Architecture-Level Backdoors in Malware Detection Models for Trustworthy Artificial Intelligence

Ziyi Yu, Xiaobing Xiong, Ju Yang, Fei Kang*
Key Laboratory of Cyberspace Security, Ministry of Education, China
Information Engineering University, Zhengzhou 450001, China
yuziyi_999@163.com, bingxiaoxiong@foxmail.com, yangyang987677@163.com, kfminnie@163.com

***Abstract*—Deep learning–based static malware detection models face severe supply chain security threats. Existing defense strategies against backdoored static malware detection models are largely built upon a core assumption: that fine-tuning contaminated models or retraining them on clean data can effectively remove backdoors. However, our study reveals critical limitations of this widely trusted security assumption. To expose this defense blind spot, we are the first to propose a *Model Architecture Backdoor* attack in the domain of static malware detection. Considering the strict format and semantic constraints of binary files, we design a universal architecture-level backdoor scheme based on a specific trigger mechanism. Unlike traditional data poisoning or weight manipulation attacks, the proposed attack introduces malicious structural components that are independent of model weights, enabling the backdoor to survive and persist even after full retraining in a clean environment. Extensive experiments conducted on the SOREL-20M and MOFIT benchmarks demonstrate that mainstream architectures, including CNNs, LSTMs, and Transformers, remain vulnerable to this architecture-level attack despite rigorous retraining procedures. Our findings expose the inherent limitations of defenses that rely solely on weight verification and highlight the necessity of incorporating architecture auditing and structure-aware defenses into the model supply chain to enhance the security and trustworthiness of AI systems in real-world deployment scenarios.**

***Index Terms*—Static Malware Detection, Model Architecture Backdoors, Model Supply Chain Security, Trustworthy AI, Backdoor Attacks**

I. Introduction

Malware detection is undergoing a critical transition from traditional approaches that rely on handcrafted rules and signatures [1] toward automated detection systems centered on deep learning. Along with the increasing engineering maturity and large-scale deployment of learning-based models, deep learning–based malware detection systems have exhibited pronounced *supply-chain characteristics*: network architectures, pretrained weights, and even complete models are frequently distributed through open-source communities or third-party platforms. In this context, building *trustworthy artificial intelligence* (Trustworthy AI) poses significant challenges. When the construction of detection models involves multi-party collaboration without end-to-end auditing, the model itself may become a potential source of systemic risk.

Existing research on the security of malware detection models, particularly backdoor attacks, has primarily focused on how adversaries encode malicious behaviors into model weights through data poisoning [2] or parameter manipulation [3]. However, such *weight-level* threats exhibit inherent fragility throughout the model lifecycle. Standard maintenance operations—including fine-tuning, continual learning, and retraining from scratch—can often effectively disrupt or eliminate malicious distributions embedded in the weights [4]. Consequently, industry practice frequently treats retraining as a "gold standard" for restoring model trustworthiness and relies on weight hash verification to establish a root of trust.

Recent studies in the computer vision (CV) domain have revealed a more fundamental threat: *architecture-level backdoors*, in which adversaries implant malicious logic directly into the computation graph, making it independent of specific parameter values [5], [6]. However, directly transferring architecture-level backdoor paradigms from CV to binary malware detection faces substantial cross-domain barriers. These challenges stem from fundamental differences in the mapping between the *problem space* and the *feature space* in the two domains.

First, inputs in the CV domain reside in a continuous pixel space, allowing adversaries to easily superimpose specific trigger patterns without violating image format constraints. In contrast, malware detection operates on highly discrete and strictly structured binary programs in the problem space. Any trigger that is effective in the feature space must be invertible to the problem space and comply with binary file format constraints: it must preserve program executability and, after being mapped through the embedding layer, precisely activate the implanted malicious substructures. Second, existing adversarial attacks against binaries typically require complex search procedures under strict semantic-preservation constraints. Architecture-level backdoors, by contrast, require adversaries to design a *structural shortcut* that enables strong activations in the feature space through extremely low-cost input modifications—such as padding non-executable regions—thereby bypassing normal detection logic. Achieving such structure–feature alignment across discrete constraints constitutes a novel and largely unexplored attack surface for malware detection models.

Motivated by these observations, this paper presents the first systematic study of the feasibility and security implications of architecture-level backdoors in static malware detection. We propose a novel attack paradigm that implants malicious components sensitive to specific binary feature patterns directly into neural network architectures, thereby constructing

*Corresponding Author: Fei Kang.

a *hard-coded* attack path decoupled from normal weight updates. This path exploits format redundancies in the binary problem space—such as Data Directory entries or overlay regions—to establish a stable mapping between problem-space modifications and malicious components in the feature space. Our results demonstrate that even when defenders possess high-quality clean datasets and retrain models from scratch, the backdoor remains effective as long as the network architecture itself contains the malicious component. This finding fundamentally challenges the conventional assumption that retraining guarantees safety and exposes critical gaps in current Trustworthy AI practices at the level of architectural auditing.

