Immune-Based System to Enhance Malware Detection

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

Immune-Based System to Enhance Malware Detection

Manel Jerbi
SMART Lab
University of Tunis, ISG
Tunis, Tunisia
manel.jerbi@isg.u-tunis.tn

Zaineb Chelly Dagdia
Université Paris-Saclay
UVSQ, DAVID
Versailles, France
zaineb.chelly-dagdia@uvsq.fr

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

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

Abstract—Malicious apps use various methods to spread viruses, take control of computers and/or IoT devices, and steal sensitive data such as credit card numbers or other personal information. Despite the numerous existing means of intrusion detection, malware code is not easily detectable. The primary issue with current malware detection approaches is their inability to identify novel attacks and obfuscated malware, as they rely on static bases of malware examples, making them susceptible to new unseen malware behaviors. To address this, we propose a new method for malware recognition, which consists of two processes: the first process creates new instances of malware using a memetic algorithm, and the second process detects these new instances of attacks through solid detectors produced by an artificial immune system-based algorithm. Our new malware recognition method has proven its merits through thorough experiments on widely used datasets and evaluation metrics, and has been compared to prominent state-of-the-art methods.

Index Terms—Malware, Memetic algorithms, Artificial immune systems

I. Introduction

Malware is malicious software created to access a computer or any other device and cause damages to it. Malicious programs try to find their way to the targeted systems, which are usually connected to Internet most of the time, either for work or for personal use, with the widespread of IoT technology. We can easily notice that IoT technology is vulnerable to malware attacks especially due to the fact that IoT devices lack robust security measures [1]. In this concern, different methods were proposed which try to detect malware programs relying on specific features within the apps. Those features can be classified with regards to their type either as static or dynamic, which depends on the nature of the used detection method [2] (i.e., signature-based, behavioral-based or heuristic-based method). Various methods and techniques were proposed in literature to enhance the array of computer security means where the use of (deep) neural networks [3] and evolutionary algorithms (EAs) [4], [5], was particularly present in recent works. Those methods showed interesting results in detecting malware when assessed using static stored malicious samples which is no longer the case when tested against new unknown variants of malware. This can be explained by the lack of diversity of the malware samples. In another perspective, many works relied on machine learning classifiers [6] to set new detection rules but those rules led to high percentages of false positives. In this paper, we propose a two-step detection approach, named IMMU-Det, which is distinguished by the combination of a Memetic Algorithm (MA) and an Artificial Immune system (AIS) based algorithm relying on a clonal selection process to generate a diverse population of immune cells (detectors in our case). The first step is the one that will generate a new set of "memes", those are the malicious variants (vectors of Application Programming Interface (API) calls) that will serve, in the following step, as input (antigens) to the AIS based algorithm which, in turn, will produce detectors. Those detectors will help reveal the true nature of unknown applications.

Let us mention that an API is a collection of protocols, procedures, and functions that enables data exchange between numerous programs and gadgets. An analysis of the API calls invoked within an app's running process regarding their number or their nature (i.e., sensitive or not) will greatly help anti-malware producers to examine the app's behavior and categorize it afterwards as a safe or a dangerous app. The key contributions of our work are as follows:

1) The suggestion of a new malware detection process based on AIS combined with MA, where the AIS-based clonal selection algorithm generates a set of effective and reliable detectors capable of detecting new malicious codes generated by a MA.
2) The MA-based schema serves as an excellent example of the advantages of genetic algorithm extended by a local search module which optimizes the search space of potential generated malicious codes (i.e., memes). It helps selecting the most challenging memes which will have impact on the detection quality of the detectors, output of the AIS-based clonal selection algorithm.
3) The advantages of coupling a MA and an AIS-based clonal selection algorithm are highlited in the experimental results showing a more accurate prediction

of the nature of new apps and a consequent decrease in erroneous decisions.

4) Our IMMU-Det approach has outperformed a number of malware detection techniques and engines in terms of accuracy maximization and false alarm decrease.

The rest of this paper is organized as follows: Section III-A presents the essential descriptions tied to MA and AIS-based clonal selection algorithm used in this work. Section II presents related previous works. Section III describes our proposed approach. The experimental setup and the results of the performance analysis are given in Section IV, and the conclusion is given in Section V.

II. Related work

In this section we will present recent related works that essentially focus on malware detection.

For instance, we can refer to the work of [7] where authors designed three self-supervised attack techniques: (1) the first technique considers the IoT input data and the adversarial sample generation in real-time. (2) The second technique builds a generative adversarial network model to generate adversarial samples in the self-supervised structure. Finally, the third technique utilises three well-known perturbation sample techniques to develop adversarial malware and inject it over the self-supervised architecture. Also, authors applied a defense method to mitigate these attacks, namely, adversarial self-supervised training, to protect the malware detection architecture against injecting the malicious samples. The method showed good detection rates when assessed using a known dataset but we can expect lower detection rates as the detection task and the generation task are done separately. In another work [8], authors suggested to consider the detection rules generation process as a bi-Level optimization problem, 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 of combined patterns. The inner task aims to generate a set of artificial malicious patterns that escape the rules of the outer task. The main limitation of the method is the existence of irrelevant malicious patterns which lead to produce detection rules with consequent rate of false alarms. In the work of [9] authors proposed a hybrid deep generative model that exploits global and local features together to detect the malware variants. They first transformed malware into an image to represent global features with pre-defined latent space, then, they extract local features using the binary code sequences. The two features extracted from the data with their respective characteristics are concatenated and entered into the malware detector. While achieving interesting detection results, we can notice that the transformation phase can cause data loss. The study of [10] investigated whether genetic algorithm-based feature selection helps Android malware detection. Authors applied nine machine learning algorithms with genetic algorithm-based feature selection. In fact, experiments were conducted to select permission and API method information features to apply machine learning classification algorithms. The main flaws within this method are: (1) the use of only static features and, (2) the run-time needed is relatively important (approximately 15 hours). Authors in [11] proposed an approach that combines global search and local search heuristics, through a memetic evolutionary search process. The tabu-search algorithm is used as the local search technique, to improve the quality and fitness of solutions through scouring the neighbourhood of each solution for better individuals. Even so, the overall method needs to improve the ability of the search technique in order to converge towards high-quality solutions. Also, we can refer to the work of [12] where authors proposed an approach, called MDEA, an Adversarial Malware Detection model that uses evolutionary optimization to create attack samples to make the network robust against evasion attacks. The proposed model suffers from overfitting shortcomings and the generation of dead species of malware patterns. In the work of [13], authors used an ensemble classifier consisting of multiple one-class classifiers to detect known and unknown malware misusing registry keys and values for malicious intent. The main limitation is that they relied only on the benign programs to build the registry model.

Among the works that relied on AIS to detect malicious codes, we can cite [5] where authors developed a virus-oriented computer defense immune system (CDIS) based on biological strategies. The model showed its limits when confronted to unknown malware bases.

Generally, the main limitation in these models lies in the fact that their training phase relies on particular datasets that might fast become outdated as new viral types continue to emerge.

III. Memetic - AIS - based detection method: IMMU-Det

A. Preliminaries

a) Memetic Algorithms: Memetic algorithms [14] are a type of optimization algorithms that combine elements of genetic algorithms and local search techniques. They are named "memetic" because they are inspired by the concept of memes, which are self-replicating ideas or behaviors that can spread and evolve within a population.

In memetic algorithms, a population of solutions to a problem is evolved through a process of selection, mutation, and recombination, similar to genetic algorithms. However, in addition to these genetic operators, memetic algorithms also incorporate local search procedures that can fine-tune individual solutions and improve their quality. The general steps for implementing a memetic algorithm are as follows:

1) Define the optimization problem and determine the parameters of the memetic algorithm, such as the population size, the number of generations, and the crossover and mutation rates.

2) Initialize the population with a set of random solutions.
3) Evaluate the fitness of each solution in the population.
4) Apply genetic operators (e.g., crossover and mutation) to the population to generate new solutions.
5) Apply local search to each solution in the population to improve its quality.
6) Repeat steps 3-5 for the specified number of generations or until a satisfactory solution is found.
7) Return the best solution found by MA.

It is important to note that the specific steps for implementing a memetic algorithm may vary depending on the problem being solved and the specific requirements of the algorithm. In fact, memetic algorithms have been applied to a variety of optimization problems, including combinatorial optimization, continuous optimization, and multiobjective optimization [15].

In the context of malware detection, memetic algorithms could eventually be used to search for patterns or features that are indicative of malware. The genetic operators could be used to explore different combinations of features, while the local search procedures could be used to fine-tune the selection of features and improve the accuracy of the detection model. An adaptation of MA is adopted within our proposed method which can be found in Section III.

b) Artificial immune systems and clonal selection: Artificial immune systems (AIS) are computational models inspired by the immune system of living organisms. They are used for various tasks such as pattern recognition, novelty detection, and anomaly detection. AIS have been applied to the task of detecting malware. In fact, in the context of malware detection, the immune cells in the AIS population may be trained to recognize patterns or features that are indicative of malware. These patterns may include specific code sequences, file header values, or other characteristics that are commonly found in malware but not in benign software. When the AIS-based detection method is presented with a new file, it can use the immune cells to search for these patterns and determine whether the file is likely to be malware. If the file is deemed to be malicious, the AIS-based method can take appropriate actions, such as quarantining the file or alerting the user. One advantage of using AIS for malware detection is that it can adapt to new types of malware as they emerge, by creating new immune cells to recognize them. This can make it more effective at detecting zero-day vulnerabilities, which are vulnerabilities that have not yet been identified and patched.

One approach to implementing AIS is to use a Clonal Selection Algorithm (CSA) [16], which involves creating a population of "immune cells" that are trained to recognize certain patterns or anomalies. These cells are then used to search for and identify similar patterns in new data. The principles of clonal selection are based on the following concepts:

1) Initialization: Randomly initialize a population of individuals (N).
2) Evaluation: Given a set of patterns to be recognized (P), for each pattern, determine its affinity (match) with each element of the population.
3) Selection and cloning: Select a number (n) of the best highest affinity elements of N and generate copies of these individuals proportionally to their affinity with the antigen.
4) Hypermutation: Mutate all the copies with a rate proportional to their affinity with the input pattern.
5) Receptor editing: Add the mutated individuals to the population and reselect a number (d) of the maturated (optimized) individuals as memory.
6) Repeat steps 2–5 until a termination criterion is met.

In this work, a new method for Android malware detection based on MA and a clonal selection based algorithm, referred to as "IMMU-Det", is proposed and Figure 1 shows its overall process.

From Figure 1, we can distinguish two main phases: the first phase is based on a MA and is responsible for the diversification of malware variants whereas the second phase is responsible for the detection process by adapting an AIS-based clonal selection algorithm, i.e., an adapted version of the clonal selection algorithm. More precisely, in the first part (phase 1), we use a MA in order to generate new variants, called memes, of malware. Those memes will be fed to ac based algorithm (phase 2) to produce efficient set of detectors capable of revealing unseen malware behaviors.

B. Memes generation

Let us recall that the aim of our MA is to produce "memes" assessed during the generation process in order to keep the "best" ones (Figure 2). To be able to perform this generation process, we need to precise that the individuals processed by the MA are vectors composed of a succession of API calls (i.e., sequence of API calls) coupled with their respective weights as illustrated in Figure 3.

The weight of a given API call is its appearance frequency in a specific API call sequence and is used to determine the Quality (Equation 2) of a meme. This calculated Quality, together with the affinity of a specific meme (Equation 3), will determine the fitness of the generated meme. In order to be more accurate, the computation of a meme's affinity is based on the difference between two measures: the similarity measure, given by the Jaccard index [18], of a meme compared to the input set of individuals (vectors of API calls) and the penalty measure (Equation 4) which could be considered as the average similarity distance between the generated meme and all the other generated memes.

More precisely, and as detailed in Algorithm 1, the evolutionary step (Algorithm 1, line 1) generates the memes which will go through (1) an evaluation process (Algorithm 1, lines 2-4) followed by (2) a solution variation

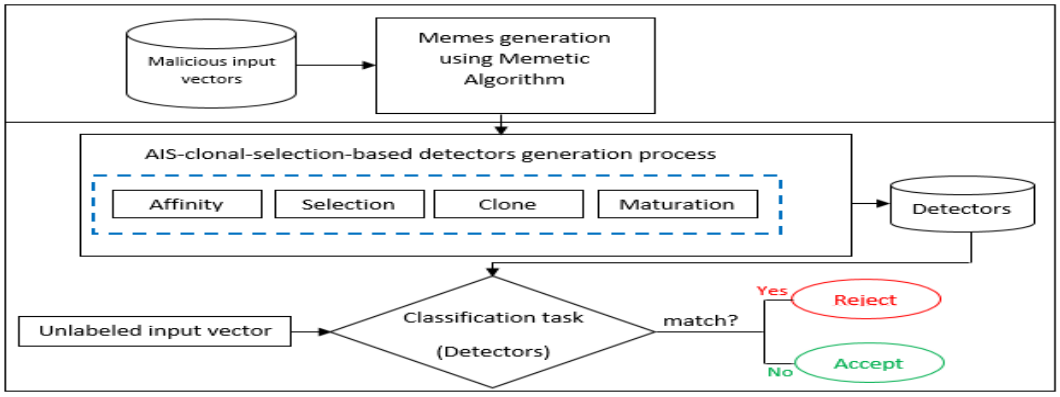

Fig. 1. IMMU-Det's overview.

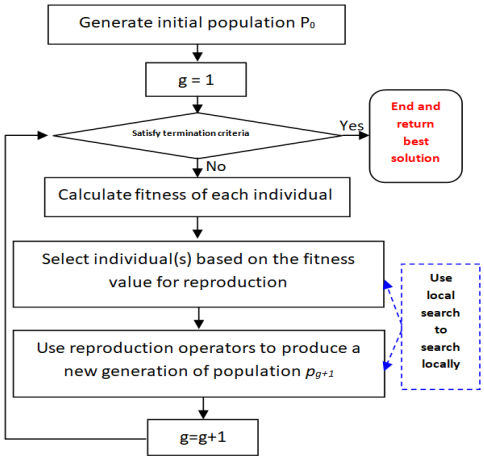

Fig. 2. The process of generating individuals. Inspired by [17].

V_k	API call$_1$	API call$_2$	...	API call$_m$
	0	1	...	1
w_k	w_1	w_2	...	w_m

Number 1 indicates if the API call exists in the vector in question and 0 otherwise.

Fig. 3. Representation of an individual (vector of API calls).

process relying on evolutionary operators (crossover – create new candidate individuals by merging information contained in selected parents– and mutation –modifying one individual at a time–) (Algorithm 1, lines 5-11). The evolutionary step is then extended by a local search (Algorithm 1, lines 12 and 15) which could be seen as performing local refinements to the produced memes.

a) Fitness function: The evaluation process relies on a fitness function, Equation 1, that mainly uses two metrics for the memes' evaluation process. These metrics are introduced in Equations 2 and 3 where the first one (Equation 2) measures the quality of a meme based on the API calls composing it and their corresponding weight. The weight (w_k) of an API call (V_k) is the normalized number of its occurrence in the corresponding meme. The

Algorithm 1 Evolving Individuals Algorithm

Inputs:N: Population size, G: Number of generations, MS: Malicious input set (individuals)
Output: MeM: Solutions (memes)
1: $P_0 \leftarrow$ Initial-Population(MS)
2: For each I_0 in P_0 do
3: $I_0 \leftarrow$ Evaluation(P_0)
4: End For
5: $g \leftarrow 1$
6: While g < G do
7: $C_g \leftarrow$ Variation(P_{g-1})
8: $C_g \leftarrow$ Evaluation(C_g,MS)
9: $Com_g \leftarrow C_g \cup P_{g-1}$
10: $P_g \leftarrow$ Selection(N, Com_g)
11: $FP \leftarrow$ BestSelection(P_g)
12: $BL \leftarrow$ LocalSearch(P_g)
13: $g \leftarrow g + 1$
14: End While
15: $MeM \leftarrow$ LocalSearch(BL)

second metric (Equation 3) measures the affinity between the generated meme and the initial set of individuals (i.e., the malicious input set) while guaranteeing an overlapping factor which denotes the overlap distance between a meme M_i and other memes.

$$Fitness_M = \frac{\frac{\sum_{i=1}^{n} Quality(M_i)}{|M|} + \frac{\sum_{i=1}^{n} aff(M_i,I)}{|I|}}{2} \quad (1)$$

$$Quality(M_i) = \sum_{k=1}^{m} V_k w_k \quad (2)$$

$$aff(M_i, I) = \sum_{j=1}^{l} S(M_i, I_j) - Pty(M_i) \quad (3)$$

$$Pty(M_i) = \frac{\sum_{M_j, i \neq j} S(M_i, M_j)}{|M|} \quad (4)$$

For more easiness regarding the reading of Equations 1 to 4, Table I gives the variables' signification.

TABLE I
Variables signification. (Equations 1;2;3;4)

Variable	Signification
M_i	Generated meme (i ∈ [1,n]; n is the total number of generated memes).
I_j	Initial set of malicious individuals i.e., malicious input vectors; j ∈ [1,l]; l is the total number of malicious input vectors).
V_k	ID of an API call. (Figure 2).
w_k	The corresponding weight of an API call V_k.

As already mentioned, the similarity between two individuals X and Y is S(X,Y) and is calculated using Jaccard coefficient which measures similarity between finite sample sets, two vectors of API calls (memes) in our case, and is defined as the size of the intersection divided by the size of the union of the memes.

b) Local search: An important aspect to consider when using a MA is the determination of a local search algorithm to improve the quality of each individual in a given population as fast as possible. In our study Hill-climbing [19] is implemented as it uses very little memory, and can often find reasonable solutions in large or infinite state spaces. Let us briefly recall that Hill-climbing is an iterative process that starts with a randomly chosen solution to a problem and then makes small adjustments to the solution in an effort to find a better one. Other local search algorithms could be possibly used but this is beyond the scope of this paper.

C. Detectors generation

In this section, we present how an AIS-based clonal selection algorithm could be applied to detect malware. We propose to protect an operating system from malware intrusions by modeling a detection system that generates a set of detectors which will serve to reveal potential malicious behaviors of given apps.

Algorithm 2 Detectors generation process

Inputs:SMI: Set of malicious individuals
Output: SD: Set of final detectors
1: $D_0 \leftarrow$ Initial-Detectors(SMI)
2: For each MI_0 in SMI do
3: $D_0 \leftarrow$ Evaluation(SMI)
4: $g \leftarrow 1$
5: While g < G do
6: $D_g \leftarrow$ Evaluation-Sort(D_{g-1},SMI)
7: $SD_g \leftarrow$ Aff-Prunning(SD_g,)
8: $SD_g \leftarrow$ Aff-Insertion(SMI)
9: $SD_g \leftarrow$ Maturation(SD_g)
10: $g \leftarrow g + 1$
11: End While
12: End For
13: $SD \leftarrow$ Prunning(SD_g)

The generation of detectors' step, as shown in Algorithm 2, starts by gathering the antigens (vectors of API calls also referred to as malicious individuals (SMI)). A new population of detectors (SD) is produced from the antigen group (Algorithm 2, line 1). The detectors set's size is equal to the number of original antigens. Then, the population of detectors is exposed to each antigen, and for each detector, an affinity value to the antigen is computed. After that, the detectors are sorted according to decreasing affinity (Algorithm 2, lines 2-6). Detectors with better affinity value are selected and other detectors are deleted (Algorithm 2, lines 7-8). Afterwards, the maturation operation (Algorithm 2, line 9) takes a detector as an input and with a change on that detector generates a new sample. In fact this operator is the substitute operator of mutation operator in genetic algorithms and is responsible for creating diversity in a population. Each detector is exposed to a new antigen, and detectors use their affinity to compete with one another and the detector that has more affinity value is allowed to copy more samples of itself.

Euclidean distance (Equation 5) was employed in our study to quantify the affinity between the set of malicious individuals (SMI) and the detectors.

$$dist(D^k, MI^u) = \sqrt{\sum_{i=1}^{Q}(D_i^k - MI_i^u)^2} \tag{5}$$

where $SD = \{D^1, D^2, \ldots, D^{nd}\}$ is the set of detectors, and $SMI = \{MI^1, MI^2, \ldots, MI^{ns}\}$ is the set of malicious individuals.

The detection task is about predicting an input vector (IV) of API calls which is either malicious or normal. Expression 6 is used for such prediction:

$$Classify(IV, \theta, dist, SD) = \begin{cases} \text{malicious if } dist(SD, IV) < \theta \\ \text{normal else.} \end{cases} \tag{6}$$

where θ is a threshold value that reflects the minimum similarity value beyond which the input vector cannot be considered as malicious. In this study, the value of 0.5 is adopted for θ.

IMMU-Det's detection process can classify application, either as malicious or normal, based on a set of detectors which are non other than the final output of the AIS-based clonal selection algorithm. Figure 1 shows the classification process.

IV. Experimental analysis

The goal of our analysis is to assess our proposed detection method and to give an insight about the registered results specially when compared to the most recent state-of-the-art methods. In order to do that, we need to answer the following research questions (RQs):

- RQ1: To which extent can our detection method unmask malicious code? To answer RQ1, we will

discuss the registered detection rates and the metrics used for this purpose.

- RQ2: Can our detection method be considered competitive among the existing state-of-the-art detection methods? To answer RQ2 a set of comparisons to most recent state-of-the-art methods is conducted relying on different evaluation metrics.
- RQ3: How can our method score low false alarms rates? To answer RQ3, we will show the added value of the use of the MA in enhancing the quality of the produced detectors and its role in reducing the false alarms rates.

Samples from the Canadian Institute for Cybersecurity were used in our experimental study: CIC-MalMem-2022 (58 596 records) used to produce the detectors, and CCCS-CIC- AndMal-2020 (400 000 Android static/dynamic malware samples)[1] used for the evaluation of our method. To show that our method is not fitting the base of samples, we selected a dataset that is distinct from the one used for training. In https://www.unb.ca/cic/datasets/andmal2020.html, the datasets are described in depth. In all of the conducted experiments, a 10-fold cross validation test is performed, and are run on an *Intel® Xeon®* Processor CPU E5-2620 v3, 16 GB RAM.

A. Evaluation metrics, results and comparisons to state-of-the-art methods

In order to answer RQ1 and to assess our method a set of evaluation metrics were used. These are the recall (RC), the specificity (SP), the accuracy (AC), the precision (PR), the F_score (FS), the area under curve (AUC), the false positive rate (FPR) and the false negative rate (FNR). Also, for the sake of running the MA and AIS-based algorithm, a set of parameters need to be fixed. Table II shows the values set to run our algorithm. From

TABLE II
Used parameters.

	IMMU-Det			
	MA	AIS-based algorithm		
Population size	200	100		
Generation size	6000	1000		
Mutation rate	0.05	0.01	0.05	0.1
Crossover rate	0.8	–	–	–
Accuracy	–	97.67	97.29	97.65
Precision	–	97.31	97.01	97.29
Training time	–	49h43mn 03s	50h29mn 00s	76h01mn 34s

Table II, we can conclude that the mutation value of 0.01 helped the AIS-based clonal algorithm reach the best accuracy and precision rates (97.67% and 97.31% respectively) with a reasonable training time. This can be explained by the fact that high mutation rates can cause a premature convergence of the algorithm and also a high computational cost.

[1]https://www.unb.ca/cic/datasets/andmal2020.html

TABLE III
Comparison between IMMU-Det and a set of different classifiers.

Classifier/ approach	RC	SP	AC	PR	FS	AUC	FPR	FNR
IMMU-Det	97.79	97.32	97.67	97.31	97.65	85.12	01.09	01.13
LR	79.49	77.46	78.44	76.66	78.05	60.62	02.31	03.34
NB	49.89	50.11	50.78	49.22	50.33	55.11	05.01	04.92
RF	75.38	73.48	74.39	72.45	73.38	72.94	02.752	02.36
J48	70.11	70.96	71.03	70.86	70.98	63.90	03.91	03.87
k-NN	70.50	70.44	70.47	70.39	70.45	60.44	03.96	03.94
SVM	95.59	94.79	95.35	94.73	95.32	71.88	01.27	01.05
Light GBM	92.29	89.73	90.97	89.41	90.83	69.96	02.18	01.70
SGA	96.06	89.73	95.74	95.90	95.73	95.89	04.27	03.92

J48: Decision Tree; NB: Naive Bayes; k-NN: k-Nearest Neighbours; LR: Logistic Regression; SVM: Support Vector Machine;RF: Random Forest; Light GBM: Light Gradient-Boosting Machine; SGA: Simple Genetic Algorithm.

Table III represents the obtained results of IMMU-Det and compares them to other classifiers. This can show the performance of our model when confronted to new data and also to answer RQ2. Concerning IMMU-Det's results and their comparison to the state-of-the-art methods, IMMU-Det achieved an accuracy of 97.67%, a recall of 97.79%, a precision of 97.31% and a false positive and a flase negative rates of 01.09% and 01.13 %, respectively. Also, SGA, SVM and Light GBM performed better than the other classifiers while ranking below IMMU-DET. In fact, SGA, SVM and Light GBM registered 95.90%, 95.35% and 90.97% of accuracy respectively, and 95.73%, 94.73% and 89.41% of precision. Also, SGA registered 04.27% of false positive rate whereas SVM and Light GBM reached respectively 01.27% and 02.18% of false positive rate. The IMMU-Det obtained interesting values of true positives and true negatives can be explained by the quality of detectors produced by the clonal selection algorithm. We can thus conclude that our model was capable of accurately determining the true nature of the unknown apps.

Table IV shows the classification performance of all compared classifiers to IMMU-Det in terms of CPU time consumption in the training and the prediction processes. In fact, IMMU-Det required important time for training but is comparable in terms of execution time for prediction to other methods. For instance, IMMU-Det required about 50mn to predict the unknown samples which can be considered reasonable when compared to the time required by Light GBM and SGA (about 48mn) and the RF classifier (approximately 1h 33 mn). The fastest prediction time was obtained with K-NN classifier which needed about 30mn to perform the prediction task. The training time consumption can be explained by the required time to run the MA and the AIS-based algorithm.

Figure 4 shows the TPR-FPR ROC curve of IMMU-Det and the different classifiers used for the sake of comparisons. The Naive Bayes approach has the lowest curve when compared to IMMU-Det and all other classifiers, when we rely on the AUC values of all the models to

TABLE IV
Performance of the classifiers during training and prediction processes.

Classifier	CPU time	
	Training process	Prediction process
IMMU-Det	49h43mn03s	00h52mn30s
LR	03h03mn12s	00h31mn54s
NB	00h59mn01s	00h28mn42s
RF	30h45mn35s	01h32mn15s
J48	01h42mn23s	00h31mn00s
k-NN	01h12mn08s	00h29mn54s
SVM	82h33mn41s	89h02mn00s
Light GBM	64h03mn00s	00h48mn23s
SGA	63h45mn23s	00h48mn02s

measure their ability to differentiate classes. Also, Figure 4 shows the models' PRC (Precision-Recall Curve) graph. It can be seen that when the Naive Bayes algorithm's recall values rise, the precision value quickly falls after a certain point and is below the 0.7 precision threshold. There is some variation in the precision numbers when the recall levels of the J48 and k-NN algorithms drop. However, it may be claimed that SVM and Light GBM have the ideal balance over the rest of the classifiers. Finally, IMMU-Det produced the least amount of misclassifications while performing good classification results.

B. Importance of MA and the clonal selection algorithm

In order to rate IMMU-Det's performance and to answer RQ3, a comparison to recent works is performed. We compared our work to six state-of-the-art works. Two among them use either a MA or an AIS-based algorithm and those are the methods of [5] and [13] and were previously introduced in Section II. And the four other methods are more recent ones ([20] [3] [10] [12]). The work of [20] proposed to use an algebraic topological approach called topological-based data analysis (TDA) to analyze and detect malware patterns. Authors compared the different TDA techniques (i.e., persistence homology, tomato, TDA Mapper) and existing techniques (i.e., PCA, UMAP, t-SNE) using different classifiers. Also, the work of [3] suggested to use data obtained through memory analysis to provide insights into the behavior and patterns of malware based on the fact that malware leave traces on memories relying on different classifiers as well. The works of [10], [12] were previously presented in Section II as well.

As reported from Table V, we can deduce that, when compared to the state-of-the-art methods, IMMU-Det outperformed the work of [20], [3], [12] and also the work of [10] in terms of registered accuracy using the CIC-MalMem-2022 dataset. For instance, [20] registered an accuracy value ranging between 80% and 100% when using the Persistence Diagram whereas [3] and [12] reached an accuracy of 97.67% and 94.00% respectively. Also, [10] registered 88.40% of accuracy when using AdaBoostM1. IMMU-det's interesting results show its ability to reveal the true nature of unknown apps and that can be explained

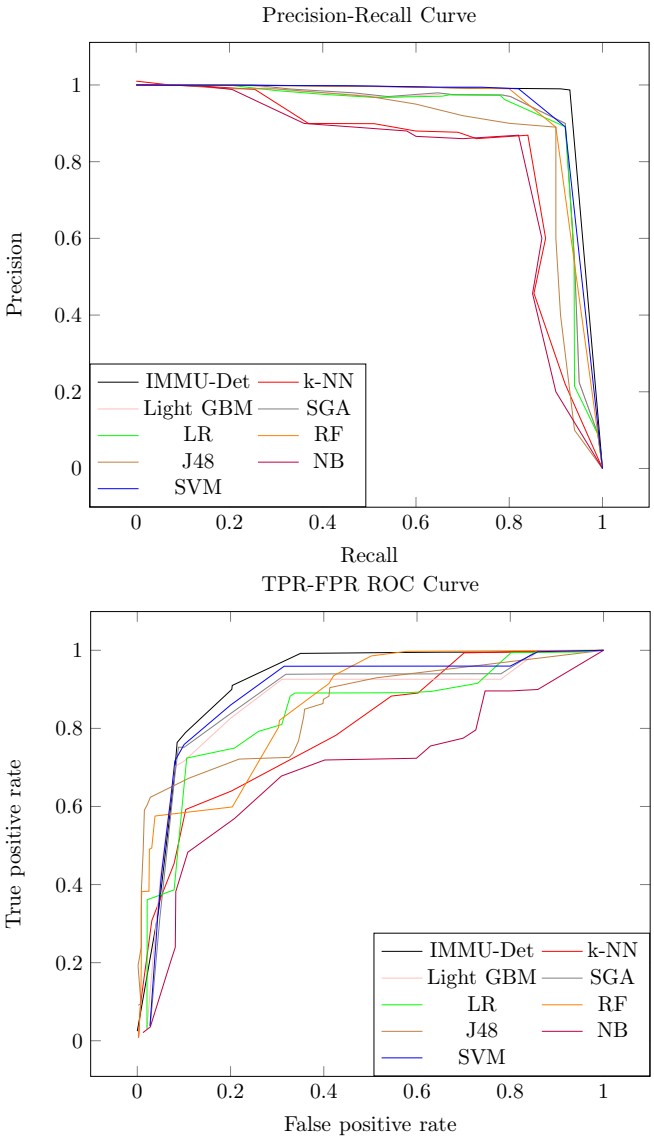

Fig. 4. IMMU-Det's obtained ROC curves with different classifiers.

by the fact that the memetic algorithm helped produce a set of unseen malware behaviors that, in turn, helped the AIS clonal selection algorithm in its task of producing reliable detectors. More precisely, the clonal selection algorithm when fed with challenging malicious codes succeeded in improving the quality of the detectors. For instance, those detectors succeeded in registering 97.67% of accuracy when tested against the CIC-MalMel-2022 dataset. Also, when compared to methods relying on MA [13] or AIS [5], IMMU-Det showed high performance in terms of accuracy and that can confirm the fact that the MA helped produce better malicious samples. Those malicious samples, when fed to the clonal selection algorithm helped produce more effective detectors. So, we can say that by improving the quality of the input dataset (input malicious vectors) of the clonal selection algorithm,

TABLE V
IMMU-Det's detection results compared to state-of-the-art methods on the CIC-MalMem-2022. (except [5], [10], [12], [13])

Anti-malware	Accuracy (%)
IMMU-Det	97.67%
[20]	approximately 80% with Persistence Diagram
[3]	97.67 %(with MLP)
[5]	89% (using TIMID virus and EICAR)
[13]	96.99% (using Windows samples)
[10]	98.10%(using MultilayerPerceptron) 88.40% (using AdaBoostM1)
[12]	94.00%

we succeeded in producing better quality detectors.

V. Conclusion, discussion and future directions

The fundamental problem with current malware detection methods is their limitation when tested against unseen malware due to their use of static datasets of malware behaviors during the training phase. To overcome such problems, in this paper, an innovative malware detection method, IMMU-Det, is proposed. IMMU-Det's overall process is composed of two steps. The first one relies on a MA to produce "memes", malicious API call vectors, that mimic the behaviour of the real API calls extracted from malware but are at the same time characterized by their new unseen behavior, and will hence serve as input for the clonal selection algorithm which, in turn, produces "detectors". Those detectors are used to unmask malicious unknown apps. IMMU-Det brought important contributions: (1) a detection process combining a MA with an AIS-based clonal selection algorithm, (2) leverage the benefits of both algorithms when combined together, (3) improvement of the detection rates and decrease of false alarms rates (4) assessment using various evaluation metrics and comparison to the most recent state-of-the-art detection methods. IMMU-Det succeeded to register 97.67% of accuracy and only 01.09% of FPR.

