Malware Evolution and Detection Based on the Variable Precision Rough Set Model

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

Malware Evolution and Detection Based on the Variable Precision Rough Set Model

Manel Jerbi
SMART Lab, CS department
University of Tunis, ISG
Tunis, Tunisia
manel.jerbi@gmail.com

Zaineb Chelly Dagdia
Université Paris-Saclay,
UVSQ, DAVID, France
LARODEC, ISG, Université de
Tunis, Tunis, Tunisia
zaineb.chelly-dagdia@uvsq.fr

Slim Bechikh
SMART Lab, CS department
University of Tunis, ISG
Tunis, Tunisia
slim.bechikh@fsegn.rnu.tn

Lamjed Ben Said
SMART Lab, CS department
University of Tunis, ISG
Tunis, Tunisia
lamjed.bensaid@isg.rnu.tn

***Abstract*—To offer innovative malware evolution techniques, it is appealing to integrate approaches that handle imperfect data and knowledge. In fact, malware writers tend to target some precise features within the app's code to camouflage the malicious content. Those features may sometimes present conflictual information about the true nature of the content of the app (malicious/benign). In this paper, we show how the Variable Precision Rough Set (VPRS) model can be combined with optimization techniques, in particular Bilevel-Optimization-Problems (BLOPs), in order to establish a detection model capable of following the crazy race of malware evolution initiated among malware-developers. We propose a new malware detection technique, based on such hybridization, named Variable Precision Rough set Malware Detection (ProRSDet), that offers robust detection rules capable of revealing the new nature of a given app. ProRSDet attains encouraging results when tested against various state-of-the-art malware detection systems using common evaluation metrics.**

I. Introduction

THE amount of malware is increasing exponentially thanks to the use of advanced malware-development tools [1]. Detection models struggle to keep up with these tricky intrusion malicious apps that do not refrain from using the most effective techniques, like the obfuscated malware, to invade the targeted systems. In this course of malware development, one can encounter some data inconsistency specially when dealing with conflictual features that may appear in both benign and malicious apps. In this context, few research has focused on dealing with the inconsistency encountered when extracting relevant features to either produce or detect malware. Authors in [2] proposed to adjust the malware detectors in order to deal with the change that occurs in the data labeling. More precisely, authors tackled the problem that appears when the labels used in the training set are different from the labels used in the testing set and proposed to empirically quantify the epistemic uncertainty of four combined deep-learning based Android malware detectors. Santos et al., in [3], proposed a semi-supervised learning based method to deal with the existing unlabelled apps (unknown nature beforehand) in the training process of a detection process. Also, Nauman et al. [4] looked into a three-way decision-making process based on acceptance, rejection, or deferment. When there is not enough knowledge, the extra deferment choice option gives the opportunity to postpone a decision. It also seeks to reduce incorrect decisions at the model level by finding a trade-off between decision-making attributes like accuracy, generality, and uncertainty. The authors focused on three-way decisions using two probabilistic rough set models: game-theoretic rough sets (GTRS) and information-theoretic rough sets (ITRS). RoughDroid [5] is a floppy analysis technique proposed by the authors that can detect Android malicious programs straight on the smartphone. It is based on seven feature sets extracted from the XML manifest file of an Android application and three feature sets extracted from the Dex file. Those feature sets pass through the Rough Set algorithm to classify the app either as benign or malicious. In this paper, we propose to specially focus on malware motif production and handling the "false" produced ones that may lead to data inconsistency. We genuinely propose to handle this challenging task and address it by evolving effective malicious motifs, a succession of frequent Application Programming Interface (API) call sequences, and exploit them afterwards in a bilevel-based method in order to produce detection rules capable of detecting them. In this work, we aim to attend the following contributions:

- Generate fraudulent motifs and then exploit them to produce robust detection rules by adopting a bilevel architecture where two Evolutionary Algorithms (EAs), an outer one (Genetic Programming algorithm (GP)) and an inner one (Genetic Algorithm (GA)), are in a mutual competition.
- Inspect the generated fraudulent motifs, which are generated by the inner algorithm within the second layer using the GA, using the Variable Precision Rough Set (VPRS) model before sending them to the outer algorithm within the first layer, i.e., the GP.
- Demonstrate the benefits of the selection made by VPRS reinforced by the bi-level competition between both algorithms since for every detection rule, there exists a whole search space of possible generated malicious motifs that should be effectively sampled to come up with fit and

This work is supported by ANR PIA funding: ANR-20-IDEES-0002.

challenging generated motifs that positively affect the detection quality of the corresponding first layer rule.
- Evaluate the outperformance of our ProRSDet approach compared to several state-of-the-art detection methods in terms of accuracy maximization and false alarms minimization.

The remainder of this paper is structured as follows: Section II emphasizes past work that is most similar to our approach. The fundamentals of BLOP and VPRS used in this work are presented in Section III. Our suggested detection method is described in Section IV. The experimental setup and performance analysis results are presented in Section V. Finally, the conclusion and a description of some future directions are presented in Section VI.

II. Related work

Different malware detection techniques [6], [7], [8] have been proposed in literature focusing, particularly, on generating new malware. These can be categorized into two heads. A first category is based on using machine learning based approaches and second category based on the use of evolutionary algorithms.

Among the works proposed in the first head, we mention the work of [9] where an Android malware detection system (DroidEvolver) was proposed that can automatically update itself during malware detection using online learning techniques with evolving feature set and pseudo labels. There were some methods which were based on generating adversarial samples. Among these, we mention, the work proposed in [10], where the feasibility of generating adversarial samples specifically through the injection of system API calls was investigated. In [11], a generative adversarial network based algorithm (MalGAN) was proposed to generate adversarial malware examples able to bypass black-box machine learning based detection models. The paper of Moti et al. [12] presented a deep generative adversarial network to generate the signature of unseen malware samples; the generated signature is potentially similar to the malware samples that may be released in the future.

Other works proposed automated signature generation systems, such as the work proposed in [13], where a system for automatic generation of intrusion signatures from honey net packet traces was developed. The work of [14], proposed an automated approach called "content sifting" that generates precise signatures that can then be used to filter or moderate the spread of a worm. In [15], a string signature generation system (Hancock) was designed to create a minimal set of N-byte sequences from a set of malware samples. Another work, the work of [16], used a 5-gram Markov chain model of good software to estimate the probability that a given byte sequence would show up in good software. In the paper of Li et al. [17], a network-based automated signature generation system (Hamsa) for polymorphic worms was proposed. The proposed model allowed to analyze the invariant content of polymorphic worms in order to make analytical attack-resilience granted for the signature generation algorithm. In [18], Newsome et al. proposed a signature generation system, Polygraph, that produces signatures that match polymorphic worms. Polygraph generates signatures that consist of multiple disjoint content sub-strings and which typically correspond to protocol framing, return addresses, and poorly obfuscated code.

Within the second head, several works [19], [20], [21], [22], [23], focused on applying evolutionary algorithms to generate malware samples. Among the most recent and efficient ones, we mention the work of [24], where an Android Malware Detection System (AMD) was proposed that produces patterns using a GA in order to mimic real malware patterns. This is to keep the dataset used in the conception of the detection system as varied as possible, which allows AMD to be resistant to obfuscated malware. Also, the work of [23], opted for a system using co-evolutionary algorithms where a first population generates detection rules, and a second population generates artificial malware. In this work, both populations are executed in parallel without any hierarchy. In the works of [25], [26], authors adopted a co-evolutionary algorithm as a search engine to ensure better detection rules.

Despite the good reached results of the above mentioned state-of-the-art methods, they still suffer from some limitations. First, they refer to a limited number of malware samples which makes the produced base of malicious malware not varied enough which cannot be of much help for a detection system when facing real attacks. Second, there is no check of the structure of the generated malicious patterns as to be sure enough that they fit among the real samples. And third, the malware generation and detection tasks are achieved separately without interaction which leads to a lack of a "harmony" and hence creates an incompatibility between the tasks.

In in paper, we will introduce our newly developed ProRSDet malware detection technique that overcomes the state-of-the-art shortcomings via the hybridization of both evolutionary algorithms and the Variable Precision Rough Set model.

III. BLOP and VPRS basic concepts

In this section, we introduce the main concepts and fundamentals of both BiLevel OPtimization and the Variable Precision Rough Set model as two tools, used in a hybrid fashion, to ensure the development of our proposed ProRSDet.

A. BiLevel OPtimization

BLOP is a distinctive optimization process where one problem is embedded within another. The inner problem, which is also referred to as the lower-level task, represents a constraint of the outer problem, which is also referred to as the upper-level task, where only an optimal lower-level solution can be a possible solution to the upper-level one. Each level has its own fitness function to optimize where the considered solutions of each level affect the decision-making space of the other one. The technical formalization of a BLOP problem can be found in [27] and it can be presented as follows:

A BLOP contains two classes of variables: (1) the upper-level variables $x \in X \subset R^n$, and (2) the lower-level variables

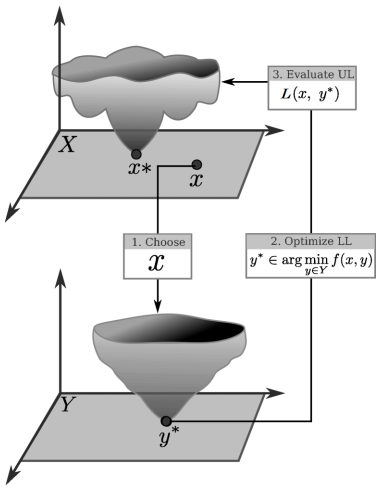

Fig. 1. Representation of a bilevel optimization problem (Inspired by [28])

$y \in Y \subset R^m$. For the follower problem, the optimization task is performed with respect to the variables y while the variables x act as fixed parameters. Thus, each x corresponds to a different follower problem, whose optimal solution is a function on Y and needs to be determined. All variables (x, y) are considered in the leader problem for given values of y $(y*)$. In what follows, we give the formal definition of BLOP.

Assuming $L : R^n \times R^m \rightarrow R$ to be the leader problem and $f : R^n \times R^m \rightarrow R$ to be the follower one, a BLOP could be defined as follows:

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

In the given formulation, L represents the first layer objective function, f represents the second layer objective function, x represents the first layer decision vector and y represents the second layer decision vector. G_k and g_j represent the inequality constraint functions at both layers, respectively. The representation of a bilevel optimization problem is illustrated in Figure 1.

B. Variable Precision Rough Set

VPRS [29], an extension of RST, is a mathematical tool that deals with inconsistent information and came mainly to overcome the maybe found strictness within the rough set notions which may be too restricted in the sense that they ignore the degree of an overlap between a set and a concept. Let us consider a universe of objects $\mathbf{U}$ referred to as elementary events and let $s(\mathbf{U})$ be the $\partial - algebra$ of measurable subsets of $\mathbf{U}$ referred to as random events. It is presumed that new objects e belonging to the universe are generated by a random process (on $\mathbf{U}$). For each new object e, the event $X \in s(\mathbf{U})$ occurred if the object $e \in X$. In addition, it is presumed the existence of the prior probability function P assigning probabilities $P(X)$ to sets X belonging to $s(\mathbf{U})$. $P(X) > 0$ means that all members of the family of sets $s(\mathbf{U})$ are likely to occur, and, $P(X)$ < 1means that their occurrence is not certain. These assumptions are justified by the fact that there is no need to construct a predictive model for events about which it is known that they are unlikely to occur or that they do occur with certainty [29]. In the context of defining the structure of rough approximation space, R denotes an equivalence relation on $\mathbf{U}$ with the finite number of equivalence classes (elementary sets) $E_1, E_2, \ldots, E_n$ such that $P(E_i)$ > 0 for all $1 \leq i \leq n$. The assumption of finite number of equivalence classes does not mean that the universe $\mathbf{U}$ is finite. Each elementary set E can be assigned a measure of overlap with the set X by the conditional probability function defined as $P(X \mid E)$ = $P(X \bigcap E)$ / $P(E)$. The values of the conditional probability function are normally estimated from sample data by taking the ratio $P(X \mid E) = card(X \bigcap E)$ / $card(E)$. The VPRS generalization of the original rough set model is based on the values of the probability function P and two lower and upper limit certainty threshold parameters l and u such that $0 \leq l < P(X) < u \leq 1$. The requirement l < $P(X)$ is an extra constraint on the values of the parameters which was proposed in [29]. The VPRS model is said to be symmetric if $l = 1 - u$. In this study, the symbol β such that $0 < P(X) < \beta \leq 1$ is used instead of the symbol u to denote the model upper threshold parameter. Also, the symbol α will substitute the previously defined l parameter.

IV. ProRSDet: The Variable Precision Rough Set Malware Detection Technique

Figure 2 depicts ProRSDet's overall running process, which is divided into two principal layers (levels): (1) *First layer* is built on a GP with the goal of generating a set of effective detection rules ($FDRB$) and (2) *Second layer* relies on a GA to generate harmful (malicious) motifs (SHM) (first step) and on a VPRS based component that exclusively preserves the most dependable set of harmful motifs with no structural flaws, referred to as "Relevant" motifs (FMM) (second step).

Each of these two layers runs through a series of iterations in order to find the optimal solutions in both levels, which are interdependent. As presented in Figure 2, the evaluation of every upper detection rule solution (among DRB) requires running a search algorithm to find the best undetectable harmful motifs (FMM) by this rule. The final set of detection rules produced by our ProRSDet ($FDRB$) is a set of detection rules that will perform the malware detection task.

1) First layer: The first layer's first step, as shown in Figure 2 and Algorithm 1, is to generate a set of detection rules (Algorithm 1, line 1), which will go through an evaluation procedure (Algorithm 1, lines 2-3). The coverage of the base of samples (input) as well as the coverage of the fraudulent motifs created by the second layer are used to make this evaluation. These two measures are used to be maximized by the population of detection rules solutions (Algorithm 1, lines 4-6). This module produces a collection of final detection rules ($FDRB$) that will be used by the detection job, which is in charge of classifying new apps as malicious or benign. The GP evolutionary operators require a specific formalization to cope

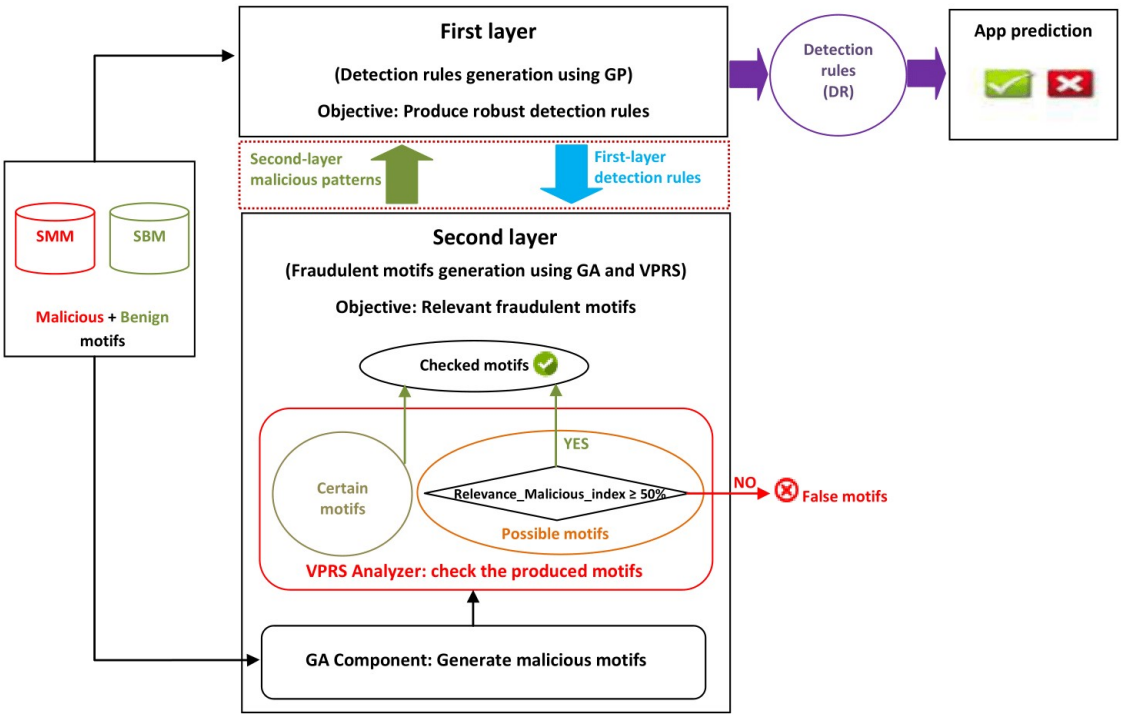

Fig. 2. Illustration of the ProRSDet functioning process.

with the generated solutions (i.e., the detection rules) by the first layer that relies on a GP process. These are the following:

- *Solution representation:* The solution is expressed as a series of terminals that relate to various motifs (API call sequences) and functions (Intersection (AND) and Union (OR)), respectively.
- *Solution variation:* By selecting one of the functions or terminals at random, the GP *mutation* operator is applied. If a terminal is selected then it is replaced by another terminal; if it is a function then it is replaced by a new function. As for the GP *crossover operator*, two parent individuals are selected, and a sub-node is picked on each selected parent. The *crossover* swaps the nodes and their related sub-nodes from one parent to the other.
- *Solution evaluation:* An individual's encoding is quantified using a mathematical metric called the "fitness function", which measures the quality of a proposed detection rule and fraudulent motifs. For the GP adaptation, we used the fitness function f_{outer} defined in Equation 2 to evaluate detection-rules solutions (DR).

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

where $\#damp$ refers to the number of detected fraudulent motifs and $\#amp$ refers to the number of fraudulent motifs and

$$Precision(DR) = \frac{\sum_{i=1}^{p} DR_i}{t}, Recall(DR) = \frac{\sum_{i=1}^{p} DR_i}{p} \quad (3)$$

Algorithm 1 Outer Algorithm (First layer)

Input: SMM**: set of malicious motifs,** SBM**: set of benign motifs,** FMM**: set of "Relevant" fraudulent motifs,** NDR**: number of detection rules,** NFM**: number of "Relevant" fraudulent motifs in** SHM**,** NF**: number of iterations in the first layer,** NS**: number of iterations in the second layer**

Output: Final set of detection rules $FDRB$

$DRB_0 \leftarrow$ Initialization(NDR,SHM,SBM) /*First generation of detection rules*/
for each DR_0 in DRB_0 **do** /*DR means detection rule*/
  $SFM_0 \leftarrow$ FMGeneration(DR_0,FMM,NFM,NS) /*call second layer*/
  $DR_0 \leftarrow$ Evaluation(DR_0,SHM,SFM_0)
end for
$k \leftarrow 1$
while $k < NF$ **do**
  $Q_t \leftarrow$ Variation(DRB_{t-1})
  for each DR_t in Q_t **do** /*Evaluate each rule based on upper fitness function*/
    $DR_t \leftarrow$ OuterEvaluation(DR_t,SHM)
    $SFM_t \leftarrow$ FMGeneration(DR_t,SHM,NFM,NS)
    $DR_t \leftarrow$ EvaluationUpdate(DR_t,SFM_t)
  end for
  $U_t \leftarrow Q_t \cup DRB_t$
  $DRB_{t+1} \leftarrow$ Selection(NDR,U_t)
  $k \leftarrow k$+1
end while
$FDRB \leftarrow$ FittestSelection(DRB_t)

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

$$DR_i = \begin{cases} 1 \text{ if the } i^{th} \text{ detected malicious motif exists in } SMM \\ 0 \text{ otherwise} \end{cases} \quad (4)$$

2) Second layer: The generation process of "Relevant" motifs (FMM, Algorithm 2, line 7) is performed as follows:

- *Step 1:* A GA is used to maximize the distance between the generated malicious motifs (SHM) and the reference benign motifs (input, not-generated motifs (SBM) while minimizing the distance between the generated malicious motifs (SHM) and the reference malicious ones (SMM). The GA also increases the amount of malicious motifs generated that are not detected by the first layer, i.e., the detection rules (DRB) (Algorithm 2, lines 1-5). The GA evolutionary operators need a special formalization to deal with the manipulated solutions in order to generate the motifs. The following are the adopted formalizations:
  - *Solution representation:* The GA solutions are represented as chromosomes made up of API call sequences. These are identifiable by their identifiers (IDs) and defined by their class (labels), which indicate their nature (malicious or benign), calling depths, and a collection of binary values indicating whether or not an API call appears in the entire API call sequence.
  - *Solution variation:* For the GA *crossover operator*, two parent individuals (chromosomes) are chosen, and a gene from each parent is chosen. Crossover involves the transfer of genes from one parent to the other. Only parents with the same nature can be used with the crossover operator (malicious or benign). The *mutation* operation starts by randomly selecting a gene on the chromosome. The selected gene is then replaced with another gene from the same class if it belongs to that class.
  - *Solution evaluation:* A fraudulent motif (FM) is evaluated based on the following GA fitness function:

$$f_{inner}(FM) = Max((\#gamp - \#dagmp) + \sum_{i=1}^{n} f_{Qual}(FM_i)) \quad (5)$$

where $i \in [1, n]$; n indicates the total number of fraudulent motifs, and $\#gamp$ refers to the number of fraudulent motifs and $\#dagmp$ refers to the number of detected fraudulent motifs. The function $f_{Qual}()$ defined in Equation 6 ensures the diversification of the fraudulent motifs.

$$f_{Qual}(FM_i) = \frac{Sim_1 + Sim_2 + Overlap(FM_i)}{3} \quad (6)$$

$$Sim_1 = Sim(MS, FM_i) = \frac{\sum MS_{j \in MS} Sim(FM_i, MS_j)}{|MS|} \quad (7)$$

where $j \in [1, m]$;m indicates the total number of malicious motifs. The similarity between the generated motif FM_i and the malicious set of motifs (MS).This measure of similarity needs to be maximized.

$$Sim_2 = Sim(BS, FM_i) = \frac{\sum BS_{k \in BS} Sim(FM_i, BS_k)}{|BS|} \quad (8)$$

The similarity between the generated motif FM_i where $k \in [1, p]$; p indicates and the benign motifs (BS) the total number of benign set of motifs and which has to be the lowest.

$$Overlap(FM_i) = 1 - \frac{\sum_{FM_l, i \neq l} Sim(FM_i, FM_l)}{|FM|} \quad (9)$$

$Overlap()$ is measured as the average value of the individual Sim(FM_i, FM_l) between the generated motif FM_i and all the other generated motifs FM_l in the generated dataset SFM. l refers to the total number of the generated motifs.

We updated the Needleman-Wunsch alignment algorithm formula [30] to our context to determine the similarity $Sim()$ between two motifs. This measure of similarity was employed in the above equations but with different parameters. A detailed description of the similarity function $Sim()$ can be found in [24].

- *Step 2:* The GA evolutionary operators mentioned above may cause the manipulated solutions to be distorted, and hence ambiguous, in different ways and with different degrees. Technically, a set of motifs is declared to be ambiguous when they share the same values of the features (API calls) but do have different label values (malicious/benign). An illustration of this ambiguity is presented in Table I. The manipulated motifs by the lower-level are API call sequences. Each API call sequence is named MFX_i (as shown in Table I) and is composed of different API calls named MLX_j. A conflict (or inconsistency) may exist between objects (fraudulent motifs). 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.

To handle this ambiguity issue and to guarantee the reliability of the generated malicious motifs, a VPRS component, namely Variable Precision Rough Set Analyzer (VPRS Analyzer in Figure 3) which uses mainly

TABLE I
EXAMPLES OF AMBIGUOUS MOTIFS.

Malicious fraudulent motifs	Condition attributes (API call)				Decision (Nature)
	MLX_1	MLX_2	...	MLX_n	
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

Algorithm 2 Inner Algorithm (Second layer)

Input: SMM**: set of malicious motifs,** SBM**: set of benign motifs,** DRB**: set of detection rules,** R**: number of generations,** N**: population size**

Output: Set of Relevant generated motifs FMM

$SFM_0 \leftarrow$ Initialization(SBM,SMM,N,R) /*SFM means set of fraudulent motifs*/
2: $SFM_0 \leftarrow$ Evaluation(SFM_0,SBM,SMM,DRB)/*Evaluation depends on DRB*/
$k \leftarrow 1$
4: **while** $k < R$ **do**
$Q_t \leftarrow$ Variation(SFM_{t-1})
6: $Q_t \leftarrow$ Evaluation(Q_t,SBM,SMM,DRB)
$U_t \leftarrow Q_t \cup SFM_t$
8: $SFM_{t+1} \leftarrow$ Selection(N,U_t)
$k \leftarrow k$+1
10: $SHM \leftarrow$ FittestSelection(SFM_t)
$(RFM, AFM) \leftarrow$ RelevanceCheck(SHM)/*Set of relevant FM and a set of ambiguous FM*/
12: $SCFM \leftarrow$ LowerCertainty(AFM) $\cup$ RFM
$SPFM \leftarrow$ UpperCertainty(AFM)
14: $(FCFM, FPFM) \leftarrow$ Pruning($SCFM$,$SPFM$)
$FMM \leftarrow FCFM \cup FPFM$
16: **end while**

the VPRS lower and upper limit certainty thresholds concepts, is plugged to the inner algorithm. Specifically, the VPRS Analyzer checks first the reliability of the generated malicious motifs (SHM). Among this set, the VPRS Analyzer keeps the most relevant motifs (RFM) which do not need any further check, and investigates the remaining ambiguous set (AFM). Among the AFM set, the VPRS Analyzer calculates the lower limit certainty threshold to only keep the certain set of fraudulent motifs $SCFM$ (Algorithm 2, lines 11-12), and the upper limit certainty threshold to keep the approved fraudulent motifs among the *possible* set of generated fraudulent motifs $SPFM$, together with $SCFM$. During the pruning operation (Algorithm 2, line 14), redundant motifs are removed. Finally, the joint sets of FFM and $FPFM$ form the relevant, and the most relevant artificially generated fraudulent motifs (FMM) (Algorithm 2, line 15).

