Journal of Management Information Systems

ISSN: 0742-1222 (Print) 1557-928X (Online) Journal homepage: www.tandfonline.com/journals/mmis20

Defending Deep Learning-Based Raw Malware Detectors Against Adversarial Attacks: A Sequence Modeling Approach

Reza Ebrahimi, James Lee Hu, Ning Zhang, Jay F. Nunamaker Jr. & Hsinchun Chen

To cite this article: Reza Ebrahimi, James Lee Hu, Ning Zhang, Jay F. Nunamaker Jr. & Hsinchun Chen (2025) Defending Deep Learning-Based Raw Malware Detectors Against Adversarial Attacks: A Sequence Modeling Approach, Journal of Management Information Systems, 42:4, 1118-1148, DOI: 10.1080/07421222.2025.2561384

To link to this article: https://doi.org/10.1080/07421222.2025.2561384

JOURNAL OF MANAGEMENT INFORMATION SYSTEMS
2025, VOL. 42, NO. 4, 1118–1148
https://doi.org/10.1080/07421222.2025.2561384

Routledge
Taylor & Francis Group

Defending Deep Learning-Based Raw Malware Detectors Against Adversarial Attacks: A Sequence Modeling Approach

Reza Ebrahimi, James Lee Hu[b], Ning Zhang[c], Jay F. Nunamaker Jr.[b], and Hsinchun Chen[b]

School of Information Systems, University of South Florida, Tampa, FL, USA; [b]Eller College of Management, University of Arizona, Tucson, AZ, USA; [c]Acium Inc., FL, USA

ABSTRACT

Malware detectors are the first line of defense against cyber-attacks that damage Information Technology (IT) infrastructure. Recently, deep learning (DL)-based malware detectors have yielded breakthrough results in identifying unseen attacks without requiring feature engineering and expensive dynamic malware analysis in a sandbox. However, these detectors are susceptible to adversarial malware attacks. Emulating effective adversarial malware variants is instrumental in revealing the vulnerabilities of such systems and developing automated cyber defense. Current methods for launching such attacks often assume scenarios that require accessing insider knowledge about the architecture of the malware detector and/or cannot operate directly on raw malware files. We propose Adversarial Malware example Generation and Defense (AMGD), a novel framework to defend the detectors by automatically generating malware variants from raw executables without assuming any prior detector knowledge. AMGD is generalizable to multiple detectors as it can be trained on multiple malware detectors simultaneously. AMGD employs Independent Recurrent Neural Nets (IndRNNs) to offer a novel generative byte-level malware sequence model, named Mal-IndRNN, to evade DL-based malware detectors. Mal-IndRNN effectively evades three renowned DL-based malware detectors and outperforms benchmark methods. We utilize malware variants generated by Mal-IndRNN to improve the robustness of malware detectors against adversarial attacks on a real dataset. AMGD offers a practical approach to proactively accounting for the Artificial Intelligence (AI)-enabled adversary in the design and development phase of DL-based malware detectors rather than reactive measures after deployment.

KEYWORDS

IT infrastructure; cyber defense; adversarial malware attacks; adversarial malware; sequence models; Independent Recurrent Neural Nets (IRNNs); cybersecurity; cybercrime prevention

Introduction

Cybercrime is predicted to cost the world $12.2 trillion yearly by 2031 [9]. Among various types of attacks threatening our cyberspace, malware attacks cause the most financial loss [7]. The average annual cost of malware attacks has increased to $2.6 million per mid-sized company worldwide [7]. Malware detectors are the first line of defense against these attacks [65]. Traditional malware detectors mainly rely on signature-based approaches that match manually defined patterns against known malicious files [3]. Signature-based methods

CONTACT Reza Ebrahimi ebrahimim@usf.edu School of Information Systems and Management, University of South Florida, 4202 E. Fowler Avenue, BSN 3403, Tampa, FL 33620, USA.

significantly rely on the quality and recency of the pre-defined rules hand-crafted by malware analysts. While useful, signature-based methods can be ineffective in dealing with newly evolved variants of malware (i.e., "unseen" variants known as zero-days) since they rely on manually defined rules that cannot keep up with the rapid evolution of malware variants [13, 21],21. To address this issue, researchers have presented machine learning (ML)-based malware detection. While classic ML algorithms require manual feature engineering, Deep Learning (DL)-based static malware detectors have emerged that consume the whole raw malware executable as input. These malware detectors can operate effectively without relying on hand-crafted defined rules, manual feature engineering, or execution of the malware in an isolated sandbox environment as is required in expensive and time-consuming dynamic analysis. As a result, successful DL-based detectors have emerged [23, 38, 55], 38, 55. Today, deep learning models are shown to be crucial for the success of static malware detection [14]. Given the breakthrough results of DL-based malware detectors in proactively identifying unseen malware variants without requiring human involvement and expensive dynamic malware analysis, leading cybersecurity firms such as Avast, Symantec, and Endgame are adopting advanced DL-based malware detectors in their proprietary antivirus products [8, 61],61.

Despite their success, it has been recently shown that the nascent DL-based malware detectors can be vulnerable to adversarial malware attacks, in which an adversary modifies a known malware executable to deceive the malware detector into recognizing the modified variant as a benign executable and thus evading detection [37, 60],60. These attacks associate with an emerging branch of ML, known as Adversarial ML [24]. Adversarial ML mainly studies how to mislead ML models by creating deceptive variants resulted from subtle modifications of known inputs (e.g., malware files). The automated approach to generate these adversarially modified variants for evading ML models is referred to as Adversarial Example Generation (AEG) [15]. While most AEG studies in the malware analysis domain focus on devising effective attack methods [12, 22, 26, 52, 62, 69, 70], 22, 26, 52, 62, 69, 70, the defensive potential of AEG has not been fully realized in extant cybersecurity studies [13, 66],66. Specifically, AEG can emulate adversarial attacks to train and improve the robustness of malware detectors [24]. Many malware AEG methods rely on some knowledge about the malware detector's internal architecture, internal system parameters, or the confidence score feedback obtained from the malware detectors. As this insider information is often unknown, such AEG methods can lead to unrealistic adversarial malware variants. We consider a practical threat model in which the adversary aims to produce an optimized byte sequence that when appended to the original malware program conceals the true nature of the malware. To preserve the malicious functionality of the new adversarial malware variant, the adversary ensures that the bytes are injected in empty sections of the malware file such that they are guaranteed to not interfere with the execution to obfuscate the malicious identity of a known malware. Most extant studies provide methods specific to a certain malware detector and, thus, require re-training on other detectors. This can pose challenges when these models are deployed in practical settings that conduct static malware detection against multiple malware detectors at the same time (e.g., cloud-based environments). Malware Independent Recurrent Neural Nets (Mal-IndRNN) allows generating adversarial examples that are generalizable to multiple malware detectors without the need to be trained separately against each malware detector. The proposed Mal-IndRNN is a novel generative model that is capable of effectively generating

adversarial malware variants that can evade well-established open-source deep learning-based malware detectors simultaneously. Being equipped with such a method helps effectively uncover the search space of the adversarial malware variants to design more robust systems on the defender side (malware detector developer).

Contributions

Given the crucial role of malware detectors in defending cyber infrastructure and improving organizations' security posture, it is necessary to automatically protect deep learning-based malware detectors against adversarial attacks with effective methods. Our study offers a novel approach to directly learn a sequence model on raw executables and generate benign-looking content that is generalizable to multiple malware detectors without requiring any knowledge of the targeted detector to improve the detector's resistance against adversarial attacks. To our knowledge, the proposed method is among the first automated approaches without these restrictive assumptions that is capable of being trained on multiple malware detectors simultaneously to ensure generalizability. Furthermore, our approach does not require expensive dynamic malware analysis. To foster reproducibility, we made Mal-IndRNN implementation available to the cybersecurity research community on GitHub at http://github.com/johnnyzn/MalRNN.

From a cybersecurity perspective, our contributions are two-fold. (1) The proposed Mal-IndRNN is a novel sequence model that operates on raw malware files to generate adversarial malware variants that are capable of evading deep learning-based malware detectors. Being equipped with such an attack mechanism can help effectively adapt to current adversaries and design more robust systems on the defender (malware detector developer) side. The attack mechanism naturally forms a sequence modeling problem. As the raw malware byte sequences are often long and can prohibit learning effective sequence models, most static malware detectors use either features that are extracted from the binary or the sequence of its application programming interfaces (APIs) called in the executable as an alternative to consuming and analyzing the whole raw malware file. We expect that a design that is capable of directly operating on raw malware executables can unleash the full potential of DL models in automatically extracting salient features from the raw malware file. (2) While some methods can effectively operate against certain malware detectors, the generated adversarial malware variants might not be optimized to evade other detectors. Adversarial Malware example Generation and Defense (AMGD) is uniquely positioned in being generalizable to multiple detectors as it can be trained on multiple malware detector simultaneously. As a corollary to our design, while most recent works are focused on mostly devising attacks in isolation, we empirically highlight in our experiments that a careful coupling of sequence modeling-based attacks with adversarial training is a powerful tool to defend against AI-enabled adversaries in cyber defense. Accordingly, we conduct a practical case study yielding applicable insights for practitioners.

Literature Review

Three major areas of research are examined. First, to position our study within information systems (IS) cybersecurity knowledge base, we review recent relevant IS cybersecurity literature on defending information technology (IT) infrastructure. Second, we review

adversarial malware example generation to identify the state-of-the-art in automatically emulating adversarial malware attacks against IT infrastructure. Lastly, we examine the state-of-the-art in generative RNN-based sequence models as the overarching methodological foundation for operationalizing effective AEG in malware analytics context. Based on the review, we identify research gaps and pose our research questions.

Recent IS Cybersecurity Research on Defending IT Infrastructure

At a high level, recent IS cybersecurity research on defending IT infrastructure falls into three major streams: (1) behavioral security research that aims to model organizational/individual behavior, policy compliance, and security risk for defending IT infrastructure [30]; (2) economic security research focusing on modeling the impact of investment, market effects, and other economic factors on the security of IT infrastructure [30]; and (3) ML-based security research, which focuses on developing new IT artifacts that employ ML to protect IT infrastructure guided by computational design science [1, 20, 56], 20, 56. Behavioral and economic security studies have often provided theoretical justification for an inherently adversarial environment [11, 33],33. These studies account for the adversary's existence in their modeling of security phenomena. For instance, Karhu et al. [33] focus on the adversarial role of hostile firms in protecting software assets of an organization from exploitation by other firms. Similarly, Burns et al. [11] consider this view in studying the impact of insiders' behavior on information security. On the contrary, the artifacts provided in the ML-based security research are often not designed to be deployed in adversarial environments, where the adversary can generate adversarial attacks to mislead the artifact and reduce its applicability. The ML-based security research stream often only focuses on leveraging ML to devise highly useful IT artifacts that learn from human security analysts for automating cybersecurity tasks [5, 6, 18, 19, 44, 58, 59, 68], 6, 18, 19, 44, 58, 59, 68. Nevertheless, the security of these IT artifacts can be compromised, which is detrimental to the effectiveness of modern cyber defense [43, 64],64. As an integral part of cyber defense, defending against malware attacks is not an exception to this issue. It is unclear how to automatically strengthen the robustness of ML-based[1] cyber defense against adversarial malware attacks. Verifying the resistance of malware detectors against adversarial attacks is a viable defense mechanism that helps improve the resistance of detector models and provides insights for designing better models [24]. This requires equipping cyber defenders with the automatic generation of realistic adversarial malware variants.