The main contributions of this paper are summarized as follows:

- **Cross-domain architecture-level backdoor threat model.** We present the first formal definition of architecture-level backdoor threats in the domain of malware detection. By overcoming the strict format and semantic constraints of binary files, we establish a stable trigger mapping mechanism that bridges the discrete problem space of binaries and the continuous feature space of deep learning models.
- **A retraining-resilient and generic attack framework.** We design malicious structural components and trigger strategies that are independent of model weights, achieving a complete decoupling between the attack logic and model parameters. As a result, the proposed attack maintains a high attack success rate even when the model undergoes retraining from scratch, fine-tuning, and pruning.
- **Empirical evaluation and defensive implications.** Through systematic experiments on large-scale datasets such as SOREL-20M and mainstream architectures including CNNs and Transformers, we demonstrate the stealthiness and generalization capability of the proposed attack. Our results expose critical blind spots in existing weight integrity verification mechanisms and highlight the necessity of incorporating architecture auditing into the model supply chain.

II. Related Work

A. Deep Learning–Based Malware Detection Systems

Malware detection techniques are broadly categorized into dynamic analysis and static analysis. Dynamic analysis relies on sandbox execution and runtime behavior monitoring [7], [8], but incurs high deployment costs and is vulnerable to evasion strategies such as anti-sandbox techniques and delayed execution. In contrast, static analysis directly operates on binary files and their metadata, offering higher efficiency and scalability; as a result, it has been widely adopted in large-scale detection scenarios [9].

Learning-based static malware detection methods can be broadly divided into feature-engineering and end-to-end approaches. The former rely on manually extracted metadata features, such as file headers and section statistics [10], and are susceptible to variations in compilation strategies and obfuscation techniques; the latter directly operate on raw bytes and leverage deep models to automatically learn discriminative representations, thereby avoiding cumbersome manual feature design.

Regarding end-to-end architectures, MalConv [11] was the first to model byte sequences using one-dimensional convolutions. Subsequent work introduced multi-scale convolutions, chunk-based modeling, or Transformer-based architectures to alleviate challenges posed by extremely long executable sequences and sparse discriminative information [12], [13]. Some methods further integrate multi-granularity features to improve detection robustness and generalization [14]. With advances in computational resources and engineering practices, deep learning models have become a central component of industrial-scale malware detection systems.

B. Adversarial Attacks Against Malware Detection Systems

Despite the strong detection performance achieved by deep learning models, their security remains exposed to various adversarial threats [15]. Attacks against malware detection systems mainly fall into two categories: evasion attacks and data poisoning attacks.

Evasion attacks occur at the inference stage, where adversaries apply functionality-preserving modifications to bypass detection, such as appending benign content, inserting no-op instructions, manipulating binary file fields, or applying code obfuscation techniques [3], [16]–[18]. In contrast, data poisoning attacks take place during training by injecting a small number of poisoned samples to interfere with the model learning process [19]. Among these, backdoor attacks have attracted particular attention due to their stealthiness: the model maintains normal detection performance on clean inputs while producing attacker-desired misclassifications under specific trigger conditions [2], [20], [21]. However, most existing backdoor attacks rely on specific weight distributions and often fail to remain effective after model updates or retraining.

C. Model Backdoor Detection and Mitigation

Research on defending against model backdoors primarily focuses on detection and mitigation. One class of methods leverages training data or a small set of trusted samples to identify poisoning artifacts by analyzing intermediate activations or spectral statistics [22], [23], but these approaches are sensitive to low poisoning ratios or diversified triggers. Another class of methods relies solely on the target model itself, detecting backdoors via trigger inversion or minimal-perturbation analysis, such as Neural Cleanse [24] and its extensions [25], as well as several black-box detection strategies [26], [27].

On the mitigation side, prior work has proposed online defenses based on input perturbation [28], as well as model repair techniques such as pruning and unlearning to weaken backdoor pathways [29], [30], often at the cost of performance degradation. Public evaluation efforts such as the NIST TrojAI

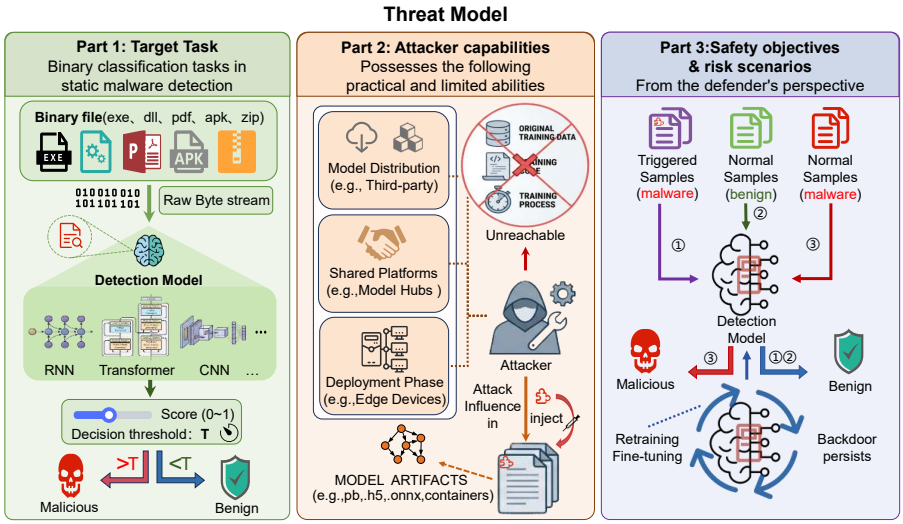

Fig. 1. Illustration of the model supply chain threat model. The adversary introduces an architecture-level backdoor by modifying the computation graph during model distribution or deployment.

benchmark [31] further indicate that existing methods exhibit limited generalization across different architectures and attack strategies.

