A Hard-Label Black-Box Evasion Attack against ML-based Malicious Traffic Detection Systems

Zixuan Liu*, Yi Zhao†✉, Zhuotao Liu*‡, Qi Li*‡, Chuanpu Fu*, Guangmeng Zhou*, Ke Xu*‡✉
*Tsinghua University, †Beijing Institute of Technology, ‡Zhongguancun Lab

***Abstract*—Machine Learning (ML)-based malicious traffic detection is a promising security paradigm. It outperforms rule-based traditional detection by identifying various advanced attacks. However, the robustness of these ML models is largely unexplored, thereby allowing attackers to craft adversarial traffic examples that evade detection. Existing evasion attacks typically rely on overly restrictive conditions (e.g., encrypted protocols, Tor, or specialized setups), or require detailed prior knowledge of the target (e.g., training data and model parameters), which is impractical in realistic black-box scenarios. The feasibility of a hard-label black-box evasion attack (i.e., applicable across diverse tasks and protocols without internal target insights) thus remains an open challenge.**

To this end, we develop NetMasquerade, which leverages reinforcement learning (RL) to manipulate attack flows to mimic benign traffic and evade detection. Specifically, we establish a tailored pre-trained model called Traffic-BERT, utilizing a network-specialized tokenizer and an attention mechanism to extract diverse benign traffic patterns. Subsequently, we integrate Traffic-BERT into the RL framework, allowing NetMasquerade to effectively manipulate malicious packet sequences based on benign traffic patterns with minimal modifications. Experimental results demonstrate that NetMasquerade enables both brute-force an stealthy attacks to evade 6 existing detection methods under 80 attack scenarios, achieving over 96.65% attack success rate. Notably, it can evade the methods that are either empirically or certifiably robust against existing evasion attacks. Finally, NetMasquerade achieves low-latency adversarial traffic generation, demonstrating its practicality in real-world scenarios.

I. Introduction

Machine learning (ML)-based malicious traffic detection systems identify attack behaviors by learning the features of traffic [28], [62], [30]. As an emerging security paradigm, it is promising for identifying multiple sophisticated attacks [51], [53], [49] and thus outperforms traditional rule-based detection [98], [39], [93] in both effectiveness [20], [4] and efficiency [104], [7]. Currently, ML-based systems are deployed to complement the traditional systems due to their ability to detect unknown [23] and encrypted [4], [89] attack traffic.

Unfortunately, as in many other ML-based domains [86], [66], [32], robustness issues are prevalent in ML-based traffic detection systems [84], [5], [47]. That is, attackers can craft adversarial traffic by adding, delaying, or otherwise modifying packets [37], [96], causing detection models to misclassify these deceptive flows as benign. The research community has put forward a range of advanced evasion methods (see Table I), yet many of these methods operate under narrowly defined conditions, such as leveraging encrypted protocols [65], [58], [82], tunnels [65], Tor networks [50], or circumventing specialized third-party censorship systems [58]. The effectiveness of these protocol-related and task-related approaches drops significantly when the attack environment changes. Moreover, some solutions rely heavily on prior knowledge about the target or dataset, requiring full (white-box) [82], [65], [96] or partial (gray-box) [65], [37] access, which is impractical in a more realistic black-box setting.

To bridge these gaps, we aim to design a black-box adversarial attack targeting widely used ML-based traffic detection systems that rely on statistical patterns [28], [62], [104], [7]. In particular, the attack must be protocol-agnostic and task-agnostic, allowing it to be seamlessly applied to any malicious traffic, regardless of whether it is encrypted, tunneled, or otherwise constrained. Moreover, the attacker can generate adversarial malicious traffic with minimal modifications, relying solely on whether the target system drops malicious packets (i.e., hard-label attack [94], [88]). In contrast to feature-space attacks [57], [3], i.e., impractical settings that require attackers to interfere with ML execution, our traffic modifications must preserve the effectiveness of the attacks [82], [37].

This paper presents NetMasquerade, a hard-label black-box evasion attack, which utilizes deep reinforcement learning (RL) to transform malicious traffic into adversarial examples by mimicking benign traffic patterns. At its core, we propose Traffic-BERT, a tailored pre-trained model for capturing diverse and complex benign traffic distributions. Subsequently, we develop an RL framework that decides the location and type of packet modification step-by-step, leveraging Traffic-BERT's embedded knowledge of benign behaviors. The only feedback required for the RL training process is the blocked-or-not signal from the targeted detection system. The detailed design ensures that NetMasquerade achieves minimal, yet effective, modifications across diverse traffic types and detection models, thereby evading detection systems under black-box conditions. We address two main challenges in constructing effective adversarial traffic.

First, we must capture rich benign traffic patterns in order to mimic them convincingly. To address this, we study the distribution of Internet packet patterns, then pad and chunk traffic from public datasets using an optimal setup to improve diversification. Afterwards, we pre-process the traffic with network-specific tokenizers. Finally, we extract dependencies among the tokens with a novel attention block in Traffic-BERT, providing a robust representation of benign traffic across various protocols and scenarios.

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.230916
www.ndss-symposium.org

TABLE I
COMPARISON OF EXISTING EVASION ATTACKS AGAINST TRAFFIC ANALYSIS SYSTEMS.

Scenarios	Evasion Techniques	Attack Applicability		Without Prior Knowledge			Attack Performance	
		Protocol-agnostic	Task-agnostic	Datasets	Features	Model	Low Overhead	Low Latency
White-box	Gradient Analysis [82], [65]	✗	✓	✗	✗	✗	✗	✓
	Optimization [96]	✗	✓	✗	✗	✗	✗	✓
Gray-box	Sample Transferability [65]	✗	✗	✗	✗	✓	✗	✓
	Feature Manipulation [37]	✓	✓	✓	✗	✓	✗	✗
Black-box	Packet reassembly [58]	✗	✗	✓	✓	✓	✗	✗
	Traffic Mimicking **(Ours)**	✓	✓	✓	✓	✓	✓	✓

Second, the RL model for generating optimal evasion policies must maintain both low training overhead and low online inference latency. To this end, we formulate the traffic modifications as a Finite Horizon Markov Decision Process, enabling a multi-step decision strategy with explicit incentives for minimal and targeted modifications, effectively reducing inference costs. Meanwhile, we utilize a lightweight policy network and ensure rapid convergence by leveraging already-learned benign distributions in Traffic-BERT, significantly reducing training overhead. In addition, we introduce an effectiveness penalty, which safeguards the malicious functionality of the attack.

We prototype NetMasquerade[1] with Pytorch [74] and Intel DPDK [45]. Experiments demonstrate that NetMasquerade enables both high-rate and low-rate attacks to evade 6 top-performing detection systems in 80 attack scenarios, achieving over 96.65% attack success rate (ASR). Note that NetMasquerade can evade the methods that are either empirically [28] or certifiably [95] robust against existing evasion attacks. Compared with other attacks [28], [37], NetMasquerade applies minimal modifications to no more than 10 steps in all test scenarios, thus incurring little impact, e.g., a Kullback-Leibler (KL) divergence of 0.013 between the original and modified bandwidth distributions. Moreover, the evasion attacks can be constructed in time, i.e., NetMasquerade can transform 4.239K adversarial packets per second.

In general, our studies reveal that the robustness issue of traffic detection remains unaddressed, which emphasizes the necessity of enhancing robustness against advanced evasion attacks. The contributions of this paper are three-fold:

- We develop a hard-label black-box evasion attack that operates across diverse traffic types and targets widely used ML-based detection systems relying on statistical patterns, leveraging deep reinforcement learning to manipulate attack packets for efficient benign-traffic mimicry.
- We design a tailored pre-trained model, Traffic-BERT, to capture diverse and complex benign traffic patterns, equipping NetMasquerade with the ability to transform malicious flows into benign-like traffic across a wide range of protocols and tasks.

[1]Source code: https://github.com/09nat/NetMasquerade

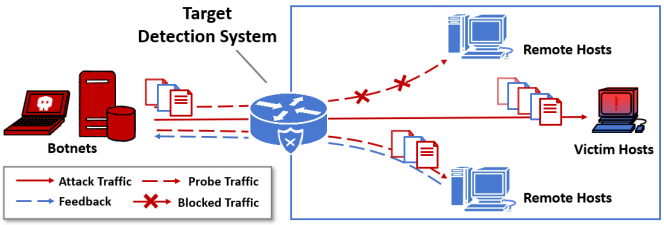


Fig. 1. **Network topology.**

- We establish an RL-based framework that transforms original attack traffic into adversarial examples with minimal modifications, and experimentally validate that NetMasquerade can generate various attack traffic to evade multiple state-of-the-art detection systems with small overhead.

II. THREAT MODEL AND ASSUMPTIONS

Threat Model. We assume an external adversary who instructs botnets to deliver malicious traffic (e.g., high-speed attack and stealthy malware behaviors) toward victim hosts [89]. The attack flows originate on the public Internet and must traverse an in-line, ML-based traffic detection system deployed at the victim's ingress link, as shown in Figure 1. The detection system operates on certain flow- or packet-level statistical patterns (e.g., sizes, delays) for traffic classification and does not inspect the payload [28], [30], [62], [7]. Such pattern-based models are increasingly popular as they are effective on encrypted traffic. Meanwhile, the system forwards benign traffic and drops or rate-limits malicious traffic. This behavior is consistent with both existing academic proposals [102] and default configurations of open-source in-line detectors [68], which prioritize network availability by choosing low-latency actions like packet dropping over more aggressive responses (e.g., blocking entire source IPs, which could be a NAT gateway serving many legitimate users) that may cause significant collateral damage. To evade detection, the attacker craft each malicious flow to construct adversarial traffic that misleads detection systems into classifying them as benign, while retaining the flow's original intent.

Assumptions. The attacker cannot obtain any details of the target detection systems, such as ML models, parameters,

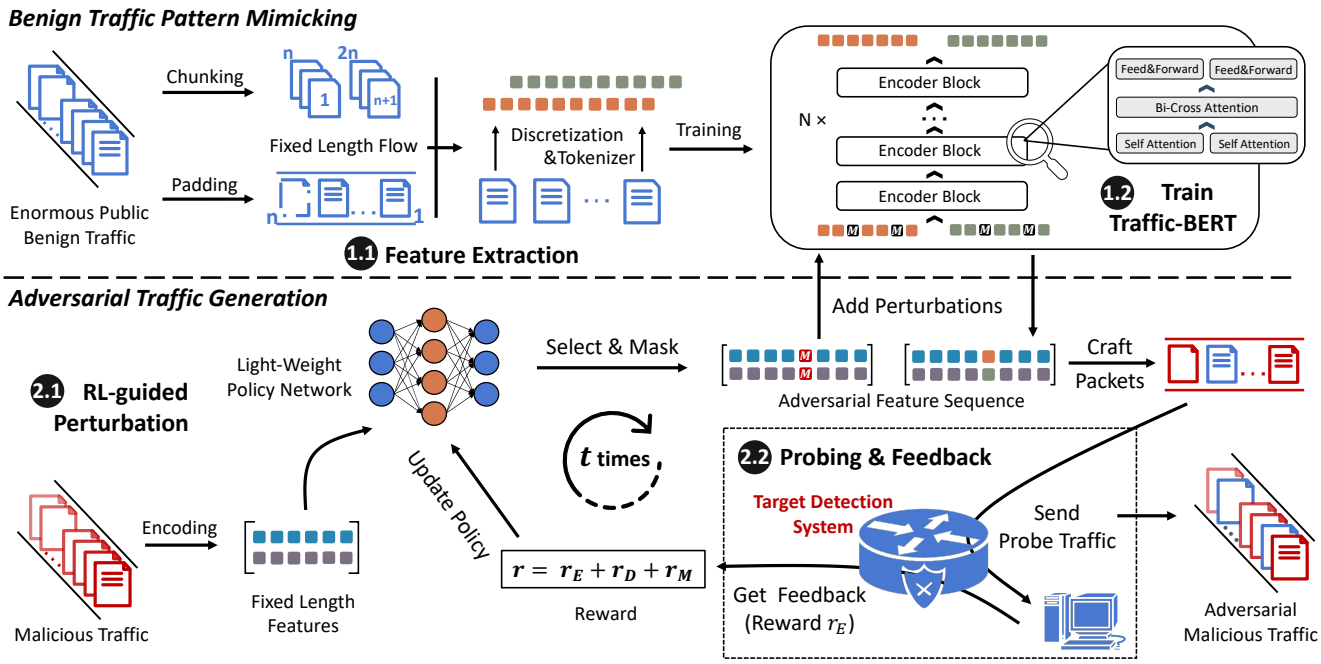


Fig. 2. **High-Level Design of NetMasquerade.**

feature extractors and training datasets. That means, the attacker treats target systems as a strict (i.e., hard-label) black-box, which differs from traditional white-box and gray-box attacks that either need to access ML models (e.g., obtain gradients) [96] or datasets [65] for better transferability. This black-box setting meets the real-world scenarios, as most traffic detection systems are closed-source software [12] or outsourced cloud services [16], [1], effectively preventing attackers from obtaining any information.

However, the attacker can conduct a reconnaissance phase to gather pass/fail feedback from the target detection system. Specifically, the attacker sends probe traffic to remote hosts behind the target detection systems. The probe traffic should exactly mirror the malicious traffic's intended traffic pattern (i.e., matching packet sizes and inter-packet delays) without embedding the original malicious payload (i.e., the attacker can freely embed payloads of the same size). For TCP flows, any return packet (e.g., RST, ACK, SYN-ACK) from the destination [97] signals that the probe has traversed the IDS, whereas the complete absence of a reply within the retransmission window indicates the blockage [24]. For UDP flows, the attacker could employ a stateful application-layer protocol (e.g., QUIC [46]) to induce a response. If the destination port is closed, the attacker would typically observe ICMP Unreachable when the traffic successfully passes the detection [73], [80]. However, no such message will be received if the flow is blocked. By checking for a response within a fixed timeout, the attacker assigns a pass/fail label to each probe. Besides, side-channel techniques (e.g., hop-limited TTL [18], [48], IPID [26], [75]) can further reveal whether the traffic reaches the destination. Overall, this binary indicator (i.e., hard-label) enables the attacker to refine subsequent adversarial traffic generation. In addition, the attacker can access benign traffic from public datasets [99], which differ from those used to train the traffic detection model. We supplement additional discussions regarding our assumptions in § VI.