Adversarial Malware Example Generation

Recently, DL models have been shown to fail when an adversary carefully modifies their input data through subtle perturbations. AEG pertains to applying automated feature perturbations that can cause a target DL model to make wrong decisions. Depending on the information available from the targeted DL model, AEG can be carried out in four scenarios, also known as threat models [3, 53],53: white-box, gray-box, black-box, and binary black-box. In white-box attacks, the adversary has full access to the internal structure and system parameters of the targeted DL model [40]. Gray-box AEG pertains to situations in which the parameters of the attacked DL model are not available, but the adversary has access to the features that are important

for its actions. In black-box (BB) attacks, the adversary cannot access the model's specification or features; however, it can observe confidence scores (i.e., a real-valued feedback signal) from the attack target. Finally, binary black-box (BBB) attacks are black-box attacks that neither assume a priori knowledge about the attack target nor require access to any real-valued feedback from the target. Instead, in the binary black-box scenario, the adversary can only observe a binary response signifying the success or failure of the generated sample in evading the attack target. While being the most restrictive, binary black-box AEG is the most realistic scenario in adversarial attacks, since oftentimes, the specification and confidence score of the attack target are unknown [23].

Conducting AEG in the malware analytics domain gives rise to a new stream of research that employs AEG to automatically perturb malware samples and generate variants that evade malware detectors while preserving the functionality of the original malware via injecting byte sequences in non-executable sections of malware files [15, 29, 61, 69], 29, 61, 69. For conciseness, we refer to AEG for malware as Adversarial Malware example Generation (AMG). AMG can be categorized based on the type of the implemented threat model (i.e., white-, gray-, black-, and binary black-box). Consistent with our goal of proposing a more realistic adversarial attack scenario, we examine extant studies that support black-box and binary black-box AMG. We summarize recent relevant studies in five dimensions shown in Table 1: data source, attack method, threat model (BB, or BBB), targeted malware detector model, and its input type. While the table is not meant to be exhaustive, it is intended to reflect the recent state of the adversarial malware example attacks field. Studies that offer black-box attacks do not require knowing the specifications of the targeted malware detectors [12, 13, 15, 28, 52, 62], 13, 15, 28, 52, 62. These studies employ a wide range of methods such as genetic algorithm [15], random perturbations [12, 13],13, dynamic programming [52], decompiling the malware code [60], RNN [28], and GAN [29, 45, 69, 70], 45, 69, 70. However, these methods heavily rely on the confidence score feedback obtained from the malware detectors in order to generate their perturbations, restricting their applicability [57]. The confidence score is a real-valued quantity ranging between 0 and 1, indicating the probability that the input is malware. The confidence score is internal to the malware detector and thus is not visible to the adversary in practice.

Binary black-box attacks, however, do not require observing the confidence score [3, 17, 22, 57], 17, 22, 57 and, thus, are applicable to real attack scenarios. As seen in Table 1, most current binary black-box studies target detectors that require manual feature engineering [3, 17, 22], 17, 22. Among binary black-box studies that target DL-based detectors, Rosenberg et al. [57], Hu and Tan [28], Zhong et al. [69], and Zhu et al. [70] propose methods that require API call sequences obtained from expensive dynamic analysis of the malware executable in a sandbox, making their utility limited in practice [3]. The methods proposed by Song et al. [61], MAB-Malware, and Demetrio et al., [15], GAMMA, yield breakthrough result in binary black-box AEG using multi-arm bandit and genetic programming, respectively. Furthermore, most studies on attacking DL-based malware detectors are designed to only target a specific detector architecture with certain parameter settings. Focusing on evading one specific architecture limits the generalizability of such methods to other malware detectors. We expect that learning

Table 1. Selected recent AMG research on malware with black-box and binary black-box threat models.

Year	Author(s)	Data Source	Attack Method	Threat Model	Malware Detector	Detectors' Input Type
2024	Kargarnovin et al. [32]	VirusShare	Benign feature append	BB	DL-based	Malware Source Code
2023	Chen et al. [14]	VirusShare, SourceForge	Deep RL	BB	DL-based	Malware Executable
2023	Zhong et al. [69]	VirusShare	GAN	BB	Signature-based	API sequences
2022	Hu & Tan [29]	Malwr	GAN	BB	DL-based	Malware Executable
2022	Zhu et al. [70]	Kaggle	GAN	BB	MLP, SVM, DT, LR, RF	API sequences
2022	Song et al. [61]	VirusTotal	MAB	BBB	DL-based	Malware Executable
2021	Demetrio et al. [15]	VirusTotal	Genetic algorithm	BBB	DL-based	Malware Executable
2020	Li et al. [45]	Contagio PDF malware dump	GAN	BB	SVM	Engineered Features
2019	Castro et al. [12]	VirusTotal	Randomized perturbations	BB	Signature-based	Engineered Features
2019	Chen et al. [13]	VirusShare, Malwarebenchmark	Enhanced randomized perturbations	BB	DL-based	Malware Executable
2019	Dey et al. [17]	Contagio PDF malware dump	Genetic programming	BBB	RF	Engineered Features
2019	Fang et al. [22]	VirusTotal	Deep RL	BBB	Gradient Boosting Trees	Engineered Features
2019	Park et al. [52]	Malmig & MMBig	Dynamic programming	BB	DL-based	Malware Executable
2019	Rosenberg et al. [57]	VirusTotal	GAN	BBB	DL-based	API sequences
2019	Suciu et al. [62]	VirusTotal, Reversing Labs, FireEye	Benign feature append	BB	DL-based	Malware Executable
2019	Sharif et al. [60]	Malware evasion competition	Binary diversification	BB	DL-based	Malware Executable
2018	Anderson et al. [3]	VirusTotal	Deep RL	BBB	Gradient Boosting Trees	Engineered Features
2018	Hu & Tan [28]	Malwr dataset	RNN	BB	DL-based	API sequence

Note: DT, decision tree; GAN, Generative Adversarial Network; LR, linear regression; MAB, Multi-Arm Bandit; MLP, multi-layer perceptron; RL, reinforcement learning; RNN, recurrent neural network; SVM, support vector machine.

a sequence model from benign executables can facilitate attaining more generalizable AMG.

Generative Sequence Models

Constructing a sequence model amounts to learning a probability distribution over a sequence of elements. Once learned, a sequence model can be used to generate the next element in a given sequence [36]. At each time step t, a generative RNN takes an input x_t and the current hidden state h_t to emit a continuous value. This value is used to generate future elements in the sequence. The generative property of RNNs makes them suitable for sequence modeling tasks [4] (e.g., malware byte analysis), where the input elements are time-dependent. Once trained, generative RNNs yield sequence models that can predict the next element based on a given input sequence [71].

Despite their promise, one major challenge that arises in utilizing RNN-based sequence models directly on raw malware content is the need to process very long sequences of bytes, which renders learning impractical due to a large number of time steps [55]. This problem is also common in various RNN variants such as long short-term memory (LSTM) and gated recurrent unit (GRU) [23]. The issue exacerbates with Transformers-based sequence models such as Bidirectional Encoder Representations from Transformers (BERT) [16], generalized autoregressive pre-training (XLNet) [67], and Generative Pre-trained Transformer (GPT) [10, 54],54 due to the fact that the computational complexity is quadratic in the size of the input [34, 35],35. To alleviate this issue, a well-recognized deep learning architecture, namely Independently Recurrent Neural Networks (IndRNN), has been proposed by Li et al. [41, 42],42 that enables processing sequences that are longer by an order of magnitude. IndRNN achieves a state-of-the-art sequence length by eliminating some interconnections between the neurons in a simple RNN, and replacing expensive matrix multiplications with less expensive Hadamard products [41]. While in a conventional RNN all neurons are connected via recurrent connections through time, in an IndRNN each neuron is independent of other neurons and connected through layers. Li et al. [41] show that the independence property of recurrent connections in IndRNN lowers the computational complexity so that learning long sequences becomes practical. This feature is expected to be helpful when dealing with long sequences of bytes in malware executables. As such, although IndRNN has been proposed for image applications, we expect with proper modifications, it could provide significant value to AMG.

Another important challenge in adopting RNN-based sequence models for malware content analysis is that generative RNNs (including IndRNN) require the input and output sequences to have the same dimensions. While this is useful in natural language processing (NLP) tasks such as part of speech tagging, it limits their applicability in the malware domain. A recent DL architecture, known as the sequence-to-sequence model, addresses this issue by adding an additional encoding step before feeding the data to the generative RNN [50]. Sequence-to-sequence models have recently yielded breakthrough results in sequence analysis tasks, such as machine translation [50] and speech recognition [31]. As such, they can map the input sequence of a fixed length to a generated output sequence of a different length.

Given the promise of IndRNNs in processing long sequences combined with the success of sequence-to-sequence models in generating sequences with arbitrary length, we expect that enhancing sequence-to-sequence RNN-based sequence models with IndRNN can provide an effective tool to automatically generate benign-looking adversarial examples for the malware domain. To this end, we propose Mal-IndRNN, which is a novel, adversarial, byte-level, sequence-to-sequence model that can operate on the malware byte sequences (as opposed to the sequence of API calls in a sandbox) to accomplish binary black-box AMG without requiring dynamic analysis or a priori knowledge of the target detector.

Research Gaps and Questions

Based on our review of the literature, we identified methodological and domain gaps and corresponding questions in AMG. From a methodological perspective, despite the practical success of sequence models in other fields such as text analytics,

enhancing adversarial malware example generation on *raw* malware executables using sequence models is not a trivial task. We expect that a novel sequence-to-sequence design based on independent RNNs can be an effective solution to this end. In the AMG domain, most AMG methods focus on evading only a specific architecture. That is, while their proposed method operates effectively on a specific malware detector, it may not be as effective on a different malware detector with a different DL architecture. This reduces their generalizability to other DL-based architectures. We hypothesize that leveraging simultaneous training of an encoder-decoder sequence-to-sequence model against multiple architectures is a viable choice to achieve generalizability on different architectures. Considering these gaps, our proposed Mal-IndRNN design aims to specifically answer the following question: "How can we learn a byte-level sequence-to-sequence model from *raw* malware executables that evades multiple DL-based malware detectors in black-box settings?" Furthermore, we expect that the adversarial attack capabilities using such sequence models, when coupled with adversarial training on the defender side, can effectively lead to a proactive and automatic approach that contributes to being a step ahead of the adversary. Consequently, we are interested in "how it can be empirically demonstrated that an effective sequence model such as Mal-IndRNN can contribute to defending against adversarial malware variants through adversarial training." Motivated by these questions, we propose AMGD, a novel framework to automatically emulate adversarial malware attacks and defend against them to improve the resiliency of DL-based malware detectors.

Proposed Design: AMGD

To address the aforementioned research questions, we propose a novel framework for AMGD to generate realistic adversarial malware variants with the goal of improving the robustness of ML-based malware detectors against adversarial malware attacks. The AMGD framework consists of three major components. (1) Malware Data Acquisition and Pre-processing, (2) Automated Adversarial Malware Example Generation, and (3) Performance Evaluation and Case Study. Figure 1 provides an

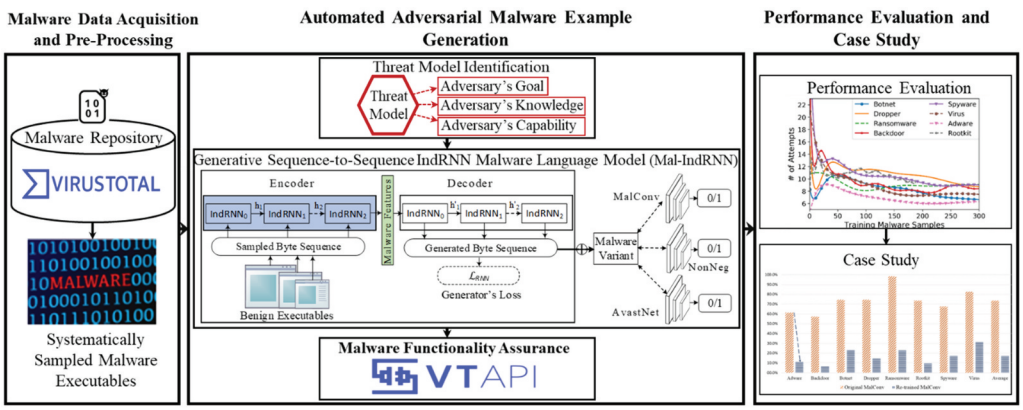

Figure 1. Overview of AMGD framework for Adversarial Malware Example Generation and Defense.

overview of the AMGD framework. Each component is described in the following subsections.

Malware Data Acquisition and Pre-processing

Following Anderson et al. [3] and Chen et al. [13], for malware testbed construction, we developed a repository comprising 31,188 (14.69 GB) executable files collected from VirusTotal. Consistent with [39], our testbed includes a development set (used for training MalRNN and conducting benchmark evaluations) and an evaluation set of recent, unseen malware executables collected in 2019-2021 to measure improvement in detection performance for our case study. Dividing the development and evaluation sets based on the recency of the malware files is a common practice in malware analysis literature, which is often preferred over cross-validation that neglects the chronological order of malware files [39]. Table 2 shows the breakdown of the malware executables in the development and evaluation sets by malware type. We describe each set next.

The development set includes two subsets of executable files with a total volume of 8.78 GB: (1) a malware executable dataset that serves as the initial seed to generate evasive malware variants (3.74 GB, as shown in Table 2), and (2) an additional benign executable dataset to train the sequence model (5.04 GB, not shown in Table 2). To obtain the former subset, we compiled an up-to-date collection of malware samples by obtaining an academic license from VirusTotal. The subset includes 6,037 (3.74 GB) recent malware executables collected from 2017 to 2019 in eight categories, including botnet, ransomware, spyware, adware, virus, dropper, backdoor, and rootkit. To obtain the benign executable dataset, following Ebrahimi et al. [21], we collected 13,554 (5.04 GB) benign executables from a clean installation folder of Microsoft Windows. In addition to the development dataset, we included 11,597 (5.91 GB, as shown in Table 2) recent, unseen malware executables from 2019 for evaluating the malware

Table 2. Breakdown of the malware executables in the development and evaluation sets by malware type.

Malware Type	Description	Examples	Development Set Size	Evaluation Set Size
Adware	Shows unwanted ads and force internet traffic to websites	eldorado, razy, gator	1,947 (1.50 GB)	4,185 (2.70 GB)
Backdoor	Negates normal authentications to access the host	lunam, rahack, symmi	678 (0.50 GB)	1,218 (0.60 GB)
Botnet	A network of bots connected through the internet	virut, salicode, sality	526 (0.15 GB)	1,184 (0.30 GB)
Dropper	Secretly installs other malware on the host	dinwod, gepys, doboc	904 (0.55 GB)	1,942 (1.1 GB)
Ransomware	Encrypts data and files, restricting access and usage until decrypted by malware authors	vtflooder, msil, bitman	900 (0.45 GB)	911 (0.54 GB)
Rootkit	Grants admin privilege to malware author	onjar, dqqd, shipup	53 (0.02 GB)	136 (0.09 GB)
Spyware	Allows malware authors to steal personal information covertly	mikey, qqpass, scar	640 (0.38 GB)	941 (0.32 GB)
Virus	Corrupts files on the host system	nimda, shodi, hematite	659 (0.19 GB)	1080 (0.26 GB)
Total	–	–	6,037 (3.74 GB)	11,597 (5.91 GB)

detection performance improvement resulting from Mal-IndRNN's generated adversarial variants in our case study.

Pre-processing

In accordance with previous malware analysis work [13, 55],55, in both development and evaluation sets, we converted the executables' content to hexadecimal byte format. In this format, each byte is represented by its hexadecimal value ranging from 0 to 15. This format is suitable for processing bytes and constructing a byte-level sequence model in Mal-IndRNN. Furthermore, to decrease training time in practice, we employed systematic sampling for all methods. This procedure samples the input binary sequence in fixed intervals. The systematic sampling was used for practical reasons and as a technique to decrease the training time.

Automated Adversarial Malware Example Generation

The automated adversarial malware example generation component of AMGD includes three sub-components. Following Dey et al. [17], Fleshman et al. [23], and Sharif et al. [60],23,60, the first sub-component characterizes the threat model for automated adversarial malware example generation. Based on the identified threat model, the second sub-component conducts adversarial malware example generation via a novel generative sequence-to-sequence RNN malware sequence model. Lastly, the third sub-component ascertains malware functionality using the VirusTotal API to ensure that the generated malware variants can be executed and maintain their original malicious functions.

Threat Model Identification

To guide the design of our adversarial malware example generation model, consistent with Sharif et al. [60], we characterize the threat model for launching binary black-box adversarial malware attacks against DL-based malware detectors by identifying three major characteristics: the *adversary's goal, knowledge*, and *capability*.

Adversary's Goal. The adversary aims to automatically generate malware variants that can evade several DL-based malware detectors simultaneously to ensure generalizability. Unlike the threat model proposed by Anderson et al. [3] for black-box settings, which targets only feature-based malware detectors, our threat model focuses on launching attacks against DL-based malware detectors. The output is evasive, functionality-preserving malware variants generated based on an initial set of known malware files.

Adversary's Knowledge. The structure and parameters of the malware detector model are unknown to the adversary. To eliminate the reliance on strong assumptions that require knowing internal architecture of the malware detector, we focus on binary black-box adversarial example generation against DL-based malware detectors. Consequently, the adversary does not have access to the confidence score produced by the malware detector. The only information available to the adversary is whether the generated malware variant can evade the malware detector or not.

Adversary's Capability. The adversary can apply functionality preserving modifications on malware executables, while the maximum modification size is limited. Following Li et al. [45] and Suciu et al. [62],62, to preserve the malicious functionality of the new adversarial malware variant, the adversary ensures that the bytes are injected in empty sections of the malware file such that they are guaranteed to not interfere with the execution to obfuscate the malicious identity of a known malware. These modifications inject the generated content to segments of the original malware file, which will not be executed by the operating system.

To operationalize this threat model, we propose Mal-IndRNN, a novel generative sequence-to-sequence RNN malware sequence model. We next detail Mal-IndRNN.

Generative Sequence-to-Sequence IndRNN Malware Sequence Model (Mal-IndRNN)

Consistent with the above threat model, it is expected that mimicking the patterns of benign executables could be a viable AMG approach. Mal-IndRNN achieves this through learning a novel sequence model on bytes that can generate benign-looking samples.

As noted, despite the success of Transformer-based sequence models, they rely on the self-attention mechanism, which incurs a time- and space-complexity that is quadratic in the length of the input [34, 35],35—making these models impractical for extremely long sequences in binary executables. Figure 2 shows the abstract view of our byte-level generative sequence-to-sequence RNN for learning an adversarial sequence model.

The model accepts a systematically sampled malware executable as input to the generative sequence-to-sequence RNN sequence model. We first describe the main objective of this sequence model, followed by the description of its components.

To alleviate the issue of processing long sequences from malware executable content, Mal-IndRNN adopts IndRNN as the building block of its sequence-to-sequence model. Mal-IndRNN aims to maximize the adversarial loss [46] of the malware detector model formulated as $\underset{\delta \in \Delta}{\text{maximize}}\, \mathcal{L}(\mathcal{H}_\theta(x+\delta), y)$, where x is the input malware sample, $\mathcal{H}_\theta$ is the attacked DL-based malware detector model parameterized by θ, and $\delta \in \Delta$ denotes the allowable perturbations that preserve functionality (appending byte sequences). The loss function $\mathcal{L}$ represents binary cross-entropy loss in most DL-based malware detectors. Given that the perturbations are generated by the IndRNN-based sequence-to-sequence model parameterized by w, the maximization can be written as Equation 1:

$$\underset{w}{\text{maximize}}\, \mathcal{L}(\mathcal{H}_\theta(x + IndRNN_w(\mathcal{H}_\theta, x)), y) \tag{1}$$

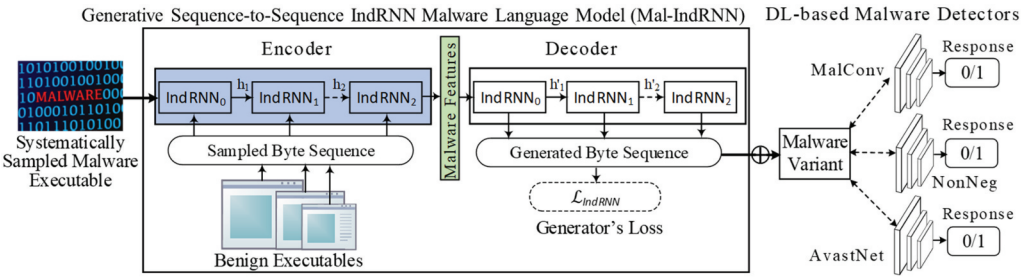

Figure 2. Abstract view of Mal-IndRNN Malware Evasion Architecture.

However, in a (binary) black-box setting, the exact loss function from the malware detector is not accessible and cannot be incorporated into the model's loss. Accordingly, the direct maximization of Eq. 1 is impractical. Accordingly, one key idea in Mal-IndRNN's design is that maximizing the loss in Eq. 1 is empirically tantamount to minimizing the loss of the adversarial model that generates benign-looking samples that can evade the malware detector. Motivated by the desirable properties of recent sequence-to-sequence and IndRNN models for processing malware content, Mal-IndRNN's adversarial malware variant generator consists of two main IndRNN components: An encoder IndRNN and a decoder IndRNN.

Encoder. The encoder aims to encapsulate the salient features of the input byte sequence into a feature vector. This vector is obtained from the final hidden state of the last RNN unit in the encoder architecture and is fed to the decoder RNN (shown in the vertical inner box in Figure 2).

In a traditional RNN-based encoder, the current hidden state h_t is obtained as a function of both its previous state h_{t-1} and the current input element x_t. More formally, h_t is given by $h_t = f\left(W^h h_{t-1} + W^x x_t + b\right)$. Where W^h denotes the network weights between the hidden units multiplied by the Hadamard (i.e., element-wise) product, W^x represents the network weights between hidden units and the input elements. Function f is a non-linear activation such as $\tanh(\cdot)$ and b represents a vector of initial bias values. IndRNN attains long-range memory via replacing the matrix multiplications with the Hadamard product [42]. As such, employing IndRNN in the encoder component of Mal-IndRNN yields Eq. 2:

$$h_t = f\left(W^h \odot h_{t-1} + W^x x_t + b\right) \tag{2}$$

Decoder. The decoder receives the feature vector from the encoder and reconstructs the byte sequence that minimizes a cross-entropy loss between the generated bytes and benign samples ($\mathcal{L}_{RNN}$) at each timestep. Unlike the encoder, the decoder's hidden state at each time step is only a function of the previous hidden state and is given by Eq. 3:

$$h_t = f\left(W^h \odot h_{t-1} + b\right) \tag{3}$$

After training, the generator learns to append benign-looking binary content to the malware executable in order to maximize the adversarial loss and construct an evasive malware variant. To generate a candidate malware variant, after completion of each training iteration, the generated byte sequence from Mal-IndRNN's generator is attached to the original malware. The candidate malware variant is checked against several black-box malware detector models to assess if it can evade all of them. The malware detectors yield a binary output of 1 and 0, denoting detection and evasion, respectively. This process repeats until the new variant evades the malware detector or the maximum number of attempts is reached. In case the maximum number of attempts for a specific malware sample is reached, the model proceeds to the next malware sample. If the generated malware variant successfully evades the malware detectors, it is marked as an evasive variant and will be further processed to ensure its functionality, as will be discussed in the next sub-section.

Content generation with Mal-IndRNN amounts to an emulated attack that is agnostic to the architecture of the targeted malware detector. It is worth noting that, following Anderson et al. [3], in order to comply with the binary black-box attack scenario, the

confidence score provided by the malware detector was masked to mimic a binary output. We implemented Mal-IndRNN using PyTorch. Mal-IndRNN was run on a single NVIDIA RTX 2080 GPU with 4,352 CUDA cores and 8 GB internal memory. The code is designed to run on both GPU and CPU environments. Mal-IndRNN's specifications, including the architecture and (hyper) parameter settings, are given in Appendix 1. Additionally, Appendix 2 includes the empirical assessment of the computation time.

Malware Functionality Assurance

To ensure the functionality of the modified malware variants, we employed a two-step process. In the first step, the generated malware variant was scanned by non-DL-based VirusTotal scanners to make sure that it will be recognized as malicious. In the second step, we used VirusTotal's API, for assessing the functionality report of the generated malware variant. This report provides a malware behavior report that includes static and dynamic analysis of the modified malware sample. The report includes information such as network behavior and file access patterns. Using the VirusTotal API, we compared the behavior reports for the modified evasive variants and original (i.e., unmodified) malware samples. Through this process, we ensured that the key parts of the VirusTotal's report stayed the same after modification, showing that the generated malware samples are fully functional. All 6,037 malware samples in our dataset were checked and confirmed to be functional after injecting the generated bytes.

Performance Evaluation and Case Study

Following the guidelines prescribed by the computational design science paradigm, we systematically evaluate the performance of the proposed design against alternative solutions [25, 56],56. Additionally, we conduct a proof-of-value for our designed system via a case study [27, 49],49. The overview of our evaluation and case study are discussed in the next two sub-sections, respectively.

Performance Evaluation

The characteristics of the testbed were described earlier. As our attack target, we selected three renowned DL-based static malware detectors. All three detectors are cited frequently by security researchers and are made available by authors through GitHub repositories. We note that while evaluating the proposed methods on open-source deep learning based is a common practice, our method might also be useful for closed-source tools that use deep learning components. However, due to licensing limitations, following previous works, we focus on the common practice of evaluating performance on the following publicly available detectors. We note that none of these malware detectors have been trained on adversarial malware samples.

- MalConv [55] is among the most successful DL-based malware detectors, developed via a collaboration between the Laboratory for Physical Sciences (LPS) and NVIDIA. The model incorporates a deep convolutional neural network architecture trained on approximately half a million malware executables and achieves an area under the receiver operating characteristic (ROC) curve of 98.5% on a large real-world test set of

with more than 77,000 binaries, including 40,000 malware files collected from VirusShare and OpenMalware.

- NonNeg [23] is a successor of MalConv developed by LPS, which modifies MalConv's architecture with non-negative weight constraints. The model was trained on 2 million malware executables and obtained an AUC of 95.3% on the MalConv's test set.
- AvastNet [38] is developed by Avast research group and features a deeper neural network than MalConv and NonNeg, with a total of eight layers. It was trained on 20 million proprietary malware samples from Avast and achieved 96% accuracy without using any hand-crafted features.

Both MalConv and NonNeg were featured in a malware evasion competition hosted by Endgame in 2019 [2]. All original malware samples in our dataset were recognized as malicious by all three malware detector models. Following Anderson et al. [3] and Fleshman et al. [23],23, we adopted the *evasion rate* as our evaluation criterion. The evasion rate of an AMG method against a given malware detector is defined as

$$Evasion\ Rate = |E \cap F|/N \tag{4}$$

where E and F denote the sets of evasive and functional modified malware obtained from the AMG method, respectively; and N denotes the total number of malware samples given as input to the AMG method. This statistic yields the efficacy of a given method in evading a malware detector. We later use this metric to evaluate Mal-IndRNN against benchmark methods. We performed two sets of internal and external evaluations and systematically conducted five experiments (three internal and two external) as described in Table 3.

Internal Evaluations. Experiments 1.1, 1.2, and 1.3 are designed to verify the design of Mal-IndRNN by conducting a series of *internal* evaluations of the model. This set of experiments aims to validate the design by providing preliminary evidence that shows Mal-IndRNN operates as expected in practice. Accordingly, we note that the purpose of Experiment 1 is not comparative benchmark analysis (unlike Experiments 2.1 and 2.2). Experiments 2.2 and 2.3 are dedicated to *external* evaluations to compare Mal-IndRNN's performance with alternative benchmark methods on different malware types and against different malware detectors. Experiment 1 aims to thoroughly validate our design and provide empirical evidence of "sign-of-life" for the proposed Mal-IndRNN. Experiment 1.1 empirically verifies whether Mal-IndRNN is capable of evading malware detectors

Table 3. Evaluation overview in AMGD framework.

Evaluation Type	Experiment	Description	Evaluation Metric	Related References
Internal	Experiment 1.1	Verify if Mal-IndRNN can learn to evade detectors during training	# of attempts to evade	[37]
	Experiment 1.2	Verify the effect of modification size and identify the size of generated byte sequences for modification	Evasion Rate	[37]
	Experiment 1.3	Verify Mal-IndRNN's design		[55, 57]
External (Benchmark Evaluation)	Experiment 2.1	Compare to the benchmark AMG methods for evading a single malware detector	Evasion Rate	[12, 13, 55, 62]
	Experiment 2.2	Compare to the benchmark AMG methods for evading multiple malware detectors simultaneously		

during training. Experiment 1.2 examines the effect of the size of generated byte sequences appended to the original malware executable. Experiment 1.3 compares IndRNN, the building block of Mal-IndRNN, with other RNN alternatives, including LSTM [57] and GRU [55].

External Evaluations. Experiment 2 compares Mal-IndRNN's performance against recent benchmark AMG methods in evading the described DL-based malware detector architectures (i.e., MalConv, NonNeg, and AvastNet). We compare the evasion rate against individual malware detectors (Experiment 2.1), as well as against several malware detectors simultaneously (Experiment 2.2). To ensure a holistic evaluation, we identified five widely-used AMG benchmarks, including four binary black-box and one black-box methods for which the implementation was made open-source by their authors:

- Random Append (RA) [12, 62],62: Appends sequences of random bytes to the end of a malware sample until the evasion occurs.
- Benign Append (BA) [12]: Appends random sections from benign files to the end of a malware sample until evasion occurs.
- Genetic Adversarial Machine learning Malware Attack (GAMMA) [15]: A renowned binary black-box method optimizing modifications via genetic programming.
- Multi-Arm Bandit (MAB-malware) [61]: A recently proposed binary black-box method based on multi-arm bandit algorithm that optimizes adversarial modifications.
- Enhanced Benign Append (EBA) [13]: Injects specific byte sequences that lower the confidence score of the anti-malware in a brute-force manner. It is worth noting that since EBA requires access to the confidence score, it is qualified as black-box and has an unfair advantage compared to the other two benchmarks and our proposed method.

To gain insight into each specific malware type in Experiment 2, we train Mal-IndRNN's performance on each malware type separately. Utilizing the functionality assessment process described earlier, we checked the functionality of all modified malware executables to ensure they retain their functionality after modification.

Results

We note that that all experiments reported in our results took less than 5.5 hours to complete on our testbed dataset with tens of thousands of files on a single NVIDIA RTX 2080 GPU, allowing for frequent re-training of the model at a low cost.

Experiment 1.1: Verifying the Behaviour of Training

It is crucial to verify if a machine learning model learns during training by monitoring the training loss or number of iterations required to solve the problem at hand. In order to assess whether Mal-IndRNN learns to generate evasive variants, we monitor the number of iterations (i.e., attempts) required for evasion during the training of Mal-IndRNN. For this experiment, we considered adversarial malware example generation against MalConv.

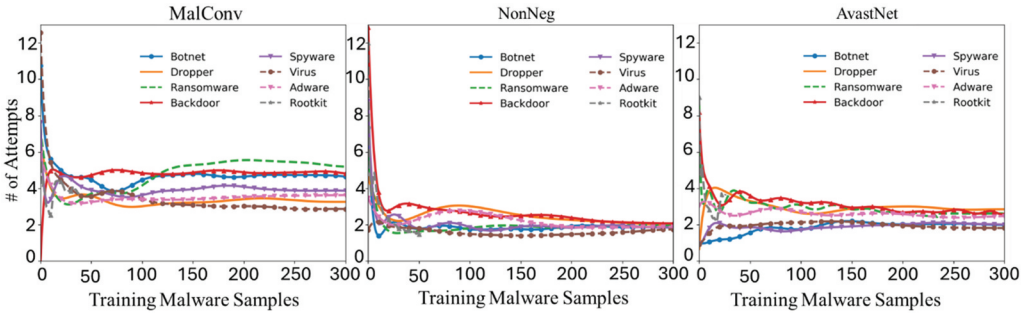

Figure 3. Running average of the number of iterations required to evade MalConv (left), NonNeg (middle), and AvastNet (right) for each malware type.

Figure 3 shows the running average of the number of iterations required to evade malware detectors for each malware sample. As shown in Figure 3, when training starts, Mal-IndRNN needs between 8 and 12 attempts to modify a given malware sample such that it can evade the malware detector. However, as the training proceeds, this number significantly decreases.

As a result, in the latest stages of training (after processing almost 300 malware samples), the number of required attempts reduces to less than a third on average. This behavior is consistent among all eight malware types and across all malware detectors, suggesting that Mal-IndRNN improves during the training process and learns to generate evasive content to be appended to the original malware executable.

Experiment 1.2: Verifying the Effect of Modification Size

Generating an excessively large number of bytes to append to the original malware binary can defeat the purpose of developing an effective AMG method [37]. It is necessary to accomplish evasion through minimal modifications of the original malware to avoid large-scale modifications that could be an indicator of anomaly or maliciousness. Thus, in practice, it is crucial to limit the maximum modification size of AMG methods [37]. To empirically observe the effect of append size on the evasion rate, we track the changes in evasion rate for various modification sizes in the virus category as it is one of the most damaging malware types. Table 4 summarizes the results of this empirical experiment. Two major observations are made from Table 4. First, it is seen that by adding only 6 KB (5%

Table 4. Mal-IndRNN's evasion rates against MalConv and the number of required training iterations at different modification sizes for virus.

Modification Rate (%)	Modification Size (KB)	# of Evaded Samples	Evasion Rate (%)
5	6.0	543	82.40
10	10.0	613	93.09
20	25.0	628	95.25
40	66.0	645	97.84
80	132.8	648	98.27
100	166.0	649	98.49
120	199.2	654	99.24
180	298.8	655	99.46

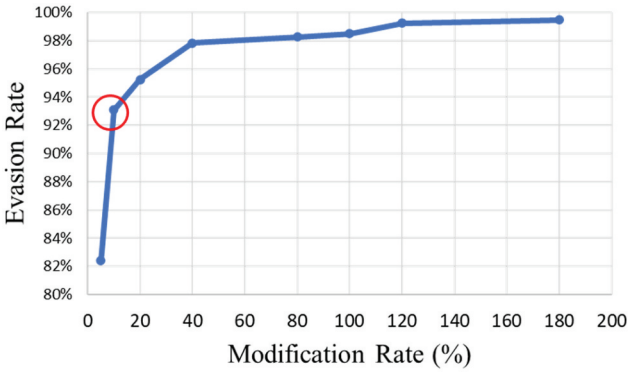

Figure 4. Changes of Mal-IndRNN's evasion rate against MalConv by modification rate.

modification rate) on average to an original malware binary, Mal-IndRNN is able to achieve an evasion rate of 82.4%.

This speaks to the effectiveness of the bytes generated by the proposed method, as will be thoroughly examined later in Experiment 2. Second, and more importantly, as the modification rate increases from 5% to 10% of the original malware size, the evasion rate rapidly increases significantly by approximately 11% (from 82.40% to 93.09%). After this point, the rate of increase does not drastically change and almost stabilizes. Figure 4 visualizes this pattern by plotting the evasion rate against the changes in modification rate.

The 10% modification rate is shown as an elbow point, denoting where the evasion rate stops to increase significantly. Accordingly, we fixed the modification rate for all methods in the benchmark evaluations (Experiment 2) to 10%. Moreover, although our model implements the binary black-box threat model, the number of bytes it appends is on par with white-box gradient-based attacks proposed by Kolosnjaji et al. [37] (i.e., 10% in achieving a 60-70% evasion rate).

Experiment 1.3: Verifying Mal-IndRNN Design

Among the RNN architectures, LSTM and GRU are two alternative RNN architectures used in the malware domain for processing API call sequences resulting from dynamic analysis [57] or processing representations obtained from parts of the malware executables [55]. To justify utilizing IndRNN as the building block of our proposed Mal-IndRNN, we compared the evasion rate against MalConv in three different variations of Mal-IndRNN architecture: (1) implemented with LSTM, (2) implemented with GRU, and (3) implemented with IndRNN (as described in our proposed design). The results of our experiment are summarized in Table 5.

Table 5. Mal-IndRNN's evasion rates against MalConv with different architectures (GRU, LSTM, and IndRNN).

Mal-IndRNN Architecture	# of Evaded Samples	Evasion Rate (%)
Mal-IndRNN with GRU	488	74.11
Mal-IndRNN with LSTM	534	81.03
Mal-IndRNN with IndRNN	544	82.55*

Note: Asterisks denote statistical significance with p-values < 0.01.

As shown in Table 5, the Mal-IndRNN implementation with GRU yields 488 evasive malware variants (evasion rate of 74.11%). Mal-IndRNN's implementation with LSTM leads to 534 evasive malware variants (evasion rate of 81.03%). Finally, Mal-IndRNN's implementation with IndRNN yields 544 evasive malware variants (evasion rate of 82.55%). The results are statistically significant based on a paired *t*-test. IndRNN's significantly higher performance compared with LSTM and GRU confirms the ability of IndRNN to better handle the long-range byte sequences in malware executables, as was expected in our design. We next evaluate Mal-IndRNN's performance against other AMG alternatives.

Experiment 2: Comparing Mal-IndRNN to the State-of-the-Art Black-Box Benchmark Methods

We conducted two series of benchmark evaluations to compare the performance of our proposed Mal-IndRNN against the identified benchmark AMG methods (i.e., Random Append (RA), Benign Append (BA), GAMMA, MAB-Malware, and Enhanced Benign Append (EBA) [12,13,15,61,62,13,15,61,62). While Experiment 2.1 compares Mal-IndRNN's ability to conduct targeted attacks against a specific DL-based malware detector (i.e., MalConv, NoNeg, or AvastNet), Experiment 2.2 assesses Mal-IndRNN's capability to evade at least two of the three malware detectors simultaneously. In both experiments, we conducted benchmark evaluations separately on each malware type to gain cybersecurity-relevant insights.

Experiment 2.1: Evading Individual Malware Detectors

We summarize the results of comparing the performance of Mal-IndRNN with benchmark methods (RA, BA, GAMMA, MAB, and EBA) against individual malware detectors (MalConv, NonNeg, and AvastNet) across eight malware types in Table 6. Benchmark methods that yielded 0% evasion rate across all categories are not shown in the table. Statistical significance was established against the second best-performing method (i.e.,

Table 6. Comparing the evasion rate of the proposed Mal-IndRNN with different AMG benchmarks against individual malware detectors (MalConv, NonNeg, or AvastNet).

		Malware Type								
Model	AMG Method	Adware (%)	Backdoor (%)	Botnet (%)	Dropper (%)	Ransom-ware (%)	Rootkit (%)	Spyware (%)	Virus (%)	Average (%)
MalConv	RA	2.52	1.03	2.47	4.65	3.33	7.55	2.50	3.79	3.48
	BA	7.45	2.51	9.51	9.29	2.89	7.55	10.31	14.42	7.99
	GAMMA	12.63	13.26	13.28	7.39	14.11	7.40	9.55	18.97	12.07
	MAB	27.79	34.50	34.60	24.20	24.26	30.50	33.33	35.18	30.55
	EBA	46.53	30.09	29.85	52.43	35.00	47.17	40.47	50.38	41.49
	Mal-IndRNN	**61.27***	**57.37***	**74.52***	**74.34***	**98.44***	**73.58***	**67.66***	**82.55***	**73.72***
NonNeg	RA	0.26	0.00	0.00	0.44	0.11	1.89	0.47	0.3	0.43
	BA	4.88	2.06	4.94	5.86	2.33	7.55	5.94	6.07	4.95
	GAMMA	9.07	4.30	4.21	11.00	11.25	4.89	10.98	24.13	9.98
	MAB	23.01	12.99	12.98	14.30	18.64	15.73	17.68	30.35	18.21
	EBA	61.79	50	52.66	62.39	51.33	77.36	56.09	59.18	58.85
	Mal-IndRNN	**78.68***	**95.54***	**98.29***	**96.02***	**97.00***	**92.45***	**95.94***	**96.43***	**93.79***
AvastNet	BA	5.61	4.61	6.77	4.91	3.22	24.93	6.02	6.44	7.81
	EBA	38.44	35.93	44.17	38.05	30.87	36.82	41.77	43.53	38.70
	Mal-IndRNN	**55.88***	**48.21***	**73.39***	**54.7***	**32.76***	**50.94***	**66.07***	**71.43***	**56.67***

Note: Asterisks denote p-values < 0.01.

EBA) via paired *t*-test shown by asterisks. Wherever applicable the best setting reported in the original paper or source code of the benchmark methods were used. The highest performances are shown in bold font. From Table 6, it is observed that Mal-IndRNN outperforms all other AMG benchmarks in all malware types for all three malware detectors with statistically significant margins.

Interestingly, not only does Mal-IndRNN outperform its binary black-box AMG counterparts, but it also outperforms EBA, which has access to the confidence scores. Mal-IndRNN achieves on average approximately 30% (73.72% vs. 41.49% for EBA) more evasive malware variants against MalConv, almost 35% (93.79% vs. 58.85% for EBA) against NonNeg, and approximately 18% (56.67% vs. 38.70% for EBA) against AvastNet versus the best benchmark AMG method. These significant results confirm the effectiveness of Mal-IndRNN in generating evasive adversarial malware variants compared to the state-of-the-art AMG methods.

Experiment 2.2: Evading Multiple Malware Detectors Simultaneously

Unlike Experiment 2.1, in Experiment 2.2, evasion occurs only if the generated malware variant can successfully evade multiple (at least two or all three) malware detectors. Such a benchmark evaluation allows us to verify Mal-IndRNN's generalizability to different DL-based models (as opposed to targeting a specific malware detector model). Table 7 summarizes the result of Experiment 2.2 for methods that were able to yield non-zero evasion rates on multiple malware detectors (i.e., BA, EBA, and Mal-IndRNN).

Statistical significance was established against the second best-performing method (i.e., EBA) via paired *t*-test shown by asterisks (highest performances are in bold font). As shown in Table 7, Mal-IndRNN significantly outperforms state-of-the-art benchmark methods against all combinations of the three malware detector models. Overall, Mal-IndRNN leads to an average performance increase of of almost 40% (71.34% vs. 30.28% for EBA) in evading MalConv and NonNeg, a 20% increase (53.29% vs. 35.37%) in evading NonNeg and AvastNet, and a 5% increase (41.84% vs. 37.9%) in evading MalConv and AvastNet simultaneously. More importantly, Mal-IndRNN is able to improve the evasion rate by 8% (32.06% vs. 24.15%) when evading all three DL-based malware detectors

Table 7. Comparing the evasion rate of the proposed Mal-IndRNN with different benchmarks against combinations of all three malware detectors (MalConv, NonNeg, and AvastNet).

		Malware Type								
Malware Detector	AMG Method	Adware (%)	Backdoor (%)	Botnet (%)	Dropper (%)	Ransom-ware (%)	Rootkit (%)	Spyware (%)	Virus (%)	Average (%)
MalConv and NonNeg	EBA	29.58	41.74	13.31	25.77	41.56	22.64	39.06	27.47	30.28
	Mal-IndRNN	**61.02***	**54.72***	**73.38***	**72.46***	**98.44***	**67.92***	**65.47***	**78.00***	**71.34***
NonNeg and AvastNet	BA	1.4	1.06	1.27	2.2	0.33	1.74	2.12	3.48	1.69
	EBA	39.55	28.37	45.29	36.1	22.14	32.08	37.09	37.94	34.88
	Mal-IndRNN	**53.67***	**42.92***	**82.7***	**51.22***	**26.22***	**47.17***	**62.81***	**70.41***	**54.49***
MalConv and AvastNet	EBA	34.64	33.34	51.48	39.7	27.46	25.88	40.96	51.18	38.06
	Mal-IndRNN	**37.08***	29.35	**66.16***	**41.26***	26.22	**37.74***	**48.13***	**65.71***	**41.84***
MalConv, NonNeg and AvastNet	EBA	25.32	18.58	37.26	25.44	11.67	20.75	29.22	26.56	24.33
	Mal-IndRNN	**29.41***	**20.54***	**46.79***	**39.32***	**12.07**	**22.64***	**34.82***	**55.36***	**32.06***

Note: Asterisks denote p-values < 0.01.

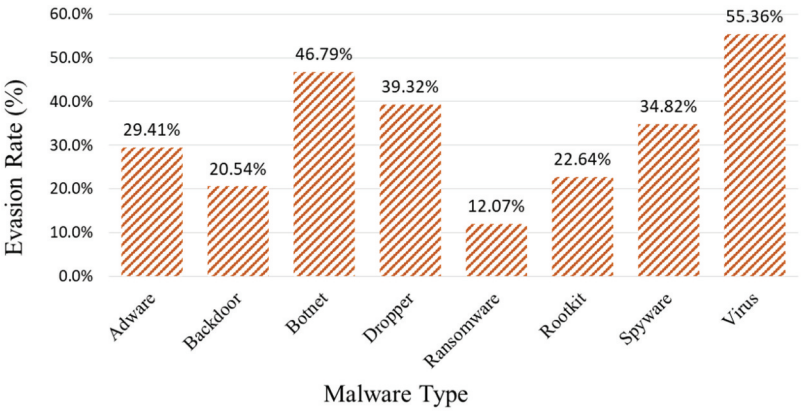

Figure 5. Mal-IndRNN's evasion rate on simultaneously evading all three DL-based detectors (MalConv, NonNeg, and AvastNet).

simultaneously. To further facilitate comparisons across each malware type, Figure 5 illustrates Mal-IndRNN's evasion rate for collectively evading all three malware detectors for each malware type.

As shown in Figure 5, Mal-IndRNN has the lowest overall evasion rate for ransomware (12.07%), which may suggest that this type of malware is more resistant to AMG. This could be attributed to the fact that ransomware executables often have large sections dedicated to data encryption routines and libraries, which render adversarial append modifications less effective, thus allowing this type of malware to be more resistant to AMG. On the contrary, it is also observed that Mal-IndRNN obtained the highest evasion rate (55.36%) for virus malware type. This may suggest that AMG methods are more effective in generating virus variants automatically. This could be due to the fact that virus executables are often embedded in benign programs, which increases the evasion likelihood of AMG compared to other malware types with conspicuously malicious content such as ransomware. Overall, the promising results confirm the generalizability of Mal-IndRNN's design in generating evasive adversarial malware variants against multiple DL-based malware detectors without prior knowledge.

Case Study

Following the guidelines of design science research [25, 49],49, we conduct a case study to assess the applicability of our design. The case study component of our AMGD framework involves utilizing the generated adversarial malware variants in improving the resistance of DL-based malware detectors. To this end, we employed the most common defense mechanism against adversarial evasion attacks, known as adversarial training [51]. Adversarial training for malware starts with generating evasive malware variants via an AMG method (e.g., Mal-IndRNN). Next, the malware detector model is re-trained on a training set that is augmented by the generated adversarial malware variants. Lastly, the re-trained model is tested on unseen malware variants that could not be detected by the original detector to verify the performance improvement in detecting previously undetected malware executables [57]. Accordingly, to conduct the case study, we first generated 6,037 malware

Table 8. Malware detection performance before and after re-training with adversarial malware variants generated by Mal-IndRNN and MAB.

	Mal-IndRNN			Multi-Arm Bandit (MAB)		
Malware Type	Original MalConv (%)	Re-trained MalConv (%)	Performance Increase (%)	Original MalConv (%)	Re-trained MalConv (%)	Performance Increase (%)
Adware	61.27	10.79	50.48	27.79	4.15	23.64
Backdoor	57.37	6.34	51.03	34.50	22.82	11.68
Botnet	74.52	23.38	51.14	34.60	11.63	22.97
Dropper	74.34	14.93	59.41	24.20	10.63	13.57
Ransomware	98.44	12.78	85.66	24.26	9.00	15.26
Rootkit	73.58	9.43	64.15	30.50	2.31	28.19
Spyware	67.66	17.03	50.63	33.33	8.18	25.15
Virus	82.55	31.26	51.29	35.18	24.35	10.83
Average	73.72	15.74	57.98	30.55	11.64	18.91

variants via Mal-IndRNN and re-trained MalConv in the presence of these adversarial variants. The distribution of these generated malware variants is similar to that of our development set, as detailed in Table 2. We then measured the malware detection performance of both original and re-trained MalConv on our evaluation set of 11,597 generated unseen malware variants (detailed in Table 2). The performance of the re-trained malware detector against adversarial malware variants was compared to its original performance via performance increase in percentage—the difference in malware detection performance between the original and re-trained MalConv compared to the detection performance of the original MalConv [43]. A higher performance increase ratio indicates a higher adversarial attack resistance resulted from retraining and thus a more valuable AMG method for adversarial training. Our case study serves as a proof-of-value for improving the malware detection performance of DL-based malware detectors against unseen adversarial malware variants. To this end, consistent with Rosenberg et al. [57], we utilized our evaluation set (detailed above) to conduct the adversarial training procedure [51]. This process involves re-training the malware detector model in the presence of adversarial malware variants generated by Mal-IndRNN. We summarize the case study results in Table 8.

As shown in Table 8, as expected, the evasion rate decreases after re-training the malware detector with both Mal-IndRNN's and MAB's generated malware variants. However, re-training with Mal-IndRNN yields a higher performance increase ratio across all malware types, indicating a more effective adversarial training process compared to the one conducted with MAB. Consequently, re-training with Mal-IndRNN samples increased malware detection performance by 57.98% on average across all malware types, compared to an increase of 18.91% when re-trained with MAB samples. To visualize the difference before and after re-training, we plot the undetected attack rate before and after adversarial training for each malware type using Mal-IndRNN samples (Figure 6).

In each category, the undetected attack rate before and after adversarial training is shown by diagonal and horizontal stripes, respectively. As seen in Figure 6, Ransomware is associated with the highest performance improvement (75.66%). Given that the cost of global ransomware damages is estimated to be $20 billion in 2021 [47], enhancing DL-based detectors' resilience can translate to preventing sizable losses and could be an effective step towards automated, proactive cyber defense.

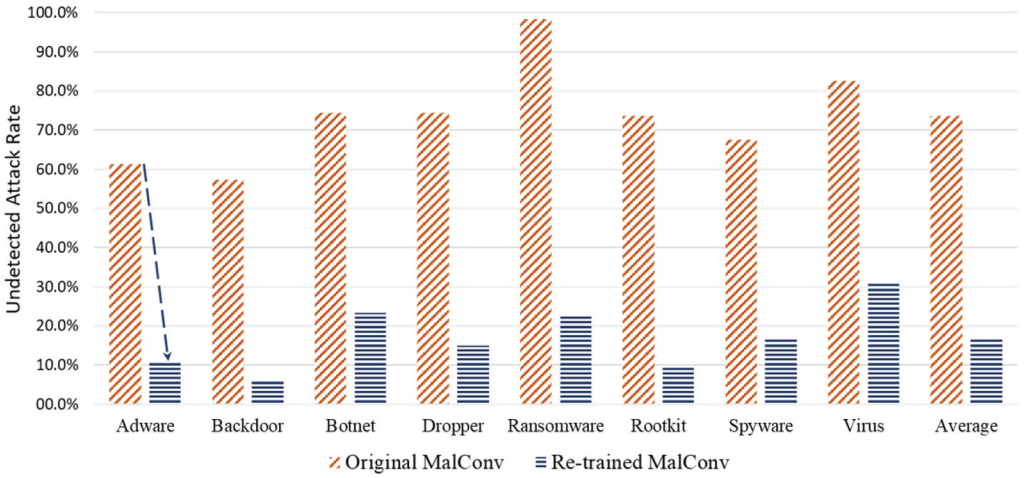

Figure 6. Comparison of malware detection performance in MalConv before (diagonal stripes) and after (horizontal stripes) adversarial training.

Contributions to the IS Knowledge Base and Managerial Implications for Cybersecurity Analytics

Informed by the computational design science paradigm [25, 56],56 and the increasing adoption of advanced DL-based malware detectors for protecting IT infrastructure, our study focuses on the emerging onslaught of AI-enabled adversarial malware attacks against DL-based cyber infrastructure. To this end, we systematically examined viable solutions and designed a novel IT artifact that aims to address this issue. The proposed IT artifact features an AMGD framework that improves automated cyber defense by generating realistic adversarial malware attacks. AMGD was systematically evaluated on state-of-the-art benchmark methods, and its proof-of-value was demonstrated through a real-world case study. We summarize our IS contributions to the cyber defense domain and methodology categorized by their stakeholders, recommended design guides, and corresponding evidence from our work in Table 9.

Managers and Security Practitioners

Our study highlights the necessity of accounting for the AI-enabled adversary in the design and development phase rather than after deployment. The arms race between the adversary and the defender is a long-standing problem that has been exacerbated by the emergence of AI-enabled adversaries. In such a hostile environment, our study establishes that in order to stay one step ahead of the adversary, the defenders need to implement AI-enable defenses that emulate adversaries during the design time. If defenders are equipped with such methods, they can adjust their models periodically to account for *evolving adversaries* in a much more effective manner. Staying one step ahead of the AI-enabled adversary requires *proactively and automatically* developing frequent effective temporary fixes such that the defender can respond quickly to adaptations from adversaries.

Table 9. Contribution to the IS knowledge base.

Contribution Category	Stakeholder	Design Guide	Evidence from this Study
Cyber Defense Domain	Managers and security practitioners	• Including the adversary and its threat model in the design and development phase of critical cyber defense systems is necessary to proactively and automatically stay ahead of the AI-enabled adversary.	• Experiments 1.1, 1.2, and 2.1 • Our Case study
	Security practitioners	• Integrating attack and defense during training, adversarial training, is a powerful approach towards protecting critical cyber defense systems.	• Our case study
Methodological	IS researchers	• Novel adversarial training methods that allow simultaneous training against several cyber defense systems can be significantly more generalizable than their counterparts that are only trained against a single defense system.	• Experiment 2.2
	IS researchers	• Applications that involve processing long sequence data such as raw malware bytes can benefit from carefully designed independent sequence models such as independent RNNs.	• Experiments 1 and 2 • Our Case study

Abbreviations: IS, information systems; RNN, recurrent neural nets.

We show that a careful coupling of sequence modeling-based attacks with adversarial training is a powerful tool to defend against AI-enabled adversaries in cyber defense. Additionally, security practitioners and malware analysts who design modern malware detectors can utilize the proposed Mal-IndRNN as an evaluation tool to evaluate the resistance of their malware detector models against adversarial malware attacks during development before deploying and releasing their models in a production environment.

For cybersecurity analytics firms that provide defense solutions for IT infrastructure, we caution against the premature adoption of modern DL-based malware detectors without implementing cybersecurity analytics frameworks such as AMGD to mitigate the risk of adversarial malware attacks. For example, enhancing the robustness in detecting ransomware by applying the AMGD framework on nascent malware detectors could help alleviate the continuous onslaught of AI-enabled attacks against the healthcare industry (e.g., Scripps Health in San Diego malware attack [48]) and other critical cyber infrastructures (e.g., Colonial Pipeline ransomware attack [63]).

IS Researchers

Adversarial training methods that are capable of simultaneously training on multiple cyber defense models can be significantly more generalizable than the ones that are only trained against a particular defense model. To achieve this in Cyber defense applications that involve processing long sequence data such as raw malware bytes, it is beneficial to carefully design independent sequence models such as independent RNNs in Mal-IndRNN. While the adversarial aspects of cyber defense have been recognized in behavioral and economic IS cybersecurity research, prior design science studies on developing automated cyber defense rarely examine the effectiveness of their proposed ML-based models in adversarial environments. Our study serves as an early step towards addressing this gap. Our AMGD framework contributes to IS cybersecurity research with a novel design for adversarial malware attack generation. At the heart of our AMGD design stands a novel Mal-IndRNN

method that enables realistic malware attack generation without requiring any prior knowledge about the specifications of the DL-based malware detectors. Mal-IndRNN is uniquely positioned in the AMG literature due to its ability to evade several DL-based anti-malware models simultaneously. Both AMGD and Mal-IndRNN contribute to automated cyber defense for protecting IT infrastructure against some of the costliest attack types (e.g., malware).

Ethical Considerations

It is crucial to attend to the dual-use nature of our study. Specifically, Mal-IndRNN contributes to generating adversarial attacks as a viable defense mechanism and gaining insight into the adversary's capabilities. Although the goal of our study is improving malware detectors, precautionary measures should be taken to monitor and prevent potentially large-scale misuse of such frameworks during the deployment of technology. Software-as-service deployment is one way to establish usage monitoring to help ensure the benevolent usage of the technology outweighs its potential malicious misuse.

To minimize the risk of malicious use of the generated adversarial malware variants, the generated malware variants in our study are stored and maintained in a securely isolated repository behind the UArizona's internal firewall. While external access to the storage is blocked, local password-protected access is granted via Eller virtual private network, which is maintained and monitored by UArizona's Eller IT support to ensure authenticated access to the repository and preventing unauthorized external access.

Conclusion and Future Directions

Today, malware attacks rank first among cyber-attacks in terms of their cost incurred to firms. Recently, DL-based malware detectors have shown promise in protecting IT infrastructure against malware attacks by enabling the rapid detection of unseen malware without manual rule definition, feature engineering, and expensive dynamic analysis. However, DL-based malware detectors can be vulnerable to large-scale, automated adversarially generated malware attacks. Adversarial malware example generation can be used by defenders to improve the resiliency of these malware detectors against future adversarial attacks. Automating this process is crucial for improving DL-based malware detectors. Current approaches to this end unrealistically assume full or partial knowledge about the targeted malware detector. In this study, we presented an AMGD framework that views adversarial malware example generation as sequence modeling. Within this framework, we developed a novel Mal-IndRNN method to generate adversarial examples without requiring prior knowledge of the targeted malware detector. Mal-IndRNN directly learns a sequence model on the malware executable's content and generates effective benign-looking byte sequences that can evade several renowned DL-based malware detectors simultaneously. Mal-IndRNN is suitable for adversarial malware example generation in the binary black-box settings described in our threat model because it neither depends on the internal architecture nor the confidence score of the targeted malware detector. Mal-IndRNN performance was evaluated against the state-of-the-art benchmark AMG methods. AMGD's proof-of-value in improving the detection of ransomware variants was demonstrated through a case study of different malware

types. Finally, contributions to the IS knowledge base and managerial implications were discussed. Our results demonstrate the vulnerability of current DL-based malware detector models to adversarial attacks and show that significant future research in this area is needed. A promising future research direction is developing automated defense mechanisms that can continuously improve the resilience of malware detectors as they encounter adversarial malware variants. Another fruitful research direction could be devising interpretable AMG methods that show the contribution of modifications leading to evasion. Such methods could reveal the most evasive perturbations that malware detectors need to guard against.

Notes

1. Due to their recent success, we focus on DL-based cyber defense models as a sub-category of ML-based models.

Acknowledgments

We would like to thank VirusTotal for providing the malware dataset and granting access to their APIs for malware functionality assessment. This study is based upon work supported by the National Science Foundation (NSF) under the grants CNS-1936370 (SaTC CORE) and DGE-1921485 (SFS).

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This work was supported by the National Science Foundation (NSF) [1921485]; National Science Foundation (NSF) [1936370].

Notes on contributors

Reza Ebrahimi (ebrahimim@usf.edu; corresponding author) is Assistant Professor and the founder of Secure Trustworthy and Reliable (STAR)-AI Lab in the School of Information Systems and Management at the University of South Florida. He received his PhD in Management Information Systems from the University of Arizona, where he was a research associate at the Artificial Intelligence (AI) Lab. His PhD dissertation on AI-enabled cybersecurity analytics received the ICIS ACM SIGMIS best doctoral dissertation award. His current research is focused on statistical and adversarial machine learning theories for AI-enabled secure and trustworthy cyberspace. Dr. Ebrahimi has published over 40 papers in peer reviewed journals, conferences, and workshops including *MIS Quarterly, Journal of Management Information Systems*, NeurIPS, numerous *IEEE Transactions, Applied Artificial Intelligence, Digital Forensics*, and other venues. He has served as a Program Chair and Program Committee member of several IEEE workshops and is an organizer of 2025 IEEE S&P Workshop on Human-Machine Intelligence for Security Analytics. He has contributed to several projects supported by the National Science Foundation.

James Hu (jameshu@email.arizona.edu) is working toward the PhD degree in the Artificial Intelligence (AI) Lab, University of Arizona. His research interests lie in the application of adversarial machine learning to cybersecurity domains including static malware analysis, adversarial malware

generation, safety of foundation AI models, and attacking and jailbreaking of large language models. His work on adversarial malware generation has been published in *IEEE Transactions* and workshop proceedings.

Ning Zhang (zhangning@arizona.edu) is a Lead Engineer and Machine Learning Researcher at Acium, Inc. He holds a BS in Electrical and Computer Engineering and an MS in Management Information System from the University of Arizona. His research focuses on the application of artificial intelligence in cybersecurity, deep learning algorithms, and data infrastructure. With a background in both engineering and business, he bridges technical depth with strategic insight. His current work involves developing a risk assessment and data breach prevention system for web browsing environments using deep learning techniques.

Jay F. Nunamaker Jr. (jnunamaker@cmi.arizona.edu) is Regents and Soldwedel Professor of MIS, Computer Science and Communication, and director of the Center for the Management of Information and the National Center for Border Security and Immigration at the University of Arizona. He received his PhD in Operations Research and Systems Engineering from Case Institute of Technology. Dr. Nunamaker has held a professional engineer's license since 1965. He was inducted into the Design Science Hall of Fame and received the LEO Award for Lifetime Achievement from the Association for Information Systems. He was featured in the July 1997 issue of Forbes Magazine on technology as one of eight key innovators in information technology. Dr. Nunamaker's specialization is in the fields of system analysis and design, collaboration technology, and deception detection. The commercial product GroupSystems ThinkTank, based on his research, is often referred to as the gold standard for structured collaboration systems. He founded the MIS Department at the University of Arizona and served as department head for 18 years.

Hsinchun Chen (hchen@eller.arizona.edu) is Regents Professor at the University of Arizona. He received his PhD in Information Systems from New York University. He is a Fellow of ACM, AAAS, and AIS. He founded the Artificial Intelligence Lab at the University of Arizona, which has received $60M+ research funding from NSF and other agencies (100+ grants, 50+ from NSF, as Principal Investigator). He received numerous awards and has served as editor-in-chief, senior editor or associate editor of several scholarly journals. Dr. Chen's COPLINK system, which has been quoted as a national model for public safety information sharing and analysis, has been adopted in more than 550 law enforcement and intelligence agencies in 20 US states. He is the director of the AZSecure Cybersecurity Program at the University of Arizona, supported by multiple grants.

ORCID

Reza Ebrahimi http://orcid.org/0000-0003-1367-3338

References

1. Abbasi, A.; Zhang, Z.; Zimbra, D.; Chen, H.; and Nunamaker Jr, J.F. Detecting fake websites: The contribution of statistical learning theory. *MIS Quarterly*, *34*, 3 (2010), 435–461.
2. Anderson, H.S. Machine Learning Static Evasion Competition. GitHub, 2019. https://github.com/endgameinc/malware_evasion_competition (accessed on September 30, 2025).
3. Anderson, H.S.; Kharkar, A.; Filar, B.; Evans, D.; and Roth, P. Learning to evade static PE machine learning malware models via reinforcement learning. *arXiv preprint arXiv:1801.08917*, (2018).
4. Belletti, F.; Chen, M., and Chi, E.H. Quantifying long range dependence in language and user behavior to improve RNNs. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Anchorage, Alaska, USA: ACM, 2019, pp. 1317–1327.
5. Benjamin, V.; Valacich, J.; and Chen, H. DICE-E: A Framework for Conducting Darknet Identification, Collection, Evaluation with Ethics. *MIS Quarterly*, *43*, 1 (2019), 1–22.

6. Benjamin, V.; Zhang, B.; Nunamaker Jr, J.F.; and Chen, H. Examining hacker participation length in cybercriminal internet-relay-chat communities. *Journal of Management Information Systems*, *33*, 2 (2016), 482–510.
7. Bissell, K.; LaSalle, R.M., and Cin, P.D. *Ninth Annual Cost of Cybercrime Study: Unlocking the Value of Improved Cybersecurity Protection*. Accenture and Ponemon, 2019. https://www.accenture.com/us-en/insights/security/cost-cybercrime-study accessed on September 30, 2025).
8. Bloomberg. Symantec Unveils Industry's First Neural Network to Protect Critical Infrastructure from Cyber Warfare. Bloomberg, 2018. https://www.bloomberg.com/press-releases/2018-12-05/symantec-unveils-industry-s-first-neural-network-to-protect-critical-infrastructure-from-cyber-warfare (accessed on January 30, 2020).
9. Braue, D. Cybercrime to cost the world $12.2 trillion annually by 2031. Cybercrime Magazine, 2025.
10. Brown, T.; Mann, B.; Ryder, N., Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; and Shyam, P. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, *33*, (2020), 1877–1901.
11. Burns, A.; Roberts, T.L.; Posey, C.; and Lowry, P.B. The adaptive roles of positive and negative emotions in organizational insiders' security-based precaution taking. *Information systems Research*, *30*, 4 (2019), 1228–1247.
12. Castro, R.; Schmitt, C.; and Rodosek, G.D. ARMED: How automatic malware modifications can evade static detection? In *International Conference on Information Management (ICIM)*. Cambridge, UK: IEEE, 2019, pp. 20–27.
13. Chen, B.; Ren, Z.; Yu, C.; Hussain, I.; and Liu, J. Adversarial examples for CNN-based malware detectors. *IEEE Access*, *7*, (2019), 54360–54371.
14. Chen, J.; Yuan, C.; Li, J.; Tian, D.; Ma, R.; and Jia, X. ELAMD: An Ensemble Learning Framework for Adversarial Malware Defense. *Journal of Information Security and Applications*, *75*, (2023), 103508.
15. Demetrio, L.; Biggio, B.; Lagorio, G.; Roli, F.; and Armando, A. Functionality-preserving black-box optimization of adversarial windows malware. *IEEE Transactions on Information Forensics and Security*, *16*, (2021), 3469–3478.
16. Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *CoRR abs/1810.04805*, (2018).
17. Dey, S.; Kumar, A.; Sawarkar, M.; Singh, P.K.; and Nandi, S. EvadePDF: Towards evading machine learning based pdf malware classifiers. In *International Conference on Security & Privacy*. Singapore: Springer, 2019, pp. 140–150.
18. Ebrahimi, M.; Chai, Y.; Samtani, S.; and Chen, H. Cross-lingual cybersecurity analytics in the international dark web with adversarial deep representation learning. *MIS Quarterly*, *46*, 2 (2022), 1209–1226.
19. Ebrahimi, R.; Chai, Y.; Li, W.; Pacheco, J., and Chen, H. 'RADAR: A framework for developing adversarially robust cyber defense AI agents with deep reinforcement learning. *MIS Quarterly*, 2025.
20. Ebrahimi, M.; Nunamaker Jr, J.F.; and Hsinchun, C. Semi-supervised cyber threat identification in dark net markets: A transductive and deep learning approach. *Journal of Management Information Systems*, *37*, 3 (2020), 694–722.
21. Ebrahimi, R.; Pacheco, J.; Hu, J.; and Chen, H. Learning contextualized action representations in sequential decision making for adversarial malware optimization. *IEEE Transactions on Dependable and Secure Computing*, *22*, 3 (2025), 2086–2102.
22. Fang, Z.; Wang, J.; Li, B.; Wu, S.; Zhou, Y.; and Huang, H. Evading anti-malware engines with deep reinforcement learning. *IEEE Access*, *7*, (2019), 48867–48879.
23. Fleshman, W.; Raff, E.; Sylvester, J.; Forsyth, S.; and McLean, M. Non-negative networks against adversarial attacks. *arXiv preprint arXiv:1806.06108*, (2019).
24. Goodfellow, I.; McDaniel, P.; and Papernot, N. Making machine learning robust against adversarial inputs. *Communications of the ACM*, *61*, 7 (2018), 56–66.
25. Gregor, S.; and Hevner, A.R. Positioning and presenting design science research for maximum impact. *MIS Quarterly*, *37*, 2 (2013), 337–355.

26. Han, D.; Wang, Z.; Zhong, Y., Chen, W.; Yang, J.; Lu, S.; Shi, X.; and Yin, X. Practical traffic-space adversarial attacks on learning-based Nidss. *arXiv preprint arXiv:2005.07519*, (2020).
27. Hevner, A.R.; March, S.T.; Park, J.; and Ram, S. Design science in information systems research. *MIS Quarterly*, *28*, 1 (2004), 75–105.
28. Hu, W.; and Tan, Y. Black-box attacks against RNN based malware detection algorithms. In *AAAI Conference on Artificial Intelligence*. The AAAI Press, Palo Alto, California, 2018, pp. 245–255.
29. Hu, W;. and Tan, Y. Generating adversarial malware examples for black-box attacks based on GAN. In *International Conference on Data Mining and Big Data*. Singapore: Springer Nature Singapore, 2022, pp. 409–423.
30. Hui, K.-L.; Vance, A., and Zhdanov, D. Securing digital assets. 2018. https://static1.squarespace.com/static/5887a660b3db2b05bd09cf36/t/5deff7043d673974a391656f/1576007428557/MISQ+Curation+Securing+Digital+Assets+July+2018.pdf (accessed on September 30, 2025).
31. Irie, K.; Prabhavalkar, R.; Kannan, A.; Bruguier, A.; Rybach, D.; and Nguyen, P. On the choice of modeling unit for sequence-to-sequence speech recognition. *Interspeech 2019*, (2019), 3800–3804.
32. Kargarnovin, O.; Sadeghzadeh, A.M.; and Jalili, R. Mal2gcn: A robust malware detection approach using deep graph convolutional networks with non-negative weights. *Journal of Computer Virology and Hacking Techniques*, *20*, 1 (2024), 95–111.
33. Karhu, K.; Gustafsson, R.; and Lyytinen, K. Exploiting and defending open digital platforms with boundary resources: Android's five platform forks. *Information Systems Research*, *29*, 2 (2018), 479–497.
34. Katharopoulos, A.; Vyas, A.; Pappas, N.; and Fleuret, F. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*. PMLR, 2020, pp. 5156–5165.
35. Keles, F.D.; Wijewardena, P.M., and Hegde, C. On the computational complexity of self-attention. In *International Conference on Algorithmic Learning Theory*. Singapore: PMLR, 2023, pp. 597–619.
36. Kim, Y.; Jernite, Y.; Sontag, D., and Rush, A.M. *Character-aware neural language models*. In *AAAI Conference on Artificial Intelligence*. New Orlean, LA: AAAI, 2016, pp. 2741–2749.
37. Kolosnjaji, B.; Demontis, A.; Biggio, B.; et al. Adversarial malware binaries: Evading deep learning for malware detection in executables. In *26th European Signal Processing Conference (EUSIPCO)*. Rome, Italy: IEEE, 2018, pp. 533–537.
38. Krčál, M.; Švec, O.; Bálek, M.; and Jašek, O. Deep convolutional malware classifiers can learn from raw executables and labels only. In *International Conference on Learning Representations (ICLR)*. British Columbia, Canada: ICLR, 2018.
39. Kyadige, A.; Rudd, E., and Berlin, K. Learning from context: A multi-view deep learning architecture for malware detection. In *3rd Deep Learning and Security Workshop*. San Francisco, CA: IEEE, 2020. pp. 1–7.
40. Li, K.; Guo, W.; Zhang, F.; and Du, J. GAMBD: Generating Adversarial Malware Against MalConv. *Computers & Security*, *130*, (2023), 103279.
41. Li, S.; Li, W.; Cook, C.; Gao, Y.; and Zhu, C. Deep independently recurrent neural network (IndRNN). *CoRR, abs/1910.06251*, (2020).
42. Li, S.; Li, W.; Cook, C.; Zhu, C., and Gao, Y. Independently recurrent neural network (IndRNN): Building a longer and deeper RNN. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Salt Lake City, USA: IEEE, 2018, pp. 5457–5466
43. Li, W.; and Chai, Y. Assessing and enhancing adversarial robustness of predictive analytics: An empirically tested design framework. *Journal of Management Information Systems*, *39*, 2 (2022), 542–572.
44. Li, W.; Chen, H.; and Nunamaker Jr, J.F. Identifying and profiling key sellers in cyber carding community: AZSecure text mining system. *Journal of Management Information Systems*, *33*, 4 (2016), 1059–1086.

45. Li, Y.; Wang, Y.; Wang, Y.; Ke, L.; and Tan, Y. A feature-vector generative adversarial network for evading Pdf malware classifiers. *Information Sciences*, *523*, (2020), 38–48.
46. Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; and Vladu, A. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations (ICLR)*. British Colombia, Canada: ICLR, 2018.
47. Morgan, S. Global Ransomware Damage Costs. Cybercrime Magazine, 2019. https://cybersecurityventures.com/global-ransomware-damage-costs-predicted-to-reach-20-billion-usd-by-2021/ (accessed on September 30, 2025).
48. National Broadcasting Company (NBC). Scripps Health CEO Confirms to Staff That Information Systems Damaged by Malware. NBC San Diego, 2021. https://www.nbcsandiego.com/news/local/what-we-know-about-scripps-health-cyberattack/2598969/ (accessed on September 30, 2025).
49. Nunamaker, J.F.; Briggs, R.O.; Derrick, D.C.; and Schwabe, G. The last research mile: achieving both rigor and relevance in information systems research. *Journal of Management Information Systems*, *32*, 3 (2015), 10–47.
50. Ono, J.; Utiyama, M.; and Sumita, E. Hybrid data-model parallel training for sequence-to-sequence recurrent neural network machine translation. In *Proceedings of The 8th Workshop on Patent and Scientific Literature Translation*. European Association for Machine Translation, Dublin, Ireland, 2019, pp. 4–12.
51. Papernot, N.; and McDaniel, P.D. Extending defensive distillation. *arXiv preprint arXiv:1705.05264*, (2017).
52. Park, D.; Khan, H.; and Yener, B. Creating adversarial malware examples using code insertion. *CoRR*, *abs/1904.04802*, (2019).
53. Qiu, S.; Liu, Q.; Zhou, S.; and Wu, C. Review of artificial intelligence adversarial attack and defense technologies. *Applied Sciences*, *9*, 5 (2019), 909–938.
54. Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; and Sutskever, I. Language models are unsupervised multitask learners. OpenAI Blog, 1, 8 (2019), 9.
55. Raff, E.; Barker, J.; Sylvester, J.; Brandon, R.; Catanzaro, B.; and Nicholas, C. Malware detection by eating a whole exe. *arXiv preprint arXiv:1710.09435*, (2018).
56. Rai, A. Editor's Comments: Diversity of design science research. *MIS Quarterly*, *41*, 1 (2017), iii–xviii.
57. Rosenberg, I.; Shabtai, A.; Rokach, L.; and Elovici, Y. Generic black-box end-to-end attack against state of the art api call-based malware classifiers. In *International Symposium on Research in Attacks, Intrusions, and Defenses*. Beijing, China: Springer, 2019, pp. 490–510.
58. Samtani, S.; Chai, Y.; and Chen, H. Linking exploits from the dark web to known vulnerabilities for proactive cyber threat intelligence: An attention-based deep structured semantic model. *MIS Quarterly*, *46*, 2 (2022), 911–946.
59. Samtani, S.; Chinn, R.; Chen, H.; and Nunamaker Jr, J.F. Exploring emerging hacker assets and key hackers for proactive cyber threat intelligence. *Journal of Management Information Systems*, *34*, 4 (2017), 1023–1053.
60. Sharif, M.; Lucas, K.; Bauer, L.; Reiter, M.K.; and Shintre, S. Optimization-guided binary diversification to mislead neural networks for malware detection. *arXiv preprint arXiv:1912.09064*, (2019).
61. Song, W.; Li, X.; Afroz, S.; Garg, D.; Kuznetsov, D., and Yin, H. MAB-Malware: A reinforcement learning framework for black box generation of adversarial malware. In *Proceedings of the 2022 ACM Asia Conference on Computer and Communications Security*. Nagasaki, Japan: ACM, 2022, pp. 990–1003.
62. Suciu, O.; Coull, S.E.; and Johns, J. Exploring adversarial examples in malware detection. In *2019 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2019, pp. 8–14.
63. The Guardian. Colonial Pipeline Confirms It Paid $4.4m Ransom to Hacker Gang After Attack. The Guardian, 2021. https://www.theguardian.com/technology/2021/may/19/colonial-pipeline-cyber-attack-ransom (accessed on September 30 2025).
64. Tolido, R.; Linden, G. van der; Delabarre, L. *Reinventing Cybersecurity with Artificial Intelligence: The new frontier in digital security*. Capgemini Research Institute, 2019. https://

www.capgemini.com/us-en/resource/reinventing-cybersecurity-with-artificial-intelligence-the-new-frontier-in-digital-security/ (accessed on September 30 2025).
65. Tounsi, W.; and Rais, H. A survey on technical threat intelligence in the age of sophisticated cyber attacks. *Computers & Security*, *72*, (2018), 212–233.
66. Usama, M.; Asim, M.; Latif, S.; Qadir, J.; and others. Generative adversarial networks for launching and thwarting adversarial attacks on network intrusion detection systems. In *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*. IEEE, 2019, pp. 78–83.
67. Yang, Z.; Dai, Z.; Yang, Y.; Carbonell, J.; Salakhutdinov, R.R.; and Le, Q.V. XLNet: Generalized autoregressive pretraining for language understanding. *Advances in Neural Information Processing Systems*, *32*, (2019), 5753–5763.
68. Yin, H.S.; Langenheldt, K.; Harlev, M.; Mukkamala, R.R.; and Vatrapu, R. Regulating cryptocurrencies: A supervised machine learning approach to de-anonymizing the bitcoin blockchain. *Journal of Management Information Systems*, *36*, 1 (2019), 37–73.
69. Zhong, F.; Cheng, X.; Yu, D.; Gong, B.; Song, S.; and Yu, J. Malfox: Camouflaged adversarial malware example generation based on conv-GANs against black-box detectors. *IEEE Transactions on Computers*, *73*, 2 (2023), 980–993
70. Zhu, E.; Zhang, J.; Yan, J.; Chen, K.; and Gao, C. N-gram MalGAN: Evading machine learning detection via feature n-gram. *Digital Communications and Networks*, *8*, 4 (2022), 485–491.
71. Zuo, F.; Li, X.; Young, P.; Luo, L.; Zeng, Q.; and Zhang, Z. Neural machine translation inspired binary code similarity comparison beyond function pairs. *arXiv preprint arXiv:1808.04706*, (2018).

Appendix 1. Mal-IndRNN Model Specification

To facilitate reproducibility, we specify Mal-IndRNN parameters used in our experiments. Both encoder and decoder in Mal-IndRNN consist of 100 IndRNN units with Tanh activation functions as their building blocks. All biases of the IndRNN units were initialized to zero. The dimension of the embedding layer (i.e., vocabulary size) in the encoder was set to 256. Also, the size of the fully connected output layer in the decoder was set to 128. Parameter settings of Mal-IndRNN were fixed throughout all experiments. The learning rate and systematic sampling rate were set to 1e-2 and 1e-3, respectively. Also, the batch size was fixed to 10 throughout all experiments. The maximum append size was set to 40% for all benchmark methods and Mal-IndRNN. Lastly, the maximum number of attempts was set to 50 for all benchmark methods and Mal-IndRNN. All parameters were selected by a light manual partial grid search. Table 1.1 summarize parameters and hyperparameters of MalInd_RNN used in our experiments, including the encoder and decoder architecture, number of units in each layer, corresponding activation functions, the shape of output for each layer, and layer-specific hyperparameters such as vocabulary size, and initial bias values.

Table 1.1. Mal-IndRNN's architecture and parameter settings in our experiments.

Layer		# of Units	Activation	Output Shape	Layer-Specific
Encoder	Embedding	256	–	(BatchSize, 256, 100)	Vocab_Size = 256; Window_Size = 2M
	IndRNN	100	Relu	(BatchSize, 100, 100)	Initial_Bias = 0
Decoder	IndRNN	100	Relu	(BatchSize, 100, 100)	Initial_Bias = 0
	FC Layer	128	Softmax	(BatchSize, 100, 256)	Initial_Bias = 0

Max_Append_Size = 40%; Max_Attempt_Num = 50; Generation_Length = 1K.

Table 1.2 summarizes Mal-IndRNN's hyperparmeter settings, including batch size, systematic sampling rate, learning rate, stochastic gradient descent optimizer, loss function, and the number of training epochs for training and testing that was used in our experiments.

Table 1.2. Mal-IndRNN's hyperparameter settings in our experiments.

Hyperparameters		Value
Training	Batch Size	10
	Systematic Sampling Rate	0.001
	Learning Rate	0.01
	Optimizer	Adam
	Loss Function	Cross Entropy Loss
Testing	Batch Size	10

Appendix 2. Empirical Analysis of Computation Time

To empirically measure the computational complexity of our proposed method, we recorded the training time for each malware category (summarized in Table 2.1).

Table 2.1. Mal-IndRNN's empirical time complexity assessment.

	Adware	Backdoor	Botnet	Dropper	Ransom-ware	Rootkit	Spyware	Virus	Average
Time (Seconds)	4316.02	2634.70	2221.71	2927.42	2885.06	2002.21	2512.88	2522.57	2752.82
Time (Minutes)	71.93	43.91	37.03	48.79	48.08	33.37	41.88	42.04	45.88
Time (Hours)	1.20	0.73	0.62	0.81	0.80	0.56	0.70	0.70	0.76

As observed in Table 2.1, the average time of training Mal-IndRNN across all malware categories is less than an hour (45.88 minutes). As expected in our design, this suggests that Mal-IndRNN is carefully designed to be notably efficient in processing raw malware binaries.