Overall, prior research has predominantly focused on backdoors at the data and weight levels, paying insufficient attention to the security risks posed by the *model architecture itself* as a potential attack carrier. This gap constitutes the primary motivation for our study of architecture-level backdoors.

III. Methodology

This section presents the design of the proposed architecture-level backdoor attack. Under the model supply chain threat model, we illustrate how an adversary can implant a persistent, semantics-preserving backdoor by modifying the neural network computation graph—without relying on data poisoning or weight manipulation. From the perspectives of *architecture design* and *activation mechanisms*, we show that the backdoor remains triggerable after model fine-tuning or retraining from scratch, while introducing only limited impact on detection performance for clean samples.

A. Threat Model

We consider a model supply chain threat model that is consistent with real-world static malware detection pipelines (as illustrated in Fig. 1). In this setting, the defender has full control over the training data as well as the training and evaluation process. Nevertheless, supply chain risks may still arise during model distribution and deployment, where model artifacts can be maliciously modified by third-party intermediaries, introducing stealthy structural malicious logic.

1) Target Task: We study the binary classification task of static malware detection: given the raw byte sequence of an executable file as input, the detection model outputs a prediction indicating whether the file is malicious or benign.

2) Security Goal and Risk Scenarios: From the defender's perspective, we ask whether a detection model can be implanted with triggerable hidden logic *without significantly degrading its standard performance*, such that it consistently produces systematic misclassifications under a trigger condition (e.g., classifying certain malicious samples as benign), and remains effective even after version iterations, fine-tuning, or retraining from scratch.

3) Adversary Capabilities: Under this threat model, we assume that the adversary possesses only realistic and constrained supply-chain–level capabilities. Specifically, the adversary can access and modify the model artifact during the distribution or deployment phase, thereby tampering with the computation graph structure. Meanwhile, the adversary is able to perform semantics-preserving equivalent modifications to binary files in order to embed specific byte sequences as trigger conditions. The adversary has knowledge of the target model architecture and serialization format, but has no access to the training data and cannot interfere with the training or evaluation process.

Based on these assumptions, we further investigate the feasibility of using the *model architecture as a backdoor carrier*: namely, whether an adversary can implant hidden logic decoupled from weight updates solely by modifying the computation graph, such that it remains effective after fine-tuning or retraining.

B. Architecture Engineering

The goal of architecture engineering is to introduce a bypass control mechanism that only takes effect under a specific trigger condition, while preserving the original functional pathway of the model. This enables controllable modulation of the output behavior at inference time. The overall architecture is shown in Fig. 2.

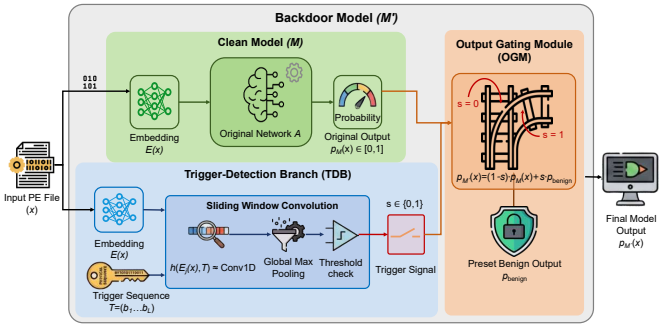

Fig. 2. Overview of architecture engineering. A trigger-detection branch (TDB, blue) and an output gating module (OGM, orange) are added in parallel to the original clean model (green), resulting in a backdoored model M'.

Let the clean detection model be defined by an architecture A and its trained parameters θ, and let its output on an input sample x be the predicted probability of being malicious:

$$p_M(x) = f_{A,\theta}(x) \in [0,1]. \tag{1}$$

On this basis, we modify the original clean model M into an architecture-backdoored model M'. Without changing the behavior of the original functional branch, the computation graph of M' introduces a trigger-detection branch (Trigger-Detection Branch, TDB) and an output gating module (Output Gating Module, OGM) in parallel. For clean inputs that do not contain the trigger, M' should produce the same output as M; when the input x contains the trigger, the OGM steers M' toward a predefined benign prediction.

1) Trigger-Detection Branch (TDB): Let the trigger be a byte sequence of length L, denoted by $T = (b_1, \ldots, b_L)$. The TDB takes the same input x as the original model and performs a sliding-window scan over its byte embedding representation $E(x)$. For the j-th local segment of length L, denoted by $E_j(x)$, the TDB computes the following similarity score:

$$h(E_j(x), T) = \|T\|^2 - \|E_j(x) - T\|^2. \tag{2}$$

Here $\|\cdot\|$ denotes the ℓ_2 norm. This score attains its maximum when $E_j(x)$ exactly matches T, and can be viewed as an equivalent transformation of the Euclidean distance. The computation can be efficiently implemented using a one-dimensional convolution operator: the convolution computes the inner-product term $2E_j(x)\cdot T$, which is combined with the sliding-window sum-of-squares term to obtain the final score.

Next, the TDB applies global max pooling over scores across all window positions:

$$s^* = \max_j h(E_j(x), T). \tag{3}$$

Finally, a thresholding function produces a binary trigger signal:

$$s = \mathbb{I}(s^* > \tau), \quad s \in \{0,1\}. \tag{4}$$

where τ is a predefined threshold and $\mathbb{I}(\cdot)$ is the indicator function. The signal satisfies $s = 1$ if and only if the input contains a byte segment that strictly matches the trigger; otherwise $s = 0$. This design enables exact trigger matching and effectively avoids accidental activations caused by byte prefixes or similar patterns in the natural distribution.

