MalDenoise: Enhancing Robustness of API-Based Malware Detection Against Adversarial Attacks

Xiaohui Chen[1], Xin Wang[2], Zuhui Yue[1], Zheng Li[1], Peipei Liu[3,4*], Hongsong Zhu[3,4]
[1]*China Mobile Research Institute, Beijing, China*
[2]*Beijing Guancheng Technology Co., Ltd., Beijing, China*
[3]*Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China*
[4]*School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China*
xiaohuichen1116@gmail.com, {yuezuhui, lizheng}@chinamobile.com,
wangxin@viewintech.com, peipliu@yeah.net, zhuhongsong@iie.ac.cn

***Abstract*—API call sequence-based machine learning (ML) models have shown promise in detecting malware with high accuracy. However, the vulnerability of ML models, particularly deep neural networks, to adversarial attacks remains a pressing concern. Unfortunately, existing research efforts have yet to provide a comprehensive defense against various types of adversarial attacks targeted at malware models relying on API call sequences. In this paper, we propose an innovative defense method, named MalDenoise, aiming to fortify the robustness of the API call sequence-based malware detection model against adversarial attacks. It employs a denoising model to filter and denoise the input data for the detection model, eliminating extraneous or irrelevant APIs while accentuating critical components. Furthermore, we have evaluated the performance of MalDenoise against four various adversarial attacks. The results conclusively showcase the superiority of MalDenoise over existing defense methods, attesting to its heightened effectiveness in safeguarding against adversarial threats.**

***Index Terms*—robustness detection, adversarial malware example, API call sequence, malware detection model**

I. Introduction

The pivotal cybersecurity task of ascertaining the maliciousness of a software is presently achieved through a sophisticated amalgamation of automated analytical methodologies. The dynamic analysis of software represents a potent and reliable approach for discerning its malicious intent. In particular, deep neural network (DNN) models that leverage dynamic analysis features, such as API call sequences [1]–[4], exhibit great promise for enhancing the efficiency of malware detection. However, a significant drawback looms in the form of the vulnerability of DNN models to adversarial attacks [5]–[9]. Adversaries possess the ability to meticulously craft adversarial examples, specifically designed to induce erroneous predictions in the target models. This strategic manipulation effectively results in the misclassification of malware as benign, highlighting a pressing concern that necessitates comprehensive solutions.

In response to the imperative of defending adversarial attacks, a plethora of efforts has been directed towards enhancing the resilience of deep neural networks [10]–[15], including adversarial training, ensemble model, defensive distillation, and logits augmentation. Nonetheless, some studies argue that newly devised attacks are tailored to exploit the specific defense mechanisms implemented, thus rendering DNN models vulnerable to these intricately crafted evasion tactics. This underscores the need to not only enhance the robustness of models but also to fortify their capabilities for detecting adversarial attacks. Regrettably, investigations reveal that these approaches remain less effective than desired in detecting various types of adversarial attacks.

A. Challenges

This paper places heightened emphasis on the defense capabilities of safety-critical systems, specifically focusing on malware detection models based on API call sequences, in the face of various adversarial attacks. There exist several challenges as follows:

Mapping perturbation to problem space. The domain of malware detection hinges on dynamic analysis features, specifically API call sequences, which constitute discrete data. Consequently, many existing defense mechanisms against adversarial attacks, such as those rooted in fast gradient sign method (FGSM) and continuous-value perturbations in feature space, are inapplicable to discrete features. The challenge lies in the inability to effectively map perturbed features back to their corresponding real-world values. E.g., suppose that the API *CreateFile* is represented as 3 in the API dictionary. The addition of a perturbation of 0.1 would result in 3.1, which does not map to any valid API call in the dictionary.

Constraints of dynamic feature perturbations. Existing research has predominantly focused on defending against adversarial attacks targeting malware detection models based on static features (like raw binary), such as data-augmentation based adversarial training [10], in-place randomization transformation [16], conserved features [17], and so on. Ensuring the integrity of binary file functionality while successfully mapping feature space to the problem space imposes significantly greater constraints when constructing adversarial examples for models based on dynamic features. Unlike static features, which can accommodate relatively fixed insertions or modifications in non-functional segments or data, dynamic features can only accommodate the insertion of no-op API calls

* Peipei Liu (peipliu@yeah.net) is the corresponding author.

in arbitrary positions. Effectively defending against adversarial attacks that involve arbitrary insertions of perturbations in dynamic features poses a substantial challenge.

Generalization of defense capabilities. Typical adversarial training involves initially subjecting the target malware detection model to craft specific adversarial examples, and then augmenting them to training dataset for retraining the defense model. However, such defense mechanisms often fall short in effectively guarding against novel adversarial attacks. They are inherently constrained by the scope and knowledge contained within the set of adversarial examples. Furthermore, the generation of real adversarial malware entails expending attack resources, incurring additional costs in the process.

B. Our Solution

In this work, we propose MalDenoise, a defense method designed to enhance the robustness of the API call-based malware detection model and fortify it against diverse adversarial attacks. It uses denoising method to denoise the input API call sequence. The task unique to the discrete nature of API calls as opposed to the continuous data typically encountered in image recognition. Traditional denoising methods are unsuitable for this context. Consequently, we introduce a denoising auto-encoder (DAE) [18] based model. During the training of the denoising model, we incorporate an attention-based API ranking method [19] to introduce "API noise" to the original dataset, emulating the perturbations encountered in real adversarial attacks. The trained denoising model becomes adept at filtering the input API call sequence. It successfully removes noise, extracts essential API call fragments, and discards irrelevant or disruptive API calls that may adversely impact the model's decision-making process.

We have implemented MalDenoise and assessed its performance against four various types of adversarial attacks, i.e., greedy-multi attack, greedy-one attack, genetic attack, and SeqGAN benign attack. To align more closely with practical application scenarios, the evaluation of defenses in this paper involves adversarial attacks conducted in a black-box setup. The results indicate that MalDenoise outperforms other state-of-the-art defense methods. Specifically, it reaches up to 91.26%, 95.69%, 98.73%, and 91.53% respectively in terms of adversarial recall on the test dataset.

II. Methodology

A. Overview

The overview architecture of MalDenoise is illustrated in Fig. 1. Our approach begins with the denoising model training, simulating the introduction of perturbations in adversarial attacks by incorporating perturbations into the original API call sequences. This process results in the creation of noisy-original API call sequence pairs, serving as input-target pairs for the auto-encoder model. Through training, the denoising model learns to generate sequences that closely resemble the original API call sequences. Notably, due to the inherent variability in the lengths of the noisy-original API call sequence pairs, the sequences after the addition of noise consistently surpass the length of the original ones. To address this disparity in sequence lengths, we utilize length reduction to ensure the denoising model generates API sequences that align with the anticipated length, optimizing the denoising process.

Subsequently, the denoising model processes the newly input API call sequences, effectively filtering and generating denoised API call sequences. Building upon the foundations laid by denoising model, we can systematically train and generate a resilient malware detection model, aptly named the MalDenoise model. This model exhibits robustness and resistance to interference, making it adept at handling unknown samples. Upon inputting various samples into the MalDenoise model—whether they be adversarial malware, new malware variants, or normal benign software—the model provides corresponding prediction results (where malicious is 1 and 0 for benign ones). Attention is paid to the ability to detect adversarial malware during the prediction process.

B. Denoising model

In the encoder-decoder system, an encoder and a decoder are composed to process sequence pairs (i.e., source and target sequences). Specifically, an encoder neural network is tasked with encoding a source sentence into a vector representation. Subsequently, a decoder generates an output sequence vector based on the encoded vector [20]. The encoder and decoder are jointly trained to maximize the probability that the sequence generated by the decoder correctly adapted the target sequence.

Denoising auto-encoders (DAEs) [18], founded on encoder-decoder systems, have found notable success in the domain of natural language processing (NLP). They have been instrumental in applying sentence compression [21], training unsupervised translation models [22], and facilitating natural language generation within specific domains [23]. In these applications, noise is deliberately introduced into the data through random word deletions, word swaps, or shuffling.

The denoising auto-encoders model inputs the sequence after adding noise into the encoder for learning, and its output further advances the decoder to reconstruct the new sequence, so that the sequence generated by the decoder is gradually close to the target sequence. This reduces the degree to which noise influences the reconstructed sequence, decreasing perturbation performance in the adversarial example. In this paper, we adapt the DAE model to make it more suitable for noise reduction scenarios in adversarial malware.

The architecture of our proposed denoising model is shown in Fig. 2. The structure is modified on the basis of an attentional encoder-decoder [21] to apply the denoising auto-encoding paradigm to API call sentence denoising. The encoder consists of multiple layers bi-directional recurrent neural network (Bi-RNN), the decoder applies multiple RNN layers and an attention layer, and the loss function employs the negative-log likelihood. The output and hidden of the encoder are fed into the decoder, and the length reduction is set in the decoder to limit the length of the reconstructed sentence.

We take the original sequence and the noisy sequence as a sequence pair, where the noisy sequence is used as the

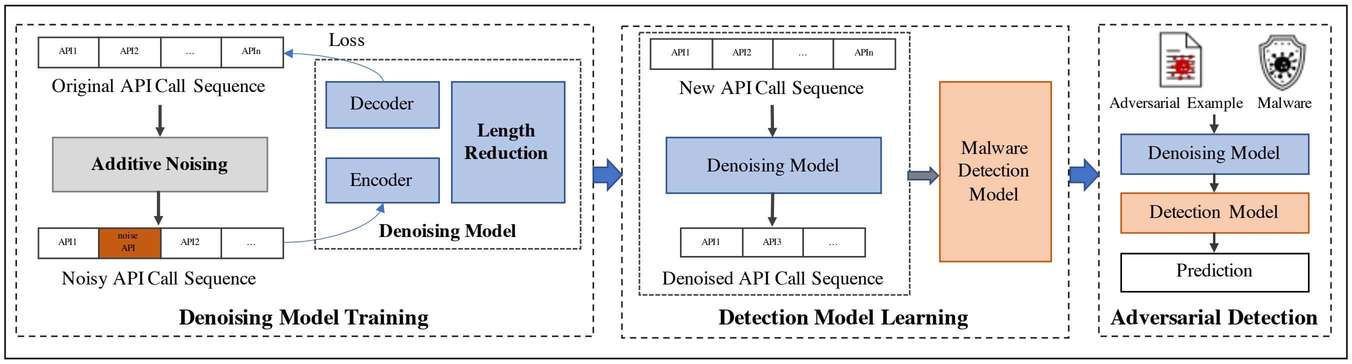


Fig. 1. Overview of MalDenoise.

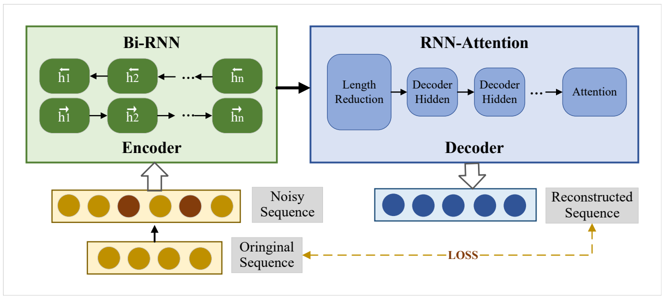


Fig. 2. Architecture of denoising model.

input to the encoder, so that the sequence reconstructed by the decoder is constantly close to the original sequence. In this way, the denoising model is motivated to learn in the direction of removing excess noise from the sequence. Additionally, the imposition of length constraints on the output sequences compels the model to distill salient information from the input. Collectively, all components of the denoising model undergo joint training with the overarching goal of maximizing the log-probability of producing accurate denoised API sequences.

C. Additive noising

Unlike the way DAE models add noise in the NLP (like Shen et al. [24]), we add noise by inserting no-op APIs to API call sequences. This is to imitate the capability of adversarial attack against API call sequences based malware detection models in real scenarios. Since the way of inserting APIs does not affect the functional structure of the binary, while the way of adding noise such as deleting, swapping, and replacing may cause the binary to be inoperable.

Motivated by the attention-based API ranking method proposed by Chen et al. [19], we apply the ranking method to emulate the process of introducing perturbations in real-world scenarios. Specifically, given an original sequence, we first use the pre-trained hierarchical attention networks to evaluate and rank the importance score of each API call. Then, we randomly select the length of the insertion noise within a limit range. Next, we insert randomly selected the noise API from the API dictionary (i.e., 97 API types in our dataset) in order of the API call importance scores. Finally, a linear insertion of the noise is performed until the added noise reaches the specified length. The denoising model must exclude noisy APIs and thus learn to output shorter and undisturbed sequences of API calls upon training process.

D. Length reduction

Since noise is inserted into the original sequence during the training process, the sequence lengths before and after adding noise are not consistent. Therefore, we use the length countdown method proposed by Thibault et al. [21] to make the reconstructed sequence and the target sequence both equal in length. The length countdown is the extension of the decoder's input to include a countdown:

$$h_t = RNN(h_{t-1}, x_t, L - t), \tag{1}$$

where h_{t-1} is the hidden state at the previous step and x_t denotes an external input (typically in the form of an embedding of the token that was decoded in the previous step). L denotes the desired length of the output sequence. When the length of the sequence generated by the decoder reaches the desired length L, then $L-t$ is 0 and negative values thereafter.

Furthermore, Cho et al. [25] demonstrated that the effectiveness of a basic encoder–decoder framework diminishes considerably as the length of the input sentence grows. However, unlike short sentences in the field of NLP, API call sequences consist of thousands of API calls, which may affect the efficiency as well as the accuracy of the encoder-decoder generated sequences. Therefore, we first slice the API sequences and perform denoising learning on the sliced fragments, and then recover the generated sequences in the order of the slices after noise reduction. In other words, the generated sequence fragments are spliced and recovered in order. In this way, we can generate denoising sequences for very long API sequences without affecting the efficiency of the encoder-decoder model.

III. Evaluation

A. Experimental Setup

1) Dataset: Our dataset has a total of 50K execution traces of Windows PE files, including 32K in the training set, 8K in the validation set, and 10K in the test set. Balanced distribution of benign and malicious samples is ensured within the training and validation sets, primarily utilized for training the malware detection model. And the test set only contains malicious samples. Adversarial attacks generate adversarial malware examples within the test set, enabling the evaluation of the malware detection model's resilience against adversarial attacks. Crucially, these sets are mutually exclusive, with no sample overlap. The dataset used by MalDenoise is publicly accessible on GitHub [26], provided by a third party.

2) Baseline model: We employ a classic bi-directional long short-term memory model (BiLSTM) for malware detection to serve as the baseline model. The API length is limited to 1,000, the size of API embedding is 300, the batch size is 64, and the dropout is 0.6.

3) Settings: The encoder and decoder utilize BiLSTM with a hidden size of 512 and three layers. Linear is applied for the decoder output. The default slice length for the API call sequence is set to 200, and the proportion of additive noising ranges randomly from 20% to 50% of the sequence length. In the denoising process, the desired sequence length is set by default to 50% of the input sequence.

4) Metrics: We employ four metrics: i) recall – the ratio of successful detections to the total number of malware for the detector, ii) attack success rate (ASR) – the ratio of successful attacks to the total number of attacks for the adversarial attack, iii) number of adversarial example (NAE) – the count of adversarial examples generated by adversarial attacks and iv) adversarial recall (AR) - the ratio of successful detections to the total number of adversarial examples for the detector, for experimental evaluation. Among these metrics, the adversarial recall is utilized to assess the robustness of the malware detection model in detecting adversarial examples.

B. Adversarial Examples Generation

To assess the resilience of a malware detection model against adversarial examples, it is essential to subject the model to various adversarial attacks, generating diverse adversarial examples. We assume adversarial attacks in a black-box setting to align more closely with real-world scenarios. Three types of adversarial attacks are conducted: greedy attack, genetic attack, and SeqGAN benign attack.

Greedy Attack. We adapt the greedy attack proposed by Yang et al [27] for adversarial attacks against API call sequencces based malware detection models. Notably, we exclusively use insertion perturbations to avoid potential impacts on the real execution and functional integrity of the program, in contrast to the replacement perturbation in the original greedy attack. Additionally, based on the number of perturbations inserted at each position, the greedy attack is categorized into two types: the multiple perturbation (greedy-multi attack) and the single perturbation (greedy-one attack).

Genetic Attack. Inspired by the attack proposed by Alzantot et al. [28], we design an adversarial attack based on genetic algorithms, called genetic attack. It initiates by randomly ordering all positions in the API call sequence, guiding subsequent iterations. Each iteration involves the random selection

TABLE I
THE PERFORMANCE OF VARIOUS TYPES OF ADVERSARIAL ATTACKS AGAINST THE MALWARE DETECTOR

Attack Method	**Metric**	**Dataset Type**	
		Valid	**Test**
No Attack	Recall(%)	99.25	99.95
Greedy-Multi Attack [27]	ASR(%)	91.76	90.37
	NAE	3,643	9,032
Greedy-One Attack [27]	ASR(%)	90.35	88.52
	NAE	3,587	8,848
Genetic Attack [28]	ASR(%)	79.85	73.21
	NAE	3,170	7,317
SeqGAN Benign Attack [29]	ASR(%)	99.92	99.98
	NAE	3,967	9,993

of a population of candidate API perturbations (set at 50). The candidate API with the most decreasing predicted probability is then selected for perturbation insertion.

SeqGAN Benign Attack. Building upon Rosenberg et al.'s [29] adversarial attack rooted in SeqGAN benign perturbation, we present an enhanced version termed SeqGAN benign attack. It begins by employing the SeqGAN model to create diverse virtual benign API call sequences based on real benign samples. Multiple benign fragments, constituted by n consecutive APIs, are then randomly chosen for perturbation. We use SeqGAN model to generate 10,880 virtual benign samples for perturbation. The default length of the benign segment is set to 3, denoted by $n = 3$.

Table I shows the attack results of various adversarial attacks against the baseline BiLSTM model. The results underscore a notable attack success rate (ASR) for the four black-box adversarial attacks, surpassing 70% in both the validation set and the test set. This demonstrates the efficacy of the black-box adversarial attack, particularly in its capacity to compromise malware detection models.

C. Comparison with State-of-the-Arts

To assess the defense performance of MalDenoise, we execute a comprehensive experimental evaluation, comparing MalDenoise against five state-of-the-art defense methods, i.e., adversarial training, greedy attack based adversarial training, RNN ensemble, sequence squeezing, and knowledge distillation. Our primary focus lies in evaluating the adversarial recall (AR) of these defense methods on the validation set and test set, particularly concerning adversarial examples generated by four adversarial attacks, i.e., greedy-multi attack, greedy-one attack, genetic attack, SeqGAN benign attack.

Adversarial Training. In adapting Wong et al.'s defense method [30], we employ it on the API embedding of the malware detection model. Specifically, we apply the PGD and FGSM algorithms in the training process of the malware detection model, which are called AdvTrain-PGD and AdvTrain-FGSM respectively. Besides, we set *epsilon*, *alpha*, and *attack_iters* to 0.1, 0.1, and 5, respectively.

Greedy Attack Based Adversarial Training. We incorporate the defense method introduced by Lucas et al. [10] into

TABLE II
DEFENSE PERFORMANCE COMPARISON WITH THE STATE-OF-THE-ARTS

Defense Method	Recall(%), Detector		AR(%), Greedy--Multi Attack		AR(%), Greedy--One Attack		AR(%), Genetic Attack		AR(%), SeqGAN Benign Attack	
	Valid	Test	Valid	Test	Valid	Test	Valid	Test	Valid	Test
BiLSTM	**99.25**	**99.95**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AdvTrain-PGD [30]	97.70	99.84	57.45	48.68	64.01	53.73	62.15	65.94	79.78	81.64
AdvTrain-FGSM [30]	98.10	99.90	63.35	55.71	74.13	70.56	69.05	75.02	80.84	83.34
AdvTGreedy [10]	97.35	99.77	74.88	78.20	75.75	85.49	53.91	69.84	68.72	73.64
RNN Ensemble [31]	97.67	99.88	87.57	89.07	89.09	88.99	94.23	97.40	83.56	85.16
Sequence Squeezing [31]	83.28	86.22	67.80	58.23	66.38	59.20	79.18	75.86	63.42	59.02
Knowledge Distillation [32]	98.02	99.90	78.42	69.66	80.65	72.58	85.52	87.44	81.50	79.48
MalDenoise	97.06	99.70	**90.39**	**91.26**	**94.37**	**95.69**	**96.25**	**98.73**	**85.13**	**91.53**

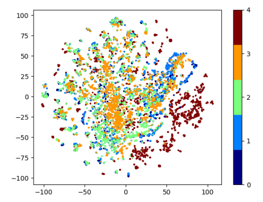
the malware detection model based on API call sequences, naming it AdvTGreedy. In this approach, each iteration of the greedy attack involves randomly selecting an API call in the sequence. Subsequently, all API types in the API dictionary are considered for substitution, with the selected API being replaced by the one causing the most substantial decrease in the prediction probability of the target model.

RNN Ensemble. Rosenberg et al [31] applied various variant models of the recurrent neural network for ensemble. In this paper, we choose three RNN variant models for ensemble, namely BiLSTM, BiGRU and RNN.

Sequence Squeezing. We implement the sequence squeezing technique proposed by Rosenberg et al. [31]. Our implementation involves pre-training API embeddings in our dataset using the Glove model [33] and subsequently applying sequence squeezing. In the inference phase, we set the size of the squeezed API dictionary to $D/2$, where D represents the size of our API dictionary. We set the thresholds for predicting adversarial examples at 0.18, the same as [31].

Knowledge Distillation. Continuing from Heo et al [32], who introduced boundary supporting sample (BSS) to enhance the generalization ability of knowledge distillation models, we integrate that into the training process to bolster the robustness of the student model. Initially, a clean teacher model is trained using the BiLSTM on the training set. Subsequently, during the training of the student model, perturbations, involving both deletion and replacement, are randomly introduced to the original API call sequence to generate adversarial examples. The BSS loss is then computed through these adversarial examples upon the student model training.

Table II provides an overview of the defense performance of MalDenoise compared to five state-of-the-art methods against adversarial examples generated by four adversarial attacks in both the validation and test sets. The results demonstrate that MalDenoise consistently outperforms all other state-of-the-art methods, achieving the highest adversarial recall (AR) for all four types of adversarial malware examples in both sets. For instance, in the test set, MalDenoise exhibits adversarial recalls of 91.26%, 95.69%, 98.73%, and 91.53% for detecting different adversarial examples. This effectiveness enhances the robustness of the detection model, elevating its capability to detect a diverse array of adversarial examples.


(a) Feature visualization on the baseline model.

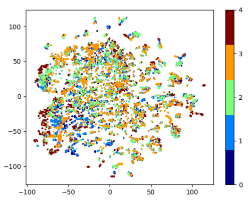

(b) Feature visualization on the MalDenoise model.

Fig. 3. Visualization of the features of the original malware and multiple adversarial malware examples on different detection models. Note that labels 0-3 denote the adversarial examples generated by the four adversarial attacks, and label 4 denotes the original malware with no added perturbations.

Note that none of the defense methods, including MalDenoise, managed to surpass the baseline model BiLSTM in terms of original malware recall, both in the validation and test sets. This observation suggests that the additional measures implemented to enhance model robustness, while effective against adversarial attacks, might introduce some trade-offs that impact the model's ability to detect malware. The primary objective of this study is to strike a balance between the performance of malware detectors in identifying unknown malware and resisting adversarial attacks. While the capability to detect malware may experience a moderate reduction, the primary focus is on enhancing the model's robustness to adversarial attacks to the greatest extent possible.

D. Understanding the Feature Representation

We explore the spatial distribution of perceptual features of original and adversarial malware. Figure 3 provides a visualization of the features on different detection models. The dataset includes 5,000 original malware samples and four types of adversarial malware, with 3,000 adversarial examples for each type of attack.

Figure 3(a) presents the feature visualization on the baseline model BiLSTM. Notably, the dark red samples form a distinct boundary with other colors. This indicates that the baseline model can clearly discern the features of the original malware and make a clear distinction in the feature space. Conversely, adversarial examples, after adding perturbations, diverge from the original malware, rendering the baseline model ineffective in identifying these adversarial malware.

Figure 3(b) depicts the feature visualization on the MalDenoise model. Unlike the distinct boundaries seen in the baseline model, the samples of all colors in the figure lack clear boundaries, appearing as neighbors that merge seamlessly. This indicates that in the feature space, the MalDenoise amalgamates the features of the original malware with those of other adversarial examples. Consequently, the denoising model effectively enable MalDenoise to identify the key malicious behaviors of the original malware within adversarial examples, eliminating the interference of adversarial perturbations and thereby enhancing the robustness of the model.

IV. Conclusion

This paper presents MalDenoise, an innovative defense approach designed to enhance the robustness of API call sequences based malware detection models against adversarial attacks. It employs denoising method to defend against various adversarial attacks. The denoising model filters input data, eliminating irrelevant or noisy APIs while extracting critical information. Evaluation against four types of adversarial attacks in a black-box setup demonstrates that MalDenoise outperforms other state-of-the-art defense methods, showcasing its effectiveness in real-world scenarios.

References

[1] Lei Cui, Jiancong Cui, Yuede Ji, Zhiyu Hao, Lun Li, and Zhenquan Ding, "Api2vec: Learning representations of api sequences for malware detection," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023, pp. 261–273.

[2] Xiaohui Chen, Zhiyu Hao, et al., "Cruparamer: Learning on parameter-augmented api sequences for malware detection," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 788–803, 2022.

[3] Xiang Huang, Li Ma, Wenyin Yang, and Yong Zhong, "A method for windows malware detection based on deep learning," *Journal of Signal Processing Systems*, vol. 93, no. 2, pp. 265–273, 2021.

[4] Ce Li, Qiujian Lv, Ning Li, et al., "A novel deep framework for dynamic malware detection based on api sequence intrinsic features," *Computers & Security*, vol. 116, pp. 102686, 2022.

[5] Qi-An Fu, Yinpeng Dong, Hang Su, et al., "{AutoDA}: Automated decision-based iterative adversarial attacks," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 3557–3574.

[6] Yijun Yang, Ruiyuan Gao, Yu Li, Qiuxia Lai, and Qiang Xu, "What you see is not what the network infers: Detecting adversarial examples based on semantic contradiction," *arXiv preprint arXiv:2201.09650*, 2022.

[7] Battista Biggio, Igino Corona, et al., "Evasion attacks against machine learning at test time," in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2013, pp. 387–402.

[8] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel, "Adversarial examples for malware detection," in *Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II 22*. Springer, 2017, pp. 62–79.

[9] Octavian Suciu, Scott E Coull, and Jeffrey Johns, "Exploring adversarial examples in malware detection," in *2019 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2019, pp. 8–14.

[10] Keane Lucas, Samruddhi Pai, et al., "Adversarial training for {Raw-Binary} malware classifiers," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 1163–1180.

[11] Yantao Li, Song Ruan, et al., "Transformer based defense gan against palm-vein adversarial attacks," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 1509–1523, 2023.

[12] Iuri Frosio and Jan Kautz, "The best defense is a good offense: Adversarial augmentation against adversarial attacks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 4067–4076.

[13] Nicolas Papernot, Patrick McDaniel, et al., "Distillation as a defense to adversarial perturbations against deep neural networks," in *2016 IEEE symposium on security and privacy (SP)*. IEEE, 2016, pp. 582–597.

[14] Fangzhou Liao, Ming Liang, Yinpeng Dong, Tianyu Pang, Xiaolin Hu, and Jun Zhu, "Defense against adversarial attacks using high-level representation guided denoiser," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 1778–1787.

[15] Deqiang Li and Qianmu Li, "Adversarial deep ensemble: Evasion attacks and defenses for malware detection," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3886–3900, 2020.

[16] Keane Lucas, Mahmood Sharif, Lujo Bauer, et al., "Malware makeover: Breaking ml-based static analysis by modifying executable bytes," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, 2021, pp. 744–758.

[17] Liang Tong, Bo Li, Chen Hajaj, et al., "Improving robustness of {ML} classifiers against realizable evasion attacks using conserved features," in *28th USENIX Security Symposium*, 2019, pp. 285–302.

[18] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th International Conference on Machine Learning*, New York, NY, USA, 2008, ICML '08, p. 1096–1103, Association for Computing Machinery.

[19] Xiaohui Chen, Lei Cui, Hui Wen, et al., "Malader: Decision-based black-box attack against api sequence based malware detectors," in *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2023, pp. 165–178.

[20] Dzmitry Bahdanau, Kyunghyun Cho, et al., "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[21] Thibault Févry and Jason Phang, "Unsupervised sentence compression using denoising auto-encoders," in *Proceedings of the 22nd Conference on Computational Natural Language Learning*, Brussels, Belgium, Oct. 2018, pp. 413–422, Association for Computational Linguistics.

[22] Mikel Artetxe, Gorka Labaka, Eneko Agirre, and Kyunghyun Cho, "Unsupervised neural machine translation," *arXiv preprint arXiv:1710.11041*, 2017.

[23] Markus Freitag and Scott Roy, "Unsupervised natural language generation with denoising autoencoders," *arXiv preprint arXiv:1804.07899*, 2018.

[24] Tianxiao Shen, Jonas Mueller, et al., "Educating text autoencoders: Latent representation guidance via denoising," in *International conference on machine learning*. PMLR, 2020, pp. 8719–8729.

[25] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[26] kericwy1337, "Malicious-code-dataset," https://github.com/kericwy1337, 2019.

[27] Puyudi Yang, Jianbo Chen, et al., "Greedy attack and gumbel attack: Generating adversarial examples for discrete data," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 1613–1648, 2020.

[28] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang, "Generating natural language adversarial examples," *arXiv preprint arXiv:1804.07998*, 2018.

[29] Ishai Rosenberg, Asaf Shabtai, et al., "Query-efficient black-box attack against sequence-based malware classifiers," in *Annual Computer Security Applications Conference*, 2020, pp. 611–626.

[30] Eric Wong, Leslie Rice, and J Zico Kolter, "Fast is better than free: Revisiting adversarial training," *arXiv preprint arXiv:2001.03994*, 2020.

[31] Ishai Rosenberg, Asaf Shabtai, Yuval Elovici, and Lior Rokach, "Sequence squeezing: A defense method against adversarial examples for api call-based rnn variants," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–10.

[32] Byeongho Heo, Minsik Lee, Sangdoo Yun, and Jin Young Choi, "Knowledge distillation with adversarial samples supporting decision boundary," in *Proceedings of the AAAI conference on artificial intelligence*, 2019, vol. 33, pp. 3771–3778.

[33] Jeffrey Pennington et al., "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.