III. The Overview of NetMasquerade

A. Key Observation

We begin by noting that modern traffic detection [1], [12], [16] operates as a strict (i.e., hard-label) black-box system from the perspective of typical malicious attackers. All model implementations and training details are concealed, which significantly degrades the performance of existing white-box [65], [82] and gray-box [37], [65] methods. In contrast, such systems often drop or throttle [102], [100], [29] malicious traffic while allowing benign traffic to pass. This behavioral asymmetry inherently yields a feedback signal, motivating us to interactively probe the detector's decision boundary and steer malicious traffic within that boundary. We accomplish this process with reinforcement learning.

However, Internet traffic spans a wide variety of protocols and use cases. A naive design can degrade RL performance (see § IX-D), while task-specific attacks fail in heterogeneous environments [58]. Fortunately, existing studies show that benign traffic distributions tend to be denser, whereas malicious traffic is often more sparse [76]. This density gap encourages us to morph the malicious flow so that its features migrate toward the benign manifold while preserving attack semantics. To achieve this, we develop an effective semantic model (i.e., Traffic-BERT) to capture benign traffic patterns, thereby guiding RL training through its learned representations. Finally, we introduce a two-stage divide-and-conquer training framework, along with several additional mechanisms, to significantly

reduce the overhead introduced by integrating Traffic-BERT with the RL process while preserving the effectiveness of the generated adversarial traffic.

B. High-Level Architecture

Figure 2 shows the two stages of NetMasquerade, a black-box evasion attack method. In the first stage, it captures the benign traffic patterns. In the second stage, it generates adversarial traffic based on the traffic patterns.

Benign Traffic Pattern Mimicking. In this stage, we focus on comprehensively modeling benign traffic distributions to provide a solid foundation for subsequent adversarial flow generation. To this end, we propose Traffic-BERT, a variant of BERT [19], capable of processing multiple feature sequence inputs and outputs. Specifically, we first extract basic flow features (i.e., packet size sequence and inter-packet delay sequence). Next, we introduce dedicated feature extraction and embedding schemes to reconcile the gap between continuous, variable-length traffic data and the fixed-length, discrete input format typically required by Traffic-BERT. Building on these enriched representations, we propose a cross-feature bidirectional attention mechanism to simultaneously capture global dependencies within each individual feature sequence and across heterogeneous feature modalities. By training Traffic-BERT under a Mask-Fill task, we enable it to learn deep bidirectional dependencies and acquire the capability to contextually complete fine-grained benign features. The trained Traffic-BERT can be directly used in **Adversarial Traffic Generation** to guide the RL optimization process. We will detail the Benign Traffic Pattern Mimicking in § IV.

Adversarial Traffic Generation. In this stage, our goal is to embed the pattern of benign traffic into malicious traffic with minimal modifications while preserving attack semantics and domain constraints. We model this as a standard Markov Decision Process (MDP), and employ deep RL to address complex sequential decision-making. Specifically, NetMasquerade utilizes the Gated Recurrent Units (GRUs) [11], a lightweight neural network, as the policy network and the state-value networks (a.k.a. Q-Networks). This design significantly reduces training time and inference latency while still effectively capturing temporal flow features. By learning an optimal policy to select packet-level feature positions for masking, NetMasquerade leverages Traffic-BERT to fill the masked tokens with benign traffic patterns. The resulting adversarial flow is used to probe the target system. The response provides the core feedback, which we integrate with two novel penalty terms to form a comprehensive reward signal: a *dissimilarity penalty*, which ensures that the final adversarial flow remains close to the original malicious flow while also reducing the required inference steps, and an *effectiveness penalty*, which retains the underlying attack function. This complete reward signal then guides the optimization of the policy network using the Soft Actor-Critic (SAC) [34] algorithm. We will detail the Adversarial Traffic Generation in § V.

Two-Stage Framework Advantages. In many RL applications, models are initialized from expert demonstrations via behavior learning and then deployed as policy networks in downstream tasks [41], [92]. However, this *Pretraining-Finetuning* framework is not suitable for the traffic domain because it introduces significant overhead. By contrast, our design cleanly decouples benign traffic modeling (Stage 1) from adversarial RL optimization (Stage 2). Traffic-BERT learns high-fidelity benign traffic embeddings without entangling with the RL process, avoiding repeated large-scale retraining. Meanwhile, the lightweight policy network incrementally references the embeddings to weave benign patterns into malicious flows, preserving both the efficiency and the effectiveness of the generated adversarial traffic.

IV. Benign Traffic Pattern Mimicking

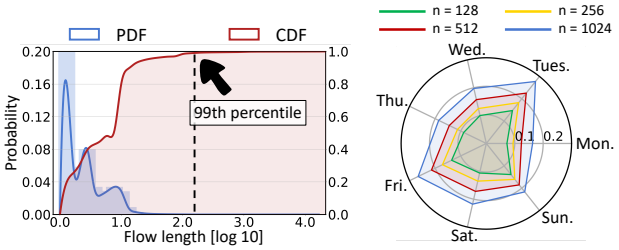
A. Feature Extraction

The feature extraction module encodes network traffic into Traffic-BERT's token sequences. Although various traffic studies have explored related encoding strategies in other contexts [29], [56], [76], [103], they are not directly applicable to Traffic-BERT for two reasons. First, as a language model, Traffic-BERT demands fixed-length inputs, while real-world flows vary widely in size and duration. It is essential to set a base input length that accommodates the majority of flows and provide a mechanism to capture extended flows without information loss. Second, Traffic-BERT requires tokens to reside in a uniformly discrete space, whereas raw network features (e.g., inter-packet delays) may be highly skewed or continuous. We overcome these issues and characterize flows based on statistical insights, as detailed below.

Flow Classification. Traffic-BERT takes sequences of fixed length as input. Typically, the input length is a power of 2, and we assume it to be n, where $n = 2^k$. Figure 3(a) shows the probability density function (PDF) and cumulative distribution (CDF) for flow lengths in the MAWI internet traffic dataset (Jun. 2023) [99]. We randomly sample over $1e7$ flows to plot the figure. Clearly, the distribution of flow lengths exhibits a long-tail pattern, with short flows dominating the majority. We obtain the 99th percentile from the cumulative distribution and select the closest n as our hyperparameter for fixed length. Nonetheless, studies of flow distributions [25] indicate that long flows hold most of the packets. Figure 3(b) shows the bytes retained for different fixed truncations (i.e., n), which can be approximately considered as information entropy, as a proportion of the total bytes of the flow. We randomly sample one week and analyze the complete flow data daily, finding that the information entropy ratios for common values of n do not exceed 0.27. To address this, we apply two complementary strategies:

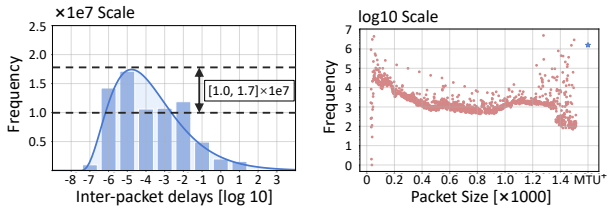
- ***Short Flow Padding.*** If $m \leq n$, we append $n-m$ special padding tokens (i.e., $[PAD]$) to the end of its feature sequence.
- ***Long Flow Chunking.*** If $m > n$, we divide its feature sequence into $m - n + 1$ segments, with each segment's index with a range of $[i, i + n)$, $0 \leq i \leq m - n$.

Feature Representation. Having standardized flow lengths via padding and chunking, we next convert these sequences



(a) Flow length distribution. (b) Flow entropy ratios.

Fig. 3. **Flow length distribution and entropy ratios.**



(a) IPD distribution. (b) Packet size distribution.[1]

[1] Size exceeds MTU, due to misconfigurations/specialized protocols.

Fig. 4. **Packet size and IPD distribution.**

into discrete tokens that Traffic-BERT can ingest. Specifically, we focus on two per-packet attributes: packet sizes and inter-packet delays (IPDs). To optimize this tokenization process, we study the distribution of packet sizes and IPDs in benign internet traffic. For the IPD feature, we observe that after taking the base-10 logarithm, the data exhibits a more uniform distribution across magnitudes. We randomly sample over 80 million packets for our analysis and plot these in Figure 4(a). The analysis shows that frequencies in the range of $[-6, -2]$ (corresponding to $1e-6$ to $1e-2$ seconds) consistently range between $1.0e7$ and $1.7e7$, while the total for other magnitudes falls below $1e7$. Based on this, we set several logarithmically equal-length intervals and hash the IPDs into the intervals, using the interval indices as the corresponding tokens. We adjust the interval lengths to balance the count of elements within each. The packet sizes exhibit a bimodal distribution: predominantly very short or very long (due to fragmentation), with a more uniform distribution in between, as shown in Figure 4(b). Due to its discrete nature, we directly use its value as the token. We use a standard Maximum Transmission Unit (MTU) length as the capacity for the packet size vocabulary. This is because we manipulate the traffic on a per-packet basis, making it impossible to generate a single packet that exceeds the MTU. We categorize features significantly exceeding the MTU under a single class and represent them with the $[UNK]$ token. The token vocabularies for packet sizes and IPDs are independent. Moreover, we add special tokens $[PAD]$ and $[MASK]$ to the vocabulary as placeholders and for masking certain tokens, respectively.

We represent each token by two embeddings: token embedding and position embedding. The complete embedding is constructed by summing both of them together.

- ***Token Embedding.*** A high-dimensional vector representing the token, which is randomly initialized and trained jointly with Traffic-BERT.
- ***Position Embedding.*** A vector annotating the token's relative position in the sequence using sinusoidal functions, similar to Transformer [91]. For chunked features, apart from the first segment, the indices of other segments do not start from 0. This helps the model learn long flow representations more effectively.

B. Traffic-BERT

Although transformer-based architectures have been applied to network traffic modeling [56], [103], most existing BERT-like models are constrained to handling only a single sequence of discrete tokens [19], [59]. In contrast, benign network flows typically involve multiple feature sequences, whose interactions are crucial for capturing real-world traffic patterns. Therefore, the first challenge lies in how to effectively model these interwoven modal features without increasing computational overhead. The second challenge is how to define the Traffic-BERT's training task that enables the model to learn representations of these features directly applicable to adversarial traffic generation, without additional training costs. To address these challenges, we propose a novel bi-cross attention mechanism for efficient fusion of multi-feature data and design a Mask-Fill task to guide Traffic-BERT in acquiring the ability to fill benign traffic packets based on flow context.

Traffic-BERT Structure. The core innovation of Traffic-BERT is the introduction of a bi-cross attention layer within a stack of bidirectional encoder blocks, which is shown in Figure 5. Each block comprises three principal components: (i) self-attention, (ii) bi-cross attention, and (iii) a feed-forward network (FFN), with residual connections linking these layers. Generally, the attention mechanism [91] can be represented as:

$$\mathit{Attn.}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{1}$$

where Q, K, and V represent the Query, Key, and Value, respectively. They are three sets of linear transformations derived from the packet size and inter-packet delay features. The parameter $\sqrt{d_k}$ represents the dimension of the Key.

Each encoder in Traffic-BERT takes size features P and IPD features H as input, which are first processed by the self-attention layer to yield respective hidden states h_P and h_H. Formally:

$$\begin{aligned} h_P &= P + \mathit{Attn.}(Q_P, K_P, V_P), \\ h_H &= H + \mathit{Attn.}(Q_H, K_H, V_H). \end{aligned} \tag{2}$$

Self-Attention is characterized by deriving both the Query and Key from the same sequence, thereby computing the relative importance of each element with respect to all other elements in that sequence. Next, h_P and h_H are fed into the bi-cross attention layer to compute the interrelated attention features, that is, using one feature sequence to query another feature sequence, and vice versa. This can be formulated as:

$$\begin{aligned} h'_P &= h_P + \mathit{Attn.}(Q_{h_P}, K_H, V_H), \\ h'_H &= h_H + \mathit{Attn.}(Q_{h_H}, K_P, V_P). \end{aligned} \tag{3}$$

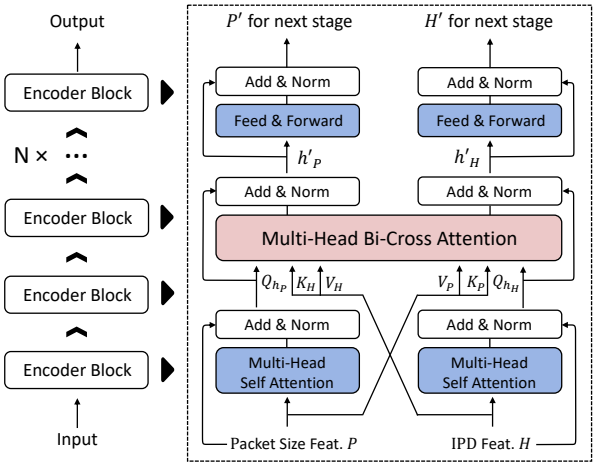


Fig. 5. **Core design of Traffic-BERT.**

Unlike self-attention, bi-cross attention uses the hidden states h_P and h_H as the Query to compute similarity with the other sequence's output from the previous block, and assigns attention weights to the other's Value. The bi-cross attention shares the same complexity $O(n^2 d_k)$ with self-attention, providing an efficient solution for multi-feature sequences from distinct feature spaces. It enables the model to better capture the long-term interactions and dependencies between different feature sequences in network traffic, significantly enhancing the semantic understanding of the benign flow.

The outputs from the bi-cross attention layer are then passed through the feed-forward network layer, serving as the input for the next encoder block. The output from the last encoder block is then passed through a linear layer to obtain the probability distribution of the output tokens.

Traffic-BERT Optimization. We train Traffic-BERT with a Mask-Fill task that generalizes RoBERTa's Dynamic Masking [59] to handle multiple correlated feature sequences. In each training step, we select 15% of token positions for masking. When a position is chosen, tokens in both sequences are masked simultaneously: 80% are replaced with $[MASK]$, 10% with a random token, and 10% remain unchanged. This dual-sequence masking scheme not only compels Traffic-BERT to master deep bidirectional semantics within individual feature sequences, but also reinforces the cross-feature interactions introduced by our bi-cross attention. The trained Traffic-BERT can reconstruct realistic per-packet attributes from partial observations and thus can be directly applied to the second stage in § V.

In a few high-speed traffic generation cases, we optionally apply the Constrained Decoding mechanism [71]. This mechanism restricts the model's output to a predefined range, ensuring that only tokens within this range are considered. However, in most cases, such explicit constraints are unnecessary because the Mask-Fill task itself already biases Traffic-BERT toward valid, realistic traffic patterns.

V. Adversarial Traffic Generation

In this section, we present the technical details of generating adversarial traffic based on deep RL, focusing on its formulation, optimization, and runtime inference. NetMasquerade addresses four main challenges:

1) **Efficient Learning from Limited Feedback.** We carefully design the state space (§ V-A), and leverage Traffic-BERT to guide RL (§ V-B), to achieve efficient exploration under black-box settings.
2) **Preserving Malicious Functionality.** We adopt a tailored action space and an effectiveness penalty (§ V-A) to maintain malicious behavior within domain constraints.
3) **Reducing Training & Inference Overhead.** We introduce a dissimilarity penalty (§ V-A) and employ a two-stage decoupled training scheme (§ V-B) to minimize overhead while ensuring stability.
4) **Enabling Attacks Without Feedback.** We propose an offline reward estimation mechanism (§ V-C) that supports real-world adversarial generation when direct feedback is unavailable during inference.

A. MDP Formulation

NetMasquerade aims to generate effective adversarial malicious traffic that evades black-box detection systems with minimal modifications. To achieve this, we perturb only two domain-agnostic features learned by Traffic-BERT from benign traffic: packet sizes and inter-packet delays. We incorporate specific penalty terms to maintain malicious effectiveness and adhere to domain constraints. This leads to the following definition of our adversarial traffic generation process.

Definition 1 (Adversarial Traffic Generation). Given an ML-based malicious traffic detection system $f \in \mathcal{F}$ and an instance x from the malicious traffic space $\mathcal{X}$, the objective of the adversarial traffic generator $G(\cdot,\cdot) : \mathcal{F} \times \mathcal{X} \to \mathcal{X}$ can be defined as:

$$\begin{aligned} \underset{\tilde{x}}{\text{argmax}} \quad & \mathbb{I}(f(\tilde{x}) \neq f(x)) \\ \text{s.t.} \quad & \tilde{x} = G(f, x), \\ & D(\tilde{x}, x) \leq \tau, \\ & M(\tilde{x}, x, \mathcal{X}) = 1, \end{aligned} \tag{4}$$

where $D(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^+$ is a distance metric that measures the dissimilarity between the adversarial traffic and the original malicious traffic, ensuring that the perturbed instance $\tilde{x}$ is close to the original instance x within the threshold τ, and $M(\cdot,\cdot,\mathcal{X}) : \mathcal{X} \times \mathcal{X} \to \{0,1\}$ is an equivalence indicator that indicates whether the perturbed instance $\tilde{x}$ is equivalent in effectiveness to the original malicious instance.

Note that this optimization problem is difficult to address directly because the objective and the malicious equivalence constraint M are binary and non-differentiable functions, and the distance metric D depends on the way adversarial traffic $\tilde{x}$ is generated. However, we can overcome these challenges by leveraging RL. To do this, we model the attack procedure as a Finite Horizon Markov Decision Process $MDP = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mathcal{T})$. The definition of such an MDP is as follows:

State Space ($\mathcal{S}$): The state $s_t \in \mathcal{S}$ at time t is represented by the tuple (P_t, H_t), where $P_t = [p_{0,t}, p_{1,t}, \ldots, p_{n,t}]$ represents the sequence of packet sizes at time t and $H_t =$

$[h_{0,t}, h_{1,t}, \ldots, h_{n,t}]$ represents the sequence of inter-packet delays at time t. The initial state $s_0 = (P_0, H_0)$ represents the features extracted from the original malicious traffic.

Action Space ($\mathcal{A}$): The attacker is allowed to modify the features of a single packet or insert a chaff packet (i.e., an intentionally injected, non-functional packet) into the flow per step. Therefore, for each feature sequence within the state s_t of length n, the size of the action space is $2n + 1$, where each action $a_t \in \mathcal{A}$ represents the index of the modification or insertion. Specifically, when a_t is odd, it indicates the attacker's intention to modify the element at position $\lfloor a_t/2 \rfloor$ in each sequence; when a_t is even, it indicates the intention to add a new element at position $a_t/2$ in each sequence. NetMasquerade does not perturb existing payloads to maintain traffic functionality and avoid introducing detectable artifacts.

Transition Probabilities ($\mathcal{P}$): Our environment is deterministic. That is, for any given state s_t and action a_t, the state transition probability $\mathcal{P}$ can be represented as:

$$\mathcal{P}(s_{t+1}|s_t, a_t) = \begin{cases} 1 & \text{if } s_{t+1} = \textit{Trans}(s_t, a_t), \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Reward Function ($\mathcal{R}$): The reward function is a composite of three distinct components, formalized as

$$r(s_t, a_t) = r_E(s_t, a_t) + \beta \cdot r_D(s_t, a_t) + \gamma \cdot r_M(s_t, a_t), \tag{6}$$

where β and γ are non-negative hyperparameters.

The term r_E aligns with the optimization objective in (4), which can be defined as:

$$r_E(s_t, a_t) = \frac{N_{\text{evade}}(s_{t+1}) - N_{\text{evade}}(s_t)}{N_{\text{total}}}, \tag{7}$$

where $N_{\text{evade}}(\cdot) : \mathcal{S} \to \mathbb{Z}_0^+$ represents the number of non-chaff packets that evade the target, and $N_{\text{total}} = n$ represents the total number of packets in the original malicious traffic.

The term r_D is the dissimilarity penalty which is derived from the distance metric $D(\cdot, \cdot)$. In our scenario, we use the *Edit Distance* between the current state s_t and the previous state s_{t-1}. Note that since the attacker modifies or inserts exactly one packet at each step, we have:

$$r_D(s_t, a_t) = -1. \tag{8}$$

r_D serves to minimize the distance between the adversarial and original malicious traffic, ensuring that NetMasquerade achieves its adversarial objectives in as few steps as possible, as each additional step incurs a non-positive reward. In general, this design preserves the stealthiness of adversarial traffic and reduces the probability that the perturbations are detected. On the other hand, NetMasquerade achieves its goals with fewer modifications, thereby accelerating inference speed.

The term r_M is the effectiveness penalty that depends on the specific intent of the attack. For instance, for DoS traffic, r_M can be defined as the rate of the traffic flow. Conversely, for maliciousness that stems from the payload, such as phishing traffic, r_M can be set to zero, as our adversarial traffic generation process does not impact the payload.

Horizon ($\mathcal{T}$): The process terminates in either of two situations: first, in the step $t = \tau$, which is consistent with the constraint $D(\tilde{x}, x) \leq \tau$ in (1) as a measure of the maximum permissible distance between the adversarial and original malicious traffic; second, when the reward $r_E(s_t, a_t) > \xi$, indicating a successful evasion ratio greater than the threshold ξ. This dual-condition criterion guarantees a bounded process.

B. Policy Optimization

Algorithm 1 shows the process of training NetMasquerade. Given the MDP setup as defined in § V-A, a sampled MDP trajectory will be $(s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_{\tilde{t}}, a_{\tilde{t}}, r_{\tilde{t}})$, where $\tilde{t} \leq \tau$. To handle the problem's large discrete action space, we employ the Soft Actor-Critic (SAC) algorithm for optimization, which is well known for strong exploration capabilities.

The SAC algorithm is an off-policy maximum entropy RL method, aiming to maximize the balance between expected return and entropy, where the entropy signifies the randomness of the policy. Its objective function is given by:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_t \eta^t \left(r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t)) \right) \right], \tag{9}$$

where π is the stochastic policy to be optimized, α is the temperature hyperparameter that controls the trade-off between exploration and exploitation, η represents the discount factor, and the entropy of the policy $\mathcal{H}(\pi(\cdot|s_t))$ is defined as the expected value of the negative log-probability of the actions taken according to the policy:

$$\mathcal{H}(\pi(\cdot|s_t)) = \mathbb{E}_{a_t \sim \pi(\cdot|s_t)}[-\log \pi(a_t|s_t)]. \tag{10}$$

Given the optimization objective, we build a policy network to approximate the optimal policy π^*, for which we employ Gated Recurrent Units (GRUs) as the backbone. We choose GRUs for two reasons: on the one hand, as a classical type of Recurrent Neural Network (RNN), GRUs are capable of understanding the semantics within the traffic feature sequences; on the other hand, compared to the more computationally demanding Traffic-BERT, GRUs offer a balance between complexity and performance, enhancing the training efficiency of the reinforcement learning model.

In each step t, the policy network takes as input the concatenated feature sequences of packet sizes and IPDs at state s_t, and outputs a distribution over the action space $\mathcal{A}$, as detailed in § V-A. An action a_t is then sampled from this distribution. Notably, when a_t is odd, the $\lfloor a_t/2 \rfloor$-th element of the inter-packet delay sequence of state s_t is replaced with a $[MASK]$ token, indicating the attacker's intent to modify the transmission timestamp of the packet at that position. Consequently, the state s_t is transformed into

$$\begin{aligned} s'_t \triangleq (P'_t, H'_t) = &([p_{0,t}, p_{1,t}, \ldots, p_{n,t}], \\ &[h_{0,t}, h_{1,t}, \ldots, h_{\lfloor a_t/2 \rfloor - 1,t}, [MASK], h_{\lfloor a_t/2 \rfloor + 1,t}, \ldots, h_{n,t}]). \end{aligned} \tag{11}$$

In this context, the packet size sequence remains unaltered as changes to packet sizes might violate the domain constraints

Algorithm 1 NetMasquerade Training Process

Initialize policy network $\pi_\phi(a|s)$, Q-networks $Q_{\omega_1}(s,a)$, $Q_{\omega_2}(s,a)$, and experience replay buffer $\mathcal{B}$
for each iteration **do**
  Sample a malicious flow and get initial state s_0
  for each environment step t **do**
    Observe state s_t and select action $a_t \sim \pi_\phi(\cdot|s_t)$ based on the current policy
    Modify s_t by inserting $[MASK]$ or replacing features with $[MASK]$ to produce s'_t
    Use Traffic-BERT for Mask-Fill task to fill $[MASK]$, obtaining s_{t+1}
    Restore s_{t+1} to adversarial malicious traffic
    Send the adversarial malicious traffic and compute reward $r_t = r_E + \beta \cdot r_D + \gamma \cdot r_M$
    Store transition tuple (s_t, a_t, r_t, s_{t+1}) in $\mathcal{B}$
    if $|\mathcal{B}|$ exceeds minimum replay buffer size **then**
      Sample mini-batch $\{s_{\bar{t}}, a_{\bar{t}}, r_{\bar{t}}, s_{\bar{t}+1}\}$ from $\mathcal{B}$
      Compute target value for each Q-network: $y_{\bar{t}} = r_{\bar{t}} + \eta \min_{i=1,2} Q_{\bar{\omega}_i}(s_{\bar{t}+1}, \pi(\cdot|s_{\bar{t}+1})) - \alpha \log \pi(a_{\bar{t}}|s_{\bar{t}})$
      Update Q-networks: $\omega_i \leftarrow \omega_i - \lambda_Q \nabla_{\omega_i} \sum (Q_{\omega_i}(s_{\bar{t}}, a_{\bar{t}}) - y_{\bar{t}})^2$
      Update policy network: $\phi \leftarrow \phi - \lambda_\pi \nabla_\phi \sum (\alpha \log(\pi_\phi(a_{\bar{t}}|s_{\bar{t}})) - Q_{\omega_i}(s_{\bar{t}}, a_{\bar{t}}))$
      Update target Q-networks: $Q_{\bar{\omega}_i} \leftarrow \lambda Q_{\omega_i} + (1-\lambda) Q_{\bar{\omega}_i}$
      Update entropy temperature α: $\alpha \leftarrow \alpha - \lambda_\alpha \nabla_\alpha \sum (-\alpha \log(\pi(a_{\bar{t}}|s_{\bar{t}})) - \alpha \mathcal{H}_0)$
    end if
  end for
end for

of the packet. Correspondingly, when a_t is even, a $[MASK]$ token is inserted at the $a_t/2$ position of both feature sequences, indicating the attacker's intention to insert a chaff packet at that position. In this case, the state s_t is transformed into

$$\begin{aligned} s'_t &\triangleq (P'_t, H'_t) \\ &= \big([p_{0,t}, p_{1,t}, \ldots, p_{a_t/2-1,t}, [MASK], p_{a_t/2,t}, \ldots, p_{n,t}], \\ &\quad [h_{0,t}, h_{1,t}, \ldots, h_{a_t/2-1,t}, [MASK], h_{a_t/2,t}, \ldots, h_{n,t}]\big). \end{aligned} \tag{12}$$

Considering that the fixed length n may exceed the actual length of a flow, not all actions are feasible. To address this issue, we employ an Invalid Action Masking mechanism [44], adjusting the probabilities of infeasible actions to a large negative value, and then re-normalizing the probability distribution to ensure the effectiveness of the chosen actions.

