DiffuPac: Contextual Mimicry in Adversarial Packets Generation via Diffusion Model

Abdullah Bin Jasni
Graduate School of Engineering
Nagaoka University of Technology
Nagaoka, Japan
s203108@stn.nagaokaut.ac.jp

Akiko Manada
Graduate School of Engineering
Nagaoka University of Technology
Nagaoka, Japan
amanada@vos.nagaokaut.ac.jp

Kohei Watabe
Graduate School of Science and Engineering
Saitama University
Saitama, Japan
kwatabe@mail.saitama-u.ac.jp

Abstract

In domains of cybersecurity, recent advancements in Machine Learning (ML) and Deep Learning (DL) have significantly enhanced Network Intrusion Detection Systems (NIDS), improving the effectiveness of cybersecurity operations. However, attackers have also leveraged ML/DL to develop sophisticated models that generate adversarial packets capable of evading NIDS detection. Consequently, defenders must study and analyze these models to prepare for the evasion attacks that exploit NIDS detection mechanisms. Unfortunately, conventional generation models often rely on unrealistic assumptions about attackers' knowledge of NIDS components, making them impractical for real-world scenarios. To address this issue, we present DiffuPac, a first-of-its-kind generation model designed to generate adversarial packets that evade detection without relying on specific NIDS components. DiffuPac integrates a pre-trained Bidirectional Encoder Representations from Transformers (BERT) with diffusion model, which, through its capability for conditional denoising and classifier-free guidance, effectively addresses the real-world constraint of limited attacker knowledge. By concatenating malicious packets with contextually relevant normal packets and applying targeted noising only to the malicious packets, DiffuPac seamlessly blends adversarial packets into genuine network traffic. Through evaluations on real-world datasets, we demonstrate that DiffuPac achieves strong evasion capabilities against sophisticated NIDS, outperforming conventional methods by an average of 6.69 percentage points, while preserving the functionality and practicality of the generated adversarial packets.

1 Introduction

Network Intrusion Detection Systems (NIDS) play a pivotal role in safeguarding the vast array of digital devices and infrastructures that permeate our lives. As of 2023, the global count of active IoT devices is expected to reach approximately 15.14 billion, with projections suggesting a rise to 30 billion by 2030 (Statista [2023]). This explosive growth, fueled by applications spanning from consumer electronics to industrial automation and healthcare, presents formidable security challenges. To meet these challenges, advancements in Machine Learning (ML) and Deep Learning (DL) have significantly bolstered the efficacy of NIDS in monitoring IoT traffic and detecting malicious activities (Talaei Khoei and Kaabouch [2023]; Talaei Khoei et al. [2023]).

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

However, the rapid evolution of generative AI technologies has ushered in a new era of cybersecurity threats, notably through the creation of adversarial packets designed to evade detection by even the most sophisticated NIDS. These AI-driven attacks can emulate and synthesize legitimate network behaviors, presenting an unprecedented challenge to existing security paradigms. Generative models, particularly those trained on extensive datasets of genuine network traffic, can generate adversarial packets that blend malicious functionalities within seemingly normal packet sequences, thus effectively camouflaging their malicious intents.

In response to these evolving threats, it is critical for cybersecurity defenders to deepen their understanding of these generative AI models. By scrutinizing the mechanisms through which these models generate adversarial packets, defenders can better anticipate and counteract adversarial tactics that compromise the detection capabilities of NIDS (Ibitoye et al. [2019]; Khazane et al. [2024]). The urgency to develop innovative solutions that can adapt to and preempt these adversarial tactics is paramount, ensuring the reliability and robustness of NIDS in an increasingly complex threat landscape.

Traditional methods for generating adversarial packets have primarily relied on direct engagement with NIDS or the use of surrogate classifiers, often assuming unrealistic levels of attacker access to NIDS configurations. Table 1 summarizes recent literature on adversarial packet generation, highlighting the limitations of these approaches. Studies using techniques such as Network Emulator (NetEM) and Metasploit (Homoliak et al. [2018]), Generative Adversarial Network (GAN) and Particle Swarm Optimization (PSO) (Han et al. [2021]), and Reinforcement Learning (RL) (Hore et al. [2023]) demonstrate that while these methods successfully modified the packets behavior, their efficacy in evading detection in real-world conditions remains suboptimal. The reliance on detailed knowledge of NIDS models is flawed, as attackers typically operate with limited information about the underlying security infrastructure. This gap underscores the necessity for more practical and effective adversarial generation methods that align with the realistic constraints faced by attackers.

Table 1: Summary of recent literature on adversarial packets generation.

Author	**Year**	**Data Set**	**Classifier-Techniques**	**Algorithm**
Homoliak et al. [2018]	2018	ASNM-NBPO	Surrogate Classifier	NetEM, Metasploit
Hashemi et al. [2019]	2019	CICIDS-2018	Surrogate Classifier	Trial and error
Kuppa et al. [2019]	2019	CICIDS-2018	Surrogate Classifier	Manifold Approx.
Han et al. [2021]	2021	Kitsune, CICIDS-2017	NIDS feature extractor	GAN and PSO
Sharon et al. [2021]	2021	Kitsune, CICIDS-2017	NIDS classifier	LSTM
Hore et al. [2023]	2023	CICIDS-2017, CICIDS-2018	Surrogate Classifier	RL

To address these challenges, we introduce *DiffuPac: Contextual Mimicry in Adversarial Packet Generation via Diffusion Model*, a novel solution that leverages the combined strengths of Bidirectional Encoder Representations from Transformers (BERT) and diffusion model. This innovative fusion not only promises high accuracy in generating adversarial packets but also operates under the realistic assumption that attackers lack direct access to NIDS models. DiffuPac leverages the extensive contextual understanding provided by BERT, which has been trained on diverse datasets representing a wide range of network behaviors, along with the generative capabilities of diffusion models. This fusion results in a sophisticated adversarial tactics where the elements of the attack are seamlessly integrated into the network traffic, making them indistinguishable from legitimate data. This capability represents a significant leap forward, offering a stealthy approach that outmaneuvers current NIDS through advanced mimicry rather than direct confrontation.

In summary, the principal contributions of this paper are as follows: (a) we have pioneered the integration of BERT and diffusion models to create DiffuPac, marking a first in the cybersecurity domain. This novel methodology sets a precedent in the field by blending the advanced contextual comprehension of network traffic with sophisticated generative capabilities to produce adversarial packets that are both stealthy and indistinguishable from genuine traffic; (b) we introduce a unique concatenation strategy coupled with targeted noising techniques. These innovations ensure that the adversarial packets not only blend seamlessly into the network environment but also dynamically adapt to evade modern detection systems; (c) DiffuPac advances a classifier-free approach to adversarial packet generation. This approach challenges traditional dependency on surrogate classifiers, offering a new paradigm that more accurately reflects the constraints and capabilities of real-world attackers; (d) Lastly, our extensive experimental evaluations, conducted on real-world datasets, demonstrate that DiffuPac significantly outperforms existing methods in terms of evasion effectiveness, establishing new benchmarks for the generation of adversarial packets. These contributions collectively push the boundaries of what is possible in the realm of network security, paving the way for more resilient cybersecurity defenses and a deeper understanding of adversarial tactics in network environments.

2 Related Works

Adversarial Attacks on ML/DL-based NIDS. NIDS are crucial for protecting digital infrastructures by monitoring network traffic and identifying potential threats through signature-based and anomaly-based detection paradigms. Signature-based NIDS use pattern matching against predefined threat databases, while anomaly-based NIDS employ machine learning models to detect deviations from benign traffic patterns, providing an advantage in detecting sophisticated threats. The transition to DL in anomaly-based NIDS, as highlighted by Ahmad et al. [2021], has enhanced threat detection due to the capability to learn abstract patterns, though this shift has introduced a vulnerability to adversarial attacks. These attacks modify network data subtly to evade NIDS, a phenomenon first noted in computer vision (Szegedy et al. [2014]) and now challenging DL-based NIDS. Adversarial attack tactics vary based on attacker knowledge, ranging from white-box attacks with full system knowledge to black-box attacks with no classifier knowledge, as outlined by McCarthy et al. [2022]. Black-box scenarios are common in real-world threats since attackers generally lack direct NIDS access, necessitating sophisticated modifications of data to exploit DL model vulnerabilities and blur the distinction between normal and malicious traffic.

Adversarial Attacks Evading NIDS. In the study of NIDS, researchers explore feature-level and packet-level attacks to enhance robustness. Feature-level attacks modify input network features using methods like GANs to mislead classifiers without direct knowledge of their mechanisms. For instance, Yang et al. [2018] used transfer-based and score-based attacks to deceive a Deep Neural Network (DNN) model on the NSL-KDD dataset, while Sheatsley et al. [2022] developed an Augmented JSMA (AJSMA) to ensure realistic feature modifications. However, these still fail to generate executable malicious packets since they do not provide a method for converting modified features into packet sequences. Conversely, packet-level attacks, which modify network packets directly to maintain their malicious intent while evading detection, are particularly effective. As detailed in Table 1, studies such as Hashemi et al. [2019] show successful modifications using non-payload based and mimicking operations. In the survey He et al. [2023], authors emphasized that the practicality and replayability of packet-level attacks make them more dominant, ensuring adversarial packets evade detection while remaining executable. Given this efficacy, our research model, DiffuPac, is designed to excel in this domain, outperforming previous models in robustness and detection evasion, and setting a new standard for NIDS efficacy in adversarial cybersecurity.

Diffusion Models. Represent a breakthrough in generative modeling, offering a novel framework for understanding data as a dynamic and stochastic process. Based on the works of Sohl-Dickstein et al. [2015] and Ho et al. [2020], these models conceptualize data points $\mathbf{z}_0 \in \mathbb{R}^d$ (where d is a positive integer) as the end result of a reverse Markov chain. In the forward diffusion process, data points $\mathbf{z}_0$ are gradually transformed into a noise distribution . This transformation is modeled by a Markov chain starting from the data distribution $q(\mathbf{z}_0)$ and ending in a Gaussian distribution $\mathbf{z}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, where T represents the total number of timesteps in the forward diffusion process. The transitions are defined $q(\mathbf{z}_t|\mathbf{z}_{t-1}) = \mathcal{N}(\mathbf{z}_t; \sqrt{1-\beta_t}\mathbf{z}_{t-1}, \beta_t\mathbf{I})$ for $1 \leq t \leq T$, where β_t as the variance scale. In the reverse process, the model f_θ (usually a U-Net or a Transformer) learns to reconstruct the original data $\mathbf{z}_0$ from the noised data $\mathbf{z}_T$. It does so by iteratively estimating the parameters $p_\theta(\mathbf{z}_{t-1}|\mathbf{z}_t) = \mathcal{N}(\mathbf{z}_{t-1}; \mu_\theta(\mathbf{z}_t, t), \sigma_t^2\mathbf{I})$, where μ_θ and σ_t^2 predict the mean and variance of the distribution.

BERT and Diffusion Models Approaches in Cybersecurity and Other Fields. The application of pre-trained models, such as ET-BERT (Lin et al. [2022]) and NetGPT (Meng et al. [2023]), has revolutionized understanding of network traffic by capturing the intricacies of language and interpreting complex network patterns. Meanwhile, diffusion models have emerged as powerful generative tools capable of producing high-fidelity data for applications like image generation and notably, synthetic network traffic generation. These models leverage a unique denoising training objective to closely mimic the original data distribution, making them promising for generating realistic network

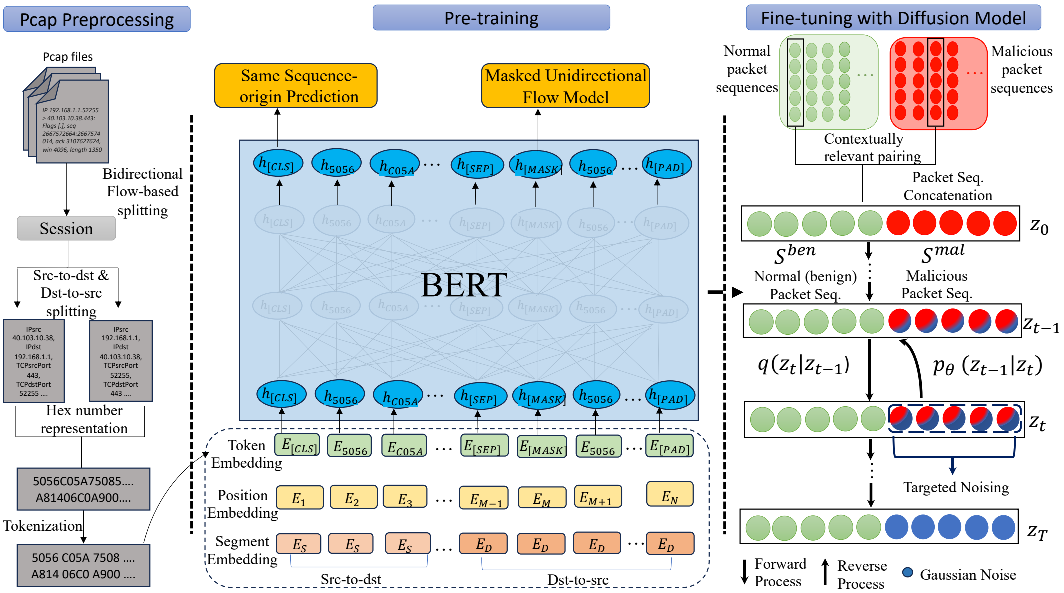

Pcap Preprocessing
Pcap files
Bidirectional Flow-based splitting
Session
Src-to-dst & Dst-to-src splitting
IPsrc 40.103.10.38, IPdst 192.168.1.1, TCPsrcPort 443, TCPdstPort 52255 ...
IPsrc 192.168.1.1, IPdst 40.103.10.38, TCPsrcPort 52255, TCPdstPort 443 ...
Hex number representation
5056C05A75085.... A81406C0A900....
Tokenization
5056 C05A 7508 .... A814 06C0 A900 ....
Pre-training
Same Sequence-origin Prediction
Masked Unidirectional Flow Model
BERT
$h_{[CLS]}$ h_{5056} h_{C05A} ... $h_{[SEP]}$ $h_{[MASK]}$ h_{5056} ... $h_{[PAD]}$
Token Embedding
$E_{[CLS]}$ E_{5056} E_{C05A} ... $E_{[SEP]}$ $E_{[MASK]}$ E_{5056} ... $E_{[PAD]}$
Position Embedding
E_1 E_2 E_3 ... E_{M-1} E_M E_{M+1} E_N
Segment Embedding
E_S E_S E_S ... E_D E_D E_D ... E_D
Src-to-dst
Dst-to-src
Fine-tuning with Diffusion Model
Normal packet sequences
Malicious packet sequences
Contextually relevant pairing
Packet Seq. Concatenation
z_0
S^{ben}
S^{mal}
Normal (benign) Packet Seq.
Malicious Packet Seq.
z_{t-1}
$q(z_t|z_{t-1})$
$p_\theta\ (z_{t-1}|z_t)$
z_t
Targeted Noising
z_T
Forward Process
Reverse Process
Gaussian Noise

Figure 1: Proposed arhitecture is divided into three phases: pcap pre-processing, pre-training and fine-tuning with diffusion models.

traffic for testing and analysis. Despite these advancements, combining the contextual understanding of pre-trained models with the generative prowess of diffusion models remains underexplored in adversarial packet generation. This integration offers a synergistic approach, merging the nuanced comprehension of network behaviors by pre-trained models with the high-quality data generation of diffusion models. By leveraging these technologies together, cybersecurity researchers can enhance the realism of simulated network environments and develop more advanced evasion tactics.

3 DiffuPac

In this section, we introduce DiffuPac, a first-of-its kind adversarial packet generation model. Figure 1 illustrates the overall framework of the proposed model, consisting of three main phases: pcap pre-processing, pre-training and fine-tuning with diffusion model.

3.1 Data pre-processing

In real network environments, traffic contains diverse flows from various applications, protocols, and services, complicating the learning of stable representations. Therefore, we first split pcap (packet capture) files into sessions (bidirectional flows) based on IP addresses, port addresses, and protocols. To refine the training of the BERT model, we further split sessions into unidirectional flows, categorizing them as either source-to-destination (src-to-dst) packet sequence or destination-to-source (dst-to-src) packet sequence. This categorization is crucial for the BERT model's pre-training. Network traffic varies due to diverse protocols and network services, resulting in different formats and patterns. To handle these variations and encoding requirements, we convert each byte to its corresponding hex number and tokenize using WordPiece (Wu et al. [2016]). Each token ranges from 0 to 65535, with a dictionary size of 65536. We also incorporate special tokens `[CLS]`, `[SEP]`, `[PAD]`, and `[MASK]` for training tasks. `[CLS]` is used at the beginning of each sequence and helps in classification tasks. `[SEP]` separates different sequences or segments within the same input. `[PAD]` ensures sequences are of uniform length and satisfy the minimum length requirement. `[MASK]` is used in pre-training task, where it temporarily substitutes tokens to be predicted.

3.2 Pre-training

As shown in Figure 1, the input tokens are processed using an embedding strategy that involves the sum of three types of embeddings: token embeddings, positional embeddings, and segment embeddings. Each embedding has a dimension of 768. *Token Embeddings* are high-dimensional vectors that uniquely represent each token, acting as their exclusive identifiers. *Positional Embeddings* are used to capture the temporal relationships of tokens, ensuring that the model learns to focus on the order of data transmission. *Segment Embeddings* differentiate packets within a single flow, as packets may not inherently share semantic associations, and preserving the order of packets to maintain the temporal sequence of events in a session flow.

We design our pre-training tasks based on the approach described in Devlin et al. [2019]. Our two proposed pre-training tasks aim to capture the contextual relationships between traffic bytes. The first task involves predicting masked tokens to learn the underlying patterns and dependencies within the traffic data. The second task predicts the transmission order by determining whether packets belong to src-to-dst or dst-to-src sequences, thereby capturing the directional flow of the network traffic.

Masked Unidirectional Flow Model. Inspired by BERT's Masked Language Model (MLM) in natural language processing, we adapt this approach for network traffic analysis through our Masked Unidirectional Flow Model. This model is designed to understand and predict the semantic patterns within bidirectional network flows; that are src-to-dst sequence and dst-to-src sequence. During pre-training, each token in the input sequence is masked with a probability of 15%. Amongst these masked tokens 80% are replaced with a `[MASK]` token, 10% are replaced with a random token from the vocabulary, and 10% are left unchanged. This introduces variability that mimics real-world data inconsistencies. The training objective is to minimize the negative log-likelihood of correctly predicting the original tokens at the masked positions. Formally, the loss function L_{MUM} for this task is defined as:

$$L_{\text{MUM}} = -\sum_{i=1}^{n} 1(m_i) \log P(v_i \mid V_{\text{masked}}; \theta), \tag{1}$$

where n is the total number of tokens in the sequence, $1(m_i)$ is an indicator function that is 1 if the token v_i was masked (0 otherwise). m_i indicates the masking status of the i-th token and V_{masked} is the masked sequence. v_i is the actual token at position i and $P(v_i \mid V_{\text{masked}}; \theta)$ is the probability of predicting the original token v_i for given masked sequence and model parameters θ. The transformer encoder, characteristic of BERT, processes V_{masked} to predict each masked token. This architecture leverages self-attention mechanisms to capture dependencies and context effectively, which is crucial for understanding complex patterns in network traffic.

Same Sequence-origin Prediction. Inspired by Lin et al. [2022], this task employs a dedicated binary classifier to determine the directional origin of network traffic, specifically whether packets in a sequence originate from src-to-dst or dst-to-src. This classification enhances the model's understanding of network flows and the contextual relationships between packets, which is crucial for recognizing communication patterns and dependencies in network traffic. For this task, each packet a_r is paired with another packet a_s. 50% of the time, a_s is the next logical packet in the flow (src-to-dst or dst-to-src); the other 50% of the time, it is a randomly chosen packet from the opposite flow. The classifier then predicts whether a_s follows a_r in the correct directional sequence. This pairing strategy improves the model's capability to discern patterns in packet flows. Let $A = \{(a_r, a_s)\}$ be the set of packet pairs, where each pair is labeled with $b_w \in \{0, 1\}$ (1 if a_s follows a_r in the correct flow, 0 otherwise). The loss function L_{SSP} for this task can be formulated as:

$$L_{\text{SSP}} = -\sum_{w=1}^{K} b_w \log P(b_w = 1 \mid a_r, a_s; \theta) + (1 - b_w) \log(1 - P(b_w = 1 \mid a_r, a_s; \theta)), \tag{2}$$

where K is the total number of packet pairs and θ represents the trainable parameters of the classifier. $P(b_w = 1 \mid a_r, a_s; \theta)$ is the probability predicted by the classifier that a_s correctly follows a_r in the given network flow direction.

In summary, the final pre-training objective is the sum of the above two losses, which can defined as:

$$L = L_{\text{SSP}} + L_{\text{MUM}}. \tag{3}$$

Fu et al. [2021] mentioned that the number of packets within each flow can vary significantly. Given the constraints imposed by packet size and the potential volume of network traffic, computational efficiency is a paramount concern. In addressing this problems, as demonstrated in Dai et al. [2023] that the initial packets in a flow contain the most significant information, we limit our analysis to the first three packets of each heavy flow. This means that for each session, we analyze a total of six packets—three from the src-to-dst flow and three from the dst-to-src flow. This strategy ensures a comprehensive view of the session while maintaining efficiency.

