A method of generating mutated Windows malware to evade ensemble learning-based detectors

Pham Van Hau, To Trong Nghia, Phan The Duy

Abstract— Recently, the application of machine learning (ML) in the field of cybersecurity, particularly in the detection and prevention of malware, has received significant attention and interest. Numerous research works on malware analysis have been proposed, showing promising results for practical applications. In such works, the use of Generative Adversarial Networks (GANs) or Reinforcement Learning (RL) can help adversaries create mutated malware to evade detection. In this study, we propose a method for generating mutated Windows malware against malware detection based on ensemble learning by combining GANs and RL to overcome the limitations of the MalGAN model. Specifically, we develop the FeaGAN model, an extension of MalGAN, by incorporating the model with the Deep Q-network anti-malware Engines Attacking Framework (DQEAF) RL model. Furthermore, the FeaGAN model employs ensemble learning for malware detection to enhance the evasion capabilities of the generated adversarial samples. Experimental results show that 100% of the selected mutation samples maintain their format integrity. Additionally, the ability to preserve the executable functionality of the malware variants achieves promising results with a stable success rate.

***Tóm tắt*— Thời gian gần đây, việc ứng dụng học máy vào lĩnh vực an ninh mạng, đặc biệt là trong phát hiện và ngăn chặn mã độc được chú trọng và quan tâm sâu sắc. Nhiều công trình nghiên cứu về phân tích phần mềm độc hại đã được đề xuất với những kết quả hứa hẹn cho các ứng dụng thực tế. Trong những công trình như vậy, việc sử dụng mạng sinh đối kháng (Generative Adversarial Networks) hoặc học tăng cường (Reinforcement Learning) có thể giúp các kẻ xấu tạo ra phần mềm độc hại đột biến trốn tránh các phần mềm chống virus. Trong nghiên cứu này, nhóm tác giả đề xuất một hệ thống đột biến chống lại các trình phát hiện mã độc dựa trên học tổng hợp (Ensemble Learning) bằng cách sử dụng kết hợp mô hình mạng sinh đối kháng và học tăng cường để khắc phục hạn chế của mô hình MalGAN. Cụ thể, nhóm tác giả phát triển mô hình FeaGAN, mô hình dựa trên MalGAN, bằng cách kết hợp mô hình với mô hình học tăng cường DQEAF (Deep Q-network anti-malware Engines Attacking Framework). Ngoài ra, mô hình FeaGAN sử dụng mô hình học tổng hợp cho trình phát hiện phần mềm độc hại để cải thiện khả năng trốn tránh của các mẫu đối kháng được tạo ra. Kết quả thực nghiệm cho thấy 100% mẫu đột biến được chọn đều đảm bảo định dạng. Ngoài ra, khả năng đảm bảo khả năng thực thi của các biến thể mã độc đạt được những kết quả tiềm năng với tỷ lệ thành công đạt mức ổn định.**

Keywords— evasion attack; adversarial attack; malware mutation; generative adversarial networks; reinforcement learning; ensemble learning.

Từ khóa— tấn công trốn tránh; tấn công đối kháng; đột biến phần mềm độc hại; mạng sinh đối kháng; học tăng cường; học tổng hợp.

I. INTRODUCTION

With the rapid progress and development of information technology, computer systems play an essential and ubiquitous role in our daily lives. However, network attacks and cybersecurity threats always accompany infrastructure and technological advancements. Malware stands as one of the most potent forms of cyberattacks, enabling attackers to carry out malicious activities on computer systems, such as unauthorized theft of sensitive information, compromising information systems and demanding substantial ransom amounts. While malware proliferates across multiple operating systems, including Windows, Linux, macOS, Android, and different file formats like Portable Executable (PE), Executable and Linkable Format, Mach-O, Android Application Package,… our research focuses on malware in the Portable Executable (PE) file format within

This manuscript is received on January 16, 2023. It is commented on June 01, 2023 and is accepted on June 06, 2023 by the first reviewer. It is commented on June 01, 2023 and is accepted on June 10, 2023 by the second reviewer.

the Windows 32-bit operating system. In 2018, Symantec reported the emergence of 246,002,762 new variants of malware [1]. According to Kaspersky Lab's statistics at the end of 2020, Kaspersky detects an average of 360,000 malware samples daily, with over 90% of them being Windows PE malware [2].

To mitigate this serious issue, many researchers have applied machine learning and deep learning techniques for malware detection. These approaches provide intelligent ways to extract and classify features [3, 26]. This has also somewhat driven the development of malware detection tools (on the defensive side) as well as evasion methods (on the offensive side). However, machine learning and deep learning models are believed to be vulnerable to adversarial attacks [2], which are created by subtly perturbing suitable inputs, leading to incorrect predictions by targeted models. To generate such adversarial samples, GANs are a promising approach. Goodfellow et al. [4] introduced GANs, a framework for generating mutated samples using the idea of two competing artificial neural networks - the generator and the discriminator engaging in a minimax game to converge on an optimal solution [5]. GANs have shown potential not only in generating images, sounds, and texts but also in the realm of information security, particularly in malware, by enhancing the detection capabilities of malware detectors or simply generating adversarial samples. However, research on GANs for generating adversarial samples in the malware domain still has limitations, as most studies focus on the feature space rather than the problem space.

Meanwhile, more generalized and powerful machine learning methods have continuously emerged and developed to cope with these variant forms. Ensemble Learning is one of those choices. It is a technique that combines multiple learning algorithms to enhance the overall prediction performance [6]. This also motivates malware researchers to devise new methods to improve the evasion effectiveness of adversarial malware samples. There are two main approaches to enhance the effectiveness of adversarial samples: using multiple attack methods and attacking multiple classifiers [1]. In the first approach, multiple attack methods are employed to introduce perturbations and increase the likelihood of misclassifications by the classifier. Tramèr et al. proposed attacking a classifier with various mutations, yielding promising results in the evasion [7]. In the second approach, this method utilizes multiple classifiers for the adversarial samples to interact with as many of them as possible, aiming to enhance the evasion capability of these adversarial samples. Specifically, Liu et al. proposed improving the transferability of samples by attacking a group of combined deep-learning models instead of a single model [8].

After reviewing relevant studies, we decided to build a system to enhance the evasion rate of malicious Windows PE files by utilizing a GAN model to generate adversarial malware called FeaGAN, which is developed based on the MalGAN model proposed by Hu and Tan [9]. In FeaGAN, the malware detector employs an ensemble learning approach for training. Additionally, our system utilizes RL to incorporate the mutation vectors generated from FeaGAN into the original malicious PE software to create mutation samples. This action helps increase the evasion capability of mutated malware and ensures a higher level of execution and sustained maliciousness.

In summary, we have the following contributions:

- We build a method of generating mutated Windows malware on the problem space, i.e., generate a real mutation pattern instead of just generating adversarial feature vectors.
- We build a range of ensemble learning-based malware detectors to evaluate the effectiveness of the mutation patterns. Additionally, we subject the mutant samples to testing across various attack contexts to ensure a comprehensive and objective evaluation against adversarial samples.

- We establish a test environment to assess the preservation properties of the mutated patterns.

The remaining paper is divided into the following sections: Part II discusses related studies and motivations that underpin our proposed method. In Section III, we outline our proposed threat modeling and methodology. Part IV describes the dataset and experimental configuration utilized in our research. Section V presents the results and analysis derived from the conducted experiments. Finally, Section VI concludes the paper and explores the limitations of our work.

II. RELATED WORKS

In contrast to images, audio, or text, malicious PE (Portable Executable) files must adhere to strict formatting rules. However, guaranteeing the executable functionality of these files remains a challenge, even when the file format is preserved. Previous research has explored different approaches for generating adversarial malware samples in various problem spaces. For instance, Liu et al. proposed the Adversarial Texture Malware Perturbation Attack (ATMPA) [14], which converts malware samples into grayscale images and applies small perturbations using adversarial attack techniques. However, the generated grayscale images disrupt the original malware structure, rendering ATMPA impractical. Another study by Lucas et al. [15] introduced an adversarial attack based on binary diversification techniques, modifying binary instructions at a functional level. While these white-box attacks demonstrate effectiveness, directly applying gradient-based optimization algorithms for constructing adversarial PE malware samples is infeasible due to the intractability issue of the feature space [16].

Black-box attacks offer a more realistic approach by requiring less knowledge about the target malware detection systems. For example, Rosenberg et al. proposed BADGER [17], a set of efficient black-box query attacks that misclassify malware detection systems based on API call sequences. Hu and Tan introduced MalGAN [9], a GAN-based model that generates adversarial PE malware samples by adding unrelated API calls. Kawai et al. proposed Improved-MalGAN [18], which addresses practical issues in adversarial sample generation. Yuan et al. presented GAPGAN [19], a byte-level GAN-based attack framework that successfully evades deep-learning-based malware detection.

In the case of RL-based attacks, according to Ebrahimi et al. [20], actor-critic or DQN approaches are commonly used in adversarial RL attacks against PE malware detectors, but they have limitations in handling situations with large combinatorial state spaces. By utilizing diverse actor-critic, which has been proven to be effective in managing situations with large combinatorial state spaces, they propose an adversarial RL attack framework called AMG-VAC based on gym-malware [12]. Additionally, Fang et al. present DQEAF [10], another framework using DQN to evade PE malware detectors, similar to gym-malware in methodology but with some implementations to enhance the model's effectiveness. Labaca-Castro et al. also provide AIMED-RL [21], an automatic intelligent malware evasion framework based on RL. The main difference between AIMED-RL and other RL-based adversarial attacks is that AIMED-RL introduces a novel penalty term for the reward function to increase the diversity of generated transformation sequences while minimizing their corresponding lengths.

In terms of preserving attributes (i.e., format preservation, executability, and maliciousness), most adversarial attacks against PE malware detectors can only guarantee format preservation rather than preserving executability or maliciousness [2]. Some adversarial attack methods (e.g., ATMPA [14]) can disrupt the layout and fixed semantics of the PE format, which are essential for loading and executing PE files. Furthermore, there are various ways to interact with the feature space [9, 14, 17, 18] that do not interact with the problem space like [10, 15], [19 - 21] which may become cumbersome due to the intractable issues related to the feature space.

Moreover, to enhance the effectiveness of adversarial samples, multiple attack methods and attacking multiple classifiers have been proposed. Tramer et al. used multiple mutation methods to attack a classifier [7], while Liu et al. targeted a group of mixed deep learning models [8]. Additionally, Luca provided a framework for systematic and scalable attack methods [22]. It helps alleviate four issues (application-specific, semantic preservation, automatability, tunability) hindering the application of attacks.

Motivated by these studies, we propose a model that combines FeaGAN and DQEAF to generate mutated adversarial malware samples while preserving the format and execution capabilities of PE files. Ensemble learning-based approaches are employed for substitute black-box detectors, increasing the evasion capability of mutated samples. To ensure reliability, we also implemented the Cuckoo testing environment to assess the format and execution capabilities of the generated samples.

III. METHODOLOGY

A. Threat Models

There are three types of attack scenarios based on the attacker's level of knowledge: White-box attacks, Gray-box attacks, and Black-box attacks. In white-box attack scenarios, the attack context assumes that the attacker has complete knowledge of the training data, algorithms, and machine learning model, along with the hyperparameters used in model training. In gray-box attack scenarios, the attacker has less information, possibly being a part of the machine learning algorithm but with restricted access to the training data. In the case of black-box attacks, the attacker has no knowledge about the target machine learning system whatsoever. With this scenario, many argue that truly black-box attacks are not feasible because the attacker must have some specific information, such as the location of the target machine learning model or the output of the machine learning model.

In this study, we leverage MalGAN to build the FeaGAN model for black-box and gray-box attacks against feature-based ensemble learning-based detectors. It does not generate new malware samples but creates new feature vectors from the original features. Then, to generate actual mutated samples, we employ RL and use the feature vectors from FeaGAN for the generation process.

Our goal is to generate new feature vectors and malware patterns that can distort the predictions of the detector, thereby compromising the integrity of the CIA (Confidentiality - Integrity - Availability) model. The generated adversarial malware samples are expected to deceive ensemble learning-based models in the task of malware detection.

B. Proposed method

Our proposed method is a system created by combining a GANs model and an RL model to generate adversarial samples. An overview of the proposed system is shown in Figure 1, which includes the FeaGAN model and the RL model. In this system, FeaGAN is used to generate feature vectors for import functions and sections to reduce the True Positive rate of the detector which has been validated and experimentally demonstrated in the MalGAN model [9]. The FeaGAN model is capable of generating adversarial feature vectors, but ultimately, it operates only in the feature space and does not directly generate actual malware patterns. By using the RL model, we overcome some limitations encountered when using FeaGAN independently. The RL model helps us determine the sequence of modifications. In the case where the action selected by the agent is to add a section, import a function, or change the section name, the adversarial feature vectors generated from FeaGAN are used. The actions and operations we use are all transformations of the malware file without breaking its format.

1. FeaGAN model

Our FeaGAN model consists of three main components, including the Generator, the

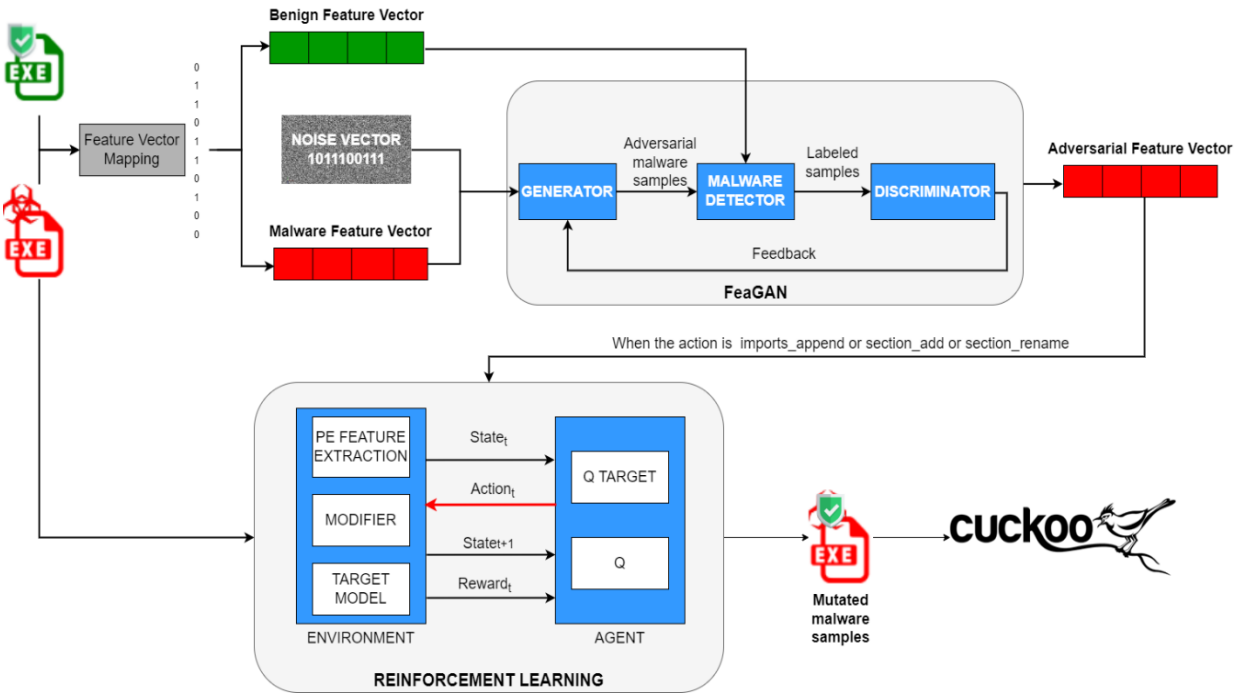

Figure 1. Overview of the Adversarial Sample Generation System

Malware Detector and the Discriminator. The model leverages the API call sequence to generate PE malware samples capable of evading the Malware Detector. FeaGAN assumes that the attacker knows the complete feature space of the target Malware Detector. It attempts to construct the Discriminator that closely resembles the target Malware Detector as much as possible. Then, it trains the Generator to minimize the malicious probability of the generated adversarial malware, which is predicted by the Discriminator, by adding some unrelated API calls to the initial malware patterns.

The Generator: Based on the learned classification rules from the Discriminator, the Generator - a multi-layer feed-forward neural network - is used to transform the malicious feature vector *m* into its adversarial counterpart by adding noise *z* as input. Each element of *m* represents the presence or absence of a specific feature, and the 10-dimensional noise is randomly generated within the range [0, 1]. The Generator consists of two hidden layers, each with 256 nodes, and the activation function used is the Leaky ReLU defined in (1). The output layer has 11041 nodes, with each node representing a feature in the dataset of malware samples. All of these nodes use the Sigmoid activation function to ensure their outputs are within the range (0, 1).

$$g(x) = \begin{cases} x, & x \geq 0 \\ \alpha x, & Otherwise \end{cases} \quad (1)$$

The Malware Detector: It serves as a third-party malware detection system, and for each input sample, it returns a label indicating whether it is malicious or benign. In the Malware Detector, we employ both single algorithms and ensemble algorithms to evaluate and compare their effectiveness. Allowing the generated adversarial malware samples to interact with multiple classifiers during the adversarial sample generation process is believed to produce more effective mutated samples that can evade malware detection better than interacting with a single classifier alone [1]. To gain a clearer understanding of this argument and the model's effectiveness, we apply both single algorithms and ensemble algorithms to compare their performance.

The Discriminator: Since the attacker has no knowledge of the detailed structure of the black-box malware detector, the Discriminator is used to emulate the behavior of the malware detector and provide gradient information for training the Generator. It is also a multi-layer feed-forward

neural network with a similar structure to the Generator. The main difference is that the output layer of the Discriminator consists of a single node, which uses the Sigmoid activation function to represent the probability of the input vector being malicious software. We address the issue of exploding gradients by constraining the output of the Generator within the range $[\epsilon,\ 1-\epsilon]$, where $\epsilon = 10^{-7}$.

2. RL model

We employ the DQEAF model [10] as a RL framework to assist FeaGAN in generating complete adversarial malware samples instead of just adversarial vectors like the MalGAN model. Assuming the input consists of rewards and environment observation states, we aim for the output to be a rational strategy for the agent to choose actions in a given situation. Additionally, each component will be described as a Markov Decision Process to apply DQEAF to this problem:

- The state s_t is a vector containing features of the PE malware.
- Given the state s_t, the deep RL agent will observe the environment state and select an action a_t from the action space *A* - a vector of actions that the agent can take. In our RL model, we implement 10 actions defined in Table 1.

TABLE 1. ACTIONS IN THE ACTION SPACE OF THE RL MODEL

Action	Description
overlay_append	Randomly add some bytes to the end of the file
imports_append	Randomly add an import function that is not in the file
section_rename	Rename a section contained in a file
section_add	Randomly add a section to the file
section_append	Randomly add some bytes to a section
upx_pack	Pack files with UPX
upx_unpack	Unpack the UPX packaged executable
remove_signature	Remove signed certificate for file
remove_debug	Remove the debug information contained in the file
break_checksum	Modify (break) header checksum option

The reward r_t is determined for each training TURN based on the label returned by the detector and the number of actions taken. MAXTURN is defined, meaning the agent will request error compensation if MAXTURN modifications have been made. The reward r_t is 0 if the label is malicious and calculated using (2) when the label is benign.

$$r_t = 20^{-(TURN-1)/MAXTURN} * 100 \quad (2)$$

The agent is trained to maximize the expected cumulative discounted reward calculated in (3).

$$R = \sum_{t=0}^{T} \gamma^{t-1} r_t \quad (3)$$

Where $\gamma \in [0,1]$ is the factor discounting future rewards.

The estimated optimal action-value function $\hat{Q}$ is expressed in (4), which estimates the expected reward when taking an action a in state s_t.

$$\hat{Q}(s_t, a_t) = Er_t + \gamma max \hat{Q}(s_{t+1}, a_{t+1}) \quad (4)$$

During the learning process, the agent optimizes the weights θ to minimize the error estimated by the loss function (5).

$$l_t(\theta_t) = \left[\begin{matrix}\left(r + \gamma max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_{t-1})\right) \\ -Q(s_t, a_t; \theta_t)\end{matrix}\right]^2 \quad (5)$$

3. Cuckoo Testing Environment

The testing environment is an isolated space from the external system, designed to capture suspected malicious applications in order to prevent these applications from performing undesired behaviors on the actual system. We have created a testing environment to validate the functionality based on Cuckoo, an open-source dynamic analysis system. We utilize the Cuckoo system to examine the format and execution of mutated malware samples.

IV. DATASET AND EXPERIMENTAL CONFIGURATION

A. Dataset

The dataset we used consists of 115,000 PE32 files extracted from [11], including 55,000 benign files collected from a Windows 7 virtual machine and 60,000 malware files collected from VirusTotal, with labels such as Adware, Trojan, Virus, Ransomware and Backdoor.

For the FeaGAN model, we used 5,000 benign files and 8,000 malware files for training. We extracted features including function imports and sections. These features are represented as binary code, where 1 indicates the presence of a feature in the sample and 0 indicates the absence. The dataset for FeaGAN is divided as follows:

- For benign samples used for training and testing the detection model, we employed an 8:2 ratio.
- For malware samples, a 6:2:2 ratio was applied. The first two portions were used for training and testing the detection model, while the remaining portion was used for training the GAN model.

In the RL model, we reused the GAN dataset, which consists of 3,200 malware samples, for training and testing the agent with a 50:50 ratio. In the target model, we used 50,000 benign files and 50,000 malware files for training and testing the target models before generating adversarial samples to deceive them. The data was split with an 8:2 ratio.

After the training process, we evaluated the mutation generation method on the remaining 2,000 malware samples to challenge the target models and assess the system's performance in the given contexts. Detailed information regarding the data split is presented in Table 2.

B. Experimental Configuration

The system was developed and tested on a virtual machine running Ubuntu 18.04 LTS operating system. The detailed hardware configuration consisted of a 16-core Intel Xeon E5-2660 CPU with a clock speed of 2.0 GHz, 30 GB of RAM, and a 300GB hard drive. The system’s source code was programmed in Python, utilizing main libraries such as PyTorch [23], Scikit-Learn [24], LIEF [25], and several other supporting libraries.

TABLE 2. OVERVIEW OF DATA DISTRIBUTION

Dataset			**Label**	
			Benign	**Malicious**
Dataset for FeaGAN	Malware Detector	Training	4000	4800
		Testing	1000	1600
	GANs	Training	0	1600
		Testing	0	1600
Dataset for RL model	Target model	Training	40000	40000
		Testing	10000	10000
	RL agent	Training	0	1600
		Testing	0	1600
Dataset for proposed system		**Testing**	**0**	**2000**

1. Malware Detectors based on Single Algorithm and Ensemble Learning

We define single algorithm-based detection engines as detectors that utilize independent algorithms without any other algorithms within them. We do not employ SVM (Support Vector Machine) and MLP (Multi-layer Perceptron) due to the high dimensionality of the dataset [13]. Ensemble learning-based detectors employ ensemble techniques such as Bagging (with DT as the estimation method); RF (Random Forest); Boosting techniques including Adaboost and GB (Gradient Boosting) (both employing DT as the estimation method); Voting and Stacking techniques. Due to the insufficient and non-GPU-supported hardware configuration, the training process of the FeaGAN model (training FeaGAN and training detector) is considerably time-consuming. Compared to single algorithm-based FeaGAN, most ensemble learning-based FeaGAN require a longer training time, ranging from 2 to 10 hours (with Stacking-based FeaGAN needing 10.5 hours), while single algorithm-based FeaGAN typically take 1.5 to 4 hours to train.

Malware Detectors in FeaGAN: utilizing single algorithms including DT (Decision Tree), LR (Logistic Regression), KNN (K-neighbors), Naive Bayes, Bernoulli. The ensemble algorithms consist of two types: homogeneous (RF, Bagging, AdaBoost, GB) and heterogeneous (Voting, Stacking). In the Voting technique, we employ all five aforementioned single algorithms as estimators and use Soft Voting for prediction. In the Stacking technique, we utilize the base estimators comprising all five single algorithms and employ LR as the final estimator. We implemented the detection engine using the Scikit-Learn library with a version equal to or greater than 0.22 when using the Stacking techniques. The performance results of the malware detectors in FeaGAN are presented in Table 3. In Table 3, the performance of most ensemble algorithms is slightly better than that of the single algorithms. The majority of models exhibit high AUC values, surpassing 91.5%. Nevertheless, there are a few models, such as Naive Bayes, that demonstrate comparatively lower performance. The Naive Bayes model exhibits a relatively low AUC score (61.137%), while maintaining a high Recall score (97.875%). This suggests that the model possesses limited overall classification capability, particularly when it comes to accurately identifying benign samples.

TABLE 3. PERFORMANCE OF THE MALWARE DETECTOR IN FEAGAN

Algorithm		**AUC**	**Accuracy**	**Precision**	**Recall**	**F1-score**
Single	Bernoulli	0.84906	0.73115	0.85839	0.67438	0.75534
	Naive Bayes	0.61137	0.69615	0.67442	0.97875	0.79857
	DT	0.91580	0.92231	0.95155	0.92063	0.93583
	LR	0.96459	0.92769	0.95315	0.92813	0.94047
	KNN	0.94977	0.90231	0.94335	0.89500	0.91854
Homogeneous Ensemble	RF	0.97978	0.93654	0.96320	0.93250	0.94760
	Bagging	0.97268	0.93115	0.96468	0.92188	0.94279
	AdaBoost	0.95289	0.88500	0.91249	0.89938	0.90589
	GB	0.96610	0.92423	0.93816	0.93875	0.93846
Heterogeneous Ensemble	Voting	0.96969	0.93000	0.94479	0.94125	0.94302
	Stacking	0.97576	0.93846	0.96038	0.93875	0.94943

TABLE 4. PERFORMANCE OF TARGET MODELS IN DQEAF

Algorithm		AUC	Accuracy	Precision	Recall	F1-score
Single	LR	0.66738	0.75660	0.89648	0.58020	0.70447
	DT	0.97195	0.97195	0.96705	0.97720	0.97210
	Naive Bayes	0.86779	0.75250	0.94236	0.53790	0.68487
	MLP	0.82203	0.82185	0.93641	0.69060	0.79494
	KNN	0.81440	0.76705	0.71321	0.89330	0.79316
Homogeneous Ensemble	RF	0.99895	0.99205	0.98897	0.99520	0.99207
	AdaBoost	0.99522	0.97330	0.95721	0.99060	0.97376
	GB	0.99877	0.98450	0.97259	0.99710	0.98469
	Bagging	0.99357	0.97870	0.97349	0.98420	0.97882
	GB (gym-malware)	0.99079	0.95200	0.96217	0.94100	0.95147
Heterogeneous Ensemble	Voting	0.96844	0.84550	0.96401	0.71780	0.82288
	Stacking_DT	0.92471	0.92355	0.91890	0.92910	0.92397
	Stacking_RF	0.99757	0.98775	0.97730	0.99870	0.98788

The target models in DQEAF: existing research works based on the gym-malware framework [10, 12], mostly employ models trained based on GB. In this paper, we deploy target models using different algorithms. We also compare the GB-based detector (target model) with the GB-based detector in gym-malware. The single algorithms are similar to the malware detection engine in FeaGAN, except for using MLP instead of Bernoulli. The homogeneous ensemble algorithms are the same as in FeaGAN.

Voting also utilizes all 5 single algorithms. For Stacking, there are 2 cases. In one case, the Stacking method uses all five single algorithms as base estimators, and the final estimator is DT. The reason for choosing DT as the final layer is that it has the best performance among all tested single algorithms. In the other case, we employ DT, RF, Adaboost, Bagging, and GB as base estimators, and RF as the final estimator because of its superior performance. The performance results of the target models in DQEAF are described in Table 4. The ensemble-based target models have fairly high AUC values when all exceed 92.4%. Except for LR model, the majority of single algorithm-based target models exhibit relatively stable AUC values, with most surpassing 81.4%. The LR model exhibited inferior performance compared to the other models when both AUC and Recall were below 67%.

In Table 4, we select representatives for each algorithm type with the best performance to implement in different contexts. Specifically, DT and KNN represent the single algorithm-based detector, RF and GB represent the homogeneous ensemble-based detector, and Stacking_RF represents the heterogeneous ensemble-based detector.

TABLE 5. RECALL EVALUATION SCORES ON MUTATED FEATURES OF THE DETECTOR IN FEAGAN

Algorithm		Recall
Single	Bernoulli	0
	Naive Bayes	0
	DT	0
	LR	0
	KNN	0.21500
Homogeneous Ensemble	RF	0
	Bagging	0.49625
	AdaBoost	0.04968
	GB	0
Heterogeneous Ensemble	Voting	0
	Stacking	0

In Table 5, the Recall scores for adversarial feature vectors decrease significantly, almost reaching 0 for most algorithms. We select five best results of adversarial features including DT, KNN, RF, GB, and Stacking to extract raw information for training the DQEAF RL model. DT and KNN represent single algorithms, while RF and GB represent Homogeneous Ensembles, and Stacking represents a Heterogeneous Ensemble.

2. FeaGAN model

The FeaGAN model is trained to generate new features for actions in the RL framework, such as imports_append, section_add, or section_rename. Both the Generator and the Discriminator in FeaGAN are multi-layer feedforward neural networks with two hidden layers of 256 nodes each. The activation function used is the Leaky ReLU defined in (1), with $\alpha = 0.01$. The output layer has 11,041 nodes, including 9,890 function imports and 1,151 sections. The output layer uses the sigmoid activation function to ensure outputs are in the range (0, 1). The parameters in the FeaGAN model are optimized using Adam with a batch size of 32 samples and a training duration of 100 epochs.

3. RL model

We implemented similar techniques to the DQEAF framework proposed by Fang et al. [10] However, instead of using 513 dimensions, we utilized 2350 dimensions, as in gym-malware [12] to obtain a more comprehensive understanding of malware. We defined 10 actions in Table I for the agent to choose from. Additionally, the RL agent was trained in a Deep Convolutional Q-network with two hidden layers of 256 and 64 nodes, respectively. The Rectified Linear Unit (ReLU) activation function was used in both layers.

The agent was trained over 600 episodes with a discount factor γ of 0.99. In each episode, the agent was allowed to perform a maximum of 80 actions on each PE file. If a reward of 10 was achieved before exhausting the 80 actions, the agent transitioned to a new episode to learn how to select actions on a new state.

C. Metrics and Contexts

1. Metrics

To demonstrate the effectiveness of the Malware Detection system, the team utilizes metrics such as Area Under Curve (AUC), accuracy, precision, recall, and F1-score. These metrics are computed based on the attributes in the Confusion Matrix, which consists of True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). In the context of information security, assuming that positive samples represent malicious samples and negative samples represent benign samples, the significance of these attributes is as follows:

TP: the number of correctly classified malicious samples.

TN: the number of correctly classified benign samples.

FN: the number of malicious samples misclassified as benign.

FP: the number of benign samples misclassified as malicious.

2. Contexts

We deploy our system in two contexts to evaluate and compare the performance of the

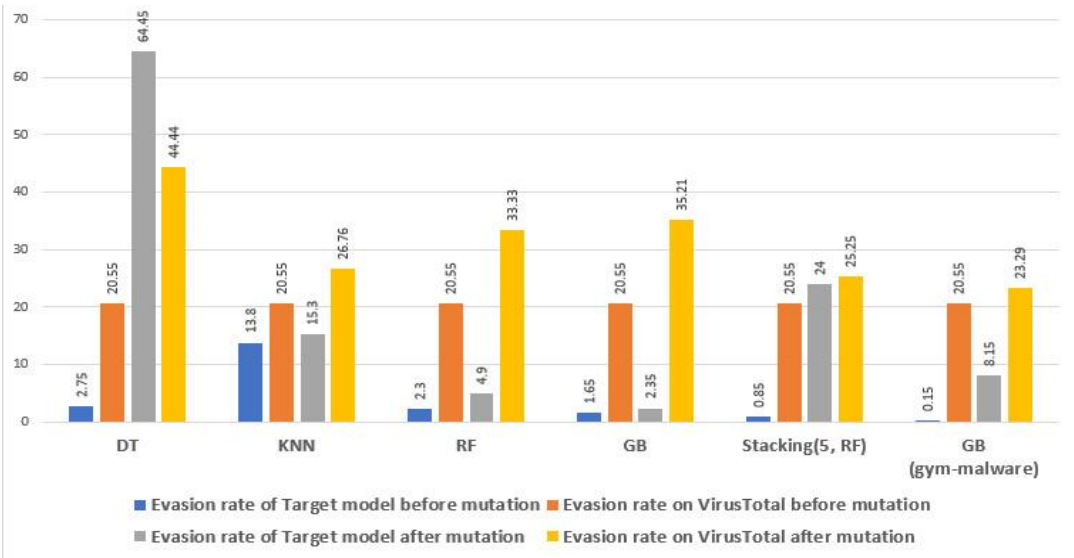

Figure 2. The chart illustrates the evasion rate of mutated samples

ensemble-based detector and the single algorithm-based detector, specifically as follows:

Targeted Attack: FeaGAN and the target model use the same machine learning algorithm in the Detector component. This type of attack includes targets that are the single algorithm and the ensemble algorithm sequentially.

Transfer Attack: FeaGAN and the target model utilize different algorithms. Our objective is to assess whether the malicious samples generated by the ensemble learning algorithm have a high evasion capability against target detectors based on both individual algorithms and other ensemble learning algorithms.

V. Experimental results

TABLE 6. DETECTION CAPABILITY OF TARGET MODELS BEFORE MUTATION FOR MALWARE SAMPLES

Target model	Before the mutation		
	Detect/Total	Evasion rate	Average score on VirusTotal
DT	1945/2000	2.75%	58/73
KNN	1725/2000	13.80%	
RF	1954/2000	2.30%	
GB	1967/2000	1.65%	
Stacking	1983/2000	0.85%	
GB (gym-malware)	1997/2000	0.15%	

Before evaluating the mutated samples, we assessed the performance of the target models on 2000 original malware samples, as shown in Table 6. The results indicate that the target models have a good ability to detect malware.

A. Targeted attack context

In this targeted attack context, our system has successfully increased the evasion rate of mutated samples against the target model. The evasion rate of mutated samples against classifiers in VirusTotal improves after mutation. Detailed results are shown in Table 7 and Figure 2.

TABLE 7. EVASION CAPABILITIES OF MUTATED SAMPLES AGAINST THE TARGET MODELS AND VIRUSTOTAL

Target model	After the mutation		
	Detect/Total	Evasion rate	Average score on VirusTotal
DT	711/2000	64.45%	40/72
KNN	1694/2000	15.30%	52/71
RF	1902/2000	4.90%	48/72
GB	1953/2000	2.35%	46/71
Stacking	1520/2000	24.00%	53/71
GB (gym-malware)	1837/2000	8.15%	56/73

TABLE 8. DETECTION CAPABILITY OF MUTATED MALWARE SAMPLES DURING TRANSFER ATTACK

Target model/Mutated pattern	DT	KNN	RF	GB	Stacking	GB (gym-malware)
DT	711/2000	1889/2000	1905/2000	1910/2000	912/2000	1905/2000
KNN	1614/2000	1694/2000	1692/2000	1697/2000	1559/2000	1692/2000
RF	1391/2000	1902/2000	1902/2000	1908/2000	1041/2000	1902/2000
GB	1915/2000	1957/2000	1957/2000	1953/2000	1961/2000	1957/2000
Stacking	1915/2000	1965/2000	1964/2000	1961/2000	1520/2000	1964/2000
GB (gym-malware)	1965/2000	1836/2000	1837/2000	1842/2000	1954/2000	1837/2000

B. Transfer attack

The results are presented in Table 8 and Figure 3. In Figure 3, to distinguish between the target model and the mutated samples, we added the symbol "M" at the end of each type of mutated sample. For example, the samples mutated by the DT algorithm in the GAN architecture are denoted as DT-M. Such DT-M samples can be tested using all machine learning algorithms, including DT, KNN, RF, GB, Stacking, and GB (gym-malware).

Regardless of the algorithm used, whether it is a single algorithm or an ensemble algorithm as the target detector, the evasion rate of the mutated samples increases against the target models. The system utilizing DT and Stacking to generate mutations showcases two models with the best performance, producing mutated samples that are more effective in evading various malware classifiers. Particularly, the Stacking ensemble algorithm yields more comprehensive results, with high evasion rates for both single algorithm-based and ensemble-based target models. This demonstrates the effectiveness of researching the generation and collection of mutated malwares as part of the data preparation strategy for real-world malware detection.

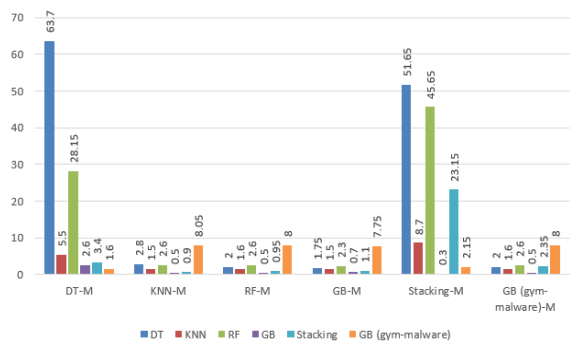

Figure 3. The chart illustrates the increased evasion rate of mutated malware samples

C. Preservation of Format and Executable functionality

To assess whether the transformation actions alter the format and execution capabilities of the malware samples, we utilized the Cuckoo testing environment on 100 randomly selected mutated samples. The results revealed that 100% of the generated adversarial malware samples ensured the PE32 file format. During the mutation process, we only manipulated components considered unrelated, without impacting the malicious code itself, which could explain why the mutated samples still preserved the format. At least 54% of the mutated malicious samples exhibited at least one dynamic feature. These dynamic features could include file system manipulation, registry key manipulation, unusual DNS queries/URL links, virtual machine detection/debugger checks,…

VI. CONCLUSION

Ideas and research related to malware are constantly being developed and achieving breakthroughs. Malicious samples continuously mutate to evade detection systems easily. Research in this field provides a deeper understanding of the methodology and provides appropriate prevention strategies. In this study, the combination of three ML methods: GANs, RL, and ensemble learning, has been

successfully employed to generate adversarial malware samples that can bypass ensemble learning-based detectors. Classifiers using ensemble learning algorithms generally outperform classifiers using single algorithms, and the adversarial malware samples they generate are highly resistant. Thus, generating adversarial malware samples by interacting with ensemble learning-based detectors holds promising results. This research has also achieved success in generating practical adversarial samples that preserve specific formats and execution capabilities. Nevertheless, our system remains constrained by several limitations such as the absence of performance comparisons with other RL models to evaluate the performance of mutated samples under various methods. Furthermore, we have yet to implement execution and malicious function testing of mutants within the RL model environment. In conclusion, this research contributes to developing effective methods for generating adversarial samples in the context of malware detection.

REFERENCE

[1] D. Li and Q. Li, "Adversarial Deep Ensemble: Evasion Attacks and Defenses for Malware Detection," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 3886–3900, 2020, doi: 10.1109/TIFS.2020.3003571.

[2] X. Ling *et al.*, "Adversarial Attacks against Windows PE Malware Detection: A Survey of the State-of-the-Art." arXiv, Dec. 19, 2022. Accessed: Jan. 03, 2023. [Online]. Available: http://arxiv.org/abs/2112.12310

[3] Toan, N. N. ., Dung, L. T., & Thang, D. Q. (2022). Static Feature Selection for IoT Malware Detection. Journal of Science and Technology on Information Security, 1(15), 74-84. https://doi.org/10.54654/isj.v1i15.844.

[4] I. Goodfellow *et al.*, "Generative Adversarial Nets".

[5] H. Lee, S. Han, and J. Lee, "Generative Adversarial Trainer: Defense to Adversarial Perturbations with GAN." arXiv, May 26, 2017. Accessed: Jan. 03, 2023. [Online]. Available: http://arxiv.org/abs/1705.03387

[6] R. Damaševičius, A. Venčkauskas, J. Toldinas, and Š. Grigaliūnas, "Ensemble-Based Classification Using Neural Networks and Machine Learning Models for Windows PE Malware Detection," *Electronics*, vol. 10, no. 4, p. 485, Feb. 2021, doi: 10.3390/electronics10040485.

[7] F. Tramèr and D. Boneh, "Adversarial Training and Robustness for Multiple Perturbations." arXiv, Oct. 17, 2019. Accessed: Jan. 03, 2023. [Online]. Available: http://arxiv.org/abs/1904.13000

[8] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into Transferable Adversarial Examples and Black-box Attacks." arXiv, Feb. 07, 2017. Accessed: Jan. 03, 2023. [Online]. Available: http://arxiv.org/abs/1611.02770

[9] W. Hu and Y. Tan, "Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN." arXiv, Feb. 20, 2017. Accessed: Jan. 03, 2023. [Online]. Available: http://arxiv.org/abs/1702.05983

[10] Z. Fang, J. Wang, B. Li, S. Wu, Y. Zhou, and H. Huang, "Evading Anti-Malware Engines With Deep Reinforcement Learning," *IEEE Access*, vol. 7, pp. 48867–48879, 2019, doi: 10.1109/ACCESS.2019.2908033.

[11] Michae Lester, "PE Malware Machine Learning Dataset," 2018. https://practicalsecurityanalytics.com/pe-malware-machine-learning-dataset/

[12] H. S. Anderson, A. Kharkar, B. Filar, D. Evans, and P. Roth, "Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning." arXiv, Jan. 30, 2018. Accessed: Jan. 03, 2023. [Online]. Available: http://arxiv.org/abs/1801.08917

[13] P. I. Wójcik and M. Kurdziel, "Training neural networks on high-dimensional data using random projection," *Pattern Anal. Appl.*, vol. 22, no. 3, pp. 1221–1231, Aug. 2019, doi: 10.1007/s10044-018-0697-0.

[14] X. Liu, J. Zhang, Y. Lin, and H. Li, "ATMPA: attacking machine learning-based malware visualization detection methods via adversarial examples," in *Proceedings of the International Symposium on Quality of Service*, Phoenix Arizona, Jun. 2019, pp. 1–10. doi: 10.1145/3326285.3329073.

[15] K. Lucas, M. Sharif, L. Bauer, M. K. Reiter, and S. Shintre, "Malware Makeover: Breaking ML-based Static Analysis by Modifying Executable Bytes," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, Virtual Event Hong Kong, May 2021, pp. 744–758. doi: 10.1145/3433210.3453086.

[16] Erwin Quiring, Alwin Maier, and Konrad Rieck, "Misleading authorship attribution of source code using adversarial learning," 2019, pp. 479–496.

[17] I. Rosenberg, A. Shabtai, Y. Elovici, and L. Rokach, "Query-Efficient Black-Box Attack Against Sequence-Based Malware Classifiers," in *Annual Computer Security Applications Conference*, Austin USA, Dec. 2020, pp. 611–626. doi: 10.1145/3427228.3427230.

[18] M. Kawai, K. Ota, and M. Dong, "Improved MalGAN: Avoiding Malware Detector by Leaning Cleanware Features," in *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, Okinawa, Japan, Feb. 2019, pp. 040–045. doi: 10.1109/ICAIIC.2019.8669079.

[19] J. Yuan, S. Zhou, L. Lin, F. Wang, and J. Cui, "Black-box Adversarial Attacks Against Deep Learning Based Malware Binaries Detection with GAN," *Santiago Compost.*, 2020.

[20] M. Ebrahimi, J. Pacheco, W. Li, J. L. Hu, and H. Chen, "Binary Black-Box Attacks Against Static Malware Detectors with Reinforcement Learning in Discrete Action Spaces," in *2021 IEEE Security and Privacy Workshops (SPW)*, San Francisco, CA, USA, May 2021, pp. 85–91. doi: 10.1109/SPW53761.2021.00021.

[21] R. Labaca-Castro, S. Franz, and G. D. Rodosek, "AIMED-RL: Exploring Adversarial Malware Examples with Reinforcement Learning," in *Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track*, vol. 12978, Y. Dong, N. Kourtellis, B. Hammer, and J. A. Lozano, Eds. Cham: Springer International Publishing, 2021, pp. 37–52. doi: 10.1007/978-3-030-86514-6_3.

[22] L. Demetrio, B. Biggio, and F. Roli, "Practical Attacks on Machine Learning: A Case Study on Adversarial Windows Malware," *IEEE Secur. Priv.*, vol. 20, no. 5, pp. 77–85, Sep. 2022, doi: 10.1109/MSEC.2022.3182356.

[23] Imambi, Sagar, Kolla Bhanu Prakash, and G. R. Kanagachidambaresan. "PyTorch." Programming with TensorFlow: Solution for Edge Computing Applications (2021): 87-104.

[24] Kramer, Oliver, and Oliver Kramer. "Scikit-learn." *Machine learning for evolution strategies* (2016): 45-53.

[25] Thomas, Romain. "Lief-library to instrument executable formats." Retrieved February 22 (2017): 2022.

[26] Ho, H. D., & Ho, H. V. (2020). Technical research of detection algorithmically generated malicious domain names using machine learning methods. Journal of Science and Technology on Information Security, 7(1), 37-43. https://doi.org/10.54654/isj.v7i1.54.

ABOUT THE AUTHOR

Pham Van Hau

Workplace: Information Security Lab, University of Information Technology, Vietnam National University Ho Chi Minh City.

Email: haupv@uit.edu.vn

Education: Received the Bachelor of degree in computer science from the University of Natural Sciences of Ho Chi Minh City in 1998. He pursued his master's degree in Computer Science from the Institut de la Francophonie pour l'Informatique (IFI) in Vietnam from 2002 to 2004. He then received his PhD degree on network security in 2009.

Recent research direction: Network security; System security; Mobile security; Cloud computing.

Cơ quan làm việc: Phòng An toàn thông tin, Trường Đại học Công nghệ Thông tin, Đại học Quốc gia Tp. Hồ Chí Minh

Email: haupv@uit.edu.vn

Quá trình đào tạo: Nhận bằng Cử nhân khoa học máy tính của Đại học Khoa học Tự nhiên Thành phố Hồ Chí Minh năm 1998. Anh theo học Thạc sĩ về Khoa học Máy tính tại Institut de la Francophonie pour l'Informatique (IFI) tại Việt Nam từ năm 2002 đến 2004. Nhận bằng Tiến sĩ về an ninh mạng vào năm 2009.

Hướng nghiên cứu hiện nay: An ninh mạng; Bảo mật hệ thống; Bảo mật di động; Điện toán đám mây.

To Trong Nghia

Workplace: Information Security Lab, University of Information Technology, Vietnam National University Ho Chi Minh City.

Email: nghiatt@uit.edu.vn

Education: Received the Bachelor of Engineering degree in Information Security from the University of Information Technology, Vietnam National University Ho Chi Minh City in 2022.

Recent research direction: Malware analysis; Software-defined networking; Blockchain; Machine learning.

Cơ quan làm việc: Phòng An toàn thông tin, Trường Đại học Công nghệ Thông tin, Đại học Quốc gia Tp. Hồ Chí Minh

Email: nghiatt@uit.edu.vn

Quá trình đào tạo: Nhận bằng Cử nhân Kỹ thuật chuyên ngành An toàn thông tin của Trường Đại học Công nghệ Thông tin, Đại học Quốc gia Thành phố Hồ Chí Minh năm 2022.

Hướng nghiên cứu hiện nay: Phân tích phần mềm độc hại; Mạng được xác định bằng phần mềm; Chuỗi khối; Học máy.

Phan The Duy

Workplace: Information Security Lab, University of Information Technology, Vietnam National University Ho Chi Minh City.

Email: duypt@uit.edu.vn

Education: Received the Bachelor of Engineering and MSc degrees in Software Engineering and Information Technology, from the University of Information Technology (UIT), Vietnam National University Ho Chi Minh City in 2013 and 2016 respectively.

Recent research direction: Vulnerabilities detection; Software-defined networking security; Malware and Cyber threat detection; Digital forensics; Machine learning and Adversarial machine learning; Private machine learning; Blockchain.

Cơ quan làm việc: Phòng An toàn thông tin, Trường Đại học Công nghệ Thông tin, Đại học Quốc gia Tp. Hồ Chí Minh

Email: duypt@uit.edu.vn

Quá trình đào tạo: Nhận bằng Cử nhân Kỹ thuật và Thạc sĩ Kỹ thuật phần mềm và Công nghệ thông tin của Trường Đại học Công nghệ Thông tin (UIT), Đại học Quốc gia Thành phố Hồ Chí Minh năm 2013 và 2016.

Hướng nghiên cứu hiện nay: Phát hiện lỗ hổng; Bảo mật mạng được xác định bằng phần mềm; Phát hiện phần mềm độc hại và mối đe dọa mạng; Pháp y kỹ thuật số; Học máy và học máy đối nghịch; Học máy riêng; Chuỗi khối.