Once s'_t is obtained, the attacker leverages Traffic-BERT in the Mask-Fill task to embed benign traffic patterns, thereby deriving the next state s_{t+1}. During this process, Traffic-BERT's parameters are fixed and considered a part of the environment. This decoupling of training significantly improves training efficiency.

According to § II, the attacker can conduct a reconnaissance phase to gather pass/fail feedback, i.e., by observing whether there is any response to the attack traffic. Based on this, the attacker restores the adversarial flow from s_{t+1} and sends it to the target. The process is straightforward as NetMasquerade only modifies timestamps or inserts new chaff packets. These chaff packets share the same source/destination IPs and ports as the malicious flow. Their payloads are randomly populated to match the packet size features. Following prior work [40], [37], for example, we use incorrect sequence numbers for TCP, set a short TTL for UDP packets, or send orphan IP fragments for other protocols which are discarded after a reassembly timeout [72]. After the traffic is sent, the attacker calculates the reward $r = r_E + \beta \cdot r_D + \gamma \cdot r_M$. Finally, the resulting transition (s_t, a_t, r_t, s_{t+1}) is stored in the experience replay buffer $\mathcal{B}$ for further policy optimization.

Given the optimization objective and the transitions, we model two state-action value functions (a.k.a., Q-networks) Q_{ω_1} and Q_{ω_2}. The utilization of double Q-networks helps to mitigate the overestimation of action values. Following the Soft Bellman Equation [33], each Q-network can be updated by minimizing the following loss function:

$$\begin{aligned} \mathcal{L}_Q(\omega) &= \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{B}}\left[\frac{1}{2}\left(Q_\omega(s_t,a_t) - y_t\right)^2\right], \\ y_t &= r_t + \eta(\min_{j=1,2} Q_{\bar{\omega}_j}(s_{t+1}, \pi(\cdot|s_{t+1})) \\ &\qquad - \alpha \log \pi(a_t|s_t)), \end{aligned} \tag{13}$$

where $Q_{\bar{\omega}}$ represents the target Q-network [63], which helps to smooth the learning updates. The target Q-networks are updated using Q-networks:

$$Q_{\bar{\omega}_i} \leftarrow \lambda Q_{\omega_i} + (1-\lambda) Q_{\bar{\omega}_i}. \tag{14}$$

The policy network optimization is achieved by minimizing the Kullback-Leibler Divergence from the exponential of the Q-network. This results in the following loss function:

$$\mathcal{L}_\pi(\theta) = \mathbb{E}_{s\sim\mathcal{B}, a\sim\pi}\left[\alpha \log(\pi(a|s)) - \min_{j=1,2} Q_{\bar{\omega}_j}(s,a)\right]. \tag{15}$$

Also, following [35], we employ an automated entropy adjustment mechanism for the temperature parameter α:

$$\mathcal{L}(\alpha) = \mathbb{E}_{s_t\sim\mathcal{B}, a_t\sim\pi(\cdot|s_t)}[-\alpha \log \pi(a_t|s_t) - \alpha \mathcal{H}_0]. \tag{16}$$

C. Runtime Inference

Algorithm 2 shows the inference process of NetMasquerade. Unlike the training phase, the attacker might not be able to receive reward feedback r_E from the target detection system during the inference phase, which prevents direct evaluation of the termination time for adversarial traffic generation. To address this, we approximate the expected total reward for each action using the maximum value from the two Q-networks. The termination condition for the inference phase of NetMasquerade is as follows:

$$(t \geq \tau) \vee \left(\max_{i=1,2} Q_{\omega_i}(s_t, a_t) \geq \xi - \beta \cdot r_D - \gamma \cdot r_M\right), \tag{17}$$

Algorithm 2 NetMasquerade Inference Process

Initialize policy network $\pi_\phi(a|s)$ with trained parameters
Initialize Q-networks $Q_{\omega_1}(s,a)$ and $Q_{\omega_2}(s,a)$ with trained parameters
Set step $t = 0$
Transform the malicious flow into initial state s_0
while $t < \tau$ **do**
    Observe state s_t and select action $a_t \sim \pi_\phi(\cdot|s_t)$ based on the policy network
    Modify s_t by inserting $[MASK]$ or replacing features with $[MASK]$ to produce s'_t
    Use Traffic-BERT for Mask-Fill task to fill $[MASK]$, obtaining s_{t+1}
    Calculate Q-Values $q_1 \leftarrow Q_{\omega_1}(s_t, a_t)$, $q_2 \leftarrow Q_{\omega_2}(s_t, a_t)$
    if $\max_{i=1,2} q_i \geq \xi'$ **then**
        break ▷ Termination condition is met
    end if
    $t \leftarrow t + 1$
end while
Restore s_t to the final adversarial malicious traffic

where the threshold ξ is determined empirically from the training phase. Specifically, we monitor the attack success rate (ASR) during training. Once the ASR stabilizes at a high level, indicating a successfully trained agent, the corresponding Q-value is recorded to serve as the threshold. In cases where the attacker cannot compute r_M, the termination condition can be transformed into:

$$(t \geq \tau) \vee \left(\max_{i=1,2} Q_{\omega_i}(s_t, a_t) \geq \xi' \right). \tag{18}$$

VI. Evaluation

A. Experiment Setup

Implementation. Traffic-BERT and the RL pipeline are written in Python v3.8.15 using PyTorch [74] v1.13.1. Each adversarial flow produced by NetMasquerade is delivered over a socket to an Intel DPDK [45] v24.11.1 worker that emits the actual packets. The DPDK process, written in C (compiled with GCC v9.4.0 -O3 via Meson v0.61.5 and Ninja v1.8.2), pre-allocates NUMA-aware mbuf pools, configures a single 1024-descriptor TX queue, and relies on TSC-based busy-wait pacing to preserve μs-level inter-packet spacing, thereby avoiding the NIC's internal burst-coalescing that would otherwise distort the on-wire delay. NetMasquerade runs on a Dell server equipped with two Intel Xeon Gold 6348 CPUs (2×28 cores, 112 threads) and a single NVIDIA Tesla A100 GPU (driver v530.30.02, CUDA [67] v12.1) under Ubuntu v18.04.6 (Linux 5.4.0-150-generic). The DPDK worker interfaces with an Intel 82599SE NIC (2×10 Gb/s SFP+ ports). All hyper-parameters are listed in Table V in the Appendix.

Datasets. We use real-world backbone network traffic traces from Samplepoint-F of the WIDE MAWI project [99], collected in June and August 2023, as background traffic. Following established practices [52], [78], we remove scanning traffic that attempts to connect to more than 10% of IP addresses and apply additional filtering rules [52] to eliminate flooding traffic. We then employ the resulting background traffic in two ways: (i) to train Traffic-BERT, using more than 1 million flows collected in June 2023, and (ii) to supplement the target system's training data with flows from August 2023 when the proportion of benign traffic in the malicious dataset is insufficient (Botnet Attacks, see Table III). Notably, this choice does not compromise the black-box setting, as there is no correlation between the distributions of the datasets.

To closely mirror real-world scenarios and highlight NetMasquerade's task-agnostic capabilities, we replay 4 groups of attacks from multiple datasets, totaling 12 attacks: (i) Reconnaissance and Scanning Attacks, including host-scanning and fuzz-scanning traffic; (ii) Denial of Service Attacks, covering SSDP and TCP SYN flood traffic; (iii) Botnet Malwares, featuring 4 common botnet strains—Mirai, Zeus, Storm, and Waledac; and (iv) Encrypted Web Attacks, encompassing webshell, XSS, CSRF, and encrypted spam traffic. The details of the datasets can be found in the Appendix IX-A.

Target Systems. We deliberately select as attack target 6 existing malicious traffic detection systems that reflect diverse designs. We use 3 advanced traditional ML-based detection systems: Whisper [28], FlowLens [7], NetBeacon [104] and 3 top-performing DL-based systems: Vanilla feature + RNNs, CICFlowMeter [22], [54] + MLP, Kitsune [62]. They operate using different learning approaches like supervised classification [104] and unsupervised anomaly detection [62], [28], at both the flow-level [7] and packet-level [62], and their implementations span both software [62], [28] and programmable switches [104], [7]. More information about malicious detection systems can be found in Appendix IX-B.

Baselines. To validate NetMasquerade, we select 2 classic and 2 state-of-the-art attack methods as baselines:

- Random Mutation. Random Mutation refers to the technique of obscuring traffic by randomly adjusting IPDs. This traditional method has been demonstrated to be powerful in several works [43], [85], and existing attack tools also employ this method [64], [8]. In our experiments, the randomization of time intervals follows a Gaussian distribution based on the malicious flow's mean and variance. The number of mutated packets matches NetMasquerade's modification steps.
- Mutate-and-Inject. We combine Random Mutation and Packet Injection to create a comprehensive attack strategy, which has been used as a standard for evaluating the robustness of advanced detection systems [28]. For Random Mutation, we follow the same rules described above. For Packet Injection, we either inject chaff packets with random sizes and intervals into the malicious flow or duplicate segments of the malicious packets. The modification steps match those in NetMasquerade.
- Traffic Manipulator [37]. Traffic Manipulator is the SOTA attack algorithm capable of generating practical adversarial traffic against malicious traffic detection systems in a gray-box scenario. Traffic Manipulator learns adversarial traffic

TABLE II
ATTACK SUCCESS RATE (ASR) OF NETMASQUERADE AND BASELINES ON DETECTION SYSTEMS

Target System		Methods	Recon.&Scan.		DoS		Botnet				Encrypted Web Attacks				Overall
			Scan	Fuzz.	SSDP	SYN	Mirai	Zeus	Storm	Waledac	Webshell	XSS	CSRF	Spam	
Traditional ML-based systems	Whisper	R.M.	- [1]	0.0100	-	0.2552	0.2324	0.1011	0.2289	0.0585	0.0812	0.0721	0.1717	0.0927	0.1087
		M.I.	0.8907	0.0756	0.1132	0.3346	0.5521	0.5719	0.4590	0.4251	0.6802	0.7010	0.7259	0.7319	0.5218
		T.M.	0.9344	0.9270	0.7712	0.2790	0.6355	0.2551	0.1820	0.3664	0.5839	0.5527	0.6055	0.9072	0.5833
		Amoeba	0.9999	0.9934	0.9999	0.9998	0.9167	0.9254	0.9844	0.8970	0.9999	0.9999	0.9966	0.8381	0.9626
		NetM.	0.9999	0.9965	0.9999	0.9467	0.9988	0.9972	0.9999	0.9355	0.9999	0.9999	0.9999	0.9795	0.9878
	FlowLens	R.M.	-	-	0.1782	0.7660	0.6893	0.0760	0.3846	0.0434	0.0100	-	0.0150	-	0.1802
		M.I.	0.9800	0.1158	0.2375	0.5950	0.9370	0.4941	0.6510	0.3114	0.6391	0.5959	0.6633	0.1313	0.5293
		T.M.	0.0222	0.1525	0.9344	0.9125	0.8591	0.2670	0.8374	0.2899	0.0760	0.0736	0.0036	0.3913	0.4016
		Amoeba	0.9976	0.9442	0.9999	0.9990	0.8776	0.8665	0.9252	0.8000	0.9990	0.9999	0.9295	0.9700	0.9424
		NetM.	0.9999	0.9335	0.9999	0.9995	0.9537	0.9102	0.9990	0.9955	0.9795	0.9999	0.9428	0.9475	0.9717
	NetBeacon	R.M.	-	-	0.5291	0.1823	0.2864	0.0230	-	0.0790	0.6294	0.3916	0.1066	0.1030	0.1942
		M.I.	0.6511	-	0.2285	0.2841	0.5544	0.3455	0.3032	-	0.8781	0.7010	0.6446	0.1134	0.3920
		T.M.	0.6494	0.2435	0.8577	0.4393	0.3047	0.1992	0.4415	0.2180	0.4585	0.5645	0.5294	0.9091	0.4846
		Amoeba	0.9900	0.9999	0.9987	0.9999	0.9999	0.5905	0.6916	0.9727	0.9550	0.9999	0.9894	N/A [2]	0.8490
		NetM.	0.9999	0.9999	0.9999	0.9999	0.9899	0.9449	0.9965	0.9999	0.9999	0.9955	0.9999	0.8448	0.9809
DL-based systems	Vanilla	R.M.	-	0.3660	0.0455	0.5815	0.1163	-	-	0.3299	0.0118	-	0.0050	0.0515	0.1256
		M.I.	0.9510	-	-	0.3355	0.8769	-	0.5415	0.6711	0.6085	0.5353	0.6751	0.1958	0.4492
		T.M.	-	0.0375	0.8600	0.6550	0.0790	0.2232	0.2595	0.1617	0.0492	0.0278	0.0264	0.8636	0.2702
		Amoeba	0.9999	0.9999	0.9999	0.9999	0.8038	0.7156	0.6540	0.2682	0.9975	0.9999	0.9455	0.2538	0.8032
		NetM.	0.9999	0.9985	0.9825	0.9890	0.9817	0.9894	0.9805	0.9687	0.9999	0.9999	0.9999	0.8485	0.9782
	CIC.	R.M.	-	0.0422	0.1100	0.6398	0.5578	0.2467	0.2922	0.0301	0.0151	0.1855	0.4467	0.1031	0.2224
		M.I.	0.2300	0.1367	0.9711	0.5735	0.7111	0.3956	0.5396	0.2122	0.7011	0.6185	0.6598	0.2886	0.5032
		T.M.	0.1444	-	0.9822	0.6520	0.6656	0.1433	0.3026	0.1021	0.0311	0.0445	0.3381	0.6391	0.3371
		Amoeba	0.9999	N/A	0.9999	0.9112	0.9980	0.9999	0.8704	0.8182	0.9800	0.9865	0.9999	N/A	0.7970
		NetM.	0.9999	0.9744	0.9999	0.9959	0.9999	0.9867	0.8898	0.9810	0.9767	0.9999	0.9999	0.7475	0.9626
	Kitsune	R.M.	-	-	0.2379	0.3744	0.2949	0.0360	0.0990	0.2901	-	0.0277	0.0374	-	0.1165
		M.I.	0.3514	0.4484	0.0913	0.1815	0.8109	0.0801	0.4424	0.6334	0.6159	0.4498	0.3493	0.5359	0.4159
		T.M.	0.9760	0.9860	0.7848	0.5590	0.9049	0.4735	0.8318	0.7878	0.8884	0.8965	0.8406	0.6949	0.8020
		Amoeba	0.9339	N/A	0.8949	0.9292	0.9915	0.9449	0.7256	0.4595	0.4355	0.7814	0.7017	N/A	0.6498
		NetM.	0.9049	0.9850	0.8218	0.9333	0.9968	0.9359	0.9911	0.9291	0.9219	0.9231	0.9177	0.7522	0.9177

[1] These methods cannot successfully attack the target (ASR < 0.01).
[2] These methods cannot generate legitimate traffic (with illegal packet sizes).

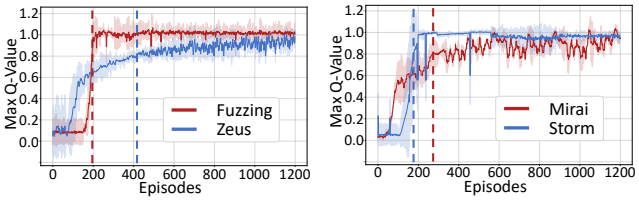
patterns with GANs and adjusts the patterns of malicious traffic with the particle swarm optimization algorithm. We use the open-source implementation of Traffic Manipulator [38] and retrain the model. We use the Kitsune Feature Extractor as the default for Traffic Manipulator, following the paper. This means that for Kitsune, it is a gray-box attack, whereas for other systems, it is a black-box attack.

- Amoeba [58]. Amoeba is designed with a per-packet adjustment technique to circumvent censorship in a black-box scenario. Specifically, Amoeba leverages RL to truncate and pad packets, which are then reassembled by the remote end after passing through the censorship system. We disregard the practical applicability of Amoeba's splitting strategy under different traffic conditions and only evaluate whether it can bypass different types of detection systems. We adopt its open-source implementation and retrain the model.

Metrics. We use AUC (Area Under the Curve) and F1 score to assess the effectiveness of the malicious traffic detection systems, and attack success rate (ASR) to measure the performance of attack methods. Specifically, ASR measures the fraction of malicious flows that are not detected when the IDS uses the threshold rendering the highest F1 score on the validation data. We also measure the bandwidth (megabits per second, Mbps) of both malicious and adversarial traffic to illustrate the impact of perturbations. Moreover, we measure throughput (packets per second, PPS) to show the efficiency.

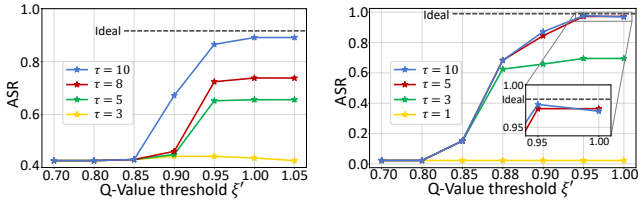
B. Attack Performance

Table II presents the attack success rate against advanced detection systems. NetMasquerade achieves $0.7475 \sim 0.999$ ASR, with an average of 0.9878, 0.9717, 0.9809, 0.9782, 0.9626 and 0.991 against Whisper, FlowLens, NetBeacon, Vanilla, CICFlowMeter and Kitsune detection systems, showing an improvement of 2.61%, 3.11%, 15.53%, 21.88%, 20.78%, and 14.42% over the best performance of the baselines. In 56 out of the 72 evaluated scenarios, NetMasquer-



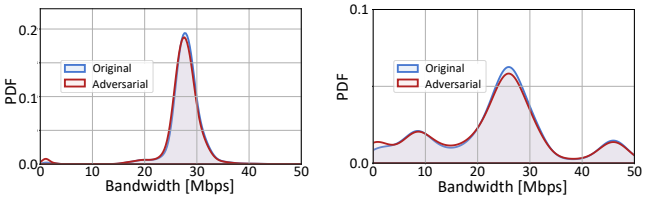
(a) NetBeacon. (b) CICFlowMeter + MLP.

Fig. 6. **Max Q-Value during the training phase.** The shaded region represents a standard deviation of the average evaluation over 5 trials. Curves are smoothed uniformly for visual clarity.



(a) FlowLens / Zeus. (b) Vanilla + RNN / Storm.

Fig. 7. **The relationship between ASR and Q-Value threshold ξ' under different step thresholds τ.** "Ideal" represents the maximum ASR when the attacker can obtain feedback.



(a) SSDP Flood. (b) SYN DoS.

Fig. 8. **Bandwidth of DoS attack.**

ade achieves the highest attack success rate. By contrast, Amoeba matches or exceeds NetMasquerade's performance in 26 scenarios, yet in certain cases its success rate plummets below 30% and even produces flows with illegal packet sizes. Moreover, Amoeba relies on truncating and padding every single packet to craft adversarial traffic, requiring cooperation from the receiving endpoint to reassemble these packets. As a result, this packet-level manipulation can fail in practical attack scenarios (e.g., spam), where such coordination is typically unavailable. Meanwhile, we observe that Traffic Manipulator's performance drops significantly under the black-box setting (i.e., when the attacker cannot access the feature extractor), while NetMasquerade maintains its capability under all scenarios. Figure 6 shows the max Q-Value during the training phase. NetMasquerade achieves 90% convergence in less than 420 episodes in 6(a) and converges within 300 episodes in 6(b), demonstrating strong convergence ability. This ensures that the RL stage incurs low training overhead. We can define the Q-Value threshold ξ' according to the training curve.

NetMasquerade is a stealthy attack capable of generating effective adversarial traffic with minimal modifications. We set the step threshold τ case-by-case while ensuring it is no more than 10. Random Mutation and Mutate-and-Inject perform poorly under the same step setting, as shown in Table II. Amoeba, Traffic Manipulator and other traffic obfuscation methods [61] typically modify most of the packets, making the attack easy to detect. Figure 7 illustrates the relationship between ASR and Q-Value threshold ξ' under different step thresholds τ. In Figure 7(a), we show the effect of attacking FlowLens on the Zeus dataset, which is a complex scenario (i.e., ASR $= 0.9102$). NetMasquerade achieves near-optimal attack rates within no more than 10 steps of modifications. According to Figure 7(b), we find that ASR is not always positively correlated with the number of modification steps at the same ξ, especially in the neighborhood of higher ASR values. An excessively high τ may lead the algorithm to make excessive modifications when the ξ cannot be satisfied.

NetMasquerade ensures the effectiveness of adversarial traffic. Our focus is on two types of malicious traffic: SSDP Flood and SYN DoS, whose effectiveness is demonstrated by high rates. We define the effectiveness penalty r_M as the sending rate of the flow. Additionally, by post-processing Traffic-BERT (see § IV-B for details), we can limit the range of IPD perturbations. We measure the bandwidth distributions of the original and adversarial traffic, as shown in Figure 8. The KL divergence between the adversarial and original traffic distributions for SSDP Flood and SYN DoS is 0.009 and 0.013, indicating that NetMasquerade does not significantly change the bandwidth distribution. On the other hand, the negligible KL divergence confirms that NetMasquerade does not introduce any new, delay-related artifacts that can be readily detected.

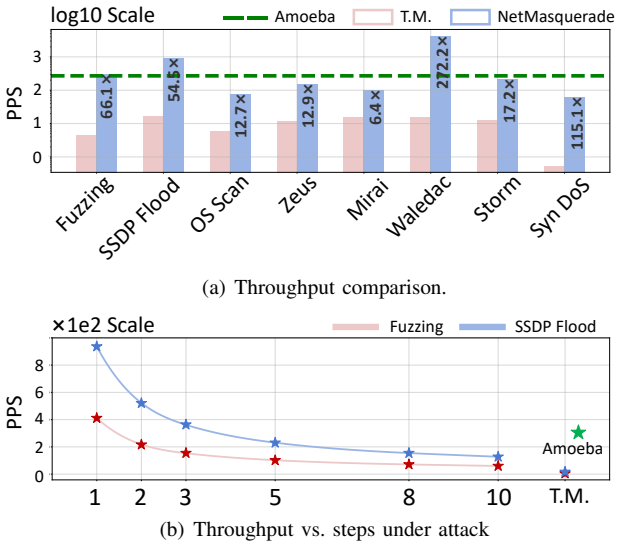
C. Overhead and Efficiency

Training Overhead. To be practical in real-world scenarios, our attack must be prepared efficiently. Our framework achieves this through the two-stage design that separates computationally intensive training from rapid online execution.

- **Stage 1 (Benign Traffic Pattern Mimicking)** is pre-trained offline on publicly available benign traces. Since this stage can be completed well before the actual attack commences, its end-to-end training cost (about $\sim$ 75 hours on our testbed) does not affect the online generation speed.
- **Stage 2 (Adversarial Traffic Generation)** runs online during the actual attack setup. We measure the time required for the RL loop (i.e., adversarial flow generation $\rightarrow$ DPDK emission $\rightarrow$ feedback reception $\rightarrow$ policy update) to converge. On our testbed, this stage is highly efficient, with the policy typically converging within just 1 hour.

Overall, the two-stage design shifts most of the overhead to offline training, thereby ensuring timely attack execution.

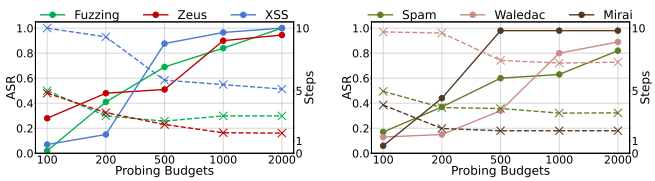
Efficiency. High inference speed is crucial for generating adversarial traffic, especially in high-throughput network environments. Our measurement is end-to-end, including packet extraction and transmitting packets via our DPDK engine. Figure 9(a) compares the throughput of NetMasquerade with baseline methods. Both NetMasquerade and Traffic Manipulator operate at the flow level, yet NetMasquerade achieves an average efficiency improvement of $\sim$ $69.6\times$ over Traffic Manipulator across eight datasets. In contrast, Amoeba performs packet-level inference, maintaining a throughput of approximately 300 ± 10 PPS under various attack scenarios.



(a) Throughput comparison.

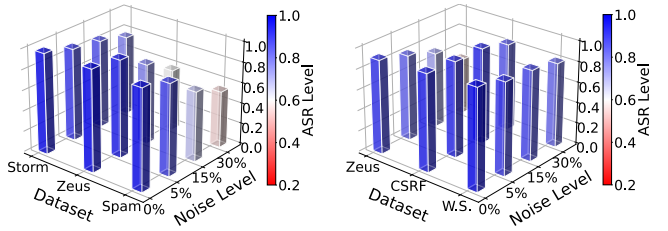
(b) Throughput vs. steps under attack

Fig. 9. **Efficiency of NetMasquerade and baselines.**



(a) NetBeacon. (b) Vanilla + RNN.

Fig. 10. **ASR under different probing budgets.** Solid lines denote ASR; dashed lines denote steps.



(a) Vanilla + RNN. (b) Whisper.

Fig. 11. **ASR under different noise levels.**

Notably, for long-flow attacks (e.g., Waledac), NetMasquerade exhibits a clear advantage in efficiency. Figure 9(b) shows the efficiency curve under different maximum inference steps. By adjusting thresholds ξ' and τ, we can make a trade-off between attack accuracy and inference efficiency. In our experiments, the maximum number of inference steps is generally no more than 10.

D. Deep Dive

Effectiveness under Limited Probes. NetMasquerade assumes that the attacker is able to perform a probing process to collect feedback for model training. However, the probing budget may be limited, as excessive probes could trigger alarms and lead to aggressive countermeasures. To quantify this trade-off, we evaluate the ASR of NetMasquerade given various probing budgets against two detection systems (NetBeacon and Vanilla + RNN) across three distinct datasets each. As shown in Figure 10 (the solid lines), the ASR exhibits a steep learning curve between 200 and $1,000$ probes. For instance, against NetBeacon, NetMasquerade achieves, on average, 35.6%, 70.4%, and 88.9% of its final ASR with budgets of 200, 500, and $1,000$ probes, respectively. The policy typically converges between $1,000$ and $2,000$ probes, although certain complex scenarios (e.g., Vanilla + RNN / Spam) may require a larger budget. This probing load is several orders of magnitude lower than the steady-state workload of a single data-center server ($\sim$500 flows per second) [79] or the capacity of a commercial telemetry system [15] (over $50,000$ flows per-second [14]). Crucially, this volume also sits below the thresholds that mainstream IDS products use to raise scan or anomaly alarms [13].

Furthermore, we examine how the probing budget affects the average number of perturbation steps per flow (the dashed lines in Figure 10). Since each step costs one probe, this count reflects the model's probe utilization efficiency. In the early training phase, the agent requires more steps to explore the action space and identify effective modifications; as training progresses and feedback accumulates, the average steps per flow drop sharply. This shows the policy is learning to use the budget more efficiently, which is key to learning optimal, minimal-modification policies and contributes to the fast convergence of our framework.

Robustness to Noisy Feedback. The feedback that an attacker can collect may be unreliable in real-world deployment. Such noise may be caused by various reasons, including misclassifications, specific configurations, or even adversarial responses from the target designed to mislead the attacker. Figure 11 shows the final ASR achieved against the Vanilla + RNN and Whisper under noise levels of 5%, 15%, and 30%. The results demonstrate that NetMasquerade maintains high efficacy under moderate noise. For example, when the noise level is 5%, NetMasquerade exhibits an average drop in attack success rate of only 0.063. When the reward signal becomes highly unreliable (i.e., given a 30% noise level), the ASR degrades gracefully rather than collapsing. Meanwhile, we observe that increased noise primarily affects the convergence efficiency, particularly the stability of the Q-value. With extended explorations, it is still possible to achieve higher ASR.

Importance of NetMasquerade's Components. NetMasquerade's success relies on its two-stage framework design. We validate the necessity of each stage through a series of ablation studies, which show that removing or replacing either stage significantly reduces the attack success rate. Detailed ablation analysis can be found in Appendix IX-D.

E. Robustness against Defenses

NetMasquerade generates adversarial traffic within the traffic (physical) space, which translates into an unrestricted adversarial attack in the feature space. Consequently, defenses that operate in the feature space, that is, defenses designed to make a model robust to variations within a certain numerical norm of its input feature vectors, such as adversarial training [32], [69] or sample randomized smoothing [17], [55], fail to counter NetMasquerade.

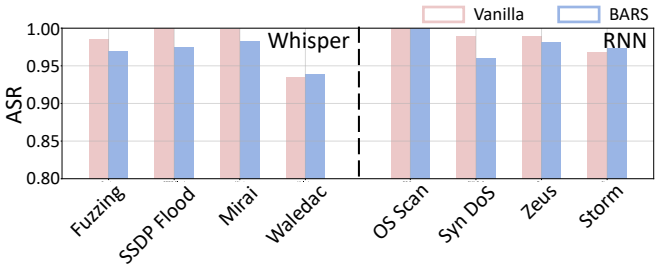


Fig. 12. **ASR against BARS.**

- **BARS** [95]. BARS employs a combined distribution transformer in feature space to map smoothed noise onto arbitrary manifolds, providing certified robustness bounds for certain perturbations in that space. We use the open-source implementation of BARS and deploy it on two detection methods: Whisper and Vanilla + RNN, and the results are shown in Figure 12. NetMasquerade maintains a high ASR across 8 test datasets even against BARS. The primary reason lies in BARS's limited certified bound in the feature space, whereas NetMasquerade performs multi-step manipulations in the traffic space (e.g., inserting additional packets), leading to perturbations that exceed BARS's bound when projected back into the feature space. Moreover, in some datasets, NetMasquerade even attains a higher ASR under BARS, likely because random noise reduces the model's accuracy, even with training data augmentation.

Guidelines for Building More Robust Detection Systems. Overall, it is difficult for the feature-space defenses to defend against NetMasquerade, because even small perturbations in traffic space can lead to large distances in feature space. A straightforward countermeasure is traffic-space adversarial training: augment the training set with packet-level perturbations to strengthen the model's decision boundaries. Second, instead of certifying an $\mathcal{L}_p$ norm in feature space, traffic space certification (e.g., bounding the number of inserted packets) may be more effective against NetMasquerade. Finally, NetMasquerade's training process implicitly assumes a static decision boundary from the target model. Thus, the defense models can introduce randomness at inference to avoid static decision boundaries, such as altering the model architecture [21] or parameters [60]. Such techniques would force the attacker to optimize against a distribution of models rather than a single target, expanding the exploration space. We will explore these strategies in future work.

VII. Related Work

ML-based Malicious Traffic Detection. For generic detection, various methods have been developed to learn flow-level features, such as frequency domain features [28], distribution features [7], statistical features [101], and graph features [30]. In particular, existing methods utilize programmable network devices to achieve high efficiency, e.g., NetBeacon [104] installed decision trees on programmable switches, N3IC implemented binary neural networks on SmartNICs [83]. In contrast to these flow-level detections, Kitsune [62], nPrintML [42], and CLAP [105] learned packet-level features of malicious traffic. For task-specific detection, several studies aimed to detect malware behaviors. For example, Tegeler *et al.* [89] detected communication traffic from botnets. Similarly, Tang *et al.* [87] detected malicious web traffic. Dodia *et al.* [20] identified malicious Tor traffic from malware. Furthermore, Sharma *et al.* [81] and Tekiner *et al.* [90] captured attack traffic targeting IoT devices.

On the Robustness of Traffic Detection. Robustness issues are prevalent in traffic analysis systems, i.e., attackers can easily construct adversarial traffic examples to trick the systems into misclassifying traffic. First, Fu *et al.* [27] revealed that attackers can easily mimic benign traffic to evade the traditional methods [62], [83] by simply injecting random noise. Such observations necessitate robust detection methods [30], [28], [27]. Second, advanced evasion strategies are developed, which optimize the adversarial traffic examples according to the outputs of white-box [96], [82], [70] and grey-box detection models [37]. These methods are different from our hard-label black-box evasion. Additionally, existing studies analyzed the robustness of traffic analysis other than traffic detection, e.g., improving robustness for web fingerprinting [65], [6], which are orthogonal to our evasion attack.

Common Issues of ML-Based Security Application. Sommer *et al.* [84] analyzed why ML-based traffic detection systems suffer from low usability, and emphasized the importance of considering evasion behaviors of real-world attackers. Arp *et al.* [5] explored the practical challenges associated with ML-based applications, highlighting issues of evasion attacks [62], [23]. Moreover, Alahmadi *et al.* [2], Vermeer *et al.* [93], and Fu *et al.* [31] further demonstrated that existing ML-based traffic detections raised massive false positive alarms. Additionally, Han *et al.* [36], Jacobs *et al.* [47], and Wei *et al.* [98] addressed the explainability of traffic detection systems.

VIII. Conclusion

In this paper, we introduce NetMasquerade, a hard-label black-box evasion attack method specifically devised for malicious traffic detection systems. NetMasquerade employs a two-stage framework. First, NetMasquerade establishes a tailored pre-trained model called Traffic-BERT for capturing diverse benign traffic patterns. Subsequently, NetMasquerade integrates the Traffic-BERT into an RL framework, effectively manipulating malicious packet sequences based on benign traffic patterns with minimal modifications. Also, NetMasquerade introduces dissimilarity and effectiveness penalties, allowing adversarial traffic to retain attack stealth and effectiveness. Extensive experiments show that NetMasquerade enables both high-rate and low-rate attacks to evade 6 top-performing detection systems in 80 attack scenarios, achieving over 96.65% attack success rate on average. Moreover, NetMasquerade applies minimal modifications to no more than 10 steps in all scenarios. Additionally, NetMasquerade can achieves low-latency adversarial traffic generation, demonstrating its practicality in real-world scenarios.

IX. Ethical Considerations

We carefully assess several ethical aspects to ensure that our study adheres to ethical standards. This work is aimed solely at assessing and improving the robustness of traffic detection models, rather than facilitating malicious or unlawful activities. We strictly follow all terms of use, and no private or sensitive data is accessed or disclosed. All analysis relies exclusively on publicly available datasets—Kitsune, PeerRush, HyperVision (malicious traffic), and MAWI (benign traffic)—without intercepting or manipulating any real-world network traffic. Likewise, we do not perform any active attacks or evasions against real detection systems, ensuring no impact on actual network traffic or stability.

Acknowledgment

We thank the anonymous reviewers for their thoughtful comments. This work was supported in part by the National Science Foundation for Distinguished Young Scholars of China (No. 62425201), the National Natural Science Foundation of China (Grant Nos. 62472036, 62472247, 62202258, 62221003, 62132011, 61932016, and U22B2031), and Beijing Nova Program. Yi Zhao and Ke Xu are the corresponding authors.

References

[1] AKamai, "Prolexic," https://www.akamai.com/products/prolexic-solutions, Accessed May 2024.

[2] B. A. Alahmadi *et al.*, "99% false positives: A qualitative study of SOC analysts' perspectives on security alarms," in *Security*. USENIX Association, 2022, pp. 2783–2800.

[3] E. Alhajjar, P. Maxwell, and N. D. Bastian, "Adversarial machine learning in network intrusion detection systems," *Expert Syst. Appl.*, vol. 186, p. 115782, 2021.

[4] B. Anderson and D. A. McGrew, "Identifying encrypted malware traffic with contextual flow data," in *AISec@CCS*. ACM, 2016, pp. 35–46.

[5] D. Arp *et al.*, "Dos and don'ts of machine learning in computer security," in *Security*. USENIX Association, 2022.

[6] A. Bahramali, A. Bozorgi, and A. Houmansadr, "Realistic website fingerprinting by augmenting network traces," in *CCS*. ACM, 2023, pp. 1035–1049.

[7] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. V. Ramos, and A. Madeira, "FlowLens: Enabling efficient flow classification for ml-based network security applications," in *NDSS*. The Internet Society, 2021.

[8] BeichenDream, "Godzilla," https://github.com/BeichenDream/Godzilla, 2021.

[9] L. Bilge *et al.*, "Disclosure: Detecting botnet command and control servers through large-scale netflow analysis," in *ACSAC*. ACM, 2012, pp. 129–138.

[10] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *KDD*. ACM, 2016, pp. 785–794.

[11] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, vol. abs/1412.3555, 2014.

[12] Cisco Systems, "Encrypted traffic analytics white paper," 2021, https://www.cisco.com/c/en/us/solutions/collateral/enterprise-networks/enterprise-network-security/nb-09-encrytd-traf-anlytcs-wp-cte-en.html.

[13] Cisco Systems, Inc., *Cisco Security Manager 4.19 User Guide*, Cisco Systems, Inc., 2018. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/security/security_management/cisco_security_manager/security_manager/419/user/guide/CSMUserGuide.html

[14] ——, *Send On-Premises Flows from Cisco Telemetry Broker or Secure Network Analytics to Secure Cloud Analytics Configuration Guide*, Cisco Systems, Inc., 2023, configuration Guide v7.5.0, Document Version 1.0. [Online]. Available: https://www.cisco.com/c/dam/en/us/td/docs/security/stealthwatch/on-premises-flows/7_5_0_Send_On_Prem_Flows_to_Secure_Cloud_Analytics_DV_1_0.pdf

[15] ——, "Cisco Telemetry Broker Data Sheet," https://www.cisco.com/c/en/us/products/collateral/security/telemetry-broker/ctb-datasheet.html, Mar. 2024.

[16] Cloudflare, "Cloudflare ddos protection products," https://developers.cloudflare.com/ddos-protection/managed-rulesets/adaptive-protection/, Accessed May 2024.

[17] J. Cohen, E. Rosenfeld, and J. Z. Kolter, "Certified adversarial robustness via randomized smoothing," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 2019, pp. 1310–1320.

[18] G. Detal, B. Hesmans, O. Bonaventure, Y. Vanaubel, and B. Donnet, "Revealing middlebox interference with tracebox," in *Proceedings of the 2013 Internet Measurement Conference, IMC 2013, Barcelona, Spain, October 23-25, 2013*, K. Papagiannaki, P. K. Gummadi, and C. Partridge, Eds. ACM, 2013, pp. 1–8.

[19] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT (1)*. Association for Computational Linguistics, 2019, pp. 4171–4186.

[20] P. Dodia, M. AlSabah, O. Alrawi, and T. Wang, "Exposing the rat in the tunnel: Using traffic analysis for tor-based malware detection," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, H. Yin, A. Stavrou, C. Cremers, and E. Shi, Eds. ACM, 2022, pp. 875–889. [Online]. Available: https://doi.org/10.1145/3548606.3560604

[21] M. Dong, X. Chen, Y. Wang, and C. Xu, "Random normalization aggregation for adversarial defense," in *NeurIPS*, 2022.

[22] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and VPN traffic using time-related features," in *ICISSP*. SciTePress, 2016, pp. 407–414.

[23] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *CCS*. ACM, 2017, pp. 1285–1298.

[24] W. M. Eddy, "Transmission control protocol (TCP)," *RFC*, vol. 9293, pp. 1–98, 2022. [Online]. Available: https://doi.org/10.17487/RFC9293

[25] C. Estan and G. Varghese, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," *ACM Trans. Comput. Syst.*, vol. 21, no. 3, pp. 270–313, 2003.

[26] X. Feng, C. Fu, Q. Li, K. Sun, and K. Xu, "Off-path TCP exploits of the mixed IPID assignment," in *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds. ACM, 2020, pp. 1323–1335. [Online]. Available: https://doi.org/10.1145/3372297.3417884

[27] C. Fu *et al.*, "Frequency domain feature based robust malicious traffic detection," *IEEE/ACM Trans. Netw.*, vol. 31, no. 1, pp. 452–467, 2023.

[28] C. Fu, Q. Li, M. Shen, and K. Xu, "Realtime robust malicious traffic detection via frequency domain analysis," in *CCS*. ACM, 2021, pp. 3431–3446.

[29] ——, "Detecting tunneled flooding traffic via deep semantic analysis of packet length patterns," in *CCS*. ACM, 2024, pp. 3659–3673.

[30] C. Fu, Q. Li, and K. Xu, "Detecting unknown encrypted malicious traffic in real time via flow interaction graph analysis," in *NDSS*. The Internet Society, 2023.

[31] C. Fu, Q. Li, K. Xu, and J. Wu, "Point cloud analysis for ml-based malicious traffic detection: Reducing majorities of false positive alarms," in *CCS*. ACM, 2023, pp. 1005–1019.

[32] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *ICLR (Poster)*, 2015.

[33] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 2017, pp. 1352–1361.

[34] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 80. PMLR, 2018, pp. 1856–1865.

[35] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic algorithms and applications," *CoRR*, vol. abs/1812.05905, 2018.

[36] D. Han, Z. Wang, W. Chen, Y. Zhong, S. Wang, H. Zhang, J. Yang, X. Shi, and X. Yin, "Deepaid: Interpreting and improving deep learning-based anomaly detection in security applications," in *CCS*. ACM, 2021, pp. 3197–3217.

[37] D. Han, Z. Wang, Y. Zhong, W. Chen, J. Yang, S. Lu, X. Shi, and X. Yin, "Evaluating and improving adversarial robustness of machine learning-based network intrusion detectors," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 8, pp. 2632–2647, 2021.

[38] ——, "Trafficmanipulator," https://github.com/dongtsi/TrafficManipulator, 2021.

[39] M. Handley, V. Paxson, and C. Kreibich, "Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics," in *USENIX Security Symposium*. USENIX Association, 2001.

[40] M. J. Hashemi, G. Cusack, and E. Keller, "Towards evaluation of nidss in adversarial setting," in *Big-DAMA@CoNEXT*. ACM, 2019, pp. 14–21.

[41] T. Hester, M. Vecerík, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, G. Dulac-Arnold, J. P. Agapiou, J. Z. Leibo, and A. Gruslys, "Deep q-learning from demonstrations," in *AAAI*. AAAI Press, 2018, pp. 3223–3230.

[42] J. Holland, P. Schmitt, N. Feamster, and P. Mittal, "New directions in automated traffic analysis," in *CCS*. ACM, 2021, pp. 3366–3383.

[43] I. Homoliak, M. Teknos, M. Ochoa, D. Breitenbacher, S. Hosseini, and P. Hanácek, "Improving network intrusion detection classifiers by non-payload-based exploit-independent obfuscations: An adversarial approach," *EAI Endorsed Trans. Security Safety*, vol. 5, no. 17, p. e4, 2019.

[44] S. Huang and S. Ontañón, "A closer look at invalid action masking in policy gradient algorithms," in *FLAIRS*, 2022.

[45] Intel, "Data Plane Development Kit," https://www.dpdk.org/, 2025, accessed Apr 2025.

[46] J. Iyengar and M. Thomson, "QUIC: A udp-based multiplexed and secure transport," *RFC*, vol. 9000, pp. 1–151, 2021. [Online]. Available: https://doi.org/10.17487/RFC9000

[47] A. S. Jacobs *et al.*, "AI/ML for network security: The emperor has no clothes," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2022, pp. 1537–1551.

[48] V. Jacobson, "traceroute," ftp://ftp.ee.lbl.gov/traceroute.tar.gz, Dec. 1988, lawrence Berkeley National Laboratory, software distribution.

[49] M. Javed and V. Paxson, "Detecting stealthy, distributed SSH bruteforcing," in *CCS*. ACM, 2013, pp. 85–96.

[50] M. Jiang, B. Cui, J. Fu, T. Wang, L. Yao, and B. K. Bhargava, "RUDOLF: an efficient and adaptive defense approach against website fingerprinting attacks based on soft actor-critic algorithm," *IEEE Trans. Inf. Forensics Secur.*, vol. 19, pp. 7794–7809, 2024.

[51] M. S. Kang, S. B. Lee, and V. D. Gligor, "The crossfire attack," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2013, pp. 127–141.

[52] D. Kopp, J. Santanna, M. Wichtlhuber, O. Hohlfeld, I. Poese, and C. Dietzel, "Ddos hide & seek: On the effectiveness of a booter services takedown," in *Internet Measurement Conference*. ACM, 2019, pp. 65–72.

[53] A. Kuzmanovic and E. W. Knightly, "Low-rate tcp-targeted denial of service attacks: the shrew vs. the mice and elephants," in *SIGCOMM*. ACM, 2003, pp. 75–86.

[54] A. H. Lashkari, G. Draper-Gil, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of tor traffic using time based features," in *ICISSP*. SciTePress, 2017, pp. 253–262.

[55] M. Lécuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified robustness to adversarial examples with differential privacy," in *IEEE Symposium on Security and Privacy*. IEEE, 2019, pp. 656–672.

[56] X. Lin, G. Xiong, G. Gou, Z. Li, J. Shi, and J. Yu, "ET-BERT: A contextualized datagram representation with pre-training transformers for encrypted traffic classification," in *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, F. Laforest, R. Troncy, E. Simperl, D. Agarwal, A. Gionis, I. Herman, and L. Médini, Eds. ACM, 2022, pp. 633–642. [Online]. Available: https://doi.org/10.1145/3485447.3512217

[57] Z. Lin, Y. Shi, and Z. Xue, "IDSGAN: generative adversarial networks for attack generation against intrusion detection," in *PAKDD (3)*, ser. Lecture Notes in Computer Science. Springer, 2022, pp. 79–91.

[58] H. Liu, A. F. Diallo, and P. Patras, "Amoeba: Circumventing ml-supported network censorship via adversarial reinforcement learning," *PACMNET*, vol. 1, no. CoNEXT, pp. 9:1–9:25, 2023.

[59] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized BERT pretraining approach," *CoRR*, vol. abs/1907.11692, 2019.

[60] Y. Ma, M. Dong, and C. Xu, "Adversarial robustness through random weight sampling," in *NeurIPS*, 2023.

[61] R. Meier, V. Lenders, and L. Vanbever, "ditto: WAN traffic obfuscation at line rate," in *NDSS*. The Internet Society, 2022.

[62] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," in *NDSS*. The Internet Society, 2018.

[63] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nat.*, vol. 518, no. 7540, pp. 529–533, 2015.

[64] R. Mudge, "Cobalt strike - adversary simulation and red team operations," https://www.cobaltstrike.com/, 2020.

[65] M. Nasr, A. Bahramali, and A. Houmansadr, "Defeating dnn-based traffic analysis systems in real-time with blind adversarial perturbations," in *USENIX Security Symposium*. USENIX Association, 2021, pp. 2705–2722.

[66] A. M. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *CVPR*. IEEE Computer Society, 2015, pp. 427–436.

[67] NVIDIA, "CUDA Toolkit: A Parallel Computing Platform on GPU," https://developer.nvidia.com/cuda-toolkit, 2025, accessed Apr 2025.

[68] Open Information Security Foundation, "Suricata – Network Intrusion Detection/Prevention and Security Monitoring Engine," Software, Version 8.0.0, Jul. 2025, released 08 Jul 2025. [Online]. Available: https://suricata.io/

[69] N. Papernot, P. D. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2016, pp. 582–597.

[70] A. Piplai, S. S. L. Chukkapalli, and A. Joshi, "Nattack! adversarial attacks to bypass a GAN based classifier trained to detect network intrusion," in *BigDataSecurity/HPSC/IDS*. IEEE, 2020, pp. 49–54.

[71] M. Post and D. Vilar, "Fast lexically constrained decoding with dynamic beam allocation for neural machine translation," in *NAACL-HLT*. Association for Computational Linguistics, 2018, pp. 1314–1324.

[72] J. Postel, "Internet Protocol," Request for Comments 791, 1981. [Online]. Available: https://www.rfc-editor.org/rfc/rfc791

[73] ——, "Internet control message protocol," *RFC*, vol. 792, pp. 1–21, 1981. [Online]. Available: https://doi.org/10.17487/RFC0792

[74] PyTorch, "A Deep Learning Framework," https://pytorch.org/, 2025, accessed Apr 2025.

[75] Z. Qian and Z. M. Mao, "Off-path TCP sequence number inference attack - how firewall middleboxes reduce security," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2012, pp. 347–361.

[76] Y. Qing, Q. Yin, X. Deng, Y. Chen, Z. Liu, K. Sun, K. Xu, J. Zhang, and Q. Li, "Low-quality training data only? A robust framework for detecting encrypted malicious network traffic," in *NDSS*. The Internet Society, 2024.

[77] B. Rahbarinia, R. Perdisci, A. Lanzi, and K. Li, "PeerRush: Mining for unwanted P2P traffic," in *DIMVA*, ser. Lecture Notes in Computer Science, vol. 7967. Springer, 2013, pp. 62–82.

[78] P. Richter and A. W. Berger, "Scanning the scanners: Sensing the internet from a massively distributed network telescope," in *Internet Measurement Conference*. ACM, 2019, pp. 144–157.

[79] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *SIGCOMM*. ACM, 2015, pp. 123–137.

[80] N. N. Scanning, "The official nmap project guide to network discovery and security scanning," *Boston: Nmap Project*, 2009.

[81] R. A. Sharma, I. Sabane, M. Apostolaki, A. Rowe, and V. Sekar, "Lumen: A framework for developing and evaluating ml-based iot network anomaly detection," in *CoNEXT*. ACM, 2022, pp. 59–71.

[82] R. Sheatsley, B. Hoak, E. Pauley, Y. Beugin, M. J. Weisman, and P. D. McDaniel, "On the robustness of domain constraints," in *CCS*. ACM, 2021, pp. 495–515.

[83] G. Siracusano, S. Galea, D. Sanvito, M. Malekzadeh, G. Antichi, P. Costa, H. Haddadi, and R. Bifulco, "Re-architecting traffic analysis

with neural network interface cards," in *NSDI*. USENIX Association, 2022, pp. 513–533.

[84] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2010, pp. 305–316.

[85] E. Stinson and J. C. Mitchell, "Towards systematic evaluation of the evadability of bot/botnet detection methods," in *WOOT*. USENIX Association, 2008.

[86] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *ICLR (Poster)*, 2014.

[87] R. Tang, Z. Yang, Z. Li, W. Meng, H. Wang, Q. Li, Y. Sun, D. Pei, T. Wei, Y. Xu, and Y. Liu, "Zerowall: Detecting zero-day web attacks through encoder-decoder recurrent neural networks," in *INFOCOM*. IEEE, 2020, pp. 2479–2488.

[88] G. Tao, S. An, S. Cheng, G. Shen, and X. Zhang, "Hard-label black-box universal adversarial patch attack," in *USENIX Security Symposium*. USENIX Association, 2023, pp. 697–714.

[89] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel, "Botfinder: Finding bots in network traffic without deep packet inspection," in *CoNEXT*. ACM, 2012, pp. 349–360.

[90] E. Tekiner, A. Acar, and A. S. Uluagac, "A lightweight iot cryptojacking detection mechanism in heterogeneous smart home networks," in *NDSS*. The Internet Society, 2022.

[91] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017, pp. 5998–6008.

[92] M. Vecerík, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. A. Riedmiller, "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards," *CoRR*, vol. abs/1707.08817, 2017.

[93] M. Vermeer, N. Kadenko, M. van Eeten, C. Gañán, and S. Parkin, "Alert alchemy: SOC workflows and decisions in the management of NIDS rules," in *CCS*. ACM, 2023, pp. 2770–2784.

[94] J. Wan, J. Fu, L. Wang, and Z. Yang, "Bounceattack: A query-efficient decision-based adversarial attack by bouncing into the wild," in *IEEE Symposium on Security and Privacy*. IEEE, 2024, pp. 1270–1286.

[95] K. Wang, Z. Wang, D. Han, W. Chen, J. Yang, X. Shi, and X. Yin, "BARS: local robustness certification for deep learning based traffic analysis systems," in *NDSS*. The Internet Society, 2023.

[96] N. Wang, Y. Chen, Y. Xiao, Y. Hu, W. Lou, and Y. T. Hou, "MANDA: on adversarial example detection for network intrusion detection system," *IEEE Trans. Dependable Secur. Comput.*, vol. 20, no. 2, pp. 1139–1153, 2023.

[97] N. Weaver, R. Sommer, and V. Paxson, "Detecting forged TCP reset packets," in *NDSS*. The Internet Society, 2009.

[98] F. Wei *et al.*, "Xnids: Explaining deep learning-based network intrusion detection systems for active intrusion responses," in *Security*. USENIX Association, 2023, p. to appear.

[99] WIDE, "Mawi working group traffic archive," http://mawi.wide.ad.jp/mawi/, 2023, accessed: June 2023.

[100] J. Xing *et al.*, "Ripple: A programmable, decentralized link-flooding defense against adaptive adversaries," in *Security*. USENIX, 2021, pp. 3865–3880.

[101] Z. Xu, S. Ramanathan, A. M. Rush, J. Mirkovic, and M. Yu, "Xatu: Boosting existing ddos detection systems using auxiliary signals," in *CoNEXT*. ACM, 2022, pp. 1–17.

[102] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, Q. Li, M. Xu, and J. Wu, "Poseidon: Mitigating volumetric ddos attacks with programmable switches," in *NDSS*. The Internet Society, 2020.

[103] G. Zhou, X. Guo, Z. Liu, T. Li, Q. Li, and K. Xu, "Trafficformer: an efficient pre-trained model for traffic data," in *SP*. IEEE Computer Society, 2024, pp. 102–102.

[104] G. Zhou, Z. Liu, C. Fu, Q. Li, and K. Xu, "An efficient design of intelligent network data plane," in *USENIX Security Symposium*. USENIX Association, 2023, pp. 6203–6220.

[105] S. Zhu, S. Li, Z. Wang, X. Chen, Z. Qian, S. V. Krishnamurthy, K. S. Chan, and A. Swami, "You do (not) belong here: Detecting DPI evasion attacks with context learning," in *CoNEXT*. ACM, 2020, pp. 183–197.