3.3 Fine-tuning with Diffusion Models

Forward Process with Packet Sequences Concatenation Strategy. In the fine-tuning phase, the goal is to train the model so that malicious packets can mimic normal packets to bypass NIDS. This involves using packet sequences that are only from the src-to-dst, reflecting the adversary's control over the packets being sent (Hore et al. [2023]). The forward process of our diffusion model begins by embedding both normal (benign) packet sequences $\mathbf{S}^{\text{ben}}$ and malicious packet sequences $\mathbf{S}^{\text{mal}}$. This transformation of discrete packet data into a continuous feature spaces uses an embedding function adapted from Li et al. [2022]. Building upon the groundwork of DiffuSeq (Gong et al. [2023]) approach, which typically involves random merging, our model innovates by leveraging the deep contextual insights provided by the pre-trained BERT model (Details in A.2). This allows us to strategically pair normal and malicious sequences with strong contextual alignments that show similarity in network behavior patterns. These contextually aligned pairs are crucial for mimicking normal traffic and increasing the chances of bypassing NIDS. By integrating these contextually aligned pairs into our diffusion process, the model extends the original forward chain to a new Markov transition $q_\phi(\mathbf{z}_0 \mid \mathbf{S}^{\text{ben}\oplus\text{mal}}) = \mathcal{N}(\text{EMB}(\mathbf{S}^{\text{ben}\oplus\text{mal}}), \beta_0\mathbf{I})$, where $\text{EMB}(\mathbf{S}^{\text{ben}\oplus\text{mal}})$ symbolizes the embedding transformation and concatenation of normal and malicious packet sequences. While $\mathbf{S}^{\text{ben}\oplus\text{mal}}$ denotes the initial concatenated sequence, the forward process gradually perturbs this initial state $\mathbf{z}_0$ through a series of transitions, producing latent variables $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_t$.

Targeted Noising. To enhance our diffusion model's capability for adversarial packet generation, we simplify the model's state transitions by defining $\mathbf{z}_t = \mathbf{x}_t \oplus \mathbf{y}_t$, where $\mathbf{x}_t$ and $\mathbf{y}_t$ correspond to the portions of $\mathbf{z}_t$ that belong to $\mathbf{S}^{\text{ben}}$ and $\mathbf{S}^{\text{mal}}$, respectively. This setup allows us to strategically inject noise into only the malicious packet sequences (represented by $\mathbf{y}_t$), rather than the entire state $\mathbf{z}_t$, during each forward step $q(\mathbf{z}_t \mid \mathbf{z}_{t-1})$. This targeted noising, inspired by DiffuSeq, is pivotal for adapting conventional diffusion models for targeted modification.

Reverse Process With Normal Packet Guidance. Building upon the foundational principles of DiffuSeq, our model introduces an innovative reverse process tailored for more nuanced handling of network traffic data. This process distinctively utilizes normal packet sequences as a guiding framework, enabling a sophisticated denoising technique that treats the concatenated sequences of normal and noise-added malicious packets as a unified unit. This approach effectively "teaches" the model to perceive these malicious elements as integral parts of the normal traffic pattern. A key aspect of this reverse process is the use of the pre-trained BERT model as the denoising engine during packet reconstruction. In this phase, the BERT model undergoes fine-tuning within the diffusion framework, ensuring that it is specifically adapted for reconstructing packets. As a result, the BERT model is not only responsible for contextual understanding but also plays a central role in recovering malicious packets that have been integrated into normal traffic patterns. The conditional denoising process effectively employs Bayesian inference to parameterize the transition probabilities between states, ensuring precise control over each step in the reverse process. These parameterizations are mathematically articulated through the equations: $p_\theta(\mathbf{z}_{0:T}) := p(\mathbf{z}_T)\prod_{t=1}^{T} p_\theta(\mathbf{z}_{t-1}|\mathbf{z}_t)$ and $p_\theta(\mathbf{z}_{t-1}|\mathbf{z}_t) = \mathcal{N}(\mathbf{z}_{t-1}; \mu_\theta(\mathbf{z}_t, t), \sigma_\theta(\mathbf{z}_t, t)$, where $\mu_\theta(\cdot)$ and $\sigma_\theta(\cdot)$ model the mean and variance necessary for the stochastic reverse transition from $\mathbf{z}_t$ to $\mathbf{z}_{t-1}$, respectively. To rigorously ensure that the malicious packets are not only recovered but also convincingly mimic normal packets, our model optimizes a specially formulated variational lower bound L_{VLB}. This objective underscores the model's effectiveness in blending malicious packets within normal packets:

$$\min_\theta L_{VLB} = \min_\theta \left[\sum_{t=2}^{T} \|\mathbf{z}_0 - f_\theta(\mathbf{z}_t, t)\|^2 + \|\text{EMB}(\mathbf{S}^{\text{ben}\oplus\text{mal}}) - f_\theta(\mathbf{z}_1, 1)\|^2 - \log p_\theta(\mathbf{S}^{\text{ben}\oplus\text{mal}} \mid \mathbf{z}_0)\right] \tag{4}$$

This objective is particularly focused on accurately reconstructing the initial state $\mathbf{z}_0$ from the noised states, with a distinct emphasis on ensuring that the malicious components are seamlessly integrated into the normal packets pattern.

Preserving Packets' Integrity. We propose an innovative approach that utilizes a parallel data structure called *"model dictionary"* that clearly distinguishes between mutable and immutable fields within each packet. By categorizing fields into mutable (e.g. TCP flags, TTL, and window size) and immutable (including the critical 5-tuple information and payload), we ensure that modifications during training do not compromise the packet's integrity. The model dictionary serves a dual purpose: it retains the original values of immutable fields and establishes a link to their mutable counterparts. This approach allows for dynamic modification of mutable fields during training without affecting the core attributes of the packet. After training, any changes made to mutable fields are seamlessly integrated with the preserved immutable fields. This recombination ensures that the modified packets maintain their operational integrity and are indistinguishable from genuine traffic in real-world scenarios. By bypassing traditional domain constraints—which typically complicate training and pose convergence challenges—our method simplifies the training process and obviates the need for additional compensatory loss functions (Letcher [2021]). The outcome is a highly effective generation of adversarial packets that are capable of evading detection without compromising the essential characteristics of the original packets.

4 Experiments

We conduct experiments to validate the performance of DiffuPac on 6 types of attacks, against 6 classifiers.

4.1 Experimental Setup

Dataset. The datasets used for this model are Kitsune Dataset (Mirsky et al. [2018]) and CICIDS-2017 Dataset (Sharafaldin et al. [2018]). Initially, *pre-training of the BERT model* utilizes a large subset of unlabeled network traffic to leverage the model's capability to capture diverse traffic patterns, accounting for 60% of the total data. *Fine-tuning phase*, the focus shifts to a smaller, labeled dataset, which constitutes 20% of the total data. This dataset is distinctly partitioned into malicious and normal packets, which is crucial for training the model to mimic malicious packets as normal. *Training the classifier and NIDS* then utilizes another 10% of the total data consisting of labeled portion. *Testing phase* is conducted with the remaining 10% of the data, reserved exclusively for evaluating the model's efficacy. This phase includes testing both original and mimicked malicious packets to rigorously assess the model's real-world applicability and its capability to generalize across unseen data.

Baseline Models. We evaluate two baseline models—Traffic Manipulator (Han et al. [2021]) and TANTRA (Sharon et al. [2021])—both of which represent current strategies for evading NIDS but share a significant limitation in their reliance on specific operational assumptions. Traffic Manipulator formulates the evasions as a bi-level optimization problem, where the first level uses GAN to identify adversarial features closely mimicking the original network features, and the second level utilizes PSO to modify packet characteristics to exploit vulnerabilities in NIDS. TANTRA employs a three-step process involving an LSTM trained to model benign traffic by predicting inter-packet delays. This LSTM then adjusts malicious traffic to mimic these benign patterns before deployment. Both models, however, assume a degree of accessibility to NIDS configurations or feature extractors that is often unrealistic in practical scenarios. This reliance on specific knowledge about NIDS configurations creates a critical dependency that can limit the deployment and scalability of these evasion techniques in diverse operational environments. Conversely, DiffuPac operates under the assumption of zero prior knowledge about NIDS configurations or feature extractors.

Implementation Details. DiffuPac leverages a BERT model with 12 transformer blocks, each featuring 12 attention heads and an embedding dimension of 768, to capture nuanced relationships in network traffic. This setup supports the maximum sequence length of 512. The diffusion model incorporates time step embedding to provide temporal context, enhancing the model's understanding of the reverse process. To manage the computational demands, particularly during the sampling phase, a projection layer reduces the embedding dimension to 128. Additionally, DiffuPac utilizes the FAISS library (Douze et al. [2024]), renowned for its capability to handle large-scale similarity search and clustering of dense vectors.

Experimental Details. Our experimental framework integrates advanced feature extraction tools and a diverse set of machine learning classifiers, building on methodologies established in the Traffic Manipulator research. We employ 2 feature extractors: AfterImage (Mirsky et al. [2018]), which provides detailed packet statistics, and CICFlowMeter (Draper-Gil et al. [2016]), which assesses connection-level metrics. Recognizing the documented shortcomings of the original CICFlowMeter, our research employs a revised version that integrates enhancements and corrections proposed in recent studies (Engelen et al. [2021], Liu et al. [2022]). These modifications are crucial in ensuring more accurate and reliable feature extraction for our analysis. AfterImage and CICFlowMeter offer a comprehensive view of network traffic, capturing both packet-level and flow-level data essential

Table 2: Comparative analysis of attack detection and evasion rates.

(a) Botnet

Feature Extractor	Classifier	Detection			Evasion Rate (*MER*)		
		P	R	F1	GAN & PSO	LSTM	**Ours**
CIC FLowMeter	KitNET	0.84	0.94	0.92	37.24%	**50.69%**	46.48%
	DT	0.79	0.91	0.82	35.88%	49.62%	**60.98%**
	IF	0.99	0.90	0.95	39.78%	42.93%	**49.71%**
	MLP	0.92	0.84	0.86	41.05%	53.87%	**63.05%**
	SVM	0.99	0.92	0.95	88.79%	**91.95%**	64.49%
	LR	0.84	0.91	0.89	24.72%	30.52%	**42.08%**
AfterImage	KitNET	0.96	0.90	0.94	99.18%	**99.79%**	74.46%
	DT	0.79	0.90	0.84	63.42%	67.30%	**72.13%**
	IF	0.99	0.90	0.94	31.48%	**61.08%**	52.79%
	MLP	0.96	0.97	0.97	48.60%	60.97%	**64.92%**
	SVM	0.99	0.90	0.94	40.31%	51.24%	**69.19%**
	LR	0.96	0.90	0.93	53.28%	50.70%	**58.98%**

(b) MITM

Feature Extractor	Classifier	Detection			Evasion Rate (*MER*)		
		P	R	F1	GAN & PSO	LSTM	**Ours**
CIC FLowMeter	KitNET	0.92	0.94	0.91	38.22%	44.39%	**53.87%**
	DT	0.74	0.79	0.76	49.98%	57.77%	**64.12%**
	IF	0.99	0.92	0.94	26.74%	38.64%	**52.99%**
	MLP	0.77	0.72	0.74	52.07%	43.19%	**73.21%**
	SVM	0.74	0.79	0.78	42.11%	45.54%	**60.46%**
	LR	0.73	0.78	0.72	35.87%	**50.04%**	47.48%
AfterImage	KitNET	0.94	0.96	0.93	68.79%	**79.29%**	58.48%
	DT	0.75	0.89	0.84	53.18%	58.15%	**70.04%**
	IF	0.81	0.83	0.86	26.53%	31.27%	**45.71%**
	MLP	0.92	0.90	0.93	50.65%	59.30%	**71.45%**
	SVM	0.99	0.90	0.94	63.51%	57.51%	**66.53%**
	LR	0.91	0.94	0.90	44.68%	46.39%	**52.38%**

for nuanced analysis. These feature extractors are integral to NIDS detection as they preprocess network data into structured features that are then analyzed by ML classifiers. This preprocessing step is crucial for transforming raw network traffic into a format that classifiers can effectively interpret, thereby enhancing the accuracy of the anomaly detection. We utilize a variety of machine learning classifiers for anomaly detection, including *KitNET* (Mirsky et al. [2018]), an ensemble of autoencoders; *Multi-Layer Perceptron (MLP)* for deep learning; *Logistics Regression (LR)*, *Decision Tree (DT)*, and *Support Vector Machine (SVM)* for traditional approaches; and *Isolation Forest (IF)* for outlier detection. Our experiments cover 6 types of attacks—*Man-in-the-Middle (MITM)*, *Botnet*, *Brute Force*, *DDoS*, *Port Scan*, and *Infiltration*—utilizing data from Kitsune and the CICIDS-2017 dataset.

Evaluation. To assess the effectiveness of the mimicked packets in evading the 6 classifiers, we employed a singular, highly illustrative metric from the Traffic Manipulator: the Malicious traffic Evasion Rate (MER). This metric is calculated using the formula: $MER = 1 - \left(N^{\text{adv}}/N^{\text{mal}}\right)$, where N^{adv} and N^{mal} represent the number of detected adversarial and detected malicious packet sequences, respectively. This equation captures the percentage of adversarial packet sequences that goes undetected as compared to the original malicious packet sequences that was detected, effectively measuring the evasion capability of the adversarial packet sequences.

We employed the two-sample Kolmogorov-Smirnov (K-S) test, a powerful non-parametric method used to determine the probabilistic differences between two data samples. Specifically, we compare the empirical cumulative distribution functions (eCDFs) of both the original malicious packets and the adversarial packets generated by our model. The K-S test calculates the maximum distance (K-S statistic, *D*) between these two eCDFs. This statistic measures the greatest deviation between the distribution of our adversarially modified packets and the original malicious packets (Hore et al. [2023]; Gretton et al. [2012]). A *D* value exceeding the critical threshold at a chosen significance level suggests a rejection of the null hypothesis—that is, the two samples are not derive from the same distribution. We demonstrated adversarial packets that were found to be out-of-distribution (OOD) at 95% significance level.

Additionally, we used Wireshark, a widely recognized network protocol analyzer to ensure that during the modifications were made, the immutable fields crucial for the packet's attack functionality were preserved. This visual verification via Wireshark confirms that our adversarial packets maintain their structural integrity and functionality, successfully mimicking genuine network traffic while evading detection (Due to limited space, results and analysis are in Appendix B.1).

We also evaluated DiffuPac's efficacy in generating adversarial packets that maintain their malicious functionality in a controlled test environment using UTM hypervisor technology. This setup included two virtual machines: one running Kali Linux (attacker) and another running Ubuntu 24.04 (victim). The isolation ensured a realistic yet controlled network environment. We conducted the evaluation on two type of attacks: *Port Scan* and *Brute Force*. (Due to space constraints, results and analysis of the two attacks are in Appendix B.3.1 and B.3.2).

4.2 Results and Analysis

Evasion Rate. We evaluated the evasion rates of the 6 attacks using DiffuPac and two baseline models, and compared the results in Table 2 (Due to space constraints, the other 4 attacks are demonstrated in Appendix B.2). As shown in Table 2, we conclude that DiffuPac is capable of generating adversarial packets that achieves comparable or even higher in evasion rate compared with both Traffic Manipulator (GAN & PSO) and TANTRA (LSTM). Importantly, DiffuPac achieves this high performance without relying on NIDS components, surrogate classifiers or specific insights into the NIDS's feature extraction methods, highlighting its effectiveness in realistic settings where attackers lack access to such insider information. The evasion performance varied significantly across different classifiers, reflecting the inherent variability in their robustness and detection capabilities. Flow-based NIDS exhibited greater resilience against our attacks compared to packet-based NIDS, likely due to DiffuPac's focus on packet-level modifications. Interestingly, the Traffic Manipulator model performed well with KitNet's AfterImage feature extractor, likely because it was trained directly on these features, enhancing its capability to modify them to evade detection. This specific alignment with NIDS features, while effective, does not typically reflect real-world attacker capabilities or constraints.

Table 3 : Percentage of successful adversarial samples found to be OOD.

Attack Type	**Percentage of OOD Samples (%)**
Botnet	48.72
MITM	29.34
Port Scan	55.23
DDoS	78.14
Infiltration	58.06
Brute Force	69.67

TANTRA also demonstrated high evasion rates, particularly with AfterImage, thanks to its use of LSTM models trained on benign traffic from the targeted network. This training allows TANTRA to reshape the timing between packets. This specific adjustment of interpacket delays directly impacts the time-based features extracted by AfterImage, making it more difficult for the classifiers to distinguish between normal and malicious packets.

Quantitatively, DiffuPac outperformed Traffic Manipulator by an average of 9.12 percentage points and TANTRA by an average of 4.26 percentage points across all attack types. The overall average improvement of DiffuPac over both baselines is 6.69 percentage points, demonstrating its superior capability to generate adversarial packets that effectively evade detection.

Based on the results of all the attacks, we can conclude that ML classifier are more robust than DNNs to adversarial attacks. Traditional ML models, like DT and IF, have simpler and more interpretable decision boundaries, making them less susceptible to subtle adversarial modifications (Sauka et al. [2022]). In contrast, DNNs with their complex and high-dimensional decision boundaries are more easily misled by the nuanced modifications introduced by DiffuPac.

Statistical Difference (K-S test). The percentage of successful adversarial packets that were found to be OOD are demonstrated in Table 3. From the result, we can obeserve that Brute Force and DDoS attacks show the highest percentage of OOD samples. The grounded reasons for these kind of attacks are due to the inherent nature of their traffic patterns. The nature of Brute Force attacks involves numerous failed login attempts, leading to highly irregular sequence numbers (Javed and Paxson [2013]). Even after modifications by DiffuPac, these attempts will stand out due to their frequency and pattern, resulting in significant deviations in the eCDFs. The sheer volume and repetitive nature of DDoS traffic (Haseeb-ur rehman et al. [2023]) make it difficult to disguise effectively. Despite efforts of intelligent mutation process, the high volume and distinctive traffic patterns will cause substantial deviations in the eCDFs as well. On the other hand, the MITM show the lowest percentage of OOD samples. Adjusting TTL values is straightforward as they can be set to typical ranges seen in normal packets without significantly altering packet behavior, resulting in minimal deviation in the eCDFs.

5 Limitations

Our evaluations demonstrated DiffuPac's strong evasion capabilities, enabling the generation of adversarial packets that convincingly mimic legitimate network traffic, particularly in packet header fields. While these results highlight DiffuPac's ability to evade detection by NIDS, there remain areas for further study.

In particular, while we assessed the malicious functionality of generated adversarial packets in a controlled environment using UTM hypervisor technology, this evaluation was limited to *Port Scan* and *Brute Force* attacks. Although DiffuPac successfully retained the malicious intent of these attacks, further evaluations are required to determine its effectiveness across a broader range of attacks, especially those with more complex behaviors such as DDoS or Botnet attacks.

We also noted that DiffuPac's evasion performance is not uniform across all scenarios, notably DDoS and Brute Force. This variability is due to the distinct nature of each attack's traffic patterns, which may not be fully captured by the model, especially in highly repetitive or anomalous behaviors. To enhance the model's adaptability and generalizability, we plan to enrich the training dataset with a broader spectrum of real-world attack scenarios.

Moreover, the dual-use nature of adversarial generation models like DiffuPac presents significant ethical and legal challenges. While designed to improve security defenses, these technologies could be misused for malicious activities, such as facilitating cyber-attacks or disrupting services. To mitigate these risks, we have opted not to publicly release saved model checkpoints, aiming to prevent exploitation by malicious entities and ensure that our advancements in adversarial packet generation are used responsibly and within ethical bounds.

6 Conclusion and Future Directions

In this study, we introduced DiffuPac, an intelligent generative model that successfully generates adversarial packets capable of evading advanced NIDS while maintaining attack functionality. Unlike previous research, which often assumes attacker access to NIDS, DiffuPac operates under constraints of limited attacker knowledge, reflecting more realistic scenarios. Here, we present the in-depths insights obtained from this study: (a) DiffuPac uniquely combines normal and malicious packet sequences using contextual alignments, ensuring seamless integration into genuine traffic while employing these normal packet sequences to guide the denoising process. Through the performance evaluations with various NIDS, our model achieved an average improvement of approximately *6.69* percentage points in evasion rate compared to the two baselines across all attack types. (b) Evasion rates varied notably across attack types, with DDoS and Brute Force attacks showing higher probabilistic differences due to their complex, repetitive nature. This highlights potential areas for further refinement in DiffuPac's approach to handling voluminous attacks. (c) Simpler ML models like DT and IF displayed surprising resilience due to their less complex decision boundaries, limiting the effectiveness of DiffuPac's modifications. In contrast, DNNs, with their intricate decision boundaries, were more vulnerable, underscoring the complexities inherent in designing robust adversarial tactics. (d) DiffuPac's capability to balance sophisticated attacks with operational stealth makes it especially suitable for environments where attackers lack comprehensive NIDS configurations, enhancing its utility in realistic defense testing.

As future studies, we will compare DiffuPac against a broader generative adversarial packets model and extend testing across more diverse NIDS to better understand its relative strengths and limitations. Next, we will further expand the range of attack types included in the malicious functionality evaluation. Moreover, adversarial defenses are a key focus of our ongoing research, particularly in relation to DiffuPac's capabilities. One promising direction is the development of adversarial purification using diffusion models, a technique successfully applied in image processing. This approach treats adversarial perturbations as noise, utilizing the diffusion model's reverse process to restore network packets to their original state before analysis by NIDS. We believe this method has significant potential for enhancing network security and aim to further develop and test it as a novel defense strategy.

Acknowledgements

This work has been partially supported by KAKENHI Grant Number JP23H03379.

References

Statista. Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2023, with forecasts from 2022 to 2030. https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/, 2023. Accessed: 2024-4-20.

Tala Talaei Khoei and Naima Kaabouch. Machine learning: Models, challenges, and research directions. *Future Internet*, 15(10):332, 2023. doi: 10.3390/fi15100332.

Tala Talaei Khoei, Hadjar Ould Slimane, and Naima Kaabouch. Deep learning: Systematic review, models, challenges, and research directions. *Neural Computing and Applications*, 35(31):23103–23124, 2023. doi: 10.1007/s00521-023-08957-4.

Olakunle Ibitoye, Rana Abou Khamis, Ashraf Matrawy, and M. Omair Shafiq. The threat of adversarial attacks on machine learning in network security – A survey. *arXiv*, 2019. URL http://arxiv.org/abs/1911.02621.

Hassan Khazane, Mohammed Ridouani, Fatima Salahdine, and Naima Kaabouch. A holistic review of machine learning adversarial attacks in IoT networks. *Future Internet*, 16(1):32, 2024. doi: 10.3390/fi16010032.

Ivan Homoliak, Martin Teknøs, Martîn Ochoa, Dominik Breitenbacher, Saeid Hosseini, and Petr Hanacek. Improving network intrusion detection classifiers by non-payload-based exploit-independent obfuscations: An adversarial approach. *EAI Endorsed Transactions on Security and Safety*, 5(17), 12 2018. doi: 10.4108/eai.10-1-2019.156245.

Mohammad J. Hashemi, Greg Cusack, and Eric Keller. Towards evaluation of nidss in adversarial setting. In *Proceedings of the 3rd ACM CoNEXT Workshop on Big DAta, Machine Learning and Artificial Intelligence for Data Communication Networks*, 2019.

Aditya Kuppa, Slawomir Grzonkowski, Muhammad Rizwan Asghar, and Nhien-An Le-Khac. Black box attacks on deep anomaly detectors. In *Proceedings of the 14th International Conference on Availability, Reliability and Security (ARES 2019)*, 2019.

Dongqi Han, Zhiliang Wang, Ying Zhong, Wenqi Chen, Jiahai Yang, Shuqiang Lu, Xingang Shi, and Xia Yin. Evaluating and improving adversarial robustness of machine learning-based network intrusion detectors. *IEEE Journal on Selected Areas in Communications*, 39(8):2632–2647, 2021. doi: 10.1109/JSAC.2021.3087242.

Yam Sharon, David Berend, Yang Liu, Asaf Shabtai, and Yuval Elovici. TANTRA: Timing-based adversarial network traffic reshaping attack. *IEEE Transactions on Information Forensics and Security*, 17:3225–3237, 2021. doi: 10.1109/TIFS.2022.3201377.

Soumyadeep Hore, Jalal Ghadermazi, Diwas Paudel, Ankit Shah, Tapas K. Das, and Nathaniel D. Bastian. Deep PackGen: A deep reinforcement learning framework for adversarial network packet generation. *arXiv*, 2023. URL https://arxiv.org/abs/2305.11039.

Zeeshan Ahmad, Adnan Shahid Khan, Cheah Wai Shiang, Johari Abdullah, and Farhan Ahmad. Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *Transactions on Emerging Telecommunications Technologies*, 32(1):e4150, 2021. doi: https://doi.org/10.1002/ett.4150.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In Yoshua Bengio and Yann LeCun, editors, *Proceedings of the 2nd International Conference on Learning Representations (ICLR 2014)*, 2014.

Andrew McCarthy, Essam Ghadafi, Panagiotis Andriotis, and Phil Legg. Functionality-preserving adversarial machine learning for robust classification in cybersecurity and intrusion detection domains: A survey. *Journal of Cybersecurity and Privacy*, 2(1):154–190, 2022. doi: 10.3390/jcp2010010.

Kaichen Yang, Jianqing Liu, Chi Zhang, and Yuguang Fang. Adversarial examples against the deep learning based network intrusion detection systems. In *Proceedings of 2018 IEEE Military Communications Conference (MILCOM 2018)*, pages 559–564, 2018. doi: 10.1109/MILCOM.2018.8599759.

Ryan Sheatsley, Nicolas Papernot, Michael J. Weisman, Gunjan Verma, and Patrick McDaniel. Adversarial examples for network intrusion detection systems. *Journal of Computer Security*, 30 (5):727–752, 2022. doi: 10.3233/JCS-210094.

Ke He, Dan Dongseong Kim, and Muhammad Rizwan Asghar. Adversarial machine learning for network intrusion detection systems: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 25(1):538–566, 2023. doi: 10.1109/COMST.2022.3233793.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2256–2265, Lille, France, 07–09 Jul 2015. PMLR.

Jonathan Ho, Ajay Jain, and P. Abbeel. Denoising diffusion probabilistic models. In *Proceedings of Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, pages 6840–6851, 2020.

Xinjie Lin, Gang Xiong, Gaopeng Gou, Zhen Li, Junzheng Shi, and Jing Yu. Et-bert: A contextualized datagram representation with pre-training transformers for encrypted traffic classification. In *Proceedings of the ACM Web Conference 2022 (WWW 2022)*. ACM, 2022. doi: 10.1145/3485447.3512217.

Xuying Meng, Chungang Lin, Yequan Wang, and Yujun Zhang. Netgpt: Generative pretrained transformer for network traffic. *arXiv*, 2023. URL https://arxiv.org/abs/2304.09513.

Yonghui Wu, Mike Schuster, Z. Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason R. Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Gregory S. Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv*, 2016. URL https://arxiv.org/abs/1609.08144.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 17th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2019)*, pages 4171–4186. ACL, 2019. doi: 10.18653/v1/N19-1423.

Chuanpu Fu, Qi Li, Meng Shen, and Ke Xu. Realtime robust malicious traffic detection via frequency domain analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS 2021)*, pages 3431–3446. ACM, 2021. doi: 10.1145/3460120.3484585.

Jianbang Dai, Xiaolong Xu, and Fu Xiao. Glads: A global-local attention data selection model for multimodal multitask encrypted traffic classification of IoT. *Computer Networks*, 225:109652, 2023. doi: https://doi.org/10.1016/j.comnet.2023.109652.

Xiang Li, John Thickstun, Ishaan Gulrajani, Percy S Liang, and Tatsunori B Hashimoto. Diffusion-LM improves controllable text generation. In *Proceedings of Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*, pages 4328–4343, 2022.

Shansan Gong, Mukai Li, Jiangtao Feng, Zhiyong Wu, and Lingpeng Kong. DiffuSeq: Sequence to sequence text generation with diffusion models. In *Proceedings of the 11th International Conference on Learning Representations (ICLR 2023)*, 2023.

Alistair Letcher. On the impossibility of global convergence in multi-loss optimization. *arXiv*, 2021. URL https://arxiv.org/abs/2005.12649.

Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: An ensemble of autoencoders for online network intrusion detection. In *Proceedings 2018 Network and Distributed System Security Symposium (NDSS 2018)*, 2018.

Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP 2018)*, pages 108–116. INSTICC, SciTePress, 2018. doi: 10.5220/0006639801080116.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. *arXiv*, 2024. URL https://arxiv.org/abs/2401.08281.

Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. Characterization of encrypted and vpn traffic using time-related features. In *International Conference on Information Systems Security and Privacy (ICISSP 2016)*, 2016. doi: 10.5220/0005740704070414.

Gints Engelen, Vera Rimmer, and Wouter Joosen. Troubleshooting an intrusion detection dataset: the cicids2017 case study. In *2021 IEEE Security and Privacy Workshops (SPW)*, pages 7–12, 2021. doi: 10.1109/SPW53761.2021.00009.

Lisa Liu, Gints Engelen, Timothy Lynar, Daryl Essam, and Wouter Joosen. Error prevalence in nids datasets: A case study on cic-ids-2017 and cse-cic-ids-2018. In *2022 IEEE Conference on Communications and Network Security (CNS)*, pages 254–262, 2022. doi: 10.1109/CNS56114.2022.9947235.

Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13:723–773, 2012.

Kudzai Sauka, Gun-Yoo Shin, Dong-Wook Kim, and Myung-Mook Han. Adversarial robust and explainable network intrusion detection systems based on deep learning. *Applied Sciences*, 12(13), 2022. doi: 10.3390/app12136451.

Mobin Javed and Vern Paxson. Detecting stealthy, distributed SSH brute-forcing. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security (CCS 2013)*, pages 85–96, New York, NY, USA, 2013. ACM. doi: 10.1145/2508859.2516719.

Rana M. Abdul Haseeb-ur rehman, Azana Hafizah Mohd Aman, Mohammad Kamrul Hasan, Khairul Akram Zainol Ariffin, Abdallah Namoun, Ali Tufail, and Ki-Hyung Kim. High-speed network ddos attack detection: A survey. *Sensors*, 23(15), 2023. doi: 10.3390/s23156850.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of Advances in Neural Information Processing Systems 30 (NIPS 2017)*, pages 6000–6010, 2017.

Zhe Zhao, Hui Chen, Jinbin Zhang, Xin Zhao, Tao Liu, Wei Lu, Xi Chen, Haotang Deng, Qi Ju, and Xiaoyong Du. Uer: An open-source toolkit for pre-training models. *EMNLP-IJCNLP 2019*, page 241, 2019.

A Model Architecture

A.1 BERT

The core of DiffuPac is the BERT model, specifically adapted to serve as the denoising component in the diffusion process. The architecture consists of 12 bidirectional Transformer blocks and each block incorporates 12 attention heads within its self-attention layers (Vaswani et al. [2017]). These layers are crucial for capturing the intricate and implicit relationships between traffic bytes. Each input token has a dimension of 768, and the model can process sequences with up to 512 tokens. This setup allows the BERT model to effectively handle extensive sequences of traffic data, making it well-suited for the task of denoising during the diffusion process.

To implement of our pre-trained BERT model, we utilized the codebase from the UER (Universal Encoder Representations)-py project (Zhao et al. [2019]). UER-py is a versatile toolkit designed for pre-training and fine-tuning various NLP models, including BERT. In UER-py, the MLM is designed to predict masked words within a sentence. We adapted this approach to our masked unidirectional flow model, which focuses on predicting masked tokens within network traffic sequences. Instead of textual tokens, our model deals with traffic units from src-to-dst and dst-to-src sequences. We modified the MLM implementation to handle these traffic tokens, ensuring that each token is masked with a probability of 15%. Our model then predicts the original token from the masked sequence, capturing semantic patterns within bidirectional network flows.

The Next Sentence Prediction (NSP) task in UER-py involves prediction step to determine whether one sentence follows another. We transformed this concept into the same sequence-origin prediction task to determine the origin of directional network traffic. We adapted the binary classifier used in NSP to classify whether packets are originated from src-to-dst or dst-to-src. By pairing packets and predicting the directional flow, our model learns the contextual relationships and patterns in network traffic, similar to how NSP learns sentence relationships.

Table 4 lists the parameters are set to run our pre-trained BERT model. The remaining parameters were set to their default values.

We also conducted a comprehensive evaluation of different BERT model configurations to identify the most efficient option that still maintains high performance. We tested four BERT model variants: Tiny, Small, Medium, and Large, comparing their specifications, training time, loss, and accuracy. The results are summarized in Table 5. The Medium BERT model was chosen as the optimal configuration for DiffuPac because it offers a balanced trade-off between accuracy and loss metrics.

Table 4: Parameters used for pre-training the BERT model.

Parameter	Value
Embedding Size	768
Feedforward Size	3072
Hidden Size	768
Hidden Activation Function	GELU
Number of Attention Heads	12
Number of Transformer Layers	12
Max Sequence Length	512
Dropout Rate	0.1
Batch Size	64
Total Steps	500000
Learning Rate	1e-4

A.2 Fine-tuning with Diffusion Model

For the fine-tuning phase of our model, we leveraged the DiffuSeq codebase (DiffuSeq's code). The DiffuSeq framework, originally designed for text data, required several significant adaptations to handle network traffic data and to meet the objectives of our diffusion model for adversarial packet generation. The primary adaptation to the DiffuSeq codebase involves the data preparation phase,

Table 5: Comparison of BERT Models.

Parameters / Metrics	Tiny BERT	Small BERT	Medium BERT	Large BERT
Model Specifications				
Embedding Size	128	512	768	1024
Feedforward Size	512	2048	3072	4096
Hidden Size	128	512	768	1024
Activation Function	GELU	GELU	GELU	GELU
Number of Heads	2	8	12	16
Number of Layers	2	4	12	24
Training				
Training Time (hours)	13.5	18	23	30.5
Performance				
MUM Loss	1.028	0.822	0.667	0.610
SSP Loss	0.295	0.204	0.099	0.061
MUM Accuracy	0.844	0.887	0.905	0.914
SSP Accuracy	0.846	0.902	0.972	0.982

specifically the concatenation strategy used to blend normal and malicious packet sequences. This adaptation is crucial for the model to learn how to generate adversarial packets that can mimic normal packets and evade detection by NIDS.

In our concatenation strategy, we identified pairs of normal and malicious packet sequences that exhibit strong contextual alignments unlike DiffuSeq, which, randomly merge the text sequences. This means that the patterns and behaviors within these sequences are similar, ensuring that the concatenated sequences are blended seamlessly. We concatenated these contextually aligned normal and malicious packet sequences to form a new sequence, $\mathbf{S}^{\mathrm{ben}\oplus\mathrm{mal}}$. The strategy leverages the inherent patterns and behaviors within normal packet sequences, which is crucial for the mimicking process of malicious packet sequences.

We utilized the pre-trained BERT embeddings in our concatenation strategy. BERT is designed to capture deep semantic relationships within sequences. This design makes BERT making it highly effective for analyzing complex patterns in network traffic. By employing the pre-trained embeddings, DiffuPac can accurately transform discrete packet data into continuous feature spaces, capturing the intricate dependencies between packets. This transformation is critical for the subsequent similarity analysis and pairing process. The pre-trained embeddings offer a rich representation of the packet sequences, so that the model can discern subtle contextual similarities between normal and malicious packet sequences. The detailed of the concatenation strategy is demonstrated in Algorithm 1.

Here, we set the similarity threshold, ϵ to 0.9. This is a strategic decision aimed at balancing the need for high contextual alignment with the practicalities of ensuring a sufficient number of matched pairs. Setting ϵ to 0.9 indicates that we are looking for pairs of packet sequences that are highly similar, capturing nearly all the essential contextual information shared between normal and malicious packet sequences. This high threshold ensures that the concatenated sequences are blended seamlessly, maintaining the realistic traffic patterns and behaviors required to evade detection by NIDS. From a practical standpoint, a threshold of 0.9 is chosen because it strikes an optimal balance between precision and recall. A higher threshold (closer to 1.0) would result in fewer matched pairs, as only the most similar sequences would be selected. This could limit the model's ability to generate a diverse set of adversarial packets. Conversely, a lower threshold would increase the number of matched pairs but might include sequences that are not contextually well-aligned, reducing the effectiveness of the adversarial packets in mimicking normal traffic.

During the reverse process, we introduced a guidance mechanism using normal packet sequences. This mechanism treats the concatenated sequences of normal and noise-added malicious packets as a unified entities, allowing the model to denoise the malicious packets in the context of the normal packet sequences. We recalibrated the variational lower bound L_{VLB} to emphasize the integration of normal sequence guidance. The objective focuses on accurately reconstructing the initial state, $\mathbf{z}_0$ from the noised states, so that the malicious components are seamlessly integrated into the normal packet sequences pattern.

Table 6 lists the parameters used to run our diffusion model. The other parameters are set according to the default values of DiffuSeq. All training, experimentation, and sampling processes are executed on a single NVIDIA AD102 (GeForce RTX 4090) GPU. The training process for BERT required approximately 23 hours, while fine-tuning with the diffusion model took roughly 10 hours.

Table 6: Parameters used for the diffusion model.

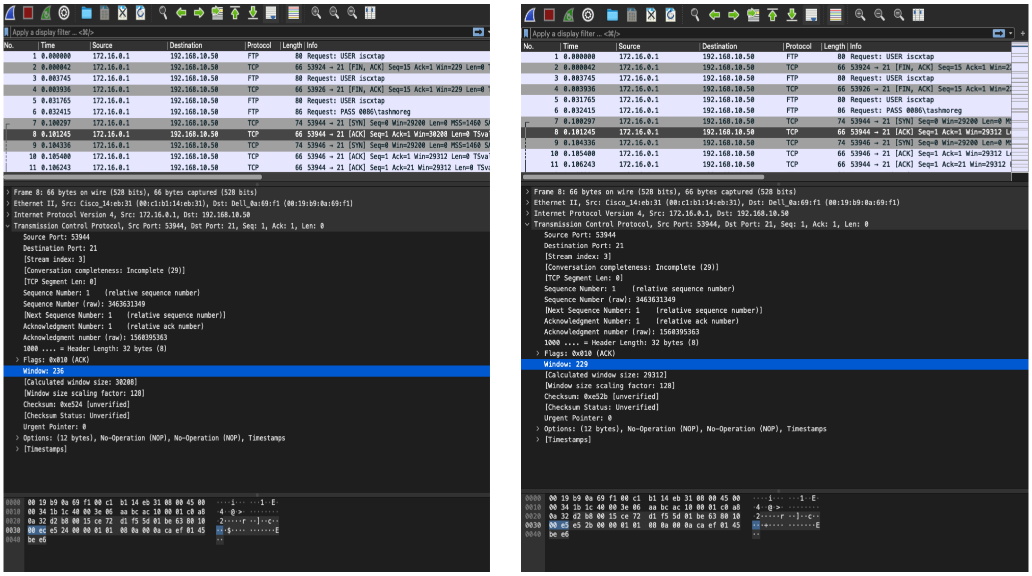
Parameter	**Value**
Diffusion Steps	2000
Learning Rate	1e-4
Learning Steps	50000
Seed	102
Noise Schedule	sqrt
Batch Size	64
Microbatch	64
Sequence Length	128
Hidden Time Dimension	128
Hidden Dimension	128
Schedule Sampler	uniform

Algorithm 1 Finding contextually relevant packet sequences.

```
Require: Normal packet sequences $\mathbf{S}^{\mathrm{ben}}$, Malicious packet sequences $\mathbf{S}^{\mathrm{mal}}$, Model parameters $\theta$,
    Similarity threshold $\epsilon$
Ensure: Matched pairs of normal and malicious packet sequences $(\mathbf{S}_i^{\mathrm{ben}}, \mathbf{S}_j^{\mathrm{mal}})$
Initialize: Load pre-trained BERT model with parameters $\theta$
Embed Normal Sequences:
for each sequence $\mathbf{S}_i^{\mathrm{ben}}$ in $\mathbf{S}^{\mathrm{ben}}$ do
    Compute embedding $\mathbf{E}_i^{\mathrm{ben}} \leftarrow \mathrm{EMB}(\mathbf{S}_i^{\mathrm{ben}}; \theta)$
end for
Embed Malicious Sequences:
for each sequence $\mathbf{S}_j^{\mathrm{mal}}$ in $\mathbf{S}^{\mathrm{mal}}$ do
    Compute embedding $\mathbf{E}_j^{\mathrm{mal}} \leftarrow \mathrm{EMB}(\mathbf{S}_j^{\mathrm{mal}}; \theta)$
end for
Find Contextually Aligned Pairs:
for each embedding $\mathbf{E}_j^{\mathrm{mal}}$ in $\mathbf{E}^{\mathrm{mal}}$ do
    Initialize best match $b_j \leftarrow$ None
    Initialize highest similarity $\sigma_{\text{max}} \leftarrow -1$
    for each embedding $\mathbf{E}_i^{\mathrm{ben}}$ in $\mathbf{E}^{\mathrm{ben}}$ do
        Compute similarity $\sigma_{ij} \leftarrow$ cosine_similarity$(\mathbf{E}_j^{\mathrm{mal}}, \mathbf{E}_i^{\mathrm{ben}})$
        if $\sigma_{ij} > \sigma_{\text{max}}$ then
            Update best match $b_j \leftarrow \mathbf{S}_i^{\mathrm{ben}}$
            Update highest similarity $\sigma_{\text{max}} \leftarrow \sigma_{ij}$
        end if
    end for
    if $\sigma_{\text{max}} > \epsilon$ then
        Add pair $(b_j, \mathbf{S}_j^{\mathrm{mal}})$ to matched pairs
    end if
end for
Return Matched Pairs: Return all matched pairs $(\mathbf{S}_i^{\mathrm{ben}}, \mathbf{S}_j^{\mathrm{mal}})$
```

B Experimental Results

B.1 Wireshark Results



```
Apply a display filter ... <⌘/>
No.   Time      Source      Destination    Protocol Length Info
    1 0.000000  172.16.0.1  192.168.10.50  FTP      80 Request: USER iscxtap
    2 0.000042  172.16.0.1  192.168.10.50  TCP      66 53924 → 21 [FIN, ACK] Seq=15 Ack=1 Win=229 Len=0 T
    3 0.003745  172.16.0.1  192.168.10.50  FTP      80 Request: USER iscxtap
    4 0.003936  172.16.0.1  192.168.10.50  TCP      66 53926 → 21 [FIN, ACK] Seq=15 Ack=1 Win=229 Len=0 T
    5 0.031765  172.16.0.1  192.168.10.50  FTP      80 Request: USER iscxtap
    6 0.032415  172.16.0.1  192.168.10.50  FTP      86 Request: PASS 0086\tashmoreg
    7 0.100297  172.16.0.1  192.168.10.50  TCP      74 53944 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SA
    8 0.101245  172.16.0.1  192.168.10.50  TCP      66 53944 → 21 [ACK] Seq=1 Ack=1 Win=30208 Len=0 TSval
    9 0.104336  172.16.0.1  192.168.10.50  TCP      74 53946 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SA
   10 0.105400  172.16.0.1  192.168.10.50  TCP      66 53946 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval
   11 0.106243  172.16.0.1  192.168.10.50  TCP      66 53944 → 21 [ACK] Seq=1 Ack=21 Win=29312 Len=0 TSva

> Frame 8: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
> Ethernet II, Src: Cisco_14:eb:31 (00:c1:b1:14:eb:31), Dst: Dell_0a:69:f1 (00:19:b9:0a:69:f1)
> Internet Protocol Version 4, Src: 172.16.0.1, Dst: 192.168.10.50
v Transmission Control Protocol, Src Port: 53944, Dst Port: 21, Seq: 1, Ack: 1, Len: 0
    Source Port: 53944
    Destination Port: 21
    [Stream index: 3]
    [Conversation completeness: Incomplete (29)]
    [TCP Segment Len: 0]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 3463631349
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 1560395363
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x010 (ACK)
    Window: 236
    [Calculated window size: 30208]
    [Window size scaling factor: 128]
    Checksum: 0xe524 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  > [Timestamps]

0000  00 19 b9 0a 69 f1 00 c1  b1 14 eb 31 08 00 45 00   ····i··· ···1··E·
0010  00 34 1b 1c 40 00 3e 06  aa bc ac 10 00 01 c0 a8   ·4··@·>· ········
0020  0a 32 d2 b8 00 15 ce 72  d1 f5 5d 01 be 63 80 10   ·2·····r ··]··c··
0030  00 ec e5 24 00 00 01 01  08 0a 00 0a ca ef 01 45   ···$···· ·······E
0040  be e6                                              ··
```

(a) Botnet attack before in Wireshark.

```
Apply a display filter ... <⌘/>
No.   Time      Source      Destination    Protocol Length Info
    1 0.000000  172.16.0.1  192.168.10.50  FTP      80 Request: USER iscxtap
    2 0.000042  172.16.0.1  192.168.10.50  TCP      66 53924 → 21 [FIN, ACK] Seq=15 Ack=1 Win=2
    3 0.003745  172.16.0.1  192.168.10.50  FTP      80 Request: USER iscxtap
    4 0.003936  172.16.0.1  192.168.10.50  TCP      66 53926 → 21 [FIN, ACK] Seq=15 Ack=1 Win=2
    5 0.031765  172.16.0.1  192.168.10.50  FTP      80 Request: USER iscxtap
    6 0.032415  172.16.0.1  192.168.10.50  FTP      86 Request: PASS 0086\tashmoreg
    7 0.100297  172.16.0.1  192.168.10.50  TCP      74 53944 → 21 [SYN] Seq=0 Win=29200 Len=0 MS
    8 0.101245  172.16.0.1  192.168.10.50  TCP      66 53944 → 21 [ACK] Seq=1 Ack=1 Win=29312 Le
    9 0.104336  172.16.0.1  192.168.10.50  TCP      74 53946 → 21 [SYN] Seq=0 Win=29200 Len=0 MS
   10 0.105400  172.16.0.1  192.168.10.50  TCP      66 53946 → 21 [ACK] Seq=1 Ack=1 Win=29312 Le
   11 0.106243  172.16.0.1  192.168.10.50  TCP      66 53944 → 21 [ACK] Seq=1 Ack=21 Win=29312 L

> Frame 8: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
> Ethernet II, Src: Cisco_14:eb:31 (00:c1:b1:14:eb:31), Dst: Dell_0a:69:f1 (00:19:b9:0a:69:f1)
> Internet Protocol Version 4, Src: 172.16.0.1, Dst: 192.168.10.50
v Transmission Control Protocol, Src Port: 53944, Dst Port: 21, Seq: 1, Ack: 1, Len: 0
    Source Port: 53944
    Destination Port: 21
    [Stream index: 3]
    [Conversation completeness: Incomplete (29)]
    [TCP Segment Len: 0]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 3463631349
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 1560395363
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x010 (ACK)
    Window: 229
    [Calculated window size: 29312]
    [Window size scaling factor: 128]
    Checksum: 0xe52b [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  > [Timestamps]

0000  00 19 b9 0a 69 f1 00 c1  b1 14 eb 31 08 00 45 00   ····i··· ···1··E·
0010  00 34 1b 1c 40 00 3e 06  aa bc ac 10 00 01 c0 a8   ·4··@·>· ········
0020  0a 32 d2 b8 00 15 ce 72  d1 f5 5d 01 be 63 80 10   ·2·····r ··]··c··
0030  00 e5 e5 2b 00 00 01 01  08 0a 00 0a ca ef 01 45   ···+···· ·······E
0040  be e6                                              ··
```

(b) Botnet attack after in Wireshark.

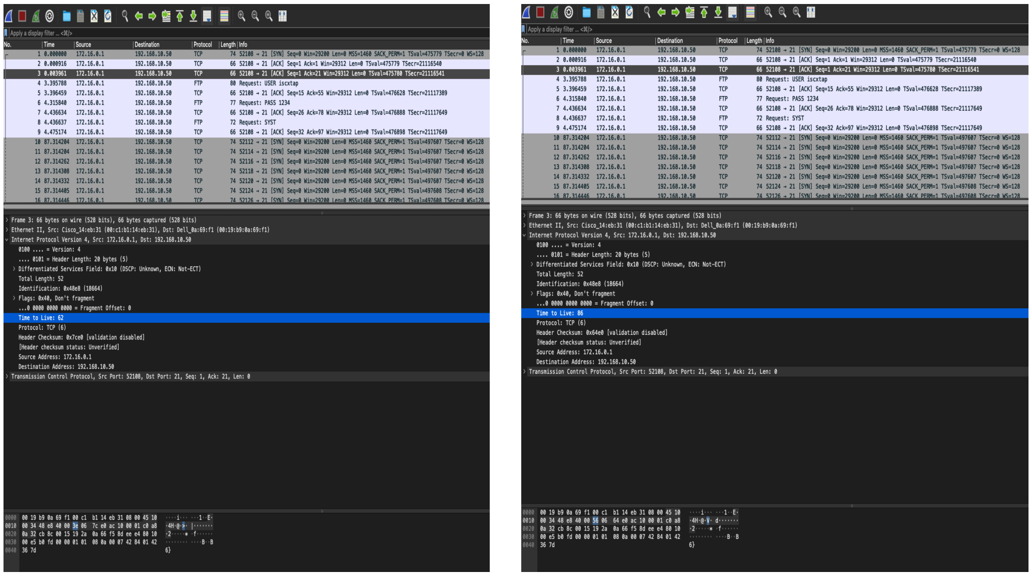
Figure 2: Comparison of Botnet attack before and after in Wireshark.

To demonstrate that our model intelligently alters specific fields to seamlessly blend malicious packets into normal traffic, we analyzed the packets using Wireshark. Our analysis covered 6 attack types: *MITM*, *Botnet*, *Brute Force*, *DDoS*, *Port Scan*, and *Infiltration*. The results showed that different specific fields were modified in the packets corresponding to each type of attack. This variation in field alteration confirms that our model adapts its modifications according to the nature of the attack, enhancing the stealthiness of the malicious packets. In Figure 2, we can see that there is a change in the window size of a Botnet packet. Botnet attack can be described as a network of infected devices (bots) controlled by an attacker to perform various malicious activities. Botnet traffic often shows abnormal patterns in window size due to automated control and data bursts. Here, we can conclude that DiffuPac made the intelligent decision by itself in modifying the window size to fall within typical user traffic patterns. This helps to camouflage the botnet activity.

MITM, on the other hand, demonstrated attacks whereby an attacker intercepts communication between two parties without their knowledge, often to eavesdrop or alter the data being sent. Abnormal TTL values (time-to-live field) can reveal this kind of interception and redirection. Here, once again, our model intelligently adjusts TTL value to match the expected values in aiding the masking of MITM presence. As shown in Figure 3, DiffuPac adjusts the TTL values to effectively conceal signs of interception. This ensures that the adversarial packets blend seamlessly with normal data exchanges, mimicking legitimate communication between parties.

DDoS attacks involve a large number of packets with similar characteristics, often reusing or predictably incrementing the IP Identification (ID) field. In Figure 4, it can be shown that DiffuPac addresses this by randomizing the IP IDs to prevent detection. When we analyzed the traffic using Wireshark, we observed varied and less predictable IP IDs in the modified packets, aligning with normal traffic patterns. This randomness helps avoid forming detectable patterns and evades detection systems that rely on identifying repetitive IP ID sequences. Thus, DiffuPac successfully masks the attack's presence while preserving the attack functionality.

Port Scan often involve specific flags, such as SYN, to probe for open ports. This behavior is distinct from normal traffic, which uses a variety of flags. In Figure 5, we found that DiffuPac diversifies the use of flags to include SYN, ACK, and FIN, similar to normal connections. By normalizing the traffic in this way, DiffuPac helps disguise scanning activity. For example, instead of sending many SYN



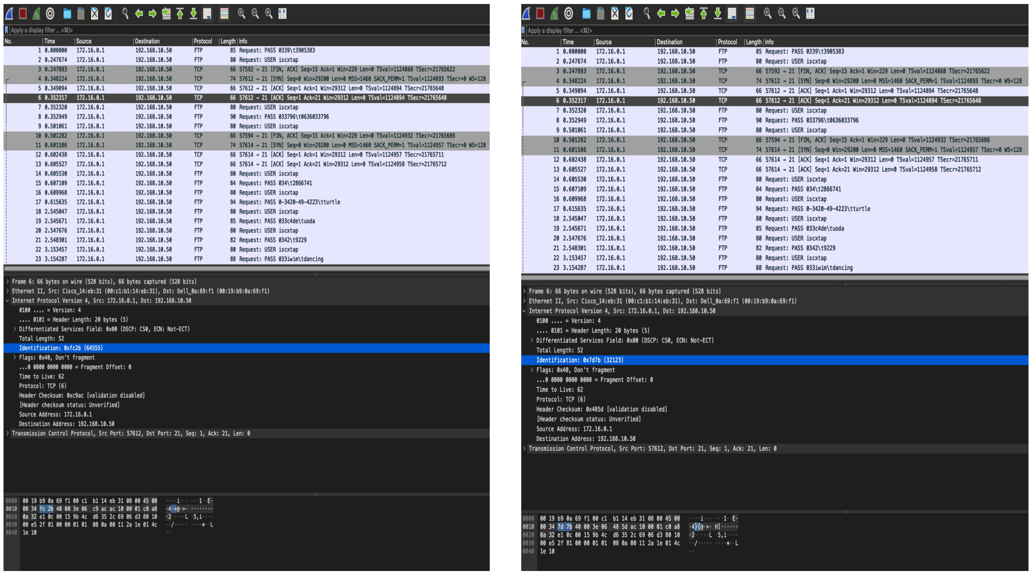
```
Apply a display filter ... <⌘/>
No.  Time       Source      Destination    Protocol Length Info
   1 0.000000   172.16.0.1  192.168.10.50  TCP      74 52108 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=475779 TSecr=0 WS=128
   2 0.000916   172.16.0.1  192.168.10.50  TCP      66 52108 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=475779 TSecr=21116540
   3 0.003961   172.16.0.1  192.168.10.50  TCP      66 52108 → 21 [ACK] Seq=1 Ack=21 Win=29312 Len=0 TSval=475780 TSecr=21116541
   4 3.395788   172.16.0.1  192.168.10.50  FTP      80 Request: USER iscxtap
   5 3.396459   172.16.0.1  192.168.10.50  TCP      66 52108 → 21 [ACK] Seq=15 Ack=55 Win=29312 Len=0 TSval=476628 TSecr=21117389
   6 4.315840   172.16.0.1  192.168.10.50  FTP      77 Request: PASS 1234
   7 4.436634   172.16.0.1  192.168.10.50  TCP      66 52108 → 21 [ACK] Seq=26 Ack=78 Win=29312 Len=0 TSval=476888 TSecr=21117649
   8 4.436637   172.16.0.1  192.168.10.50  FTP      72 Request: SYST
   9 4.475174   172.16.0.1  192.168.10.50  TCP      66 52108 → 21 [ACK] Seq=32 Ack=97 Win=29312 Len=0 TSval=476898 TSecr=21117649
  10 87.314204  172.16.0.1  192.168.10.50  TCP      74 52112 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=497607 TSecr=0 WS=128
  11 87.314204  172.16.0.1  192.168.10.50  TCP      74 52114 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=497607 TSecr=0 WS=128
  12 87.314262  172.16.0.1  192.168.10.50  TCP      74 52116 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=497607 TSecr=0 WS=128
  13 87.314308  172.16.0.1  192.168.10.50  TCP      74 52118 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=497607 TSecr=0 WS=128
  14 87.314332  172.16.0.1  192.168.10.50  TCP      74 52120 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=497607 TSecr=0 WS=128
  15 87.314405  172.16.0.1  192.168.10.50  TCP      74 52124 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=497608 TSecr=0 WS=128
  16 87.314446  172.16.0.1  192.168.10.50  TCP      74 52126 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=497608 TSecr=0 WS=128

> Frame 3: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
> Ethernet II, Src: Cisco_14:eb:31 (00:c1:b1:14:eb:31), Dst: Dell_0a:69:f1 (00:19:b9:0a:69:f1)
v Internet Protocol Version 4, Src: 172.16.0.1, Dst: 192.168.10.50
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x10 (DSCP: Unknown, ECN: Not-ECT)
    Total Length: 52
    Identification: 0x48e8 (18664)
  > Flags: 0x40, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 62
    Protocol: TCP (6)
    Header Checksum: 0x7ce0 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.16.0.1
    Destination Address: 192.168.10.50
> Transmission Control Protocol, Src Port: 52108, Dst Port: 21, Seq: 1, Ack: 21, Len: 0

0000  00 19 b9 0a 69 f1 00 c1  b1 14 eb 31 08 00 45 10   ····i··· ···1··E·
0010  00 34 48 e8 40 00 3e 06  7c e0 ac 10 00 01 c0 a8   ·4H·@·>· |·······
0020  0a 32 cb 8c 00 15 19 2a  0a 66 f5 8d ee e4 80 10   ·2·····* ·f······
0030  00 e5 b0 fd 00 00 01 01  08 0a 00 07 42 84 01 42   ········ ····B··B
0040  36 7d                                              6}
```

(a) MITM attack before in Wireshark.

```
Apply a display filter ... <⌘/>
No.  Time       Source      Destination    Protocol Length Info
   1 0.000000   172.16.0.1  192.168.10.50  TCP      74 52108 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=475779 TSecr=0 WS=128
   2 0.000916   172.16.0.1  192.168.10.50  TCP      66 52108 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=475779 TSecr=21116540
   3 0.003961   172.16.0.1  192.168.10.50  TCP      66 52108 → 21 [ACK] Seq=1 Ack=21 Win=29312 Len=0 TSval=475780 TSecr=21116541
   4 3.395788   172.16.0.1  192.168.10.50  FTP      80 Request: USER iscxtap
   5 3.396459   172.16.0.1  192.168.10.50  TCP      66 52108 → 21 [ACK] Seq=15 Ack=55 Win=29312 Len=0 TSval=476628 TSecr=21117389
   6 4.315840   172.16.0.1  192.168.10.50  FTP      77 Request: PASS 1234
   7 4.436634   172.16.0.1  192.168.10.50  TCP      66 52108 → 21 [ACK] Seq=26 Ack=78 Win=29312 Len=0 TSval=476888 TSecr=21117649
   8 4.436637   172.16.0.1  192.168.10.50  FTP      72 Request: SYST
   9 4.475174   172.16.0.1  192.168.10.50  TCP      66 52108 → 21 [ACK] Seq=32 Ack=97 Win=29312 Len=0 TSval=476898 TSecr=21117649
  10 87.314204  172.16.0.1  192.168.10.50  TCP      74 52112 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=497607 TSecr=0 WS=128
  11 87.314204  172.16.0.1  192.168.10.50  TCP      74 52114 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=497607 TSecr=0 WS=128
  12 87.314262  172.16.0.1  192.168.10.50  TCP      74 52116 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=497607 TSecr=0 WS=128
  13 87.314308  172.16.0.1  192.168.10.50  TCP      74 52118 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=497607 TSecr=0 WS=128
  14 87.314332  172.16.0.1  192.168.10.50  TCP      74 52120 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=497607 TSecr=0 WS=128
  15 87.314405  172.16.0.1  192.168.10.50  TCP      74 52124 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=497608 TSecr=0 WS=128
  16 87.314446  172.16.0.1  192.168.10.50  TCP      74 52126 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=497608 TSecr=0 WS=128

> Frame 3: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
> Ethernet II, Src: Cisco_14:eb:31 (00:c1:b1:14:eb:31), Dst: Dell_0a:69:f1 (00:19:b9:0a:69:f1)
v Internet Protocol Version 4, Src: 172.16.0.1, Dst: 192.168.10.50
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x10 (DSCP: Unknown, ECN: Not-ECT)
    Total Length: 52
    Identification: 0x48e8 (18664)
  > Flags: 0x40, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 86
    Protocol: TCP (6)
    Header Checksum: 0x64e0 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.16.0.1
    Destination Address: 192.168.10.50
> Transmission Control Protocol, Src Port: 52108, Dst Port: 21, Seq: 1, Ack: 21, Len: 0

0000  00 19 b9 0a 69 f1 00 c1  b1 14 eb 31 08 00 45 10   ····i··· ···1··E·
0010  00 34 48 e8 40 00 56 06  64 e0 ac 10 00 01 c0 a8   ·4H·@·V· d·······
0020  0a 32 cb 8c 00 15 19 2a  0a 66 f5 8d ee e4 80 10   ·2·····* ·f······
0030  00 e5 b0 fd 00 00 01 01  08 0a 00 07 42 84 01 42   ········ ····B··B
0040  36 7d                                              6}
```

(b) MITM attack after in Wireshark.

Figure 3: Comparison of MITM attack before and after in Wireshark.



```
Apply a display filter ... <⌘/>
No.  Time      Source      Destination    Protocol Length Info
   1 0.000000  172.16.0.1  192.168.10.50  FTP      85 Request: PASS 0339\t3905383
   2 0.247674  172.16.0.1  192.168.10.50  FTP      80 Request: USER iscxtap
   3 0.247893  172.16.0.1  192.168.10.50  TCP      66 57592 → 21 [FIN, ACK] Seq=15 Ack=1 Win=229 Len=0 TSval=1124868 TSecr=21765622
   4 0.348224  172.16.0.1  192.168.10.50  TCP      74 57612 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1124893 TSecr=0 WS=128
   5 0.349094  172.16.0.1  192.168.10.50  TCP      66 57612 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1124894 TSecr=21765648
   6 0.352317  172.16.0.1  192.168.10.50  TCP      66 57612 → 21 [ACK] Seq=1 Ack=21 Win=29312 Len=0 TSval=1124894 TSecr=21765648
   7 0.352320  172.16.0.1  192.168.10.50  FTP      80 Request: USER iscxtap
   8 0.352949  172.16.0.1  192.168.10.50  FTP      90 Request: PASS 033796\t0636033796
   9 0.501061  172.16.0.1  192.168.10.50  FTP      80 Request: USER iscxtap
  10 0.501282  172.16.0.1  192.168.10.50  TCP      66 57594 → 21 [FIN, ACK] Seq=15 Ack=1 Win=229 Len=0 TSval=1124932 TSecr=21765686
  11 0.601586  172.16.0.1  192.168.10.50  TCP      74 57614 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1124957 TSecr=0 WS=128
  12 0.602438  172.16.0.1  192.168.10.50  TCP      66 57614 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1124957 TSecr=21765711
  13 0.605527  172.16.0.1  192.168.10.50  TCP      66 57614 → 21 [ACK] Seq=1 Ack=21 Win=29312 Len=0 TSval=1124958 TSecr=21765712
  14 0.605530  172.16.0.1  192.168.10.50  FTP      80 Request: USER iscxtap
  15 0.607109  172.16.0.1  192.168.10.50  FTP      84 Request: PASS 034\t2866741
  16 0.609968  172.16.0.1  192.168.10.50  FTP      80 Request: USER iscxtap
  17 0.615635  172.16.0.1  192.168.10.50  FTP      94 Request: PASS 0-3420-49-4223\tturtle
  18 2.545047  172.16.0.1  192.168.10.50  FTP      80 Request: USER iscxtap
  19 2.545671  172.16.0.1  192.168.10.50  FTP      85 Request: PASS 033c4de\tuoda
  20 2.547676  172.16.0.1  192.168.10.50  FTP      80 Request: USER iscxtap
  21 2.548301  172.16.0.1  192.168.10.50  FTP      82 Request: PASS 0342\t9229
  22 3.153457  172.16.0.1  192.168.10.50  FTP      80 Request: USER iscxtap
  23 3.154287  172.16.0.1  192.168.10.50  FTP      88 Request: PASS 033iwim\tdancing

> Frame 6: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
> Ethernet II, Src: Cisco_14:eb:31 (00:c1:b1:14:eb:31), Dst: Dell_0a:69:f1 (00:19:b9:0a:69:f1)
v Internet Protocol Version 4, Src: 172.16.0.1, Dst: 192.168.10.50
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 52
    Identification: 0xfc2b (64555)
  > Flags: 0x40, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 62
    Protocol: TCP (6)
    Header Checksum: 0xc9ac [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.16.0.1
    Destination Address: 192.168.10.50
> Transmission Control Protocol, Src Port: 57612, Dst Port: 21, Seq: 1, Ack: 21, Len: 0

0000  00 19 b9 0a 69 f1 00 c1  b1 14 eb 31 08 00 45 00   ····i··· ···1··E·
0010  00 34 fc 2b 40 00 3e 06  c9 ac ac 10 00 01 c0 a8   ·4·+@·>· ········
0020  0a 32 e1 0c 00 15 9b 4c  d6 35 2c 69 06 d3 80 10   ·2·····L ·5,i····
0030  00 e5 2f 81 00 00 01 01  08 0a 00 11 2a 1e 01 4c   ··/····· ····*··L
0040  1e 10                                              ··
```

(a) DDoS attack before in Wireshark.

```
Apply a display filter ... <⌘/>
No.  Time      Source      Destination    Protocol Length Info
   1 0.000000  172.16.0.1  192.168.10.50  FTP      85 Request: PASS 0339\t3905383
   2 0.247674  172.16.0.1  192.168.10.50  FTP      80 Request: USER iscxtap
   3 0.247893  172.16.0.1  192.168.10.50  TCP      66 57592 → 21 [FIN, ACK] Seq=15 Ack=1 Win=229 Len=0 TSval=1124868 TSecr=21765622
   4 0.348224  172.16.0.1  192.168.10.50  TCP      74 57612 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1124893 TSecr=0 WS=128
   5 0.349094  172.16.0.1  192.168.10.50  TCP      66 57612 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1124894 TSecr=21765648
   6 0.352317  172.16.0.1  192.168.10.50  TCP      66 57612 → 21 [ACK] Seq=1 Ack=21 Win=29312 Len=0 TSval=1124894 TSecr=21765648
   7 0.352320  172.16.0.1  192.168.10.50  FTP      80 Request: USER iscxtap
   8 0.352949  172.16.0.1  192.168.10.50  FTP      90 Request: PASS 033796\t0636033796
   9 0.501061  172.16.0.1  192.168.10.50  FTP      80 Request: USER iscxtap
  10 0.501282  172.16.0.1  192.168.10.50  TCP      66 57594 → 21 [FIN, ACK] Seq=15 Ack=1 Win=229 Len=0 TSval=1124932 TSecr=21765686
  11 0.601586  172.16.0.1  192.168.10.50  TCP      74 57614 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1124957 TSecr=0 WS=128
  12 0.602438  172.16.0.1  192.168.10.50  TCP      66 57614 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1124957 TSecr=21765711
  13 0.605527  172.16.0.1  192.168.10.50  TCP      66 57614 → 21 [ACK] Seq=1 Ack=21 Win=29312 Len=0 TSval=1124958 TSecr=21765712
  14 0.605530  172.16.0.1  192.168.10.50  FTP      80 Request: USER iscxtap
  15 0.607109  172.16.0.1  192.168.10.50  FTP      84 Request: PASS 034\t2866741
  16 0.609968  172.16.0.1  192.168.10.50  FTP      80 Request: USER iscxtap
  17 0.615635  172.16.0.1  192.168.10.50  FTP      94 Request: PASS 0-3420-49-4223\tturtle
  18 2.545047  172.16.0.1  192.168.10.50  FTP      80 Request: USER iscxtap
  19 2.545671  172.16.0.1  192.168.10.50  FTP      85 Request: PASS 033c4de\tuoda
  20 2.547676  172.16.0.1  192.168.10.50  FTP      80 Request: USER iscxtap
  21 2.548301  172.16.0.1  192.168.10.50  FTP      82 Request: PASS 0342\t9229
  22 3.153457  172.16.0.1  192.168.10.50  FTP      80 Request: USER iscxtap
  23 3.154287  172.16.0.1  192.168.10.50  FTP      88 Request: PASS 033iwim\tdancing

> Frame 6: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
> Ethernet II, Src: Cisco_14:eb:31 (00:c1:b1:14:eb:31), Dst: Dell_0a:69:f1 (00:19:b9:0a:69:f1)
v Internet Protocol Version 4, Src: 172.16.0.1, Dst: 192.168.10.50
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 52
    Identification: 0x7d7b (32123)
  > Flags: 0x40, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 62
    Protocol: TCP (6)
    Header Checksum: 0x485d [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.16.0.1
    Destination Address: 192.168.10.50
> Transmission Control Protocol, Src Port: 57612, Dst Port: 21, Seq: 1, Ack: 21, Len: 0

0000  00 19 b9 0a 69 f1 00 c1  b1 14 eb 31 08 00 45 00   ····i··· ···1··E·
0010  00 34 7d 7b 40 00 3e 06  48 5d ac 10 00 01 c0 a8   ·4}{@·>· H]······
0020  0a 32 e1 0c 00 15 9b 4c  d6 35 2c 69 06 d3 80 10   ·2·····L ·5,i····
0030  00 e5 2f 81 00 00 01 01  08 0a 00 11 2a 1e 01 4c   ··/····· ····*··L
0040  1e 10                                              ··
```

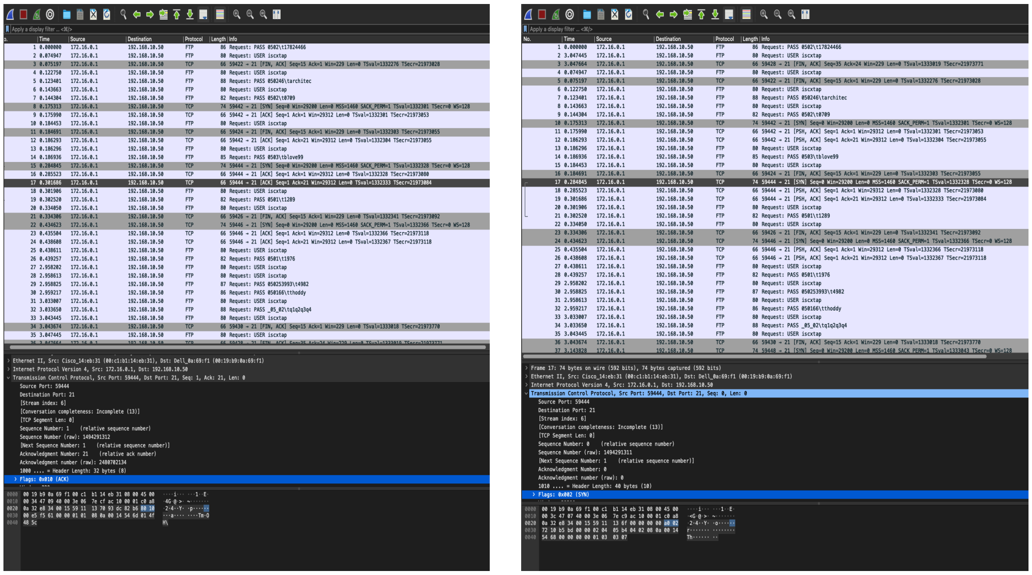
(b) DDoS attack after in Wireshark.

Figure 4: Comparison of DDoS attack before and after in Wireshark.

packets, it includes a mix of SYN, ACK, and FIN packets, which helps blend the scan into typical network behavior and evade detection.

Brute force attacks generate numerous login attempts, leading to irregular sequence numbers due to the repeated connections. By analyzing the traffic with Wireshark, we observed that DiffuPac intelligently normalizes sequence numbers to mimic regular user connections. Normal user login attempts typically show sequentially increasing sequence numbers. In Figure 6, it shows that DiffuPac adjusts these numbers to follow a similar pattern, thereby camouflaging the brute force activity. This adjustment helps blend the attack packets seamlessly into benign traffic, making it harder for NIDS to spot the abnormality.

Infiltration attacks involve moving laterally within a network, often generating unusual acknowledgment numbers as the attacker accesses various systems. In Figure 7, it demonstrated that DiffuPac



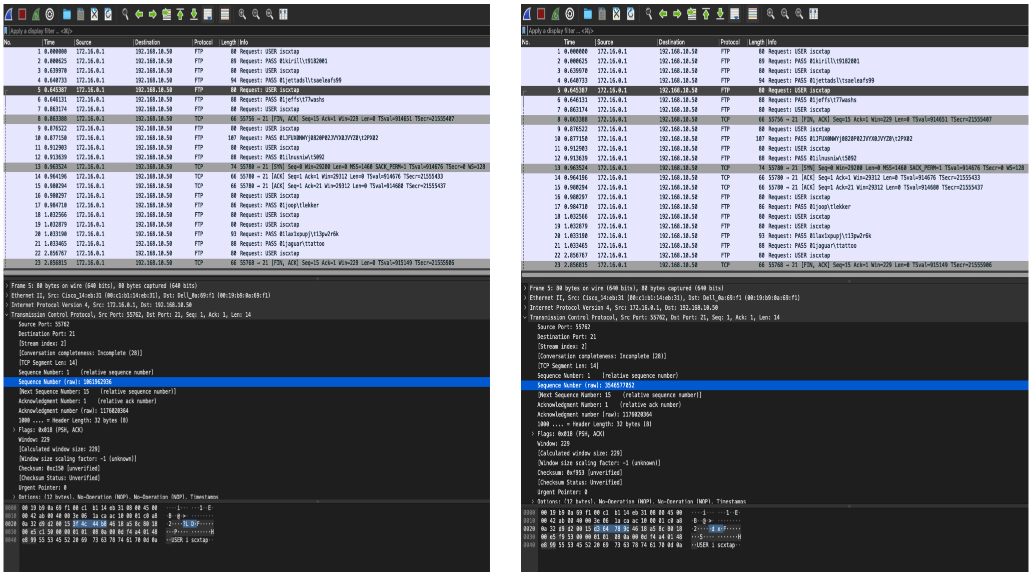
```
No.    |Time      |Source      |Destination    |Protocol |Length |Info
     1 0.000000   172.16.0.1   192.168.10.50   FTP        86 Request: PASS 0502\t17824466
     2 0.074947   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
     3 0.075197   172.16.0.1   192.168.10.50   TCP        66 59422 → 21 [FIN, ACK] Seq=15 Ack=1 Win=229 Len=0 TSval=1332276 TSecr=21973028
     4 0.122750   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
     5 0.123401   172.16.0.1   192.168.10.50   FTP        88 Request: PASS 050246\tarchitec
     6 0.143663   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
     7 0.144304   172.16.0.1   192.168.10.50   FTP        82 Request: PASS 0502\t0709
     8 0.175313   172.16.0.1   192.168.10.50   TCP        74 59442 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1332301 TSecr=0 WS=128
     9 0.175990   172.16.0.1   192.168.10.50   TCP        66 59442 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1332301 TSecr=21973053
    10 0.184453   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    11 0.184691   172.16.0.1   192.168.10.50   TCP        66 59424 → 21 [FIN, ACK] Seq=15 Ack=1 Win=229 Len=0 TSval=1332303 TSecr=21973055
    12 0.186293   172.16.0.1   192.168.10.50   TCP        66 59442 → 21 [ACK] Seq=1 Ack=21 Win=29312 Len=0 TSval=1332304 TSecr=21973055
    13 0.186296   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    14 0.186936   172.16.0.1   192.168.10.50   FTP        85 Request: PASS 0503\tblove99
    15 0.284845   172.16.0.1   192.168.10.50   TCP        74 59444 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1332328 TSecr=0 WS=128
    16 0.285523   172.16.0.1   192.168.10.50   TCP        66 59444 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1332328 TSecr=21973080
    17 0.301686   172.16.0.1   192.168.10.50   TCP        66 59444 → 21 [ACK] Seq=1 Ack=21 Win=29312 Len=0 TSval=1332333 TSecr=21973084
    18 0.301906   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    19 0.302520   172.16.0.1   192.168.10.50   FTP        82 Request: PASS 0501\t1289
    20 0.334050   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    21 0.334306   172.16.0.1   192.168.10.50   TCP        66 59426 → 21 [FIN, ACK] Seq=15 Ack=1 Win=229 Len=0 TSval=1332341 TSecr=21973092
    22 0.434623   172.16.0.1   192.168.10.50   TCP        74 59446 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1332366 TSecr=0 WS=128
    23 0.435504   172.16.0.1   192.168.10.50   TCP        66 59446 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1332366 TSecr=21973118
    24 0.438608   172.16.0.1   192.168.10.50   TCP        66 59446 → 21 [ACK] Seq=1 Ack=21 Win=29312 Len=0 TSval=1332367 TSecr=21973118
    25 0.438611   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    26 0.439257   172.16.0.1   192.168.10.50   FTP        82 Request: PASS 0501\t1976
    27 2.958202   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    28 2.958613   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    29 2.958825   172.16.0.1   192.168.10.50   FTP        87 Request: PASS 050253993\t4982
    30 2.959217   172.16.0.1   192.168.10.50   FTP        86 Request: PASS 050166\tthoddy
    31 3.033007   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    32 3.033650   172.16.0.1   192.168.10.50   FTP        88 Request: PASS _05_02\tq1q2q3q4
    33 3.043445   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    34 3.043674   172.16.0.1   192.168.10.50   TCP        66 59430 → 21 [FIN, ACK] Seq=15 Ack=1 Win=229 Len=0 TSval=1333018 TSecr=21973770
    35 3.047445   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap

> Ethernet II, Src: Cisco_14:eb:31 (00:c1:b1:14:eb:31), Dst: Dell_0a:69:f1 (00:19:b9:0a:69:f1)
> Internet Protocol Version 4, Src: 172.16.0.1, Dst: 192.168.10.50
v Transmission Control Protocol, Src Port: 59444, Dst Port: 21, Seq: 1, Ack: 21, Len: 0
    Source Port: 59444
    Destination Port: 21
    [Stream index: 6]
    [Conversation completeness: Incomplete (13)]
    [TCP Segment Len: 0]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 1494291312
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 21    (relative ack number)
    Acknowledgment number (raw): 2480702134
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x010 (ACK)

0000  00 19 b9 0a 69 f1 00 c1  b1 14 eb 31 08 00 45 00   ····i··· ···1··E·
0010  00 34 47 09 40 00 3e 06  7e cf ac 10 00 01 c0 a8   ·4G·@·>· ~·······
0020  0a 32 e8 34 00 15 59 11  13 70 93 dc 82 b6 80 10   ·2·4··Y· ·p······
0030  00 e5 f5 61 00 00 01 01  08 0a 00 14 54 6d 01 4f   ···a···· ····Tm·O
0040  48 5c                                              H\
```

(a) Port Scan attack before in Wireshark.

```
No.    |Time      |Source      |Destination    |Protocol |Length |Info
     1 0.000000   172.16.0.1   192.168.10.50   FTP        86 Request: PASS 0502\t17824466
     2 3.047445   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
     3 3.047664   172.16.0.1   192.168.10.50   TCP        66 59428 → 21 [FIN, ACK] Seq=35 Ack=24 Win=229 Len=0 TSval=1333019 TSecr=21973771
     4 0.074947   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
     5 0.075197   172.16.0.1   192.168.10.50   TCP        66 59422 → 21 [FIN, ACK] Seq=15 Ack=1 Win=229 Len=0 TSval=1332276 TSecr=21973028
     6 0.122750   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
     7 0.123401   172.16.0.1   192.168.10.50   FTP        88 Request: PASS 050246\tarchitec
     8 0.143663   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
     9 0.144304   172.16.0.1   192.168.10.50   FTP        82 Request: PASS 0502\t0709
    10 0.175313   172.16.0.1   192.168.10.50   TCP        74 59442 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1332301 TSecr=0 WS=128
    11 0.175990   172.16.0.1   192.168.10.50   TCP        66 59442 → 21 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1332301 TSecr=21973053
    12 0.186293   172.16.0.1   192.168.10.50   TCP        66 59442 → 21 [PSH, ACK] Seq=1 Ack=21 Win=29312 Len=0 TSval=1332304 TSecr=21973055
    13 0.186296   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    14 0.186936   172.16.0.1   192.168.10.50   FTP        85 Request: PASS 0503\tblove99
    15 0.184453   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    16 0.184691   172.16.0.1   192.168.10.50   TCP        66 59424 → 21 [FIN, ACK] Seq=15 Ack=1 Win=229 Len=0 TSval=1332303 TSecr=21973055
    17 0.284845   172.16.0.1   192.168.10.50   TCP        74 59444 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1332328 TSecr=0 WS=128
    18 0.285523   172.16.0.1   192.168.10.50   TCP        66 59444 → 21 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1332328 TSecr=21973080
    19 0.301686   172.16.0.1   192.168.10.50   TCP        66 59444 → 21 [PSH, ACK] Seq=1 Ack=21 Win=29312 Len=0 TSval=1332333 TSecr=21973084
    20 0.301906   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    21 0.302520   172.16.0.1   192.168.10.50   FTP        82 Request: PASS 0501\t1289
    22 0.334050   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    23 0.334306   172.16.0.1   192.168.10.50   TCP        66 59426 → 21 [FIN, ACK] Seq=15 Ack=1 Win=229 Len=0 TSval=1332341 TSecr=21973092
    24 0.434623   172.16.0.1   192.168.10.50   TCP        74 59446 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1332366 TSecr=0 WS=128
    25 0.435504   172.16.0.1   192.168.10.50   TCP        66 59446 → 21 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1332366 TSecr=21973118
    26 0.438608   172.16.0.1   192.168.10.50   TCP        66 59446 → 21 [PSH, ACK] Seq=1 Ack=21 Win=29312 Len=0 TSval=1332367 TSecr=21973118
    27 0.438611   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    28 0.439257   172.16.0.1   192.168.10.50   FTP        82 Request: PASS 0501\t1976
    29 2.958202   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    30 2.958825   172.16.0.1   192.168.10.50   FTP        87 Request: PASS 050253993\t4982
    31 2.958613   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    32 2.959217   172.16.0.1   192.168.10.50   FTP        86 Request: PASS 050166\tthoddy
    33 3.033007   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    34 3.033650   172.16.0.1   192.168.10.50   FTP        88 Request: PASS _05_02\tq1q2q3q4
    35 3.043445   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    36 3.043674   172.16.0.1   192.168.10.50   TCP        66 59430 → 21 [FIN, ACK] Seq=15 Ack=1 Win=229 Len=0 TSval=1333018 TSecr=21973770
    37 3.143828   172.16.0.1   192.168.10.50   TCP        74 59448 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1333043 TSecr=0 WS=128

> Frame 17: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
> Ethernet II, Src: Cisco_14:eb:31 (00:c1:b1:14:eb:31), Dst: Dell_0a:69:f1 (00:19:b9:0a:69:f1)
> Internet Protocol Version 4, Src: 172.16.0.1, Dst: 192.168.10.50
v Transmission Control Protocol, Src Port: 59444, Dst Port: 21, Seq: 0, Len: 0
    Source Port: 59444
    Destination Port: 21
    [Stream index: 6]
    [Conversation completeness: Incomplete (13)]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 1494291311
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    1010 .... = Header Length: 40 bytes (10)
  > Flags: 0x002 (SYN)

0000  00 19 b9 0a 69 f1 00 c1  b1 14 eb 31 08 00 45 00   ····i··· ···1··E·
0010  00 3c 47 07 40 00 3e 06  7e c9 ac 10 00 01 c0 a8   ·<G·@·>· ~·······
0020  0a 32 e8 34 00 15 59 11  13 6f 00 00 00 00 a0 02   ·2·4··Y· ·o······
0030  72 10 b5 bd 00 00 02 04  05 b4 04 02 08 0a 00 14   r······· ········
0040  54 68 00 00 00 00 01 03  03 07                     Th······ ··
```

(b) Port Scan attack after in Wireshark.

Figure 5: Comparison of Port Scan attack before and after in Wireshark.



```
No.    |Time      |Source      |Destination    |Protocol |Length |Info
     1 0.000000   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
     2 0.000625   172.16.0.1   192.168.10.50   FTP        89 Request: PASS 01kirill\t9182001
     3 0.639970   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
     4 0.640733   172.16.0.1   192.168.10.50   FTP        94 Request: PASS 01jettadsl\tsaeleafs99
     5 0.645387   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
     6 0.646131   172.16.0.1   192.168.10.50   FTP        88 Request: PASS 01jeffs\t77washs
     7 0.863174   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
     8 0.863388   172.16.0.1   192.168.10.50   TCP        66 55756 → 21 [FIN, ACK] Seq=15 Ack=1 Win=229 Len=0 TSval=914651 TSecr=21555407
     9 0.876522   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    10 0.877150   172.16.0.1   192.168.10.50   FTP       107 Request: PASS 01JFUX0NWYj0820P02JVYX0JVYZ0\t2PX02
    11 0.912903   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    12 0.913639   172.16.0.1   192.168.10.50   FTP        88 Request: PASS 01ilnusniw\t5092
    13 0.963524   172.16.0.1   192.168.10.50   TCP        74 55780 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=914676 TSecr=0 WS=128
    14 0.964196   172.16.0.1   192.168.10.50   TCP        66 55780 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=914676 TSecr=21555433
    15 0.980294   172.16.0.1   192.168.10.50   TCP        66 55780 → 21 [ACK] Seq=1 Ack=21 Win=29312 Len=0 TSval=914680 TSecr=21555437
    16 0.980297   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    17 0.984710   172.16.0.1   192.168.10.50   FTP        86 Request: PASS 01joop\tlekker
    18 1.032566   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    19 1.032879   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    20 1.033190   172.16.0.1   192.168.10.50   FTP        93 Request: PASS 01lax1xpupj\t13pw2r6k
    21 1.033465   172.16.0.1   192.168.10.50   FTP        88 Request: PASS 01jaguar\ttattoo
    22 2.856767   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    23 2.856815   172.16.0.1   192.168.10.50   TCP        66 55768 → 21 [FIN, ACK] Seq=15 Ack=1 Win=229 Len=0 TSval=915149 TSecr=21555906

> Frame 5: 80 bytes on wire (640 bits), 80 bytes captured (640 bits)
> Ethernet II, Src: Cisco_14:eb:31 (00:c1:b1:14:eb:31), Dst: Dell_0a:69:f1 (00:19:b9:0a:69:f1)
> Internet Protocol Version 4, Src: 172.16.0.1, Dst: 192.168.10.50
v Transmission Control Protocol, Src Port: 55762, Dst Port: 21, Seq: 1, Ack: 1, Len: 14
    Source Port: 55762
    Destination Port: 21
    [Stream index: 2]
    [Conversation completeness: Incomplete (28)]
    [TCP Segment Len: 14]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 1061962936
    [Next Sequence Number: 15    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 1176020364
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x018 (PSH, ACK)
    Window: 229
    [Calculated window size: 229]
    [Window size scaling factor: -1 (unknown)]
    Checksum: 0xc150 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps

0000  00 19 b9 0a 69 f1 00 c1  b1 14 eb 31 08 00 45 00   ····i··· ···1··E·
0010  00 42 ab 00 40 00 3e 06  1a ca ac 10 00 01 c0 a8   ·B··@·>· ········
0020  0a 32 d9 d2 00 15 3f 4c  44 b8 46 18 a5 8c 80 18   ·2····?L D·F·····
0030  00 e5 c1 50 00 00 01 01  08 0a 00 0d f4 a4 01 48   ···P···· ·······H
0040  e8 99 55 53 45 52 20 69  73 63 78 74 61 70 0d 0a   ··USER i scxtap··
```

(a) BruteForce attack before in Wireshark.

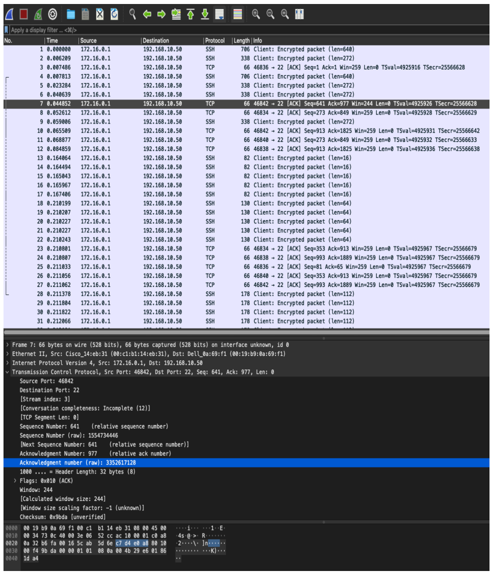
```
No.    |Time      |Source      |Destination    |Protocol |Length |Info
     1 0.000000   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
     2 0.000625   172.16.0.1   192.168.10.50   FTP        89 Request: PASS 01kirill\t9182001
     3 0.639970   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
     4 0.640733   172.16.0.1   192.168.10.50   FTP        94 Request: PASS 01jettadsl\tsaeleafs99
     5 0.645387   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
     6 0.646131   172.16.0.1   192.168.10.50   FTP        88 Request: PASS 01jeffs\t77washs
     7 0.863174   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
     8 0.863388   172.16.0.1   192.168.10.50   TCP        66 55756 → 21 [FIN, ACK] Seq=15 Ack=1 Win=229 Len=0 TSval=914651 TSecr=21555407
     9 0.876522   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    10 0.877150   172.16.0.1   192.168.10.50   FTP       107 Request: PASS 01JFUX0NWYj0820P02JVYX0JVYZ0\t2PX02
    11 0.912903   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    12 0.913639   172.16.0.1   192.168.10.50   FTP        88 Request: PASS 01ilnusniw\t5092
    13 0.963524   172.16.0.1   192.168.10.50   TCP        74 55780 → 21 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=914676 TSecr=0 WS=128
    14 0.964196   172.16.0.1   192.168.10.50   TCP        66 55780 → 21 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=914676 TSecr=21555433
    15 0.980294   172.16.0.1   192.168.10.50   TCP        66 55780 → 21 [ACK] Seq=1 Ack=21 Win=29312 Len=0 TSval=914680 TSecr=21555437
    16 0.980297   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    17 0.984710   172.16.0.1   192.168.10.50   FTP        86 Request: PASS 01joop\tlekker
    18 1.032566   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    19 1.032879   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    20 1.033190   172.16.0.1   192.168.10.50   FTP        93 Request: PASS 01lax1xpupj\t13pw2r6k
    21 1.033465   172.16.0.1   192.168.10.50   FTP        88 Request: PASS 01jaguar\ttattoo
    22 2.856767   172.16.0.1   192.168.10.50   FTP        80 Request: USER iscxtap
    23 2.856815   172.16.0.1   192.168.10.50   TCP        66 55768 → 21 [FIN, ACK] Seq=15 Ack=1 Win=229 Len=0 TSval=915149 TSecr=21555906

> Frame 5: 80 bytes on wire (640 bits), 80 bytes captured (640 bits)
> Ethernet II, Src: Cisco_14:eb:31 (00:c1:b1:14:eb:31), Dst: Dell_0a:69:f1 (00:19:b9:0a:69:f1)
> Internet Protocol Version 4, Src: 172.16.0.1, Dst: 192.168.10.50
v Transmission Control Protocol, Src Port: 55762, Dst Port: 21, Seq: 1, Ack: 1, Len: 14
    Source Port: 55762
    Destination Port: 21
    [Stream index: 2]
    [Conversation completeness: Incomplete (28)]
    [TCP Segment Len: 14]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 3546577052
    [Next Sequence Number: 15    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 1176020364
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x018 (PSH, ACK)
    Window: 229
    [Calculated window size: 229]
    [Window size scaling factor: -1 (unknown)]
    Checksum: 0xf953 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps

0000  00 19 b9 0a 69 f1 00 c1  b1 14 eb 31 08 00 45 00   ····i··· ···1··E·
0010  00 42 ab 00 40 00 3e 06  1a ca ac 10 00 01 c0 a8   ·B··@·>· ········
0020  0a 32 d9 d2 00 15 d3 64  78 9c 46 18 a5 8c 80 18   ·2····d x·F·····
0030  00 e5 f9 53 00 00 01 01  08 0a 00 0d f4 a4 01 48   ···S···· ·······H
0040  e8 99 55 53 45 52 20 69  73 63 78 74 61 70 0d 0a   ··USER i scxtap··
```

(b) BruteForce attack after in Wireshark.

Figure 6: Comparison of bruteForce attack before and after in Wireshark.

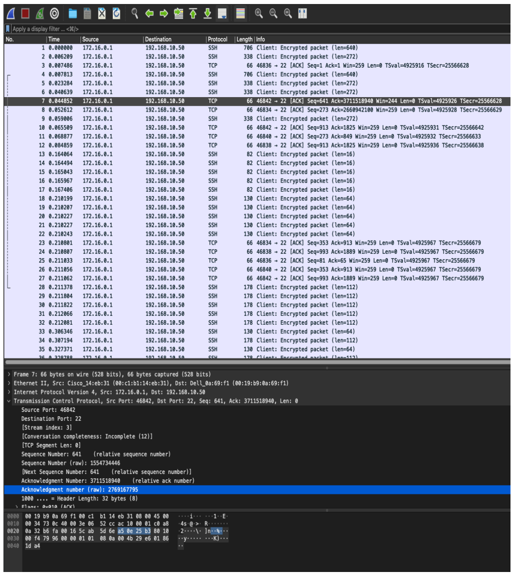
intelligently modifies acknowledgment numbers to follow expected sequences. In normal communication, acknowledgment numbers increment predictably based on the received data. DiffuPac ensures that infiltration attempts have acknowledgment numbers that follow these normal patterns, effectively masking the lateral movement within the network and blending the attack traffic with normal traffic, making it difficult for detection systems to identify the anomaly.

DiffuPac demonstrates its ability to modify specific fields intelligently in the packet headers of different attack types, such as the window size for botnet traffic and TTL for MITM, to blend malicious packets into normal packets. This capability is consistently proven through detailed packet analysis with Wireshark. Indeed, the detailed packets analysis shows that critical fields defining packet identity and practicality (e.g., IP addresses, ports, payload) remain intact after modification, ensuring the functionality of the attack while evading detection by sophisticated NIDS. The comparisons



```
Apply a display filter ... <⌘/>
No.   Time      Source       Destination    Protocol Length Info
    1 0.000000  172.16.0.1   192.168.10.50  SSH      706 Client: Encrypted packet (len=640)
    2 0.006209  172.16.0.1   192.168.10.50  SSH      338 Client: Encrypted packet (len=272)
    3 0.007486  172.16.0.1   192.168.10.50  TCP       66 46836 → 22 [ACK] Seq=1 Ack=1 Win=259 Len=0 TSval=4925916 TSecr=25566628
    4 0.007813  172.16.0.1   192.168.10.50  SSH      706 Client: Encrypted packet (len=640)
    5 0.023284  172.16.0.1   192.168.10.50  SSH      338 Client: Encrypted packet (len=272)
    6 0.040639  172.16.0.1   192.168.10.50  SSH      338 Client: Encrypted packet (len=272)
    7 0.044852  172.16.0.1   192.168.10.50  TCP       66 46842 → 22 [ACK] Seq=641 Ack=977 Win=244 Len=0 TSval=4925926 TSecr=25566628
    8 0.052612  172.16.0.1   192.168.10.50  TCP       66 46834 → 22 [ACK] Seq=273 Ack=849 Win=259 Len=0 TSval=4925928 TSecr=25566629
    9 0.059006  172.16.0.1   192.168.10.50  SSH      338 Client: Encrypted packet (len=272)
   10 0.065509  172.16.0.1   192.168.10.50  TCP       66 46842 → 22 [ACK] Seq=913 Ack=1825 Win=259 Len=0 TSval=4925931 TSecr=25566642
   11 0.068877  172.16.0.1   192.168.10.50  TCP       66 46840 → 22 [ACK] Seq=273 Ack=849 Win=259 Len=0 TSval=4925932 TSecr=25566633
   12 0.084859  172.16.0.1   192.168.10.50  TCP       66 46838 → 22 [ACK] Seq=913 Ack=1825 Win=259 Len=0 TSval=4925936 TSecr=25566638
   13 0.164064  172.16.0.1   192.168.10.50  SSH       82 Client: Encrypted packet (len=16)
   14 0.164494  172.16.0.1   192.168.10.50  SSH       82 Client: Encrypted packet (len=16)
   15 0.165043  172.16.0.1   192.168.10.50  SSH       82 Client: Encrypted packet (len=16)
   16 0.165967  172.16.0.1   192.168.10.50  SSH       82 Client: Encrypted packet (len=16)
   17 0.167406  172.16.0.1   192.168.10.50  SSH       82 Client: Encrypted packet (len=16)
   18 0.210199  172.16.0.1   192.168.10.50  SSH      130 Client: Encrypted packet (len=64)
   19 0.210207  172.16.0.1   192.168.10.50  SSH      130 Client: Encrypted packet (len=64)
   20 0.210227  172.16.0.1   192.168.10.50  SSH      130 Client: Encrypted packet (len=64)
   21 0.210227  172.16.0.1   192.168.10.50  SSH      130 Client: Encrypted packet (len=64)
   22 0.210243  172.16.0.1   192.168.10.50  SSH      130 Client: Encrypted packet (len=64)
   23 0.210801  172.16.0.1   192.168.10.50  TCP       66 46834 → 22 [ACK] Seq=353 Ack=913 Win=259 Len=0 TSval=4925967 TSecr=25566679
   24 0.210807  172.16.0.1   192.168.10.50  TCP       66 46838 → 22 [ACK] Seq=993 Ack=1889 Win=259 Len=0 TSval=4925967 TSecr=25566679
   25 0.211033  172.16.0.1   192.168.10.50  TCP       66 46836 → 22 [ACK] Seq=81 Ack=65 Win=259 Len=0 TSval=4925967 TSecr=25566679
   26 0.211056  172.16.0.1   192.168.10.50  TCP       66 46840 → 22 [ACK] Seq=353 Ack=913 Win=259 Len=0 TSval=4925967 TSecr=25566679
   27 0.211062  172.16.0.1   192.168.10.50  TCP       66 46842 → 22 [ACK] Seq=993 Ack=1889 Win=259 Len=0 TSval=4925967 TSecr=25566679
   28 0.211378  172.16.0.1   192.168.10.50  SSH      178 Client: Encrypted packet (len=112)
   29 0.211804  172.16.0.1   192.168.10.50  SSH      178 Client: Encrypted packet (len=112)
   30 0.211822  172.16.0.1   192.168.10.50  SSH      178 Client: Encrypted packet (len=112)
   31 0.212066  172.16.0.1   192.168.10.50  SSH      178 Client: Encrypted packet (len=112)

> Frame 7: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface unknown, id 0
> Ethernet II, Src: Cisco_14:eb:31 (00:c1:b1:14:eb:31), Dst: Dell_0a:69:f1 (00:19:b9:0a:69:f1)
> Internet Protocol Version 4, Src: 172.16.0.1, Dst: 192.168.10.50
v Transmission Control Protocol, Src Port: 46842, Dst Port: 22, Seq: 641, Ack: 977, Len: 0
    Source Port: 46842
    Destination Port: 22
    [Stream index: 3]
    [Conversation completeness: Incomplete (12)]
    [TCP Segment Len: 0]
    Sequence Number: 641    (relative sequence number)
    Sequence Number (raw): 1554734446
    [Next Sequence Number: 641    (relative sequence number)]
    Acknowledgment Number: 977    (relative ack number)
    Acknowledgment number (raw): 3352617128
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x010 (ACK)
    Window: 244
    [Calculated window size: 244]
    [Window size scaling factor: -1 (unknown)]
    Checksum: 0x9bda [unverified]

0000  00 19 b9 0a 69 f1 00 c1  b1 14 eb 31 08 00 45 00   ····i··· ···1··E·
0010  00 34 73 0c 40 00 3e 06  52 cc ac 10 00 01 c0 a8   ·4s·@·>· R·······
0020  0a 32 b6 fa 00 16 5c ab  5d 6e c7 d4 e0 a8 80 10   ·2····\· ]n······
0030  00 f4 9b da 00 00 01 01  08 0a 00 4b 29 e6 01 86   ········ ···K)···
0040  1d a4                                              ··
```

(a) Infiltration attack before in Wireshark.



```
Apply a display filter ... <⌘/>
No.   Time      Source       Destination    Protocol Length Info
    1 0.000000  172.16.0.1   192.168.10.50  SSH      706 Client: Encrypted packet (len=640)
    2 0.006209  172.16.0.1   192.168.10.50  SSH      338 Client: Encrypted packet (len=272)
    3 0.007486  172.16.0.1   192.168.10.50  TCP       66 46836 → 22 [ACK] Seq=1 Ack=1 Win=259 Len=0 TSval=4925916 TSecr=25566628
    4 0.007813  172.16.0.1   192.168.10.50  SSH      706 Client: Encrypted packet (len=640)
    5 0.023284  172.16.0.1   192.168.10.50  SSH      338 Client: Encrypted packet (len=272)
    6 0.040639  172.16.0.1   192.168.10.50  SSH      338 Client: Encrypted packet (len=272)
    7 0.044852  172.16.0.1   192.168.10.50  TCP       66 46842 → 22 [ACK] Seq=641 Ack=3711518940 Win=244 Len=0 TSval=4925926 TSecr=25566628
    8 0.052612  172.16.0.1   192.168.10.50  TCP       66 46834 → 22 [ACK] Seq=273 Ack=2660942100 Win=259 Len=0 TSval=4925928 TSecr=25566629
    9 0.059006  172.16.0.1   192.168.10.50  SSH      338 Client: Encrypted packet (len=272)
   10 0.065509  172.16.0.1   192.168.10.50  TCP       66 46842 → 22 [ACK] Seq=913 Ack=1825 Win=259 Len=0 TSval=4925931 TSecr=25566642
   11 0.068877  172.16.0.1   192.168.10.50  TCP       66 46840 → 22 [ACK] Seq=273 Ack=849 Win=259 Len=0 TSval=4925932 TSecr=25566633
   12 0.084859  172.16.0.1   192.168.10.50  TCP       66 46838 → 22 [ACK] Seq=913 Ack=1825 Win=259 Len=0 TSval=4925936 TSecr=25566638
   13 0.164064  172.16.0.1   192.168.10.50  SSH       82 Client: Encrypted packet (len=16)
   14 0.164494  172.16.0.1   192.168.10.50  SSH       82 Client: Encrypted packet (len=16)
   15 0.165043  172.16.0.1   192.168.10.50  SSH       82 Client: Encrypted packet (len=16)
   16 0.165967  172.16.0.1   192.168.10.50  SSH       82 Client: Encrypted packet (len=16)
   17 0.167406  172.16.0.1   192.168.10.50  SSH       82 Client: Encrypted packet (len=16)
   18 0.210199  172.16.0.1   192.168.10.50  SSH      130 Client: Encrypted packet (len=64)
   19 0.210207  172.16.0.1   192.168.10.50  SSH      130 Client: Encrypted packet (len=64)
   20 0.210227  172.16.0.1   192.168.10.50  SSH      130 Client: Encrypted packet (len=64)
   21 0.210227  172.16.0.1   192.168.10.50  SSH      130 Client: Encrypted packet (len=64)
   22 0.210243  172.16.0.1   192.168.10.50  SSH      130 Client: Encrypted packet (len=64)
   23 0.210801  172.16.0.1   192.168.10.50  TCP       66 46834 → 22 [ACK] Seq=353 Ack=913 Win=259 Len=0 TSval=4925967 TSecr=25566679
   24 0.210807  172.16.0.1   192.168.10.50  TCP       66 46838 → 22 [ACK] Seq=993 Ack=1889 Win=259 Len=0 TSval=4925967 TSecr=25566679
   25 0.211033  172.16.0.1   192.168.10.50  TCP       66 46836 → 22 [ACK] Seq=81 Ack=65 Win=259 Len=0 TSval=4925967 TSecr=25566679
   26 0.211056  172.16.0.1   192.168.10.50  TCP       66 46840 → 22 [ACK] Seq=353 Ack=913 Win=259 Len=0 TSval=4925967 TSecr=25566679
   27 0.211062  172.16.0.1   192.168.10.50  TCP       66 46842 → 22 [ACK] Seq=993 Ack=1889 Win=259 Len=0 TSval=4925967 TSecr=25566679
   28 0.211378  172.16.0.1   192.168.10.50  SSH      178 Client: Encrypted packet (len=112)
   29 0.211804  172.16.0.1   192.168.10.50  SSH      178 Client: Encrypted packet (len=112)
   30 0.211822  172.16.0.1   192.168.10.50  SSH      178 Client: Encrypted packet (len=112)
   31 0.212066  172.16.0.1   192.168.10.50  SSH      178 Client: Encrypted packet (len=112)
   32 0.212081  172.16.0.1   192.168.10.50  SSH      178 Client: Encrypted packet (len=112)
   33 0.306346  172.16.0.1   192.168.10.50  SSH      130 Client: Encrypted packet (len=64)
   34 0.307194  172.16.0.1   192.168.10.50  SSH      178 Client: Encrypted packet (len=112)
   35 0.327371  172.16.0.1   192.168.10.50  SSH      130 Client: Encrypted packet (len=64)

> Frame 7: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
> Ethernet II, Src: Cisco_14:eb:31 (00:c1:b1:14:eb:31), Dst: Dell_0a:69:f1 (00:19:b9:0a:69:f1)
> Internet Protocol Version 4, Src: 172.16.0.1, Dst: 192.168.10.50
v Transmission Control Protocol, Src Port: 46842, Dst Port: 22, Seq: 641, Ack: 3711518940, Len: 0
    Source Port: 46842
    Destination Port: 22
    [Stream index: 3]
    [Conversation completeness: Incomplete (12)]
    [TCP Segment Len: 0]
    Sequence Number: 641    (relative sequence number)
    Sequence Number (raw): 1554734446
    [Next Sequence Number: 641    (relative sequence number)]
    Acknowledgment Number: 3711518940    (relative ack number)
    Acknowledgment number (raw): 2769167795
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x010 (ACK)

0000  00 19 b9 0a 69 f1 00 c1  b1 14 eb 31 08 00 45 00   ····i··· ···1··E·
0010  00 34 73 0c 40 00 3e 06  52 cc ac 10 00 01 c0 a8   ·4s·@·>· R·······
0020  0a 32 b6 fa 00 16 5c ab  5d 6e a5 0e 25 b3 80 10   ·2····\· ]n··%···
0030  00 f4 79 96 00 00 01 01  08 0a 00 4b 29 e6 01 86   ··y····· ···K)···
0040  1d a4                                              ··
```

(b) Infiltration attack after in Wireshark.

Figure 7: Comparison of infiltration attack before and after in Wireshark.

for each attack type before and after the modificatinos further highlight DiffuPac's effectiveness in maintaining attack practicality while enhancing stealth.

B.2 Evasion Rate

We evaluated the evasion rates of the other 4 attacks—DDoS, Port Scan, Brute Force, and Infiltration—using DiffuPac and the baseline models, with the detailed results summarized in Table 7. According to the results, we can observed that the trends in the 4 attacks are consistent with the 2 attacks shown in Table 2 in the main paper.

Table 7: Comparative analysis of attack detection and evasion rates of the other 4 attacks.

(c) DDoS

Feature Extractor	Classifier	Detection			Evasion (*MER*)		
		P	R	F1	GAN & PSO	LSTM	**Ours**
CIC FLowMeter	KitNET	0.94	0.90	0.92	33.19%	41.69%	**52.06%**
	DT	0.73	0.81	0.72	39.92%	40.80%	**46.81%**
	IF	0.99	0.91	0.93	26.74%	39.49%	**51.45%**
	MLP	0.72	0.71	0.74	52.03%	48.17%	**69.87%**
	SVM	0.73	0.77	0.78	42.87%	**58.47%**	54.57%
	LR	0.75	0.79	0.76	48.79%	51.08%	**58.95%**
AfterImage	KitNET	0.97	0.95	0.96	59.84%	**74.06%**	62.31%
	DT	0.78	0.92	0.85	**72.88%**	63.27%	59.47%
	IF	0.86	0.84	0.88	22.63%	36.33%	**48.91%**
	MLP	0.91	0.94	0.92	55.45%	57.77%	**63.25%**
	SVM	0.99	0.90	0.94	78.51%	**79.97%**	68.93%
	LR	0.94	0.97	0.91	**71.68%**	69.95%	56.66%

(d) Port Scan

Feature Extractor	Classifier	Detection			Evasive (*MER*)		
		P	R	F1	GAN & PSO	LSTM	**Ours**
CIC FLowMeter	KitNET	0.96	0.91	0.94	33.46%	40.04%	**55.03%**
	DT	0.78	0.72	0.73	34.01%	**61.23%**	59.25%
	IF	0.99	0.90	0.94	22.73%	29.49%	**45.26%**
	MLP	0.76	0.71	0.74	59.72%	57.56%	**67.92%**
	SVM	0.73	0.77	0.78	**68.91%**	58.37%	60.95%
	LR	0.78	0.83	0.79	42.04%	49.78%	**54.21%**
AfterImage	KitNET	0.94	0.96	0.93	**69.26%**	64.97%	55.87%
	DT	0.77	0.91	0.82	69.78%	62.01%	**70.47%**
	IF	0.84	0.91	0.89	18.38%	22.15%	**48.30%**
	MLP	0.96	0.94	0.97	49.73%	56.62%	**64.02%**
	SVM	0.97	0.96	0.95	76.40%	**78.96%**	75.81%
	LR	0.95	0.92	0.90	67.29%	70.07%	**73.94%**

(e) BruteForce

Feature Extractor	Classifier	Detection			Evasive (*MER*)		
		P	R	F1	GAN & PSO	LSTM	**Ours**
CIC FLowMeter	KitNET	0.97	0.94	0.97	36.94%	43.13%	**50.32%**
	DT	0.75	0.77	0.72	40.56%	**44.87%**	42.79%
	IF	0.91	0.90	0.94	27.01%	39.14%	**41.64%**
	MLP	0.88	0.85	0.83	52.32%	64.58%	**67.18%**
	SVM	0.80	0.81	0.72	47.31%	51.54%	**56.74%**
	LR	0.79	0.75	0.78	49.05%	58.77%	**61.89%**
AfterImage	KitNET	0.96	0.93	0.95	81.84%	**86.39%**	69.31%
	DT	0.82	0.79	0.78	41.07%	43.56%	**55.07%**
	IF	0.88	0.98	0.92	24.43%	22.99%	**39.71%**
	MLP	0.91	0.94	0.92	59.69%	63.09%	**68.25%**
	SVM	0.94	0.92	0.96	**68.51%**	61.79%	58.93%
	LR	0.97	0.93	0.94	61.68%	57.25%	**64.06%**

(f) Infiltration

Feature Extractor	Classifier	Detection			Evasive (*MER*)		
		P	R	F1	GAN & PSO	LSTM	**Ours**
CIC FLowMeter	KitNET	0.91	0.90	0.96	41.71%	40.68%	**45.04%**
	DT	0.72	0.78	0.79	32.47%	48.81%	**62.67%**
	IF	0.99	0.91	0.94	28.10%	42.19%	**45.88%**
	MLP	0.77	0.71	0.74	50.09%	53.68%	**63.14%**
	SVM	0.74	0.76	0.79	62.29%	**64.51%**	56.11%
	LR	0.71	0.80	0.76	43.05%	50.67%	**61.94%**
AfterImage	KitNET	0.92	0.98	0.97	**73.44%**	70.58%	51.26%
	DT	0.77	0.91	0.82	69.23%	**72.32%**	62.54%
	IF	0.84	0.91	0.89	23.61%	27.51%	**42.59%**
	MLP	0.96	0.94	0.97	50.65%	48.73%	**72.25%**
	SVM	0.94	0.96	0.98	**72.48%**	68.95%	64.73%
	LR	0.93	0.92	0.95	66.04%	65.93%	**68.37%**

B.3 Malicious Functionality Evaluation

Experimental Setup. We conducted a robust evaluation of both Port Scan and Brute Force attacks to assess their malicious functionality in controlled, isolated network environments. The primary goal of this evaluation was to determine whether adversarially modified packets retain their ability to perform malicious actions. Throughout the evaluation, we focused on analyzing the response of both the original and adversarial packets using Wireshark, carefully observing how the target system reacted to each type of attack.

B.3.1 Port Scan

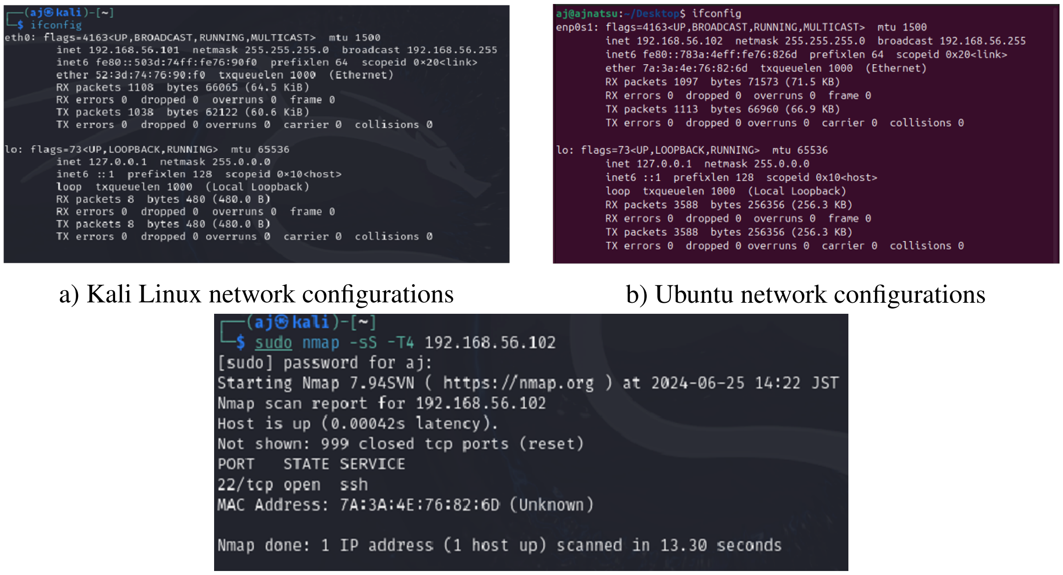
Before proceeding to evaluate the generated adversarial packets of DiffuPac, we first demonstrated a a successful Port Scan attack within our controlled environment. This initial step is a fundamental step to ensure the validity and reliability of subsequent tests with both original and adversarial packets. The network configurations depicted in Figure 8(a) and Figure 8(b) confirm that both virtual machines were correctly placed within the same subnet. As demonstrated in Figure 8(c), the Port Scan successfully identified open ports on the Ubuntu, highlighting port 22 as vulnerable.

We initiated a Port Scan attack from the Kali Linux machine targeting the Ubuntu server. During the execution of this attack, we captured all packets sent from the attacker to the victim, saving them as a pcap file. This capture was timed precisely to begin as the Nmap tool launched the Port Scan, ensuring that only the relevant src-to-dst packets were recorded. This focus aligns with the operational framework of DiffuPac, which concentrates on modifying packets sent from the source.

Using the captured src-to-dst packets, we employed DiffuPac to generate adversarial Port Scan packets, which were then saved in a separate pcap file. These adversarial packets were subsequently replayed using the Tcpreplay tool, directing them towards the Ubuntu server. Concurrently, we used Wireshark on the Ubuntu machine to capture the incoming responses, allowing us to analyze the effectiveness of the adversarial packets in real-time.

To ensure a balanced evaluation, we also replayed the original Port Scan pcap file using Tcpreplay, capturing the responses in Wireshark on the Ubuntu side. This dual replay enabled a direct comparison between the packets' responses generated by the original and adversarial Port Scan packets.

As depicted in Figure 9, the response to the adversarial Port Scan packets closely mirrors that of the original Port Scan. A detailed inspection of the TCP flags reveals that packets with SYN and ACK flags targeted port 22, corroborating the initial vulnerability identified in Figure 8(c). This consistent response across both original and adversarial packets provides compelling evidence that DiffuPac successfully generates adversarial packets that retain their intended malicious functionality.



```
┌──(aj㉿kali)-[~]
└─$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.101  netmask 255.255.255.0  broadcast 192.168.56.255
        inet6 fe80::503d:74ff:fe76:90f0  prefixlen 64  scopeid 0×20<link>
        ether 52:3d:74:76:90:f0  txqueuelen 1000  (Ethernet)
        RX packets 1108  bytes 66065 (64.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1038  bytes 62122 (60.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0×10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 8  bytes 480 (480.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 480 (480.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

a) Kali Linux network configurations

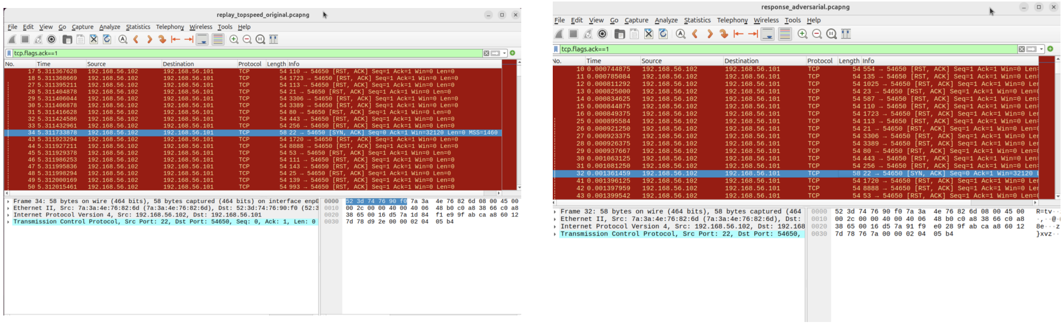
```
aj@ajnatsu:~/Desktop$ ifconfig
enp0s1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.102  netmask 255.255.255.0  broadcast 192.168.56.255
        inet6 fe80::783a:4eff:fe76:826d  prefixlen 64  scopeid 0x20<link>
        ether 7a:3a:4e:76:82:6d  txqueuelen 1000  (Ethernet)
        RX packets 1097  bytes 71573 (71.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1113  bytes 66960 (66.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 3588  bytes 256356 (256.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3588  bytes 256356 (256.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

b) Ubuntu network configurations

```
┌──(aj㉿kali)-[~]
└─$ sudo nmap -sS -T4 192.168.56.102
[sudo] password for aj:
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-25 14:22 JST
Nmap scan report for 192.168.56.102
Host is up (0.00042s latency).
Not shown: 999 closed tcp ports (reset)
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: 7A:3A:4E:76:82:6D (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 13.30 seconds
```

c) Successfully executed Port Scan using Nmap tool

Figure 8: Port scan attack demonstration on isolated environment


replay_topspeed_original.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp.flags.ack==1

```
No.   Time        Source          Destination     Protocol Length Info
   17 5.311367628 192.168.56.102  192.168.56.101  TCP        54 110 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
   18 5.311368669 192.168.56.102  192.168.56.101  TCP        54 1723 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
   27 5.311395211 192.168.56.102  192.168.56.101  TCP        54 113 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
   28 5.311404878 192.168.56.102  192.168.56.101  TCP        54 21 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
   29 5.311406044 192.168.56.102  192.168.56.101  TCP        54 3306 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
   30 5.311406878 192.168.56.102  192.168.56.101  TCP        54 3389 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
   31 5.311416628 192.168.56.102  192.168.56.101  TCP        54 80 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
   32 5.311424586 192.168.56.102  192.168.56.101  TCP        54 443 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
   33 5.311432961 192.168.56.102  192.168.56.101  TCP        54 256 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
   34 5.311733878 192.168.56.102  192.168.56.101  TCP        58 22 → 54650 [SYN, ACK] Seq=0 Ack=1 Win=32120 Len=0 MSS=1460
   43 5.311923294 192.168.56.102  192.168.56.101  TCP        54 1720 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
   44 5.311927211 192.168.56.102  192.168.56.101  TCP        54 8888 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
   45 5.311929378 192.168.56.102  192.168.56.101  TCP        54 53 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
   46 5.311986253 192.168.56.102  192.168.56.101  TCP        54 111 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
   47 5.311995836 192.168.56.102  192.168.56.101  TCP        54 143 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
   48 5.311998294 192.168.56.102  192.168.56.101  TCP        54 25 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
   49 5.312000169 192.168.56.102  192.168.56.101  TCP        54 139 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
   50 5.312015461 192.168.56.102  192.168.56.101  TCP        54 993 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

› Frame 34: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface enp0   0000  52 3d 74 76 90 f0 7a 3a  4e 76 82 6d 08 00 45 00
› Ethernet II, Src: 7a:3a:4e:76:82:6d (7a:3a:4e:76:82:6d), Dst: 52:3d:74:76:90:f0 (52:3  0010  00 2c 00 00 40 00 40 06  48 b0 c0 a8 38 66 c0 a8
› Internet Protocol Version 4, Src: 192.168.56.102, Dst: 192.168.56.101                   0020  38 65 00 16 d5 7a 1d 84  f1 e9 9f ab ca a8 60 12
› Transmission Control Protocol, Src Port: 22, Dst Port: 54650, Seq: 0, Ack: 1, Len: 0    0030  7d 78 d9 2e 00 00 02 04  05 b4
```

a) Original Port Scan Response

response_adversarial.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp.flags.ack==1

```
No.   Time        Source          Destination     Protocol Length Info
   10 0.000744875 192.168.56.102  192.168.56.101  TCP        54 554 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=
   11 0.000785084 192.168.56.102  192.168.56.101  TCP        54 135 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=
   12 0.000811292 192.168.56.102  192.168.56.101  TCP        54 1025 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Ler
   13 0.000825000 192.168.56.102  192.168.56.101  TCP        54 23 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
   14 0.000834625 192.168.56.102  192.168.56.101  TCP        54 587 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=
   15 0.000844875 192.168.56.102  192.168.56.101  TCP        54 110 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=
   16 0.000849375 192.168.56.102  192.168.56.101  TCP        54 1723 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Ler
   25 0.000895584 192.168.56.102  192.168.56.101  TCP        54 113 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=
   26 0.000921250 192.168.56.102  192.168.56.101  TCP        54 21 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
   27 0.000923375 192.168.56.102  192.168.56.101  TCP        54 3306 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Ler
   28 0.000926375 192.168.56.102  192.168.56.101  TCP        54 3389 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Ler
   29 0.000937667 192.168.56.102  192.168.56.101  TCP        54 80 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
   30 0.001063125 192.168.56.102  192.168.56.101  TCP        54 443 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=
   31 0.001081250 192.168.56.102  192.168.56.101  TCP        54 256 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=
   32 0.001361459 192.168.56.102  192.168.56.101  TCP        58 22 → 54650 [SYN, ACK] Seq=0 Ack=1 Win=32120
   41 0.001396125 192.168.56.102  192.168.56.101  TCP        54 1720 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Ler
   42 0.001397959 192.168.56.102  192.168.56.101  TCP        54 8888 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Ler
   43 0.001399542 192.168.56.102  192.168.56.101  TCP        54 53 → 54650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

› Frame 32: 58 bytes on wire (464 bits), 58 bytes captured (464    0000  52 3d 74 76 90 f0 7a 3a  4e 76 82 6d 08 00 45 00   R=tv··
› Ethernet II, Src: 7a:3a:4e:76:82:6d (7a:3a:4e:76:82:6d), Dst:    0010  00 2c 00 00 40 00 40 06  48 b0 c0 a8 38 66 c0 a8   ·,··@·
› Internet Protocol Version 4, Src: 192.168.56.102, Dst: 192.168   0020  38 65 00 16 d5 7a 91 f9  e0 28 9f ab ca a8 60 12   8e···z
› Transmission Control Protocol, Src Port: 22, Dst Port: 54650,    0030  7d 78 76 7a 00 00 02 04  05 b4                     }xvz··
```

b) Adversarial Port Scan Response

Figure 9: Comparison of Victim Response demonstrated in wireshark

B.3.2 Brute Force

For the Brute Force attack evaluation, the experimental setup followed a similar approach to that of the Port Scan, focusing on capturing and analyzing both original and adversarial packets. However, in this case, we leveraged Metasploit's ssh_login module to perform the attack. After demonstrating a successful Brute Force attack using Metasploit, we captured the relevant packets and utilized DiffuPac to generate adversarial versions of those packets. The evaluation focused on comparing the response of original and adversarial packets, observing how successful logins were achieved in both cases.

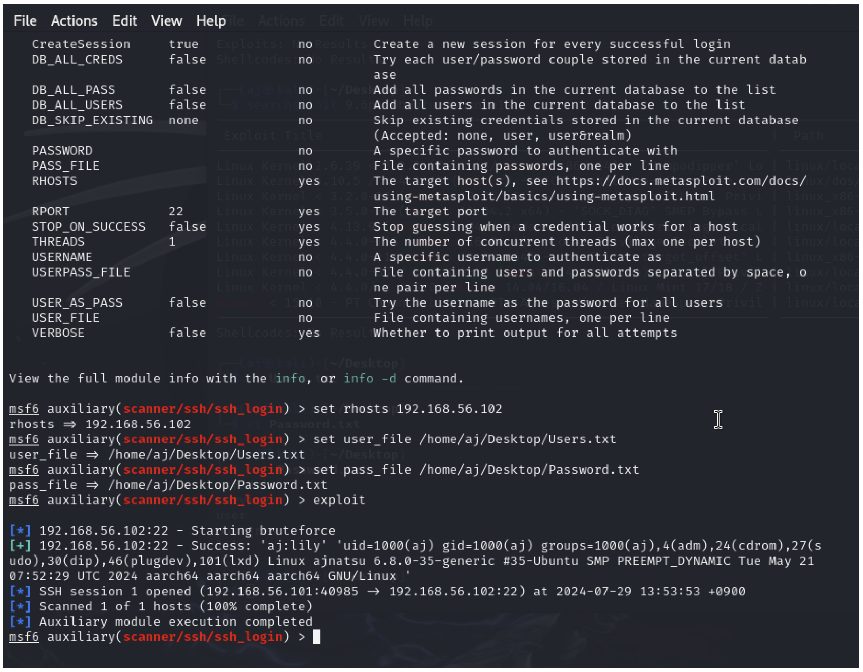
As shown in Figure 10, we utilized the Metasploit Framework to perform a Brute Force attack targeting the SSH service on the Ubuntu server. After setting the RHOSTS to the Ubuntu machine, we provided both a user file and a password file containing potential login credentials. The ssh_login module systematically attempted each combination of usernames and passwords, ultimately leading to a successful login.

The success of the Brute Force attack is demonstrated in Figure 10, where Metasploit successfully opened an SSH session on the target machine. This provided the necessary confirmation that the attack could be executed under normal conditions, laying the foundation for the subsequent evaluation of adversarial packets.

Following the successful attack, similar to the Port Scan setup, we captured the src-to-dst packets exchanged during the Brute Force attack. These packets were saved as a pcap file, which was subsequently used as input for DiffuPac to generate adversarial Brute Force packets.

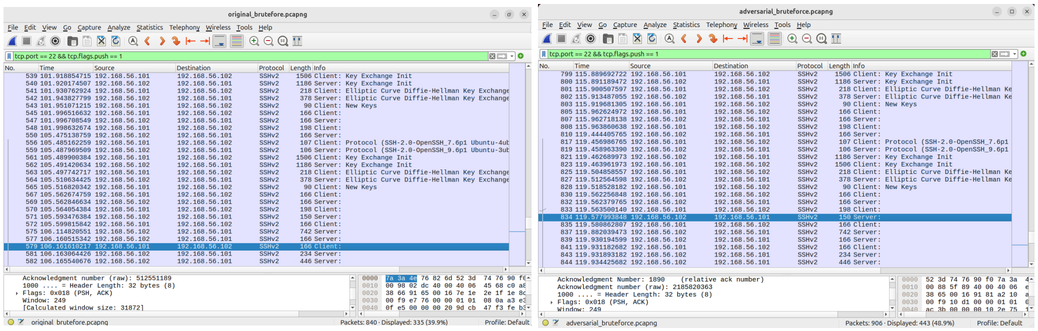
Using Tcpreplay, both the original and adversarial Brute Force packets were replayed towards the Ubuntu server. As shown in Figure 11, the response to the adversarial Brute Force packets closely mirrored that of the original Brute Force packets. The packet exchange between the attacker and the target server increased significantly as the correct login credentials were identified, culminating in a successful SSH session for both the original and adversarial attacks.

To further analyze and validate the outcomes of both the original and adversarial Brute Force attacks, we utilized pam_unix(sshd) on the Ubuntu server to log and compare successful login attempts. As depicted in Figure 12, the log entries confirmed that the SSH service recorded successful logins for both the original Brute Force attack and the adversarial Brute Force attack.



```
   CreateSession      true        no        Create a new session for every successful login
   DB_ALL_CREDS       false       no        Try each user/password couple stored in the current datab
                                            ase
   DB_ALL_PASS        false       no        Add all passwords in the current database to the list
   DB_ALL_USERS       false       no        Add all users in the current database to the list
   DB_SKIP_EXISTING   none        no        Skip existing credentials stored in the current database
                                            (Accepted: none, user, user&realm)
   PASSWORD                       no        A specific password to authenticate with
   PASS_FILE                      no        File containing passwords, one per line
   RHOSTS                         yes       The target host(s), see https://docs.metasploit.com/docs/
                                            using-metasploit/basics/using-metasploit.html
   RPORT              22          yes       The target port
   STOP_ON_SUCCESS    false       yes       Stop guessing when a credential works for a host
   THREADS            1           yes       The number of concurrent threads (max one per host)
   USERNAME                       no        A specific username to authenticate as
   USERPASS_FILE                  no        File containing users and passwords separated by space, o
                                            ne pair per line
   USER_AS_PASS       false       no        Try the username as the password for all users
   USER_FILE                      no        File containing usernames, one per line
   VERBOSE            false       yes       Whether to print output for all attempts


View the full module info with the info, or info -d command.

msf6 auxiliary(scanner/ssh/ssh_login) > set rhosts 192.168.56.102
rhosts ⇒ 192.168.56.102
msf6 auxiliary(scanner/ssh/ssh_login) > set user_file /home/aj/Desktop/Users.txt
user_file ⇒ /home/aj/Desktop/Users.txt
msf6 auxiliary(scanner/ssh/ssh_login) > set pass_file /home/aj/Desktop/Password.txt
pass_file ⇒ /home/aj/Desktop/Password.txt
msf6 auxiliary(scanner/ssh/ssh_login) > exploit

[*] 192.168.56.102:22 - Starting bruteforce
[+] 192.168.56.102:22 - Success: 'aj:lily' 'uid=1000(aj) gid=1000(aj) groups=1000(aj),4(adm),24(cdrom),27(s
udo),30(dip),46(plugdev),101(lxd) Linux ajnatsu 6.8.0-35-generic #35-Ubuntu SMP PREEMPT_DYNAMIC Tue May 21
07:52:29 UTC 2024 aarch64 aarch64 aarch64 GNU/Linux '
[*] SSH session 1 opened (192.168.56.101:40985 → 192.168.56.102:22) at 2024-07-29 13:53:53 +0900
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/ssh/ssh_login) > 
```

Figure 10: Brute Force Attack Successfully conducted using metasploit


original_brutefore.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
tcp.port == 22 && tcp.flags.push == 1

No.   Time          Source          Destination     Protocol Length Info
  539 101.918854715 192.168.56.101  192.168.56.102  SSHv2    1506 Client: Key Exchange Init
  540 101.920174507 192.168.56.102  192.168.56.101  SSHv2    1186 Server: Key Exchange Init
  541 101.930762924 192.168.56.101  192.168.56.102  SSHv2     218 Client: Elliptic Curve Diffie-Hellman Key Exchange
  542 101.943827799 192.168.56.102  192.168.56.101  SSHv2     378 Server: Elliptic Curve Diffie-Hellman Key Exchange
  543 101.951071215 192.168.56.101  192.168.56.102  SSHv2      90 Client: New Keys
  545 101.996516632 192.168.56.101  192.168.56.102  SSHv2     166 Client:
  547 101.996708549 192.168.56.102  192.168.56.101  SSHv2     166 Server:
  548 101.998632674 192.168.56.101  192.168.56.102  SSHv2     198 Client:
  550 105.475138759 192.168.56.102  192.168.56.101  SSHv2     166 Server:
  556 105.485162259 192.168.56.101  192.168.56.102  SSHv2     107 Client: Protocol (SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ub
  559 105.487969509 192.168.56.102  192.168.56.101  SSHv2     106 Server: Protocol (SSH-2.0-OpenSSH_9.6p1 Ubuntu-3ub
  561 105.489900384 192.168.56.101  192.168.56.102  SSHv2    1506 Client: Key Exchange Init
  562 105.491420634 192.168.56.102  192.168.56.101  SSHv2    1186 Server: Key Exchange Init
  563 105.497742717 192.168.56.101  192.168.56.102  SSHv2     218 Client: Elliptic Curve Diffie-Hellman Key Exchange
  564 105.510634425 192.168.56.102  192.168.56.101  SSHv2     378 Server: Elliptic Curve Diffie-Hellman Key Exchange
  565 105.516820342 192.168.56.101  192.168.56.102  SSHv2      90 Client: New Keys
  567 105.562674759 192.168.56.101  192.168.56.102  SSHv2     166 Client:
  569 105.562846634 192.168.56.102  192.168.56.101  SSHv2     166 Server:
  570 105.564054384 192.168.56.101  192.168.56.102  SSHv2     198 Client:
  571 105.593476384 192.168.56.102  192.168.56.101  SSHv2     150 Server:
  572 105.599815842 192.168.56.101  192.168.56.102  SSHv2     166 Client:
  575 106.114820551 192.168.56.102  192.168.56.101  SSHv2     742 Server:
  577 106.160515342 192.168.56.102  192.168.56.101  SSHv2     166 Server:
  579 106.161610217 192.168.56.101  192.168.56.102  SSHv2     166 Client:
  581 106.163064426 192.168.56.102  192.168.56.101  SSHv2     234 Server:
  582 106.165540676 192.168.56.102  192.168.56.101  SSHv2     446 Server:

    Acknowledgment number (raw): 512551189      0000  7a 3a 4e 76 82 6d 52 3d  74 76 90 f6
    1000 .... = Header Length: 32 bytes (8)     0010  00 98 02 dc 40 00 40 06  45 68 c0 a8
  ▸ Flags: 0x018 (PSH, ACK)                     0020  38 66 91 65 00 16 7e 1e  2e 1f 1e 8c
    Window: 249                                 0030  00 f9 e7 76 00 00 01 01  08 0a a3 e3
    [Calculated window size: 31872]             0040  0f e5 00 00 20 9d cb  47 f3 fe b3

original_brutefore.pcapng          Packets: 840 · Displayed: 335 (39.9%)          Profile: Default
```

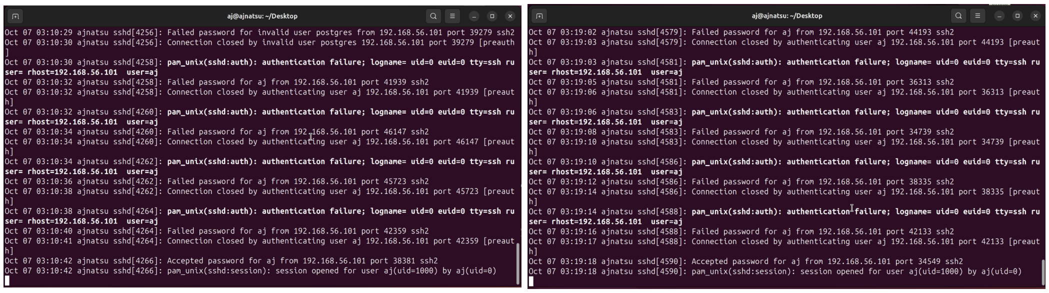
a) Original Brute Force

adversarial_bruteforce.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
tcp.port == 22 && tcp.flags.push == 1

No.   Time          Source          Destination     Protocol Length Info
  799 115.889692722 192.168.56.101  192.168.56.102  SSHv2    1506 Client: Key Exchange Init
  800 115.891189472 192.168.56.102  192.168.56.101  SSHv2    1186 Server: Key Exchange Init
  801 115.900507597 192.168.56.101  192.168.56.102  SSHv2     218 Client: Elliptic Curve Diffie-Hellman Ke
  802 115.913487055 192.168.56.102  192.168.56.101  SSHv2     378 Server: Elliptic Curve Diffie-Hellman Ke
  803 115.919681305 192.168.56.101  192.168.56.102  SSHv2      90 Client: New Keys
  805 115.962624972 192.168.56.101  192.168.56.102  SSHv2     166 Client:
  807 115.962718138 192.168.56.102  192.168.56.101  SSHv2     166 Server:
  808 115.963860638 192.168.56.101  192.168.56.102  SSHv2     198 Client:
  810 119.444405765 192.168.56.102  192.168.56.101  SSHv2     166 Server:
  817 119.456986765 192.168.56.101  192.168.56.102  SSHv2     107 Client: Protocol (SSH-2.0-OpenSSH_7.6p1
  819 119.458963390 192.168.56.102  192.168.56.101  SSHv2     106 Server: Protocol (SSH-2.0-OpenSSH_9.6p1
  821 119.462689973 192.168.56.102  192.168.56.101  SSHv2    1186 Server: Key Exchange Init
  823 119.463961973 192.168.56.101  192.168.56.102  SSHv2    1506 Client: Key Exchange Init
  825 119.504858557 192.168.56.101  192.168.56.102  SSHv2     218 Client: Elliptic Curve Diffie-Hellman Ke
  827 119.512564598 192.168.56.102  192.168.56.101  SSHv2     378 Server: Elliptic Curve Diffie-Hellman Ke
  828 119.518528182 192.168.56.101  192.168.56.102  SSHv2      90 Client: New Keys
  830 119.562256848 192.168.56.101  192.168.56.102  SSHv2     166 Client:
  832 119.562379765 192.168.56.102  192.168.56.101  SSHv2     166 Server:
  833 119.563500140 192.168.56.101  192.168.56.102  SSHv2     198 Client:
  834 119.577993848 192.168.56.102  192.168.56.101  SSHv2     150 Server:
  835 119.580862807 192.168.56.101  192.168.56.102  SSHv2     166 Client:
  837 119.882039473 192.168.56.102  192.168.56.101  SSHv2     742 Server:
  839 119.930194599 192.168.56.102  192.168.56.101  SSHv2     166 Server:
  841 119.931182682 192.168.56.101  192.168.56.102  SSHv2     166 Client:
  843 119.931893182 192.168.56.102  192.168.56.101  SSHv2     234 Server:
  844 119.934425682 192.168.56.102  192.168.56.101  SSHv2     446 Server:

    Acknowledgment Number: 1890    (relative ack number)    0000  52 3d 74 76 90 f0 7a 3a  4
    Acknowledgment number (raw): 2185820363                 0010  00 88 5f 89 40 00 40 06  e
    1000 .... = Header Length: 32 bytes (8)                 0020  38 65 00 16 91 81 a2 10  a
  ▸ Flags: 0x018 (PSH, ACK)                                 0030  00 f9 10 d1 00 00 01 01  0
    Window: 249                                             0040  ac 3b 00 00 00 10 2e 75  1

adversarial_bruteforce.pcapng          Packets: 906 · Displayed: 443 (48.9%)          Profile: Default
```

b) Adversarial Brute Force

Figure 11: Response Comparison of Original Brute Force and Adversarial Brute Force in Wireshark



```
Oct 07 03:10:29 ajnatsu sshd[4256]: Failed password for invalid user postgres from 192.168.56.101 port 39279 ssh2
Oct 07 03:10:30 ajnatsu sshd[4256]: Connection closed by invalid user postgres 192.168.56.101 port 39279 [preauth
]
Oct 07 03:10:30 ajnatsu sshd[4258]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ru
ser= rhost=192.168.56.101  user=aj
Oct 07 03:10:32 ajnatsu sshd[4258]: Failed password for aj from 192.168.56.101 port 41939 ssh2
Oct 07 03:10:32 ajnatsu sshd[4258]: Connection closed by authenticating user aj 192.168.56.101 port 41939 [preaut
h]
Oct 07 03:10:32 ajnatsu sshd[4260]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ru
ser= rhost=192.168.56.101  user=aj
Oct 07 03:10:34 ajnatsu sshd[4260]: Failed password for aj from 192.168.56.101 port 46147 ssh2
Oct 07 03:10:34 ajnatsu sshd[4260]: Connection closed by authenticating user aj 192.168.56.101 port 46147 [preaut
h]
Oct 07 03:10:34 ajnatsu sshd[4262]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ru
ser= rhost=192.168.56.101  user=aj
Oct 07 03:10:36 ajnatsu sshd[4262]: Failed password for aj from 192.168.56.101 port 45723 ssh2
Oct 07 03:10:38 ajnatsu sshd[4262]: Connection closed by authenticating user aj 192.168.56.101 port 45723 [preaut
h]
Oct 07 03:10:38 ajnatsu sshd[4264]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ru
ser= rhost=192.168.56.101  user=aj
Oct 07 03:10:40 ajnatsu sshd[4264]: Failed password for aj from 192.168.56.101 port 42359 ssh2
Oct 07 03:10:41 ajnatsu sshd[4264]: Connection closed by authenticating user aj 192.168.56.101 port 42359 [preaut
h]
Oct 07 03:10:42 ajnatsu sshd[4266]: Accepted password for aj from 192.168.56.101 port 38381 ssh2
Oct 07 03:10:42 ajnatsu sshd[4266]: pam_unix(sshd:session): session opened for user aj(uid=1000) by aj(uid=0)
```

a) Original Brute Force

```
Oct 07 03:19:02 ajnatsu sshd[4579]: Failed password for aj from 192.168.56.101 port 44193 ssh2
Oct 07 03:19:03 ajnatsu sshd[4579]: Connection closed by authenticating user aj 192.168.56.101 port 44193 [preaut
h]
Oct 07 03:19:03 ajnatsu sshd[4581]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ru
ser= rhost=192.168.56.101  user=aj
Oct 07 03:19:05 ajnatsu sshd[4581]: Failed password for aj from 192.168.56.101 port 36313 ssh2
Oct 07 03:19:06 ajnatsu sshd[4581]: Connection closed by authenticating user aj 192.168.56.101 port 36313 [preaut
h]
Oct 07 03:19:06 ajnatsu sshd[4583]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ru
ser= rhost=192.168.56.101  user=aj
Oct 07 03:19:08 ajnatsu sshd[4583]: Failed password for aj from 192.168.56.101 port 34739 ssh2
Oct 07 03:19:10 ajnatsu sshd[4583]: Connection closed by authenticating user aj 192.168.56.101 port 34739 [preaut
h]
Oct 07 03:19:10 ajnatsu sshd[4586]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ru
ser= rhost=192.168.56.101  user=aj
Oct 07 03:19:12 ajnatsu sshd[4586]: Failed password for aj from 192.168.56.101 port 38335 ssh2
Oct 07 03:19:14 ajnatsu sshd[4586]: Connection closed by authenticating user aj 192.168.56.101 port 38335 [preaut
h]
Oct 07 03:19:14 ajnatsu sshd[4588]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ru
ser= rhost=192.168.56.101  user=aj
Oct 07 03:19:16 ajnatsu sshd[4588]: Failed password for aj from 192.168.56.101 port 42133 ssh2
Oct 07 03:19:17 ajnatsu sshd[4588]: Connection closed by authenticating user aj 192.168.56.101 port 42133 [preaut
h]
Oct 07 03:19:18 ajnatsu sshd[4590]: Accepted password for aj from 192.168.56.101 port 34549 ssh2
Oct 07 03:19:18 ajnatsu sshd[4590]: pam_unix(sshd:session): session opened for user aj(uid=1000) by aj(uid=0)
```

b) Adversarial Brute Force

Figure 12: Response Comparison of Original Brute Force and Adversarial Brute Force in ssh login

C Ablution Study

This ablation study assesses the impact of substituting the pre-trained BERT model with a standard transformer architecture in the denoising process of DiffuPac. The focus is on assessing each model's ability to accurately reconstruct and generate adversarial packets that can effectively evade detection.

Non-BERT-based. This variant of DiffuPac employs a standard transformer architecture for the denoising process, as outlined in the DiffuSeq methodology (Gong et al. [2023]). Notably, this model is trained from scratch without leveraging any pre-trained weights, thus relying solely on the information provided during the training phase.

BERT-based. The existing DiffuPac configuration, which incorporates a pre-trained BERT model for the denoising process, was utilized. This setup takes advantage of BERT's pre-trained weights, enabling enhanced semantic understanding and contextual relevance during packet reconstruction.

Both models were trained under identical conditions and evaluated based on their success in generating adversarial packets that evade detection. The results, as depicted in Table 8 , highlight the comparative effectiveness of each model.

As shown in Table 8, the BERT-based DiffuPac significantly outperforms the non-BERT variant, demonstrating superior ability to reconstruct packets that blend seamlessly into normal traffic. The BERT-based model excels in pairing malicious packets with normal packets that exhibit high contextual relevance. This strategic pairing allows the generated adversarial packets to blend seamlessly into normal traffic, thereby increasing their likelihood of evading detection. In contrast, the non-BERT-based model, which lacks this pre-trained knowledge, resorts to random matching of malicious and normal packets, leading to a noticeable reduction in the effectiveness of the generated adversarial traffic.

An intriguing aspect of DiffuPac is its dual utilization of the pre-trained BERT model: first, as the denoising engine during the packet reconstruction process, and second, as a critical tool for identifying contextually relevant packet pairs. To the best of our knowledge, this dual application of a

Table 8: Comparative analysis of attack detection and evasion rates.

(a) Botnet

Feature Extractor	Classifier	Detection			Evasive (*MER*)	
		P	R	F1	Non-BERT-based	BERT-Based
CIC FLowMeter	KitNET	0.84	0.94	0.92	38.93%	**46.48%**
	DT	0.79	0.91	0.82	50.24%	**60.98%**
	IF	0.99	0.90	0.95	39.18%	**49.71%**
	MLP	0.92	0.84	0.86	53.85%	**63.05%**
	SVM	0.99	0.92	0.95	52.31%	**64.49%**
	LR	0.84	0.91	0.89	35.02%	**42.08%**
AfterImage	KitNET	0.96	0.90	0.94	69.15%	**74.46%**
	DT	0.80	0.90	0.82	65.77%	**72.13%**
	IF	0.99	0.92	0.94	46.83%	**52.79%**
	MLP	0.96	0.96	0.95	52.04%	**64.92%**
	SVM	0.99	0.90	0.94	60.19%	**69.19%**
	LR	0.97	0.90	0.93	53.98%	**58.98%**

(b) MITM

Feature Extractor	Classifier	Detection			Evasive (*MER*)	
		P	R	F1	Non-BERT-based	BERT-Based
CIC FLowMeter	KitNET	0.92	0.94	0.91	45.24%	**55.87%**
	DT	0.74	0.79	0.76	47.83%	**64.12%**
	IF	0.99	0.92	0.94	36.91%	**52.99%**
	MLP	0.77	0.72	0.74	54.98%	**73.21%**
	SVM	0.74	0.79	0.78	56.10%	**60.46%**
	LR	0.73	0.78	0.72	35.27%	**47.48%**
AfterImage	KitNET	0.94	0.96	0.93	52.55%	**58.48%**
	DT	0.75	0.89	0.84	61.47%	**70.04%**
	IF	0.81	0.83	0.86	31.33%	**45.71%**
	MLP	0.92	0.90	0.93	64.17%	**71.45%**
	SVM	0.99	0.90	0.94	56.44%	**66.53%**
	LR	0.91	0.94	0.90	44.83%	**52.38 %**

pre-trained model within a diffusion framework is unprecedented, further highlighting the innovation and effectiveness of our approach.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state the key contributions of our paper, including the development of DiffuPac, the integration of a pre-trained BERT model with diffusion model, and the novel approach to generating adversarial packets. These claims are substantiated by both theoretical analysis and experimental results presented in the paper. The scope of our research, including the focus on real-world constraints and the effectiveness of the proposed model against advanced NIDS, is accurately reflected in the paper.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We have included the limitations in the main paper.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: The proofs for all theorems and formulas are clearly provided both in the main paper and supplemental material.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper provides a detailed explanation of the model architecture and the experimental setups. This includes a thorough description of the pre-trained BERT model integration, the diffusion model, and the specific configurations used in the experiments. Additionally, all hyperparameters, dataset details, and evaluation metrics are disclosed to enable replication of the results.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: Due to ethical concerns, we have decided not to publicly share the data and code. Additionally, much of the code is based on existing codebases from other sources. However, we have provided a full and detailed explanation of the model architecture and its implementation in the paper and supplemental material. This includes comprehensive descriptions of the experimental setups, hyperparameters, and step-by-step instructions to facilitate reproducibility.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We have provided the full detailed explanation regarding the training process and the experimental details both in the main paper and supplemental material.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: The experiments conducted in this paper primarily focus on evaluating the success of generating adversarial packets. Metrics such as evasion rate, packet field analysis, and probabilistic differences between original and adversarial packets are used to assess the effectiveness of the proposed method.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: We have detailed the types of computational resources used and the time of execution for BERT and diffusion model in the supplemental materials.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

Answer: [Yes]

Justification: We have read the Code Of Ethics and strictly follow it.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We have discussed the positive societal impacts and negative societal impacts in the main paper.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [Yes]

    Justification: Due to ethical concerns of our paper, we have implemented stringent measures.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

Justification: All existing assets used in this paper, including code, data, and models, have been properly credited to their original creators. We have cited the original papers that produced these assets and included URLs where applicable.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The paper introduces a new model, DiffuPac, which integrates a pre-trained BERT with a diffusion model to generate adversarial packets. Detailed documentation for DiffuPac is provided, including descriptions of the model architecture, training process, implementation details, and usage guidelines. This documentation is included in the main paper and supplemental materials.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification:

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification:

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.