Despite the good registered detection rates, we came to realize that IMMU-Det could be improved in four main aspects. The first aspect focuses on the increase of diversity of Android sample attributes utilized to construct the detection model. More specifically, we want to rely on additional Android features (i.e., permissions) and not just be limited to API calls when it comes to the task of conceptualizing both the detectors and the memes. We also want to investigate additional feature selection techniques that might, for example, rely on Reinforcement learning. A second aspect is to address the amount of time the model needs to train. In fact, designing an adaptive parameter tuning technique that attempts to approximate the optimal parameter values for the EAs could be an interesting option to optimize the training time utilized in the system's training process. Also, the third aspect could be the search for an alternative component to produce new malicious codes. Varying the malicious codes used to build the detection model could improve the detection rates. Finally, the fourth aspect will mainly focus on expanding the scope of the current work to other operating systems.

References

[1] W. H. Hassan et al., "Current research on internet of things (iot) security: A survey," Computer networks, vol. 148, pp. 283–294, 2019.

[2] A. Damodaran, F. D. Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "A comparison of static, dynamic, and hybrid analysis for malware detection," Journal of Computer Virology and Hacking Techniques, vol. 13, no. 1, pp. 1–12, 2017.

[3] M. Dener, G. Ok, and A. Orman, "Malware detection using memory analysis data in big data environment," Applied Sciences, vol. 12, no. 17, p. 8604, 2022.

[4] L. Hong, "Artificial immune system for anomaly detection," in 2008 IEEE international symposium on knowledge acquisition and modeling workshop. IEEE, 2008, pp. 340–343.

[5] P. K. Harmer, P. D. Williams, G. H. Gunsch, and G. B. Lamont, "An artificial immune system architecture for computer security applications," IEEE transactions on evolutionary computation, vol. 6, no. 3, pp. 252–280, 2002.

[6] P. Harshalatha and R. Mohanasundaram, "Classification of malware detection using machine learn-ing algorithms: A survey," International Journal of Scientific & Technology Research, vol. 9, no. 02, 2020.

[7] M. Golmaryami, R. Taheri, Z. Pooranian, M. Shojafar, and P. Xiao, "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), vol. 18, no. 2s, pp. 1–21, 2022.

[8] M. Jerbi, Z. C. Dagdia, S. Bechikh, and L. B. Said, "Android malware detection as a bi-level problem," Computers & Security, vol. 121, p. 102825, 2022.

[9] J.-Y. Kim and S.-B. Cho, "Obfuscated malware detection using deep generative model based on global/local features," Computers & Security, vol. 112, p. 102501, 2022.

[10] J. Lee, H. Jang, S. Ha, and Y. Yoon, "Android malware detection using machine learning with feature selection based on the genetic algorithm," Mathematics, vol. 9, no. 21, p. 2813, 2021.

[11] S. M. Akandwanaho and M. Kooblal, "Intelligent malware detection using a neural network ensemble based on a hybrid search mechanism," The African Journal of Information and Communication, vol. 24, pp. 1–21, 2019.

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

[13] A. Tajoddin and M. Abadi, "Ramd: registry-based anomaly malware detection using one-class ensemble classifiers," Applied Intelligence, vol. 49, no. 7, pp. 2641–2658, 2019.

[14] N. Krasnogor, A. Aragon, and J. Pacheco, "Memetic algorithms," in Metaheuristic procedures for training neutral networks. Springer, 2006, pp. 225–248.

[15] F. Neri and C. Cotta, "Memetic algorithms and memetic computing optimization: a literature review. swarm evol comput 2: 1–14," 2012.

[16] R. M. Anderson, The population dynamics of infectious diseases: theory and applications. Springer, 2013.

[17] J. Liang and Y. Xue, "Multi-objective memetic algorithms with tree-based genetic programming and local search for symbolic regression," Neural Processing Letters, vol. 53, no. 3, pp. 2197–2219, 2021.

[18] S. Niwattanakul, J. Singthongchai, E. Naenudorn, and S. Wanapu, "Using of jaccard coefficient for keywords similarity," in Proceedings of the international multiconference of engineers and computer scientists, vol. 1, no. 6, 2013, pp. 380–384.

[19] J. Borghoff, L. R. Knudsen, and K. Matusiewicz, "Hill climbing algorithms and trivium," in Selected Areas in Cryptography: 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers 17. Springer, 2011, pp. 57–73.

[20] L. N. Tidjon and F. Khomh, "Reliable malware analysis and detection using topology data analysis," arXiv preprint arXiv:2211.01535, 2022.