEE Conference on, 46–54 (2016). IEEE

[43] Jeon, S., Moon, J.: Malware-detection method with a convolutional recurrent neural network using opcode sequences. Information Sciences **535**, 1–15 (2020)