2) Output Gating Module (OGM): The OGM gates the original model output based on the trigger indicator s. Let the original model output in the prediction space be $p_M(x)$. The OGM output is defined as:

$$p_{M'}(x) = (1-s)\, p_M(x) + s\, p_{\text{benign}}. \tag{5}$$

where p_{benign} is a predefined constant target output that forces the model to produce a benign prediction. In our implementation, p_{benign} corresponds to a fixed benign output value. Therefore, when $s = 0$, M' behaves identically to the original model M; when $s = 1$, the OGM overrides the original prediction, making the model consistently output a benign result.

The above backdoor logic is embedded into the model via a fixed computation graph structure (i.e., TDB and OGM). During subsequent fine-tuning or training from scratch, gradient-based optimization only updates trainable parameters (e.g., θ and any learnable parameters in the TDB), while the gating logic in the OGM remains as a structural constraint in the computation graph. As a result, the backdoor does not depend on specific weight values and cannot be removed by standard gradient-based updates during fine-tuning or retraining.

C. Activation Engineering

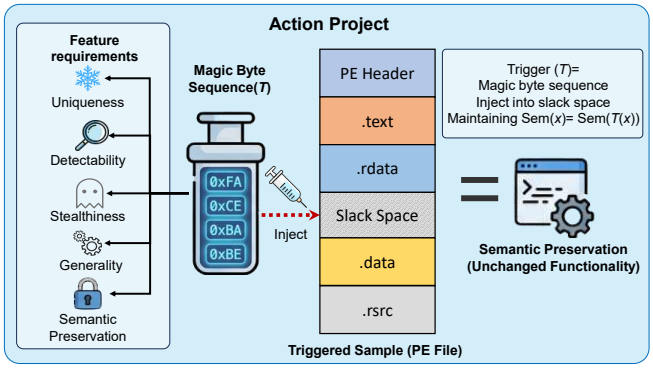

Fig. 3. Overview of the activation engineering framework. The figure illustrates trigger constraints, the construction of the magic byte sequence, and the injection workflow into redundant regions of various binary formats (e.g., PE and ELF).

1) Trigger Constraints: The core of activation engineering lies in the construction and controllable injection of triggers, as illustrated in Fig. 3. Since binary files are governed by strict format specifications, trigger design must ensure both structural integrity and functional equivalence of the modified binaries. We formalize a trigger as a constrained transformation $T(\cdot)$ applied to an arbitrary binary sample x, which must satisfy the following general constraints (shown on the left side of Fig. 3):

- **Uniqueness:** The trigger byte pattern has an extremely low probability of occurring in naturally distributed benign and malicious samples.
- **Detectability:** The trigger exhibits stable, localized, and well-defined structural features, enabling reliable recognition by the detection branch.
- **Stealthiness:** Trigger injection does not introduce anomalous behaviors or prominent static-analysis artifacts.
- **Generality:** The trigger and its injection strategy are compatible with multiple mainstream binary formats.
- **Semantic Preservation:** The transformed sample remains equivalent to the original in terms of loading and execution semantics.

Under these constraints, we abstract the trigger as a perturbation applied to a set of modifiable positions $\mathcal{R}(x)$ in the input x:

$$T(x) = x \oplus \delta, \quad \text{s.t.} \delta \in \mathcal{R}(x),\ \mathrm{Sem}(x) = \mathrm{Sem}(T(x)). \quad (6)$$

Here $\mathcal{R}(x)$ denotes the set of byte positions that can be safely modified, and $\mathrm{Sem}(\cdot)$ denotes program semantics. Eq. 6 indicates that the trigger operates only on modifiable regions while preserving program semantics.

2) Magic Byte Sequence (MBS): Based on the above abstraction, as illustrated in the middle of Fig. 3, we design a trigger based on a *Magic Byte Sequence (MBS)*. The trigger T consists of a fixed-length byte sequence, whose specific values can either be randomly sampled from a uniform distribution (e.g., `0xFA 0xCE 0xBA 0xBE`) or customized according to the content or contextual characteristics of the target binary file. By selecting a sufficiently long sequence length (e.g., 32 or 64 bytes), the probability of the sequence occurring spontaneously in natural samples can be considered negligible, thereby ensuring the uniqueness and distinguishability of the trigger in the detection space.

3) Trigger Injection: The trigger injection location must simultaneously satisfy the requirements of semantic preservation and structural stability. Compared to directly modifying code sections or introducing new sections, we choose to exploit inter-section padding regions (*slack space*) that arise from alignment rules in binary files. Such regions do not participate in control flow or data access during program loading and execution, and can therefore serve as low-risk carriers for trigger injection.

Formally, let the byte-level representation of a binary file x be denoted by $B(x)$, and let its set of sections be $\{S_1, \ldots, S_n\}$. For a section S_i, let p_i denote its physical file offset, d_i its raw data size, and A_f the file alignment granularity. The actual on-disk space occupied by this section is given by:

$$D_i = \left\lceil \frac{d_i}{A_f} \right\rceil \cdot A_f. \quad (7)$$

The resulting inter-section padding size is $P_i = D_i - d_i$. When $P_i \geq |T|$, the trigger T can be written at offset $p_i + d_i$. The corresponding injection operation can be expressed as:

$$B(x_t) = \mathrm{Inject}\big(B(x), T, p_i + d_i\big), \quad \text{s.t. } |T| \leq P_i. \quad (8)$$

This injection procedure does not change any section attributes, virtual addresses, or entry-point information, nor does it affect the program execution path. Therefore, it preserves semantics and achieves stealthiness while ensuring that the trigger can be embedded.

IV. Experimental Evaluation and Analysis

A. Datasets and Evaluation Metrics

1) Datasets: We primarily conduct our evaluation on the SOREL-20M dataset. Released by Sophos and ReversingLabs, SOREL-20M contains metadata and features for approximately 2×10^7 binary files, with roughly half of them labeled as malicious. It has been widely adopted as a benchmark for static malware detection research.

In our experiments, we randomly sample 10^6 instances from SOREL-20M (5×10^5 malicious and 5×10^5 benign) for training, and select an additional 2×10^5 instances (10^5 malicious and 10^5 benign) to construct an independent clean test set. To verify the consistency of our findings on a publicly available small-scale dataset, we further replicate the experimental setup and conduct comparative evaluations on the MOFIT dataset.

2) Evaluation Metrics: We evaluate the models from both detection performance and false-alarm control perspectives, reporting Accuracy, Precision, Recall, F1-score, False Positive Rate (FPR), and ROC-AUC. Unless otherwise specified, all metrics are computed on the independent test set.

B. Backdoor Effectiveness and Robustness Evaluation

1) Baseline Performance and Attack Stealthiness: We first evaluate the stealthiness of the proposed attack by focusing on two aspects: (i) whether trigger injection affects the normal inference behavior of the clean model M; and (ii) whether the backdoored model M' maintains detection performance comparable to M under non-triggered conditions.

Figure 4 presents the validation accuracy curves of CNN-, LSTM-, and Transformer-based models across different training epochs. For the clean model M, the accuracy curves for inputs with and without triggers almost completely overlap, indicating that the trigger—implemented as a semantics-preserving byte-level modification—does not significantly perturb the model's normal discriminative behavior.

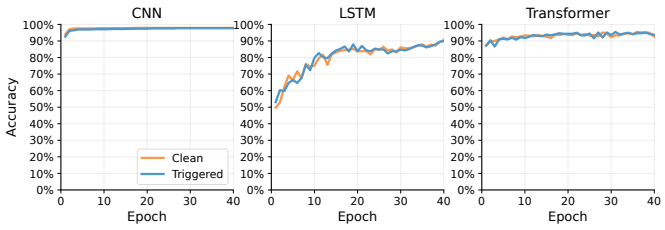

Fig. 4. Test-set accuracy of CNN, LSTM, and Transformer models across training epochs, comparing clean inputs with trigger-injected inputs.

TABLE I
COMPARISON OF DETECTION PERFORMANCE ON CLEAN SAMPLES UNDER NON-TRIGGERED CONDITIONS FOR THE CLEAN MODEL M AND THE BACKDOORED MODEL M' (ALL METRICS ARE REPORTED IN %).

Model	Version	Recall	Precision	F1	FPR	AUC	ACC
CNN	Clean	99.03	96.84	97.92	3.24	99.60	97.90
	Backdoored	98.38	97.12	97.75	2.91	99.59	97.73
LSTM	Clean	90.94	89.49	90.21	10.68	95.87	90.13
	Backdoored	91.91	87.38	89.59	13.27	95.63	89.32
Transformer	Clean	96.12	93.99	95.04	5.83	98.24	94.98
	Backdoored	93.85	95.08	94.46	4.85	96.45	94.50

Furthermore, we compare the detection performance of M and M' on an independent clean test set containing 2×10^5 samples, with the results summarized in Table I. Overall, after implanting the architecture-level backdoor, the model performance on clean inputs remains stable, with only limited fluctuations. This observation indicates that the backdoor has minimal impact on standard detection behavior when the trigger is absent. The primary reason is that the gating mechanism effectively suppresses the backdoor pathway: when the input does not contain the trigger, the trigger signal satisfies $s \approx 0$, and thus the output of M' closely matches that of M.

After confirming stealthiness, we further evaluate the attack effectiveness under triggered conditions. The test set contains 2×10^5 samples, including 10^5 benign and 10^5 malicious instances. The benign samples are kept unchanged, while triggers are injected into all malicious samples. We then perform inference using the clean model M and the backdoored model M', with the results reported in Table II. On M, the triggered malicious samples are still correctly detected, indicating that the trigger itself does not directly cause misclassification. In contrast, on M', all triggered malicious samples are consistently misclassified as benign, driving the malicious-class Recall/Precision/F1 to 0. This phenomenon is consistently observed across all three model architectures, demonstrating that architecture-level backdoors can reliably hijack the output once the trigger condition is satisfied.

TABLE II
COMPARISON OF DETECTION PERFORMANCE OF TRIGGER-INJECTED SAMPLES UNDER TRIGGERED CONDITIONS ON THE CLEAN MODEL M AND THE BACKDOORED MODEL M' (ALL METRICS ARE REPORTED IN %).

Model	Version	Recall	Precision	F1	FPR	AUC	ACC
CNN	Clean	98.71	96.83	97.76	3.24	99.57	**97.73**
	Backdoored	0.00	0.00	0.00	7.77	91.59	**46.12**
LSTM	Clean	90.94	89.49	90.21	10.68	95.87	**90.13**
	Backdoored	0.00	0.00	0.00	13.27	64.72	**43.37**
Transformer	Clean	95.47	94.25	94.86	5.83	98.24	**94.82**
	Backdoored	0.00	0.00	0.00	4.85	92.56	**47.57**

In summary, the proposed attack simultaneously achieves *stealthiness* and *effectiveness*: under standard inputs and evaluation procedures, M' behaves almost identically to M, while once the trigger condition is satisfied, the backdoor reliably hijacks the output and induces systematic misclassification.

2) Comparison of Architecture-Level and Weight-Level Backdoor Attacks: To compare the impact of different backdoor carriers on malware detection models, we evaluate the proposed architecture-level backdoor attack (MAB) against two representative weight-level backdoor attacks under a unified experimental setting. All experiments target the **MalConv** model and are conducted using the same data splits of the **MOFIT** dataset.

TABLE III
COMPARISON OF DETECTION PERFORMANCE UNDER CLEAN (CLEAN) AND TRIGGERED (TRIGGERED) CONDITIONS FOR DIFFERENT BACKDOOR ATTACK METHODS (ALL METRICS ARE REPORTED IN %).

Model	Setting	Recall	Precision	F1	FPR	AUC	ACC
None	Clean	99.03	96.84	97.92	3.24	99.60	97.90
	Triggered	98.71	96.83	97.76	3.24	99.57	97.73
BadNets	Clean	97.09	94.94	96.00	5.19	99.22	95.95
	Triggered	3.56	40.74	6.55	5.18	58.58	49.19
Handcrafted	Clean	93.20	98.29	95.68	1.62	99.56	95.79
	Triggered	6.59	85.71	12.24	10.99	78.03	52.75
MAB	Clean	98.38	97.12	97.75	2.91	99.59	97.73
	Triggered	0.00	0.00	0.00	7.77	91.59	46.12

Specifically, we consider three representative attack methods: **BadNets**, which injects 5% trigger-bearing samples labeled as benign during training; **Handcrafted**, which introduces backdoor bias via targeted fine-tuning of model weights prior to deployment; and the proposed **MAB**, which implants backdoor logic solely by modifying the computation graph structure. We report Recall, Precision, F1-score, FPR, AUC, and ACC on both the *clean test set (Clean)* and the *trigger-injected test set (Triggered)*, enabling simultaneous evaluation of normal detection performance and behavior under trigger conditions.

Table III summarizes the results on the MOFIT test set. Compared with the no-attack baseline, BadNets, Handcrafted, and MAB achieve ACC values of 95.95%, 95.79%, and 93.85% on clean samples, respectively, indicating that all three backdoor mechanisms introduce some degradation to standard detection performance, albeit with comparable magnitude. Under triggered conditions, all attacks significantly alter the model's prediction behavior: BadNets and Handcrafted cause a substantial drop in Recall, while MAB further suppresses Recall, Precision, and F1-score to zero, exhibiting a stronger output-dominating effect. These results demonstrate that even without relying on training data poisoning or weight manipulation, MAB can achieve trigger impacts comparable to—or even stronger than—typical weight-level backdoor attacks.

3) Performance After Fine-Tuning: We evaluate the robustness of different backdoor mechanisms under the setting of *fine-tuning using only clean data*. Specifically, starting from MalConv models implanted with different backdoors (MAB, BadNets, and Handcrafted), we fine-tune each model on 10^5 clean samples from the MOFIT dataset for 20 epochs. All experiments use identical training configurations, and no trigger-bearing samples are introduced during fine-tuning. The top row of Fig. 5 shows the evolution of triggered-sample detection accuracy during fine-tuning. We observe that the effectiveness of weight-level backdoors (BadNets and Handcrafted) rapidly

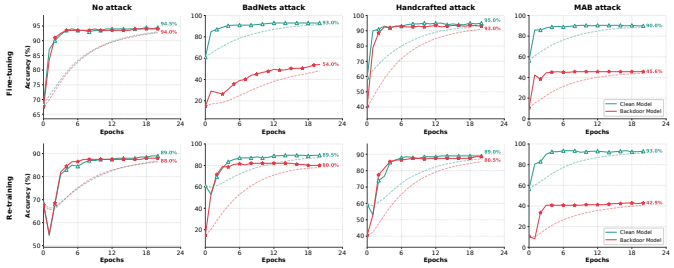

Fig. 5. Evolution of triggered-sample detection accuracy during fine-tuning (top) and retraining from scratch (bottom), comparing the behavior of different backdoor mechanisms under model updates.

degrades as fine-tuning progresses: the accuracy on triggered samples gradually increases and approaches that of the clean model. In contrast, the architecture-level backdoor (MAB) remains largely unaffected throughout the fine-tuning process, with triggered-sample accuracy consistently staying at a low level. This indicates that conventional fine-tuning with clean data alone is insufficient to disrupt structural backdoor logic.

4) Performance After Retraining From Scratch: Building on the above results, we further evaluate the effectiveness of *retraining from scratch* as a mitigation strategy against different backdoors. Specifically, we start from the backdoored model artifact as the architectural baseline, randomly reinitialize *all* learnable weight parameters in the model, and then retrain it for 20 epochs on 10^5 clean samples. This setting preserves no prior weight information and is intended to characterize a defender's full-reset retraining procedure after identifying architectural anomalies.

The bottom row of Fig. 5 presents the evolution of triggered-sample accuracy during retraining from scratch. The results show that for BadNets and Handcrafted, the triggered-sample accuracy recovers to a level comparable to that of the clean model after retraining. In contrast, for MAB, triggered-sample accuracy remains consistently low even after retraining. This demonstrates that when backdoor logic is hard-coded into the model computation graph, even strong mitigation measures such as retraining from scratch may fail to completely remove it.

C. Evaluation of Backdoor Detectability

Our experiments show that architecture-level backdoors retain pronounced *persistence* after both fine-tuning and retraining from scratch. In real-world deployments, although defenders often combine backdoor detection with data sanitization, these methods typically can only observe anomalous behaviors and struggle to localize or remove the structural root cause.

Existing defenses commonly rely on two assumptions. First, a backdoor introduces separable statistical anomalies in the weight or representation space. Second, the backdoor depends on an identifiable subset of poisoned samples and can therefore be mitigated via data sanitization. However, in the architecture-level backdoor setting, both assumptions frequently fail to hold. Such backdoors do not rely on a salient poisoned subpopulation and may not induce stable anomalies in weights or representations. Even when trigger-inversion methods reveal abnormal behaviors, it remains difficult to further attribute them to specific architectural components.

In summary, the central challenge of architecture-level backdoors is that they *do not depend on visible poisoned samples* and *do not exhibit stable statistical anomalies*. To address this threat model, a more targeted direction is to incorporate **structural auditing** and **structure-aware probing** to cover attack surfaces where behavioral anomalies are observable but cannot be explained by data- or weight-level evidence.

V. Discussion and Limitations

This work systematically introduces architecture-level backdoors into the domain of static malware detection and experimentally demonstrates that, compared to conventional backdoor mechanisms relying on data poisoning or weight manipulation, architecture-level backdoors can maintain stable triggerable behavior after model fine-tuning and retraining from scratch, while imposing only limited impact on clean-sample detection performance. It is important to emphasize that our goal is not to propose a "stronger" attack, but rather to expose a class of structural risks that have long been overlooked in model supply chain and auditing processes, thereby providing empirical evidence to inform the design of future defense and verification mechanisms.

A. Defense Implications

Against architecture-level backdoors, relying solely on weight integrity verification or model retraining is insufficient to provide reliable protection. As a baseline measure, integrity verification of model artifacts (e.g., signature validation based on SHA-256 hashes) can reduce the risk of model replacement or tampering during distribution and deployment. However, such approaches fail to address scenarios where the source model itself is untrusted, such as third-party pretrained models. More advanced defenses must therefore incorporate architecture-aware auditing mechanisms, including computation-graph diffing, operator-level static inspection, and anomalous execution-path analysis, to identify hidden branches or gating logic. Nevertheless, in complex deep learning models, distinguishing between legitimate engineering optimizations and malicious structural modifications remains an open challenge, with no unified criteria or mature tooling available. As a complementary direction, runtime monitoring may leverage activation statistics to identify suspicious modules that remain inactive under normal inputs but exert a disproportionate influence on outputs under specific conditions. Such methods, however, often rely on heuristic thresholds, and their generalization capability and false-positive control require further investigation.

B. Limitations

The trigger mechanism adopted in this work is based on static byte sequences and deterministic matching, offering simplicity and reproducibility but potentially underestimating

the stealth achievable by stronger adversaries using context-dependent or dynamic trigger conditions in real-world scenarios. In addition, our backdoor logic is implemented via explicitly introduced parallel branches, which facilitates structural analysis and experimental validation but may be more readily exposed under strict architectural auditing frameworks. Future attack variants may further integrate gating logic into existing computation paths to enhance stealthiness. For such "structurally endogenous" architecture-level backdoors, systematic modeling and effective defense strategies remain largely unexplored.

VI. Conclusion

This paper proposes and systematically investigates an architecture-level backdoor attack paradigm for static malware detection. By embedding trigger-detection and output-gating logic into the model computation graph, the backdoor remains effective after both fine-tuning and retraining from scratch. Taking into account the format constraints of binary executables, we construct semantics-preserving triggers and validate the feasibility of the proposed approach on benchmark datasets such as SOREL-20M across multiple mainstream detection models. Experimental results show that, while largely preserving clean-sample detection performance, the architecture-level backdoor can reliably induce misclassification on triggered samples. These findings indicate that relying solely on weight verification or retraining is insufficient to fully address structural risks in the model supply chain. Future defensive research should therefore incorporate structure-aware verification, auditing, and monitoring mechanisms to enhance overall trustworthiness in real-world model supply chain environments.

References

[1] E. Raff, R. Zak, G. Munoz, W. Fleming, B. Filar, C. Nicholas, and J. Holt, "Automatic yara rule generation using biclustering," 11 2020, pp. 71–82.

[2] D. Zhan, K. Xu, X. Liu, T. Han, Z. Pan, and S. Guo, "Practical clean-label backdoor attack against static malware detection," *Comput. Secur.*, vol. 150, p. 104280, 2024.

[3] G. Severi, J. Meyer, and S. E. Coull, "Explanation-guided backdoor poisoning attacks against malware classifiers," in *USENIX Security Symposium*, 2021.

[4] D. Wu and Y. Wang, "Adversarial neuron pruning purifies backdoored deep models," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 16 913–16 925.

[5] S. Goldwasser, M. P. Kim, V. Vaikuntanathan, and O. Zamir, "Planting undetectable backdoors in machine learning models : [extended abstract]," in *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, 2022, pp. 931–942.

[6] M. Bober-Irizar, I. Shumailov, Y. Zhao, R. Mullins, and N. Papernot, "Architectural backdoors in neural networks," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 24 595–24 604.

[7] Y. Li, Y. Jiang, Z. Li, and S.-T. Xia, "Backdoor learning: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 1, pp. 5–22, 2024.

[8] O. Or-Meir, N. Nissim, Y. Elovici, and L. Rokach, "Dynamic malware analysis in the modern era—a state of the art survey," *ACM Computing Surveys (CSUR)*, vol. 52, pp. 1 – 48, 2019.

[9] M. Amin, T. A. Tanveer, M. Tehseen, M. Khan, F. A. Khan, and S. Anwar, "Static malware detection and attribution in android byte-code through an end-to-end deep system," *Future Generation Computer Systems*, vol. 102, p. 112–126, Jan 2020.

[10] H. Anderson and P. Roth, "Ember: An open dataset for training static pe malware machine learning models," *ArXiv*, vol. abs/1804.04637, 2018.

[11] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole exe," in *AAAI Workshops*, 2017.

[12] E.-J. Kim, Y.-K. Lee, S.-M. Lee, J.-N. Kim, A. R. Kang, M.-S. Kim, and Y.-S. Jeong, "Malware detection using pre-trained transformer encoder with byte sequences," *PLOS One*, vol. 20, 2025.

[13] S. Khan and M. Nauman, "Interpretable detection of malicious behavior in windows portable executables using multi-head 2d transformers," *Big Data Mining and Analytics*, vol. 7, no. 2, pp. 485–499, 2024.

[14] X. Yang, D. Yang, and Y. Li, "A hybrid attention network for malware detection based on multi-feature aligned and fusion," *Electronics*, 2023.

[15] M. Noppel and C. Wressnegger, "Composite explanation-aware attacks," in *2025 IEEE Security and Privacy Workshops (SPW)*, 2025, pp. 167–176.

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

[17] M. D'Onghia, F. Di Cesare, L. Gallo, M. Carminati, M. Polino, and S. Zanero, "Lookin' out my backdoor! investigating backdooring attacks against dl-driven malware detectors," ser. AISec '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 209–220.

[18] L. Demetrio, S. E. Coull, B. Biggio, G. Lagorio, A. Armando, and F. Roli, "Adversarial exemples: A survey and experimental evaluation of practical attacks on machine learning for windows malware detection," vol. 24, no. 4, Sep. 2021.

[19] K. Aryal, M. Gupta, and M. Abdelsalam, " Analysis of Label-Flip Poisoning Attack on Machine Learning Based Malware Detector ," in *2022 IEEE International Conference on Big Data (Big Data)*. Los Alamitos, CA, USA: IEEE Computer Society, Dec. 2022, pp. 4236–4245.

[20] Y. Zhang, F. Feng, Z. Liao, Z. Li, and S. Yao, "Universal backdoor attack on deep neural networks for malware detection," *Appl. Soft Comput.*, vol. 143, no. C, Aug. 2023.

[21] C. Li, X. Chen, D. Wang, S. Wen, M. E. Ahmed, S. Camtepe, and Y. Xiang, "Backdoor attack on machine learning based android malware detectors," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 5, pp. 3357–3370, 2022.

[22] M. Ma, H. Li, and X. Kuang, "Detecting backdoor attacks on deep neural networks based on model parameters analysis," in *2022 IEEE 34th International Conference on Tools with Artificial Intelligence (ICTAI)*, 2022, pp. 630–637.

[23] B. Tran, J. Li, and A. Mądry, "Spectral signatures in backdoor attacks," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 8011–8021.

[24] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 707–723.

[25] W. Guo, L. Wang, X. Xing, M. Du, and D. X. Song, "Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems," *ArXiv*, vol. abs/1908.01763, 2019.

[26] H. Chen, C. Fu, J. Zhao, and F. Koushanfar, "Deepinspect: a black-box trojan detection and mitigation framework for deep neural networks," ser. IJCAI'19. AAAI Press, 2019, p. 4658–4664.

[27] Z. Yang, C. Luo, D. He, Y. Li, and Y. Li, "Arcgen: Generalizing neural backdoor detection across diverse architectures," *IEEE Transactions on Information Forensics and Security*, vol. 20, pp. 10 082–10 097, 2025.

[28] Y. Gao, C. Xu, D. Wang, S. Chen, D. C. Ranasinghe, and S. Nepal, "Strip: a defence against trojan attacks on deep neural networks," *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019.

[29] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," *ArXiv*, vol. abs/1805.12185, 2018.

[30] Y. Zeng, S. Chen, W. Park, Z. M. Mao, M. Jin, and R. Jia, "Adversarial unlearning of backdoors via implicit hypergradient," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2022.

[31] National Institute of Standards and Technology (NIST), "Trojai: AI trojan detection," https://pages.nist.gov/trojai/, 2025.