Realtime Robust Malicious Traffic Detection via Frequency Domain Analysis

Chuanpu Fu[1], Qi Li[2,3], Meng Shen[4], and Ke Xu[1,3,5]
[1]Department of Computer Science and Technology, Tsinghua University, Beijing, China
[2]Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China
[3]Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua University, Beijing, China
[4]School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing, China
[5]Peng Cheng Laboratory, China
{fcp20@mails., qli01@, xuke@}tsinghua.edu.cn, shenmeng@bit.edu.cn

arXiv:2106.14707v3 [cs.CR] 16 Sep 2021

ABSTRACT

Machine learning (ML) based malicious traffic detection is an emerging security paradigm, particularly for zero-day attack detection, which is complementary to existing rule based detection. However, the existing ML based detection achieves low detection accuracy and low throughput incurred by inefficient traffic features extraction. Thus, they cannot detect attacks in realtime, especially in high throughput networks. Particularly, these detection systems similar to the existing rule based detection can be easily evaded by sophisticated attacks. To this end, we propose Whisper, a realtime ML based malicious traffic detection system that achieves both high accuracy and high throughput by utilizing frequency domain features. It utilizes sequential information represented by the frequency domain features to achieve bounded information loss, which ensures high detection accuracy, and meanwhile constrains the scale of features to achieve high detection throughput. In particular, attackers cannot easily interfere with the frequency domain features and thus Whisper is robust against various evasion attacks. Our experiments with 42 types of attacks demonstrate that, compared with the state-of-the-art systems, Whisper can accurately detect various sophisticated and stealthy attacks, achieving at most 18.36% improvement of AUC, while achieving two orders of magnitude throughput. Even under various evasion attacks, Whisper is still able to maintain around 90% detection accuracy.

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**.

KEYWORDS

Machine learning; malicious traffic detection; frequency domain

ACM Reference Format:
Chuanpu Fu, Qi Li, Meng Shen, and Ke Xu. 2021. Realtime Robust Malicious Traffic Detection via Frequency Domain Analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3460120.3484585

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8454-4/21/11…$15.00
https://doi.org/10.1145/3460120.3484585

1 INTRODUCTION

Traditional malicious traffic detection identifies malicious traffic by analyzing the features of traffic according to preconfigured rules, which aims to protect legitimate Internet users from network attacks [29, 47]. However, the rule-base detection is unable to detect zero-day attacks [8, 12, 22, 65] though it can achieve high detection accuracy and detection throughput in high bandwidth networks, e.g., in Internet backbone networks.

As a promising security paradigm, machine learning based malicious traffic detection has been developed, particularly as a complement of the traditional fixed rule based methods (i.e., signature based NIDS) [6, 29, 35, 47]. Table 1 summarizes and compares rule based and typical machine learning based detection methods. Compared with rule based methods, machine learning based methods can effectively identify zero-day malicious traffic [12, 22]. Unfortunately, due to the processing overhead of machine learning algorithms, existing detection methods achieve low detection throughput and are unable to process high-rate traffic. As a result, most of these methods can only be deployed offline [2, 4, 5, 15, 28, 49] so that they cannot realize realtime detection, particularly in high performance networks (e.g., in 10 Gigabit networks) [42, 77, 78].

Meanwhile, attackers can easily interfere with and evade these methods by injecting noises, e.g., packets generated by benign applications, into attack traffic. Packet-level detection [42, 53, 68] that analyzes per-packet feature sequences is unable to achieve robust detection. Actually, even in the absence of the evasion attacks, the packet-level detection is unable to detect sophisticated zero-day attacks. Traditional flow-level methods [4, 28, 49, 77] detecting attacks by analyzing flow-level statistics incur significant detection latency. Moreover, evasion attacks can easily bypass the traditional flow-level detection that uses coarse-grained flow-level statistics [14, 63]. Thus, realtime robust machine learning based detection that is ready for real deployment is still missing.

In this paper, we develop Whisper that aims to realize realtime robust malicious traffic detection by utilizing machine learning algorithms. Whisper effectively extracts and analyzes the sequential information of network traffic by frequency domain analysis [51], which extracts traffic features with low information loss. Especially, the frequency domain features of traffic can efficiently represent various packet ordering patterns of traffic with low feature redundancy. Frequency domain feature analysis with low information

https://t.me/learningnets

Table 1: Comparing the Existing Malicious Traffic Detection Methods

Category of Detection Systems		Feature Extraction Methods	Zero-Day Detection	High Accuracy	Robust Detection	Realtime Detection	High Throughput	Task Agnostic
Rule based		Preconfigured fix rules [6, 29, 35]	×	✓	×	✓	✓	×
ML based	Packet-level	Packet header fields [53]	✓	✓	×	✓	×	✓
		Context statistics [42]	✓	✓	×	✓	×	✓
		Payload statistics [68]	✓	✓	×	×	×	✓
	Flow-level	Flow-level statistics [5, 37, 77]	✓	×	×	×	✓	×
		Application usage statistics [4, 28, 49]	✓	✓	×[1]	×	×	×
		Frequency domain features, Whisper	✓	✓	✓	✓	✓	✓

[1] Bartos *et al.* [4] only considered evasion strategies for malicious Web traffic.

loss enables accurate and robust detection, while low feature redundancy ensures high throughput traffic detection. In particular, since the frequency domain features represent fine-grained sequential information of the packet sequences, which are not disturbed by the injected noise packets, Whisper can achieve robust detection. However, it is non-trivial to extract and analyze the frequency domain features from traffic because of the large-scale, complicated, and dynamic patterns of traffic [14, 63].

To effectively perform frequency domain traffic feature analysis, we develop a three-step frequency domain feature extraction. First, we encode per-packet feature sequences as vectors, which reduces the data scale and the overhead of subsequent processing. Second, we segment the encoded vectors and perform Discrete Fourier Transformation (DFT) [51] on each frame, which aims to extract the sequential information of traffic. It allows statistical machine learning algorithms to easily learn the patterns. Third, we perform logarithmic transformation on the modulus of the frequency domain representation produced by DFT, which prevents float point overflows incurred by the numerical instability issue [23] during the training of machine learning.

Furthermore, we propose an automatic parameter selection module to select the encoding vector for efficient packet feature encoding. To achieve this, we formulate the per-packet feature encoding as a constrained optimization problem to minimize mutual interference of the per-packet features during frequency domain feature analysis. We transform the original problem into an equivalent SMT problem and solve the problem by an SMT solver. It ensures the detection accuracy by choosing vectors, while effectively reducing manual efforts of selecting encoding vectors. We utilize statistical machine learning to cluster the patterns according to the frequency domain features. Due to the rich feature presentation and lightweight machine learning, Whisper finally realizes realtime detection of malicious traffic in high throughput networks.

We theoretically prove that Whisper is more efficient than packet-level and traditional flow-level detection methods. We conduct a theoretical analysis to prove that the frequency domain features ensure bounded information loss, which lays the foundation for robust detection of Whisper. We develop a *traffic feature differential entropy model*, a theoretical framework to measure information loss of feature extraction from traffic. First, we prove the information loss in processing packet sequences in the existing flow-level methods, which further demonstrates that it cannot accurately extract features. Second, we prove that Whisper maintains the information loss in the flow-level methods and validate that the frequency domain features are more efficient. Third, we prove that Whisper effectively reduces feature redundancy by the decrease in the data scale of features.

We prototype Whisper with Intel's Data Plane Development Kit (DPDK) [26]. To extensively evaluate the performance of the Whisper prototype, we replay 42 malicious traffic datasets with the high throughput backbone network traffic. Besides the typical attacks, we collect and replay 36 new malicious traffic datasets including: (i) more stealthy attacks, e.g., low-rate TCP DoS attacks [25, 31, 33] and stealthy network scanning [38]; (ii) complicated multi-stage attacks, e.g., TCP side-channel attacks [10, 11, 17] and TLS padding oracle attacks [67]; (iii) evasion attacks, i.e., attackers inject different types of noise packets (i.e., packets generated by various benign applications) in attack traffic to evade detection. According to our experimental results, we validate that Whisper can detect the different types of attacks with AUC ranging between 0.891 and 0.999 while achieving 1,310,000 PPS, i.e., two orders of magnitude throughput more than the state-of-the-art methods. Particularly, Whisper can detect various evasion attacks with 35% improvement of AUC over the state-of-the-art methods. Furthermore, Whisper achieves realtime detection with bounded 0.06 second detection latency in high throughput networks.

In summary, the contributions of our paper are five-fold:

- We present Whisper, a novel malicious traffic detection system by utilizing frequency domain analysis, which is the first system built upon machine learning achieving realtime and robust detection in high throughput networks.
- We perform frequency domain feature analysis to extract the sequential information of traffic, which lays the foundation for the detection accuracy, robustness, and high throughput of Whisper.
- We develop automatic encoding vector selection for Whisper to reduce manual efforts for parameter selection, which ensures the detection accuracy while avoiding manual parameter setting.
- We develop a theoretical analysis framework to prove the properties of Whisper.
- We prototype Whisper with Intel DPDK and use the experiments with different types of replayed attack traffic to validate the performance of Whisper.

https://t.me/learningnets

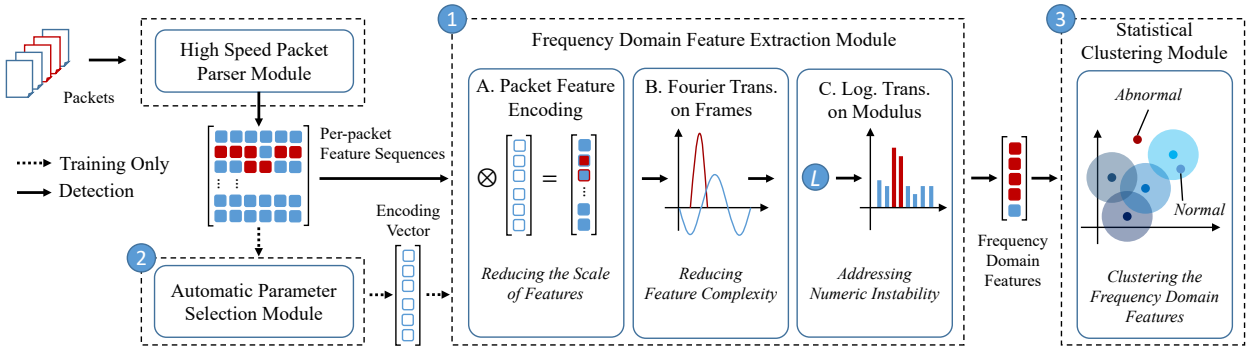


Figure 1: High-level design of Whisper.

The rest of the paper is organized as follows: Section 2 introduces the threat model and the design goals of Whisper. Section 3 presents the high-level design of Whisper. In Section 4, we present the design details of Whisper. In Section 5, we conduct a theoretical analysis. In Section 6, we experimentally evaluate the performances of Whisper. Section 7 reviews related works and Section 8 concludes this paper.

2 THREAT MODEL AND DESIGN GOALS

2.1 Threat Model

We aim to develop a malicious traffic detection system as a plug-in module of middlebox. The middlebox forwards the replicated traffic to the detection system through port mirroring, which is similar to Cisco SPAN [13]. Thus, the detection system does not interfere with benign traffic forwarding. We assume that the detection system does not have any prior knowledge on threats, which means that it should be able to deal with zero-day attacks [12, 42, 65]. Note that, we do not consider defenses against the attacks detected by Whisper and can deploy existing malicious traffic defenses [70, 71, 74] to throttle the detected traffic.

The developed detection system should be able to determine whether traffic passing through the middlebox is benign or malicious by monitoring ongoing traffic. We emphasize that the malicious traffic detection is fully different from traffic classification [48, 58, 61, 66] that aims to classify whether traffic is generated by a certain network application or a certain user. We do not consider detecting passive attacks that do not cause obvious traffic variance, e.g., eavesdropping attacks and intercept attacks [45, 46].

2.2 Design Goals

In this paper, we aim to develop a realtime robust malicious traffic detection system, which achieves high detection accuracy and task-agnostic detection. Particularly, the system should achieve the following two goals, which are not well addressed in the literature.

Robust Accurate Detection. The system should be able to detect various zero-day attacks. Especially, it can capture different evasion attacks, which try to evade detection by deliberately injecting noise packets, i.e., using various packets generated by benign applications, into the attack traffic.

Realtime Detection with High Throughput. The system should be able to be deployed in high throughput networks, e.g., a 10 Gigabit Ethernet, while incurring low detection latency.

3 OVERVIEW OF WHISPER

In this section, we present our malicious traffic detection system, Whisper. Whisper achieves high performance detection by encoding per-packet feature sequences as vectors to reduce the overhead of subsequent feature processing. Meanwhile, it extracts the sequential information of traffic via frequency domain to ensure detection accuracy. In particular, since the frequency domain features represent fine-grained sequential information of the packet sequences, which are not disturbed by the injected noise packets, Whisper can achieve robust detection. Figure 1 shows the overview of Whisper.

High Speed Packet Parser Module. High speed packet parser module extracts per-packet features, e.g., the packet length and arriving time interval, at high speed to ensure the processing efficiency in both training and detection phases. This module provides the per-packet feature sequences to the feature extraction module for extracting the frequency domain features and the automatic parameter selection module for determining the encoding vector. Note that, this module dose not extract specific application related features and thus Whisper achieves task agnostic detection.

Frequency Features Extraction Module. In both training and detection phases, this module extracts the frequency domain features from the per-packet feature sequences. This module periodically polls the required information from the high speed packet parser module with a fixed time interval. After acquiring the extracted per-packet features, it encodes the per-packet feature sequences as vectors and extracts the sequential information via frequency domain. These features with low redundancy are provided for the statistical clustering module. However, it is difficult to extract the frequency domain features of traffic in high throughput networks in realtime because of the various complicated, irregular, and dynamic flow patterns [14, 63]. We cannot apply deep learning models, e.g., recurrent neural networks, to extract features due to their long processing latency though they can extract more richer features for detection. We will present the details of this module in Section 4.1.

Automatic Parameter Selection Module. This module calculates the encoding vector for the feature extraction module. We decide the encoding vector by solving a constrained optimization problem that reduces the mutual interference of different per-packet features. In the training phase, this module acquires the per-packet feature sequences and solves an equivalent Satisfiability Modulo Theories (SMT) problem to approximate the optimal solution of the original problem. By enabling automatic parameter selection,

https://t.me/learningnets

we significantly reduce the manual efforts for parameter selection. Therefore, we can fix and accurately set the encoding vector in the detection phase. We will describe the details of the module in Section 4.2.

Statistical Clustering Module. In this module, we utilize a light-weight statistical clustering algorithm to learn the patterns of the frequency domain features from the feature extraction module. In the training phase, this module calculates the clustering centers of the frequency domain features of benign traffic and the averaged training loss. In the detection phase, this module calculates the distances between the frequency domain features and the clustering centers. Whisper detects traffic as malicious if the distances are significantly larger than the training loss. We will elaborate on the statistical clustering based detection in Section 4.3.

4 DESIGN DETAILS

In this section, we present the design details of Whisper, i.e., the design of three main modules in Whisper.

4.1 Frequency Feature Extraction Module

In this module, we extract the frequency domain features from high speed traffic. We acquire the per-packet features of N packets from the same flow by polling the high speed packet parser module. We use the mathematical representation similar to Bartos *et al.* [4] to denote the features. We use $s^{(i)}$ and M to indicate the i^{th} per-packet feature and the number of per-packet features, respectively. