Malware Detection Using Rough Set Based Evolutionary Optimization

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

Malware Detection Using Rough Set Based Evolutionary Optimization

Manel Jerbi✉[1][0000−0002−5070−5573], Zaineb Chelly Dagdia[2,3][0000−0002−2551−6586], Slim Bechikh[1][000−0003−1378−7415], and Lamjed Ben Said[1][0000−0001−9225−884X]

[1] SMART Lab, CS department, University of Tunis, ISG, Tunis, Tunisia
[2] Université Paris-Saclay, UVSQ, DAVID, Versailles, France
[3] LARODEC, University of Tunis, ISG, Tunis, Tunisia
manel.jerbi@gmail.com

Abstract. Despite the existing anti-malware techniques and their interesting achieved results to "hook" attacks, the unstoppable evolution of malware makes the need for more capable malware detection systems overriding. In this paper, we propose a new malware detection technique named Bilevel-Roughset based Malware Detection (BLRDetect) that is based on, and exploits the benefits of, Bilevel optimization and Rough Set Theory. The upper-level of the Bilevel optimization component uses a Genetic Programming Algorithm in its chase of generating powerful detection rules while the lower-level leans on both a Genetic Algorithm and a Rough-Set module to produce high quality, and reliable, malware samples that escape, to their best, the upper-level's generated detection rules. Both levels interact with each other in a competitive way in order to produce populations that depend on one another. Our detection technique has proven its outperformance when tested against various state-of-the-art malware detection systems using common evaluation metrics.

Keywords: Evolutionary optimization · Rough Set Theory · Malware detection.

1 Introduction

Malware authors tend to use obfuscation techniques to infiltrate and hack a targeted system. To counter these attacks, a "good" malware detection system has to detect every sort of attacks and neutralize them. We will focus specifically on the works that opted to generate new, and evolve, malware aiming at keeping the base of malware samples varied and rich; as a way to better detect malicious code. Among these works, we mention [7, 6], where authors applied evolutionary algorithms to generate malware samples. Among the most recent and efficient ones, we mention [5], where a malware detection system (AMD) was proposed that produces patterns using a Genetic Algorithm (GA) in order to mimic real malware patterns. This is to keep the data set 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 [6], 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 spite of their interesting detection rates, these works suffer from many limitations: (1) they refer to a limited number of malware samples which makes the produced base of malicious malware not varied enough, (2) there is no check of the structure of the generated malicious patterns, (3) the malware generation and detection tasks are achieved separately. To overcome the above mentioned shortcomings, our BLRDetect detection technique, also, generates artificial malware – technically called "patterns" which are a set of Application Programming Interface (API) call sequences – but its main distinction and novelty rely on combining evolutionary algorithms, responsible for producing both detection rules and malware patterns, following a BiLevel OPtimization (BLOP) [3] process, and Rough Set Theory [4] to guarantee the "reliability" of the generated patterns.

2 The Bi-level Rough Set Malware Detection Technique

Figure 1 presents the overall running process of BLRDetect: (1) *Module 1* is based on a GP which aims to produce a set of efficient detection rules ($FSDR$) and (2) *Module 2* leans on a GA that produces artificial malicious patterns ($SAMP$) (step 1) and a rough set based component that keeps only the reliable set of artificial malicious patterns that does not present deficiencies concerning their structure, referred to as "High-quality" artificial patterns (FAP) (step 2).

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

Fig. 1. BLRDetect overview.

First module: Upper-level In order to produce a set of effective detection rules, and as shown in Figure 1 and Algorithm 1, the upper-level's first step consists of generating a set of detection rules (Algorithm 1, line 1) which will go through an evaluation process (Algorithm 1, lines 2-3). This evaluation is based on the coverage of the base of examples (input) and also the coverage of the artificial malicious patterns generated by the lower-level. These two measures are used to be maximized by the population of detection rules solutions (Algorithm 1, lines 4-6). The output of this module is a set of final detection rules ($FSDR$) that will be used by the detection task which is responsible for labelling new apps either as malicious or as benign. As the upper-level relies upon a GP process, the GP evolutionary operators require a specific formalization to deal with the manipulated solutions (i.e., the detection rules). These are the following: *(1) Solution representation:* The solution is formalized as a set of terminals, referring to different patterns (API call sequences), and functions (Intersection (AND)

Algorithm 1: The Upper-Level Algorithm

Inputs: SMP: set of malicious patterns, SBP: set of benign patterns, FAP: set of High-quality artificial malicious patterns, NDR: number of detection rules, NAP: number of High-quality artificial malicious patterns in $SAMP$, NU: number of iterations in the upper-level, NL: number of iterations in the lower-level

Output: Final set of detection rules $FSDR$

1: $SDR_0 \leftarrow$ Initialization(NDR,$SAMP$,SBP)
2: **For** each DR_0 in SDR_0 **do**
2.1: $SAP_0 \leftarrow$ APGeneration(DR_0,FAP,NAP, NL)
2.2: $DR_0 \leftarrow$ Evaluation(DR_0,$SAMP$,SAP_0)
3: **End For**
4: $t \leftarrow 1$
5: **While** ($t < NU$) **do**
5.1: $Q_t \leftarrow$ Variation(SDR_{t-1})
5.2: **For** each DR_t in Q_t **do**
5.2.1: $DR_t \leftarrow$ UpperEvaluation(DR_t,$SAMP$)
5.2.2: $SAP_t \leftarrow$ APGeneration(DR_t,$SAMP$,NAP,NL)
5.2.3: $DR_t \leftarrow$ EvaluationUpdate(DR_t,SAP_t)
5.3: **End For**
5.4: $U_t \leftarrow Q_t \cup SDR_t$
5.5: $SDR_{t+1} \leftarrow$ Selection(NDR,U_t)
5.6: $t \leftarrow t+1$
6: **End While**
7: $FSDR \leftarrow$ FittestSelection(SDR_t)

and Union (OR)). *(2) Solution variation:* The GP *mutation operator* is applied to a function or to a terminal by randomly selecting one of them. 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-node from one parent to the other. *(3) Solution evaluation:* The encoding of an individual is formalized as a mathematical function called the "fitness function" that quantifies the quality of the proposed detection rules and the artificial malicious patterns. For the GP adaptation, we used the fitness function f_{upper} defined in Equation 1 to evaluate detection-rules solutions (DR).

$$f_{upper}(DR) = Max(\frac{\frac{Precision(DR)+Recall(DR)}{2} + \frac{\#damp}{\#amp}}{2}) \quad (1)$$

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

$$Precision(DR) = \frac{\sum_{i=1}^{p} DR_i}{t},\ Recall(DR) = \frac{\sum_{i=1}^{p} DR_i}{p} \quad (2)$$

p is the number of detected malicious patterns after executing the solution, i.e., the detection rule, on the base of malicious patterns examples (SMP), t is the total number of malicious patterns within SMP, and DR_i is the i^{th} component of a detection rule DR such that: $DR_i = 1$ if the i^{th} detected malicious pattern exists in SMP ; 0 otherwise.

Second module: Lower-level The generation process of "High-quality" artificial malicious patterns (FAP, Algorithm 2, line 4.8) is performed as follows:

- *Step 1:* A GA is applied that (i) maximizes the distance between the generated malicious patterns ($SAMP$) and the reference benign patterns (input, not-generated patterns (SBP)), (ii) minimizes the distance between the generated malicious patterns ($SAMP$) and the reference malicious ones (SMP), and (iii) maximizes the number of the generated malicious patterns that are not detected by the upper-level; i.e., by the detection rules (SDR) (Algorithm 2, lines 1-4.6). To generate the patterns, the GA evolutionary operators require a specific formalization to deal with the manipulated solutions (i.e., the patterns). These are defined as follows: *(1) Solution representation:* The GA solutions are formalized as chromosomes which are composed of API call sequences. These are identified via their identifiers (IDs) and described by their class labels which indicate their nature (malicious or benign), their different calling depths, and a set of binary values indicating if an API call shows or not in the whole API call sequence. *(2) Solution variation:* For the GA, as previously explained for the GP, the crossover and the mutation operators are applied.*(3) Solution evaluation:* An artificial malicious pattern (AP) is evaluated based on the following GA fitness function:

$$f_{lower}(AP) = Max((\#gamp - \#dagmp) + \sum_{i=1}^{N} f_{Qual}(AP_i)) \tag{3}$$

  where $i \in [1, n]$; n indicates the total number of artificial patterns, and $\#gamp$ refers to the number of artificial malicious patterns and $\#dagmp$ refers to the number of detected artificial malicious patterns. The function $f_{Qual}()$ defined in 4, and its components in Table 1, guarantees the diversity of the artificial malicious patterns.

$$f_{Qual}(AP_i) = \frac{Sim_1 + Sim_2 + Overlap(AP_i)}{3} \tag{4}$$

  A detailed description of the similarity function $Sim()$ can be found in [5].
- *Step 2:* The GA above mentioned evolutionary operators may cause the manipulated solutions to be distorted, and hence ambiguous. Technically, a set of patterns is declared to be ambiguous when they share the same values of the features (API calls) but do have different label values (malicious/benign). To handle this ambiguity issue and to guarantee the reliability of the generated malicious patterns, a rough set component which uses mainly the rough

Table 1. The different $f_{Qual}()$ components.

Similarity used	Description
$Sim_1 = Sim(MS, AP_i)$	
$\text{Sim(MS, AP}_i) = \frac{\sum MS_{j \in MS} Sim(AP_i, MS_j)}{\lvert MS \rvert}$ (5) where j ∈ [1, m];m indicates the total number of malicious patterns.	The similarity between the generated pattern AP_i and the malicious patterns (MS).This measure of similarity needs to be maximized
$Sim_2 = Sim(BS, AP_i)$	
$\text{Sim (BS, AP}_i) = \frac{\sum BS_{k \in BS} Sim(AP_i, BS_k)}{\lvert BS \rvert}$ (6) where $k \in [1, p]$;p indicates the total number of benign patterns.	The similarity between the generated pattern AP_i and the benign patterns (BS) which has to be the lowest.
Overlap(AP_i)	
$\text{Overlap (AP}_i) = 1 - \frac{\sum AP_{l, i \neq l} Sim(AP_i, AP_l)}{\lvert AP \rvert}$ (7)	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 SAP. l refers to the total number of the generated artificial patterns.

set lower approximation concept is plugged to the lower-level. (See Figure 1 and Algorithm 2, lines 4.7-4.8).

Algorithm 2: The Lower-Level Algorithm

Inputs: SMP, SBP, SDR: set of detection rules, G: number of generations, N: population size
Output: Set of High-quality artificial malicious patterns FAP

1: $SAP_0 \leftarrow$ Initialization(SBP,SMP,N,G)
2: $SAP_0 \leftarrow$ Evaluation(SAP_0,SBP,SMP,SDR)
3: $t \leftarrow 1$
4: **While** ($t < G$) **do**
4.1: $Q_t \leftarrow$ Variation(SAP_{t-1})
4.2: $Q_t \leftarrow$ Evaluation(Q_t,SBP,SMP,SDR)
4.3: $U_t \leftarrow Q_t \cup SAP_t$
4.4: $SAP_{t+1} \leftarrow$ Selection(N,U_t)
4.5: $t \leftarrow t$+1
4.6: $SAMP \leftarrow$ FittestSelection(SAP_t)
4.7: $(RAMP, AAMP) \leftarrow$ ReliabilityCheck($SAMP$)
4.8: $FAP \leftarrow$ LowerApproximation($AAMP$) $\cup$ $RAMP$
5: **End While**

Detection task based on detection rules Each pattern is labeled as benign or as malicious by comparing it to the patterns of the SMP and SBP databases. Then, the obtained patterns are compared to the antecedent of $FSDR$.

3 Experimental Setup and Results

To evaluate the performance of BLRDetect, we have considered datasets obtained from the Android Malware Data set (AMD set) [2], 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 [1], 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. Different state-of-the-art methods were considered for comparisons. These are the classical classifiers named in Table 2, tested using Weka with the proposed default parameters settings, the two most recent EA-based methods (AMD [5] and Sen et al. [6]) previously described in Section 1. To ensure the fairness of comparisons between evolutionary approaches, we set to 0.9 the crossover rate and to 0.5 the mutaion rate. All of the evolutionary approaches perform 810 000 function evaluations in each run. When running the experiments, we concluded that the fitness functions become stabilized around the 40^{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.

We compare the BLRDetect obatined results to a set of state-of-the-art non-EA based classifiers (Table 2) and two EA-based approaches (Sen et al. [6] and AMD [5]). Concerning the comparison with non-EA based classifiers, Table 2 shows that BLRDetect outperforms all classifiers based on all the used evaluation metrics. For instance, BLRDetect achieved a precision of 98.09%, an accuracy of 98.21%, an F1_score of 97.82%, and a specificity of 98.36% in comparison to the LDA and $J48$ classifiers, which achieved 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 $J48$. These interesting BLRDetect results are based on its interesting reached values of true positives (98.09%) and the low false positives (01.91%); which are, indeed, the best achieved values among the classifiers' obtained results. These satisfying results confirm that BLRDetect is powerful in performing its detection task between the two possible labels (malicious/benign). Furthermore, from Table 2, we can deduce that, when compared against EA-based methods using the unknown dataset [1], BLRDetect came first with particularly an accuracy = 96.46%, a

[3] https://www.cs.waikato.ac.nz/ml/weka/

Table 2. Comparison between BLRDetect and other detection techniques.

Classifier/ approach	TP	FP	TN	FN	RC	SP	AC	PR	FS	AUC	FPR	FNR
BLRDetect	98.09	01.91	98.15	01.65	98.37	98.36	98.21	98.09	97.82	86.83	01.90	01.65
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
BLRDetect	97.20	02.80	97.99	02.01	98.05	97.77	96.46	96.93	96.41	87.00	02.77	02.02
Sen et al.	97.10	02.80	93.25	06.75	98.24	95.37	95.15	97.13	95.88	82.10	02.91	06.49
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

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

specificity = 97.77%, a recall = 98.05%, a precision = 96.93%, and an AUC = 87.00%. As for Sen et al. and AMD, they achieved an accuracy of 95.15% and 92.28% , a precision of 97.13% and 93.60%, and an AUC of 82.10% 57.69%, respectively, which are lower than those obtained by our proposed technique. The results reported from Table 2 highlights the ability of BLRDetect – thanks to its set of efficient produced rules which are generated using the most reliable set of the generated artificial malicious malware; both guaranteed via the use of the BLOP architecture and the rough set 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 BLRDetect, we analyse the results in terms of false positive and the false negative rates. The registered BLRDetect values of those two metrics (Table 2) confirm the usefulness of a bilevel architecture to detect efficiently malicious code. The continuous competition between both levels permitted good solutions generation (detection rules and artificial malicious patterns) and this had positive impact on the values of FPR (02.77%) and FNR (02.02%). In comparison to BLRDetect, the registered FPR/FNR values for both [6] and [5], which rely on a single-layer based architecture via the use of EAs, are (02.91%/06.49%) and (06.37%/08.84%), respectively. In addition, we can state that the rough-set based module succeeded to set apart 212 000 ambiguous instances among the generated artificial patterns $SAMP$ (468 000 instances) and to produce 256 000 reliables instances. This distinction brings to light the rough set component's important contribution in improving the quality of the artificial malicious patterns by the lower-level and which, consequently, positively affected the false alarms rate.

4 Conclusion

We developed a malware detection technique named BLRDetect which leans on a bilevel architecture and a rough set module. Within the bilevel architecture, the

malware generation task (lower-level) and the rules generation task (detection task, upper-level) are in mutual competition. The lower-level generates "High-quality" malicious patterns which are generated by a GA and thoroughly checked by a rough set component that only keeps the most reliable ones, and which are capable to escape the set of detection rules which are produced by a GP within the upper-level. These efficient generated detection rules try their best to detect the set of artificial malicious patterns generated in the lower-level.

References

1. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., Siemens, C. E. R. T.: Drebin: Effective and explainable detection of android malware in your pocket. In Ndss (Vol. 14, pp. 23-26), (2014, February).
2. Wei, F., Li, Y., Roy, S., Ou, X., Zhou, W.: Deep ground truth analysis of current android malware. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (pp. 252-276). Springer, Cham, (2017, July).
3. Colson, B., Marcotte, P., Savard, G.: An overview of bilevel optimization. Annals of operations research, **153**(1), 235-256, (2007)
4. 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).
5. 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).
6. 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).
7. Xue, Y., Meng, G., Liu, Y., Tan, T. H., Chen, H., Sun, J., Zhang, J.: Auditing anti-malware tools by evolving android malware and dynamic loading technique. IEEE Transactions on Information Forensics and Security, **12**(7), 1529-1544, (2017).