As *possible* fraudulent motifs cannot be considered relevant enough to be added to the initial set of malicious motifs, they need further evaluation that reflects their quality and measures

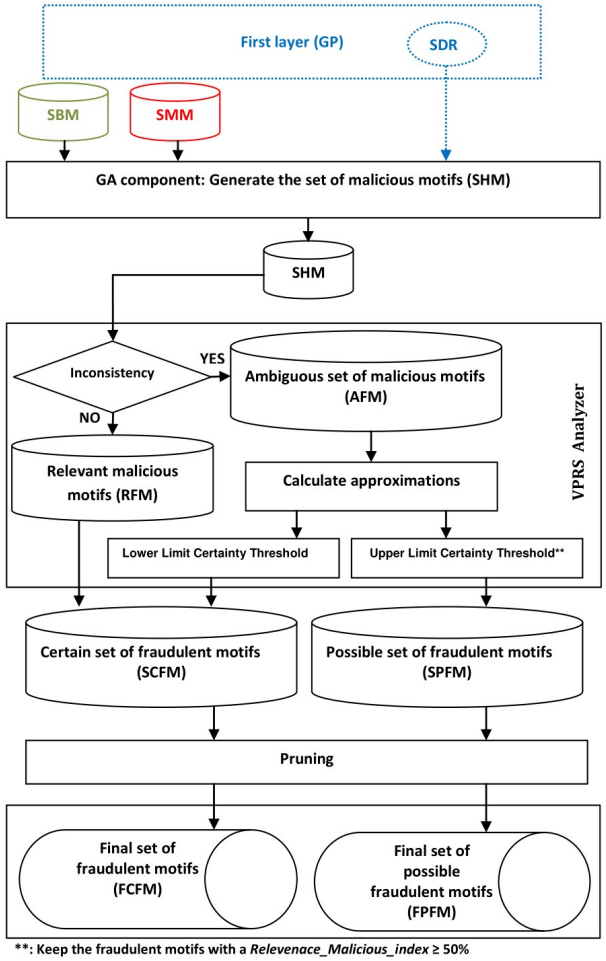

Fig. 3. Second layer functioning process.

their reliability. For every *possible* fraudulent motif, the VPRS-based component estimates its reliability using an index named $Relevance\_Malicious\_index$, which is defined as the ratio of the number of instances that belong to the *possible* set and having the same structure with a malicious label, and the number of the whole *possible* set of instances (Equation 10).

$$Relevance\_Malicious\_index = \frac{Instance\_Possible\_Malicious\_Motif}{Instance\_Possible\_Original\_Data} \quad (10)$$

where $Instance\_Possible\_Malicious\_Motif$ refers to the number of instances that share the same structure and are labelled malicious and $Instance\_Possible\_Original\_Data$ refers to the total number of instances within the whole possible set. This index can, therefore, be viewed as the probability of counting the ambiguous fraudulent motifs set (the *possible* generated motif set) correctly. It shows the extent to which a correct label can be given to a generated motif belonging to the *possible* set of generated motifs. Tacking into account that we are aiming to produce effective malicious fraudulent motifs, we will only keep the generated motifs that have a $Relevance\_Malicious\_index > 50\%$. An illustrative example of this index is given below: Sup-

pose that when generating 123 new fraudulent motifs, 100 among them were labelled as malicious and 23 were labelled as benign. The $Relevance\_Malicious\_index$ of those 123 *possible* generated motifs is (100/123). This means that the $Relevance\_Malicious\_index$ = 81,30% which is clearly greater than 50% and hence the common shared structure of these generated motifs will be added to the set of malicious motifs sent to the outer algorithm.

3) Detection task based on detection rules: Throughout this phase, our model will perform its classification task where a new app, the executable, will be classified either as a malware or as a benign. This is achieved using the set of detection rules ($FDRB$). Formally, the first step aims to extract the motifs of the executable. Each motif will be labeled as benign or as malicious by comparing it to the motifs of the SMM and SBM databases. Then, the obtained motifs are compared to the antecedent of $FDRB$. The comparison will allow the executable to be either classified as a malware or as a benign app.

V. Experimental Setup and Results

A. Experimental Setup

An experimental investigation was done to evaluate ProRSDet's performance in detecting new malware variants. For this purpose, we gathered data from a variety of sources (the theZoo dataset[1], from VirusTotal[2]) and from various portable benign tools such as Google play. We have gathered 5 540 Android apps where 3 440 are malicious and 2 100 apps are benign files. From those apps, a total of malicious motifs and a total of benign motifs were extracted. The conducted process is summarized in Table II. The Drebin dataset [31], which contains 123 453 benign applications and 5 560 malware samples, is used for the evaluation of our approach against the new variants of malware and 0-day attacks. The necessity for confirmation that ProRSDet is not fitting the base of examples led to the selection of a dataset that is different from the one used for the construction phase. For comparisons, various state-of-the-art methods were investigated. These are the classical classifiers named in Table V, tested using Weka with the proposed default parameters settings, three known methods (Rathore et al. [32], Gym-plus [33], and AMD [24]), and several commercial antimalware named in Table VII. The comparison made with the two recent state-of-the-art methods ([32] and [33]) is justified by the fact that those approaches are somehow similar to ProRSDet. In fact, there are common traits between our developed approach and those approaches: they propose a two-task solution (a malware generation task and a malware detection task). Also, to ensure the fairness of comparisons between evolutionary approaches (AMD [24] and ProRSDet), we used the parameter settings described in Table III.

Both evolutionary approaches perform 798 000 function evaluations in each run. Also, to help determine the most appropriate α and β values, a set of experiments is conducted and the results are reported in Table IV. Indeed, Table IV shows that the best results were reached with a pair of α and β value that equals 0.5, respectively. When running the experiments, we concluded that the fitness functions become stabilized around the 36^{th} generation. For these reasons, the algorithms did not suffer from premature convergence. The metrics used for the evaluation are: true positives (TP), false positives (FP), true negatives (TN), false negatives (FN), recall (RC), specificity (SP), accuracy (AC), precision (PR), F1_score (FS), and the Area Under the Receiver Operating Characteristics (ROC) Curve (AUC). All of the conducted experiments, based on a 10-fold cross validation, are run on an $Intel^{®}$ $Xeon^{®}$ Processor CPU E5-2620 v3, with a 16 GB RAM.

TABLE II
Number of obtained motifs.

	Number of apps	Number of motifs
Benign	2 100	28 019 663
Malicious	3 440	36 995 382

TABLE III
Evolutionary parameters.

	ProRSDet	AMD
Population size	(both levels) 30	180
Generation size	(both levels) 30	4500
Mutation rate	0.5	0.5
Crossover rate	0.9	0.9

B. Results Analysis

In this section, we compare the ProRSDet obtained results to a set of classifiers (Table V), three state-of-the-art approaches (Rathore et al. [32], Gym-plus [33] and AMD [24]) and five antivirus engines (Table VI and Table VII). More precisely, to determine how accurate our predictive model will perform in practice, we used 10-fold cross-validation. We considered all of the collected programs and hence all of the obtained motifs stored in SBM and SMM (see Table II). So, concerning the comparison with the top-five classifiers (Table V), and based on all of the evaluation metrics, ProRSDet surpasses all other classifiers. In comparison to the LDA and $J48$ classifiers, which produced the second best results among the rest of the classifiers with a pair of precision and accuracy of (98.36%, 97.82%) for LDA and (97.73%, 96.58%) for $J48$ and a pair of F1_score and specificity of (97.32%, 97.31%) for LDA and (98.37%, 97.13%) for $J48$, ProRSDet achieved a precision of 98.20%, an accuracy of 98.22%, an F1 score of 98.21%, and a specificity of 98.20%. These remarkable ProRSDet results are based on its high true positives (98.20%) and low false positives (1.80%), which are the best achieved values among the results of the classifiers. These encouraging findings show that ProRSDet is capable of distinguishing between the two possible designations (malicious and benign). Also, regarding the comparison between the EA-based approaches (ProRSDet and AMD [24]), we used an unknown dataset (Drebin dataset

[1]https://thezoo.morirt.com/
[2]https://www.virustotal.com/gui/home/upload
[2]https://www.cs.waikato.ac.nz/ml/weka/

TABLE IV
VPRS PARAMETERS.

		RC	SP	AC	PR	FS	AUC	FPR	FNR
Experiment 1	α = 1 β = 0	97.56	97.02	97.29	97.01	97.28	80.01	02.99	02.42
Experiment 2	α = 0.85 β = 0.15	96.68	97.66	97.17	97.69	97.18	82.13	02.31	03.28
Experiment 3	α = 0.7 β = 0.3	96.66	97.07	96.87	97.09	96.87	73.80	02.91	03.35
Experiment 4	α = 0.5 β = 0.5	97.99	97.32	97.66	97.31	96.65	87.00	02.69	01.99

[31]) and ProRSDet outperformed AMD in terms of the used evaluation metrics as stated in Table VI. This can be explained by the contribution brought by the VPRS Analyzer which helped keep the most "relevant" malicious motifs.

Moreover, we may derive from Table VI and Table VII that, when compared to competing state-of-the-art approaches, (Rathore et al. [32], Gym-plus [33] and AMD [24]) using the unknown dataset [31], ProRSDet came in top with an accuracy of 97.66%, a specificity of 97.32%, a recall of 97.99%, a precision of 97.31%, and an AUC of 86.15%. Rathore et al., Gym-plus and AMD, obtained an accuracy of 93.81%, 93.50% and 92.28%, respectively, which are lower than those obtained by our proposed technique. In addition, the interesting detection results obtained by ProRSDet are endorsed by the results presented in Table VII which refers to the comparison with the commercial antivirus engines. Table VII shows that ProRSDet reached an accuracy rate of 97.66% whereas the ESET NOD32 engine, which is ranked first among all the other malware antivirus engines, registered only 66.68% of accuracy. It is to be noted that the accuracy values of the four remaining antivirus engines varied approximately between 56% and 66%.

The results reported from Tables V, VI and VII highlight the ability of ProRSDet – thanks to its set of efficient produced rules which are generated using the most relevant set of the generated fraudulent malware; both guaranteed via the use of the BLOP architecture and the VPRS component – to achieve accurate detection operations against new and unknown variants of malware.

To better clarify the efficiency and benefits of relying on the bilevel architecture within ProRSDet, we analyse the results in terms of false positive and the false negative rates. The registered ProRSDet values of those two metrics (Table VI) confirm the usefulness of a bilevel architecture to detect a malicious code efficiently. The continuous competition between both levels (first layer and second layer) permitted generation of good solutions (detection rules and fraudulent motifs) and this had positive impact on the values of FPR (02.69%) and FNR (01.99%). In comparison to ProRSDet, the registered FPR and FNR values for AMD [24], which rely on a single-layer based architecture via the use of evolutionary algorithms, are 06.37% and 08.84%, respectively. In addition, referring to Table VIII, we can state that the Variable Precision Rough Set based module succeeded to set apart 198 522 ambiguous instances (*possible* set) among the generated fraudulent motifs SHM (468 000 instances). More precisely, a set of 92 689 of false motifs were removed from the whole set of ambiguous motifs. The removal of those false motifs was performed thanks to the $Relevance\_Malicious\_Index$ and after being processed by the lower and upper limit certainty thresholds explained and illustrated in Section IV-2. This distinction brings to light the VPRS component's important contribution in improving the quality of the fraudulent motifs by the second layer and which, consequently, positively affected the false alarms rate. Let us recall that ProRSDet provides a set of robust detection rules thanks to the set of malicious motifs produced by the second layer's GA reinforced by the VPRS module. The VPRS module helps determine the set of "certain" malicious motifs and the set of "possible" malicious motifs. When dealing with this set of *possible* motifs, a $Relevance\_Malicious\_index$ (Equation 10) that estimates the reliability degree of each *possible* malicious motif is also provided to the user (as presented in Section IV-2). This metric, specific to the evaluation of each *possible* malicious motif, will help determine the fate of this specific motif: add it to the set of malicious motifs or remove it. To be more specific, Figure 4 represents the number of the obtained *possible malicious motifs* with regards to their $Relevance\_Malicious\_index$. Figure 4 shows that 51.28% of possible malicious motifs have a $Relevance\_Malicious\_index$ that exceeds 50%. Also, 17.98% of possible rules succeeded to correctly classify apps with rates that lie between 41% and 50%. 18.23% of those rules ranked just below with a $Relevance\_Malicious\_index$ comprised between 31% and 40%. Approximately only 13% of the rules failed to have a $Relevance\_Malicious\_index$ above 30%. An example of the use of this index was previously given in Section IV-2. Also, an important aspect in our proposed ProRSDet approach that needs to be clarified concerns the setting of the α and β VPRS parameters. In fact, in this study we adopted the the trial and error method which consists of choosing randomly values and apply them in our algorithm. For instance, we conducted four different experiments (Table IV) which helped us determine the best pair of α and β values that leads to better detection results. More precisely, we tried different combination of α and β values and each time we assessed the recall (RC), specificity (SP), accuracy (AC), precision (PR), F1_score (FS), Area under ROC curve (AUC), false positive rate (FPR) and false negative rate (FNR). The best values were reached with α = 0.5 and β =

TABLE V
COMPARISON BETWEEN PRORSDET AND THE CLASSICAL CLASSIFIERS.

Classifier/ approach	TP	FP	TN	FN	RC	SP	AC	PR	FS	AUC	FPR	FNR
ProRSDet	98.20	01.80	98.24	01.76	98.23	98.20	98.22	98.20	98.21	86.79	01.80	01.76
LR	93.81	06.19	96.75	03.25	96.65	93.98	95.28	93.17	95.60	63.69	06.01	03.34
NB	92.30	07.70	28.41	71.59	56.31	78.67	60.35	92,37	93,62	65.06	02.13	09.03
RF	97.41	02.59	95.90	04.10	96.00	98.37	97.16	97.36	97.17	73.04	02.62	04.03
J48	97.18	02.82	93.98	06.02	94.27	97.13	96.58	97.73	98.37	83.90	02.91	05.83
k-NN	89.52	10.48	95.21	04.79	94.92	90.08	92.37	85.74	90.56	57.69	09.91	05.07
LDA	97.29	02.71	98.36	01.64	98.34	97.31	97.82	98.36	97.32	75.96	02.68	01.65

LR: Logistic Regression; LDA: Linear Discriminant Analysis; RF: Random Forest; J48: Decision Tree; NB: Naive Bayes; k-NN: k-Nearest Neighbours.

TABLE VI
COMPARISON BETWEEN PRORSDET AND AMD [24] USING THE DREBIN DATASET [31].

Classifier/ approach	TP	FP	TN	FN	RC	SP	AC	PR	FS	AUC	FPR	FNR
ProRSDet	97.31	02.69	98.01	01.99	97.99	97.32	97.66	97.31	96.65	86.15	02.69	01.99
AMD	93.80	06.19	90.90	09.10	96.20	92.70	92.28	93.60	92.37	57.69	06.37	08.84

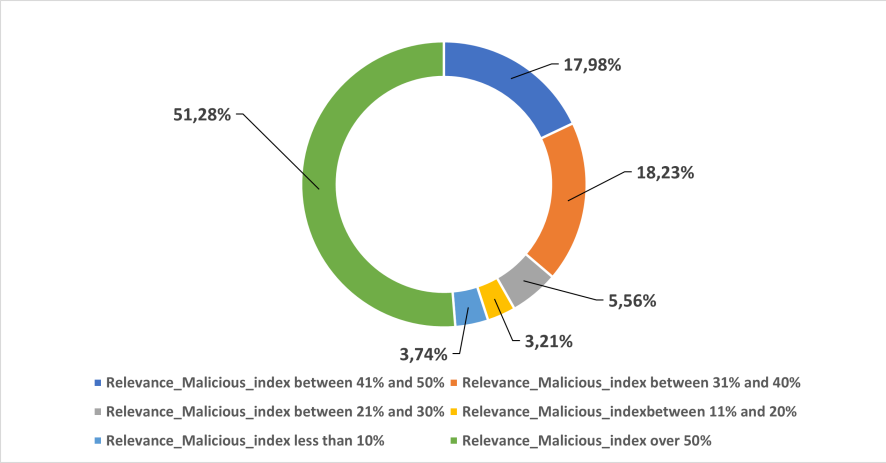

Fig. 4. Number of *possible* motifs with regards to the $Relevance\_Malicious\_index$.

TABLE VII
ACCURACY RESULTS OF PRORSDET AND TOP FIVE COMMERCIAL ENGINES BY VIRUSTOTAL [3] ON THE DREBIN DATASET [31].

Anti-malware	Reference	Accuracy (%)
ProRSDet	Our current approach	97.66
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
Gym-plus	[33]	93.50
Rathore et al.	[32]	93.81 (with RF)

0.5. Indeed, we specifically registered an AC of 97.66%, a PR of 97.31% a FPR of 02.69% and a FNR of 01.99%. Those significant values found their way thanks to great reached percentages of true positives and true negatives.

TABLE VIII
NUMBERS OF RELEVANT AND AMBIGUOUS GENERATED MALICIOUS MOTIFS.

Number of generated motifs in ProRSDet		
Possible instances		Certain instances
False motifs	Approved motifs	
92 689	105 833	269 478

VI. CONCLUSION AND FUTURE DIRECTIONS

In this research, we developed ProRSDet, a malware detection technique that combines the Variable Precision Rough Set model and bilevel optimization. Within the bilevel architecture, the malware generation task (inner algorithm or second layer) and the rules generation task (detection task, outer algorithm or first layer) are in mutual competition. The second layer generates "Relevant" malicious motifs which are generated by a GA and thoroughly checked by a VPRS component that only keeps the most "pertinent" ones, and which are capable of eluding the GP's set of detection rules in the first layer.

These effective created detection rules, in turn, attempt their hardest to detect the second layer's set of fraudulent patterns.

ProRSDet outperformed a variety of state-of-the-art approaches and commercial engines, achieving encouraging detection rates of 97.66% accuracy and 2.69% false positives. We plan to investigate other methods to help determine the best values of α and β concerning the VPRS. It would be interesting to design an adaptive parameter tuning strategy that aims to approximate the best values of the VPRS parameters. Also, we can consider other theories that deal with the inconsistency in the future (i.e., [34], [35]), as well as consider expanding the scope of the proposed work to encompass other operating systems.

REFERENCES

[1] G. Ollmann, "The evolution of commercial malware development kits and colour-by-numbers custom malware," *Computer Fraud & Security*, vol. 2008, no. 9, pp. 4–7, 2008. doi: 10.1016/S1361-3723(08)70135-0

[2] D. Li, T. Qiu, S. Chen, Q. Li, and S. Xu, "Can we leverage predictive uncertainty to detect dataset shift and adversarial examples in android malware detection?" in *Annual Computer Security Applications Conference*, 2021. doi: https://doi.org/10.1145/3485832.3485916 pp. 596–608.

[3] I. Santos, J. Nieves, and P. G. Bringas, "Semi-supervised learning for unknown malware detection," in *International Symposium on Distributed Computing and Artificial Intelligence*. Springer, 2011. doi: 10.1007/978-3-642-19934-9_53 pp. 415–422.

[4] M. Nauman, N. Azam, and J. Yao, "A three-way decision making approach to malware analysis using probabilistic rough sets," *Information Sciences*, vol. 374, pp. 193–209, 2016. doi: https://doi.org/10.1016/j.ins.2016.09.037

[5] K. Riad and L. Ke, "Roughdroid: operative scheme for functional android malware detection," *Security and Communication Networks*, vol. 2018, 2018. doi: https://doi.org/10.1155/2018/8087303

[6] S. Piparia, D. Adamo, R. Bryce, H. Do, and B. Bryant, "Combinatorial testing of context aware android applications," in *2021 16th Conference on Computer Science and Intelligence Systems (FedCSIS)*, 2021. doi: 10.15439/2021F003 pp. 17–26.

[7] F. Alotaibi and A. Lisitsa, "Matrix profile for ddos attacks detection," in *2021 16th Conference on Computer Science and Intelligence Systems (FedCSIS)*, 2021. doi: 10.15439/2021F114 pp. 357–361.

[8] P. Kishore, S. K. Barisal, and D. P. Mohapatra, "Decentralized controller for software interconnected system subject to malicious attacks." in *FedCSIS (Position Papers)*, 2021. doi: 10.15439/2021F90 pp. 211–218.

[9] K. Xu, Y. Li, R. Deng, K. Chen, and J. Xu, "Droidevolver: Self-evolving android malware detection system," in *2019 IEEE European Symposium on Security and Privacy (EuroS P)*, 2019. doi: 10.1109/EuroSP.2019.00014 pp. 47–62.

[10] F. Cara, M. Scalas, G. Giacinto, and D. Maiorca, "On the feasibility of adversarial sample creation using the android system api," *Information*, vol. 11, no. 9, p. 433, 2020. doi: https://doi.org/10.3390/info11090433

[11] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on gan," *arXiv preprint arXiv:1702.05983*, 2017. doi: https://doi.org/10.48550/arXiv.1702.05983

[12] Z. Moti, S. Hashemi, and A. Namavar, "Discovering future malware variants by generating new malware samples using generative adversarial network," in *2019 9th International Conference on Computer and Knowledge Engineering (ICCKE)*, 2019. doi: 10.1109/ICCKE48569.2019.8964913 pp. 319–324.

[13] V. Yegneswaran, J. T. Giffin, P. Barford, and S. Jha, "An architecture for generating semantic aware signatures." in *USENIX security symposium*, 2005, pp. 97–112.

[14] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated worm fingerprinting." in *OSDI*, vol. 4, 2004, pp. 4–4.

[15] K. Griffin, S. Schneider, X. Hu, and T.-c. Chiueh, "Automatic generation of string signatures for malware detection," in *International workshop on recent advances in intrusion detection*. Springer, 2009. doi: https://doi.org/10.1007/978-3-642-04342-0_6 pp. 101–120.

[16] J. O. Kephart, "Automatic extraction of computer virus signatures," in *Proc. 4th Virus Bulletin International Conference, Abingdon, England, 1994*, 1994, pp. 178–184.

[17] Z. Li, M. Sanghi, Y. Chen, M.-Y. Kao, and B. Chavez, "Hamsa: fast signature generation for zero-day polymorphic worms with provable attack resilience," in *2006 IEEE Symposium on Security and Privacy (S P'06)*, 2006. doi: 10.1109/SP.2006.18 pp. 15 pp.–47.

[18] J. Newsome, B. Karp, and D. Song, "Polygraph: automatically generating signatures for polymorphic worms," in *2005 IEEE Symposium on Security and Privacy (S P'05)*, 2005. doi: 10.1109/SP.2005.15 pp. 226–241.

[19] E. Aydogan and S. Sen, "Automatic generation of mobile malwares using genetic programming," in *European conference on the applications of evolutionary computation*. Springer, 2015. doi: 10.1007/978-3-319-16549-3_60 pp. 745–756.

[20] M. F. Zolkipli and A. Jantan, "A framework for malware detection using combination technique and signature generation," in *2010 Second International Conference on Computer Research and Development*. IEEE, 2010. doi: 10.1109/ICCRD.2010.25 pp. 196–199.

[21] H. G. Kayacık, A. N. Zincir-Heywood, and M. I. Heywood, "Can a good offense be a good defense? vulnerability testing of anomaly detectors through an artificial arms race," *Applied Soft Computing*, vol. 11, no. 7, pp. 4366–4383, 2011. doi: https://doi.org/10.1016/j.asoc.2010.09.005

[22] Y. Xue, G. Meng, Y. Liu, T. H. Tan, H. Chen, J. Sun, and J. Zhang, "Auditing anti-malware tools by evolving android malware and dynamic loading technique," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 7, pp. 1529–1544, 2017. doi: 10.1109/TIFS.2017.2661723

[23] S. Sen, E. Aydogan, and A. I. Aysan, "Coevolution of mobile malware and anti-malware," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 10, pp. 2563–2574, 2018. doi: 10.1109/TIFS.2018.2824250

[24] M. Jerbi, Z. C. Dagdia, S. Bechikh, M. Makhlouf, and L. B. Said, "On the use of artificial malicious patterns for android malware detection," *Computers & Security*, p. 101743, 2020. doi: https://doi.org/10.1016/j.cose.2020.101743

[25] M. Jerbi, Z. C. Dagdia, S. Bechikh, and L. B. Said, "Android malware detection as a bi-level problem," *Computers & Security*, p. 102825, 2022. doi: https://doi.org/10.1016/j.cose.2022.102825. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016740482200219X

[26] M. Jerbi, Z. Chelly Dagdia, S. Bechikh, and L. Ben Said, "Malware detection using rough set based evolutionary optimization," in *Neural Information Processing*, T. Mantoro, M. Lee, M. A. Ayu, K. W. Wong, and A. N. Hidayanto, Eds. Cham: Springer International Publishing, 2021. ISBN 978-3-030-92307-5 pp. 634–641.

[27] B. Colson, P. Marcotte, and G. Savard, "An overview of bilevel optimization," *Annals of operations research*, vol. 153, no. 1, pp. 235–256, 2007. doi: https://doi.org/10.1007/s10479-007-0176-2

[28] J.-A. Mejía-de Dios, E. Mezura-Montes, and M. Quiroz, "Automated parameter tuning as a bilevel optimization problem solved by a surrogate-assisted population-based approach," *Applied Intelligence*, vol. 51, pp. 1–23, 08 2021. doi: 10.1007/s10489-020-02151-y

[29] W. Ziarko, "Set approximation quality measures in the variable precision rough set model." in *HIS*, 2002, pp. 442–452.

[30] L. Nanni and A. Lumini, "Generalized needleman–wunsch algorithm for the recognition of t-cell epitopes," *Expert Systems with Applications*, vol. 35, no. 3, pp. 1463–1467, 2008. doi: https://doi.org/10.1016/j.eswa.2007.08.028

[31] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." in *Ndss*, vol. 14, 2014. doi: 10.14722/ndss.2014.23247 pp. 23–26.

[32] H. Rathore, S. K. Sahay, P. Nikam, and M. Sewak, "Robust android malware detection system against adversarial attacks using q-learning," *Information Systems Frontiers*, vol. 23, no. 4, pp. 867–882, 2021. doi: https://doi.org/10.1007/s10796-020-10083-8

[33] C. Wu, J. Shi, Y. Yang, and W. Li, "Enhancing machine learning based malware detection model by reinforcement learning," in *Proceedings of the 8th International Conference on Communication and Network Security*, 2018. doi: https://doi.org/10.1145/3290480.3290494 pp. 74–78.

[34] D. Ślęzak, "Rough sets and bayes factor," in *Transactions on Rough Sets III*. Springer, 2005, pp. 202–229.

[35] D. Slezak and W. Ziarko, "The investigation of the bayesian rough set model," *International journal of approximate reasoning*, vol. 40, no. 1-2, pp. 81–91, 2005. doi: https://doi.org/10.1016/j.ijar.2004.11.004