APPENDIX

A. Details of Datasets

To closely mirror real-world scenarios, we replay 4 categories of malicious traffic, totaling 12 attack types: Reconnaissance & Scanning, Denial of Service (DoS), Botnet, and Encrypted Web attacks. The details are shown in Table III.

- *Reconnaissance and Scanning Attacks*. These attacks identify open ports and services across a wide range of servers by sending specific packets, e.g., by sending ICMP echo requests to determine if a host is active. Adversarial traffic made by NetMasquerade does not influence the effectiveness. Moreover, since these scanning attacks typically do not involve the transmission of payloads, they can bypass payload-based detection methods. Employing a pattern-based detection system is a common way of detecting such attacks. We select two distinct scanning attacks from Kitsune [62]: OS Scan and Fuzz Scan.
- *Denial of Service Attacks*. DoS attacks incapacitate targeted services by inundating them with an overwhelming volume of requests, depleting resources and rendering the services unavailable. We selected SSDP DoS and SYN DoS traffic from Kitsune. Similar to scanning attacks, payload-based detection methods fall short against DoS attacks; however, detecting these attacks becomes feasible when focusing on the characteristics of their patterns.
- *Botnet Attacks*. Botnets, which are large networks of compromised machines, are controlled by attackers via command and control (C&C) channels to conduct malicious activities [9], [89]. We employ the Mirai dataset from Kitsune, and analyze data from three typical botnets: Zeus, Storm, and Waledac, which were collected by PeerRush [77].
- *Encrypted Web Attacks*. Typically, malicious web traffic is encrypted with HTTPS, concealing its malicious behavior within the packet payload. This encryption prevents traditional rule-based methods unable to detect the traffic. Meanwhile, most traditional ML-based detection systems cannot effectively detect the traffic due to their low packet rates [30]. We obtained four common types of Web attack traffic from [31], including automated vulnerability discovery (XSS, CSRF), Webshell, and Spam traffic.

We replay traffic from diverse sources—covering high and low throughput flows, as well as encrypted and unencrypted streams—to demonstrate NetMasquerade's general applicability across varying protocols and tasks. For the Scanning, DoS and Encrypted Web attacks, each target detection system is trained on the malicious and benign traces from the corresponding private datasets. For the Botnet attack, the original datasets contain virtually no benign traffic, so we supplement benign flows with traces from the WIDE MAWI project (August 2023). This choice does not compromise the black-box setting, as the Traffic-BERT model is trained on data from June 2023, ensuring that there is no correlation between the distributions of the datasets. To achieve class balance, we train the target models with thousands of malicious flows and an

TABLE III
DETAILS OF MALICIOUS TRAFFIC DATASETS

Malicious Traffic Dataset		Description	Source	Bandwidth	Enc. Ratio	Mal. Ratio	External Data [1]
Recon.	OS Scan	Scanning for active hosts and operating systems.	Kitsune [62]	0.96 Mbps	0.0%	0.0045	N/A
	Fuzz Scan	Scanning for vulnerabilities in protocols.	Kitsune	27.9 Mbps	0.0%	0.0089	N/A
DoS	SSDP DoS	Amplifying SSDP traffic to flood targets.	Kitsune	27.2 Mbps	0.0%	0.0321	N/A
	SYN DoS	Flooding servers with half-open TCP connections.	Kitsune	23.5 Mbps	0.0%	0.0858	N/A
Botnet	Mirai	Infects IoT devices with the Mirai malware.	Kitsune	0.12 Mbps	0.0%	0.8408	MAWI [2]
	Zeus	Botnet infected by a Zeus trojan.	PeerRush [77]	0.06 Mbps	0.0%	0.9999	MAWI
	Storm	Peer-to-peer botnet spreading malware.	PeerRush	25.3 Mbps	0.0%	0.9628	MAWI
	Waledac	Spam botnet harvesting user data.	PeerRush	13.9 Mbps	0.0%	1.0000	MAWI
Enc. Web Attacks	Webshell	Malicious script enabling remote server control.	H.V. [30]	11.2 Mbps	100.0%	0.0234	N/A
	XSS	Injects malicious scripts into legitimate websites.	H.V.	31.8 Mbps	100.0%	0.0259	N/A
	CSRF	Fools authenticated users into unintended actions.	H.V.	7.73 Mbps	100.0%	0.0236	N/A
	Spam	Bulk messages with phishing / malware.	H.V.	36.2 Mbps	100.0%	0.0238	N/A

[1] We use an external benign dataset when the malicious dataset is nearly 100% malicious.
[2] To ensure a strict black-box setting, we employ the real-world backbone network traces from the WIDE MAWI project's August 2023 dataset [99] to train the additional models, keeping them entirely separate from the June 2023 traces used to train Traffic-BERT.

equal number of benign flows. For the one-class detection system Kitsune, we use only benign samples during training.

B. Details of Target Detection Systems

Target Systems. We use three advanced traditional ML-based malicious traffic detection systems as target systems:

- Whisper [28]. Whisper transforms the patterns of flows into frequency domain features, employing clustering to unsupervisedly learn these features. Similarly, the effectiveness of original traffic is assessed by the distance between frequency domain features and the cluster centers. Whisper is particularly effective at detecting DoS traffic, so we retrain the model on the DoS dataset using its default configuration. For botnet traffic, we replace clustering with a linear classifier to enhance detection capabilities.
- FlowLens [7]. FlowLens samples the distribution of packet-level features on the data plane and uses random forests to learn these features in a supervised manner on the control plane, introducing a new paradigm of malicious traffic detection. We retrain the model with the default model structure and hyperparameters described in the paper.
- NetBeacon [104]. NetBeacon introduces the concept of Intelligent Data Plane (IDP), performing flow-level feature extraction and feature classification with tree-based models directly in the data plane. We reconstruct the feature extraction method described in the paper, select XGBoost [10] as the representative tree model for traffic classification, and adjust hyperparameters to achieve optimal accuracy.

We also implement three top-performing DL-based systems:

- CICFlowMeter [22], [54] + MLP. CICFlowMeter is a widely used feature processor that extracts over 80 time-related features from flow patterns. We employ a four-layer linear neural network to learn these features, with the number of neurons in the hidden layers set to three times that of the input layer. By adjusting the hyperparameters, we achieve the model's best classification performance.
- Vanilla + RNN. The native feature extractor extracts sequences of sizes and IPDs from the flow, without additional feature processing. Given the sequential nature of the features, we use a single-layer LSTM as the model, taking the concatenation of the two feature sequences as input.
- Kitsune [62]. Kitsune dynamically extracts and maintains per-packet features through specially designed feature extractors and uses autoencoders and clustering to learn the features of benign traffic unsupervisedly. The detection of malicious traffic is based on the discrepancy between the autoencoder's output and the original feature input. We retrain the model using its original feature extraction and model structure with default hyperparameters.

Target System Detection Performance. Table IV summarizes the detection performance of 6 target systems on 12 types of malicious traffic. Notably, Kitsune outputs a score indicating how malicious each sample is; we perform a grid search to determine a threshold and compute the corresponding F1 score. We then apply this threshold in the attack experiments to calculate the ASR. All detection systems achieve high AUC and F1 scores, demonstrating strong performance in the absence of evasion attacks.

C. Details of Hyperparameter Settings

The default hyperparameters of NetMasquerade are listed in Table V.

D. Deep Dive into NetMasquerade

Importance of Two-Stage Framework. NetMasquerade consists of two stages: benign traffic pattern mimicking and adversarial traffic generation. To study the importance of each stage, we design three ablation strategies and conduct attacks

TABLE IV
MALICIOUS TRAFFIC DETECTION SYSTEMS' PERFORMANCE

Malicious Traffic Dataset		Traditional ML-based Systems						DL-based Systems					
		Whisper		FlowLens		NetBeacon		Vanilla		CIC.		Kitsune	
		AUC	F1	AUC	F1	AUC	F1	AUC	F1	AUC	F1	AUC	F1
Recon.	OS Scan	0.9978	0.9979	0.9946	0.9947	0.9897	0.9899	0.9720	0.9728	0.9980	0.9980	0.9211	0.9780
	Fuzz Scan	0.9905	0.9899	0.9972	0.9933	0.9913	0.9910	0.9662	0.9663	0.9900	0.9901	0.9952	0.9974
DoS	SSDP DoS	0.9167	0.9231	0.9982	0.9981	0.9995	0.9994	0.9790	0.9790	0.9999	0.9999	0.9900	0.9996
	SYN DoS	0.9879	0.9823	0.9815	0.9800	0.9903	0.9833	0.9616	0.9560	0.9852	0.9849	0.9801	0.9213
Botnet	Mirai	0.9449	0.9458	0.9600	0.9463	0.9444	0.9371	0.9099	0.9156	0.9574	0.9521	0.9322	0.9762
	Zeus	0.9121	0.9056	0.9250	0.8837	0.9279	0.9516	0.9118	0.9041	0.9625	0.9484	0.9246	0.9017
	Storm	0.9495	0.9468	0.9395	0.9415	0.9972	0.9978	0.9233	0.9271	0.9968	0.9982	0.9302	0.9822
	Waledac	0.9505	0.9484	0.9660	0.9653	0.9285	0.9467	0.9299	0.9304	0.9860	0.9862	0.8964	0.8414
Enc.	Webshell	0.9980	0.9979	0.9955	0.9946	0.9943	0.9955	0.9989	0.9874	0.9965	0.9964	0.9996	0.9887
	XSS	0.9975	0.9974	0.9965	0.9965	0.9967	0.9966	0.9845	0.9937	0.9937	0.9984	0.9991	0.9990
	CSRF	0.9944	0.9950	0.9920	0.9920	0.9935	0.9934	0.9819	0.9822	0.9928	0.9927	0.9019	0.6625
	Spam	0.9200	0.9135	0.9780	0.9756	0.9635	0.9622	0.8897	0.8847	0.9900	0.9901	0.8887	0.8690

TABLE V
DETAILS OF HYPERPARAMETERS.

Stage	Hyperparameter	Value	Description
1: Traffic-BERT	n	512	Fixed length of sequences.
	d_k	128	Embedding size / total length of Q, K, V.
	N_LAYERS	6	Number of encoder blocks.
	ATTN_HEADS	8	Number of attention heads.
	D_FF	512	Dimension of feed-forward layer.
	T_SIZE	56	Size of IPD feature vocabulary.
	S_SIZE	1606	Size of size feature vocabulary.
2: RL process	β	$0.01 \sim 0.1$	Weight of r_D.
	γ	$0 \sim 0.2$	Weight of r_M.
	τ	≤ 10	Max step threshold.
	ξ'	$0.8 \sim 1.05$	Stop reward threshold.
	η	1	Discount factor.
	λ	0.9	Soft update weight of Q-networks.
	$\mathcal{B}$	$1e5$	Size of experience replay buffer.
	TARGET_ENTROPY	-10	Desired policy entropy (related to α)

TABLE VI
EFFECT OF TWO-STAGE FRAMEWORK. S_1, S_2-S, S_2-F, NETM. STAND FOR STAGE 1 ONLY, STAGE 2 ONLY WITH STOCHASTIC VALUE, STAGE 2 ONLY WITH FIXED VALUE, AND THE OVERALL NETMASQUERADE.

Target Systems	NetBeacon				CICFlowMeter + MLP			
Methods	S_1	S_2-S	S_2-F	NetM.	S_1	S_2-S	S_2-F	NetM.
OS Scan	0.940	0.990	0.990	0.999	0.479	0.990	0.999	0.999
Fuzzing	0.538	0.936	0.996	0.999	0.063	0.004	0.009	0.974
SSDP Flood	0.001	0.434	0.481	0.999	0.488	0.557	0.639	0.999
SYN DoS	0.058	0.325	0.545	0.999	0.508	0.011	0.754	0.996
Mirai	0.915	0.895	0.915	0.990	0.488	0.856	0.938	0.999
Zeus	0.355	0.508	0.508	0.945	0.347	0.813	0.891	0.987
Storm	0.201	0.233	0.239	0.997	0.647	0.797	0.817	0.890
Waledac	0.750	0.727	0.924	0.999	0.123	0.783	0.880	0.981

on two detection systems, NetBeacon and CICFlowMeter + MLP, across all eight datasets. Table VI shows the ASR.

In the first scenario, we retain stage 1 and replace stage 2 with randomly selecting positions for feature modifications(denoted as S_1). Clearly, under this setting, the attacker cannot find the optimal modification positions, resulting in a significant drop in attack capability. On both datasets, the attack success rate drops by an average of 56%. This result underscores that merely integrating BERT-generated traffic patterns is insufficient to evade detection; the RL step in Stage 2 is crucial for identifying the most strategic insertion points.

In the second scenario, we remove stage 2 and replace it with a new Mask-Fill strategy. The first strategy is to fill in the selected positions with stochastic values between the minimum and maximum values of the same flow feature sequence (denoted as S_2-S). By converting the Markov decision process from deterministic to stochastic, it becomes exceedingly difficult for the RL algorithm to converge. Consequently, we observe that the RL strategy predominantly adds chaff packets because the rewards for this type of action are relatively stable. The second strategy is filling in the selected positions with the average value of the same flow feature (denoted as S_2-F). Due to the variation in sending patterns across different flow segments, this strategy is limited in some cases. As Table 3 illustrates, the attack success rate for these two strategies decrease by an average of 37.4% and 26.8%, and neither strategy is effective against high-speed traffic. This outcome underscores how merely relying on an average-value approach or a random approach to features cannot capture dynamic and peak-driven traffic patterns—an issue that becomes even more pronounced in fine-tuning scenarios such as high-speed traffic. Traffic-BERT, on the other hand, guides the RL training process by offering stable and effective benign-pattern perturbations. Although more complex candidate Mask-Fill rules could be considered, these rules can only be applied during stage 2, which would exponentially expand the action space and lead to an action space explosion.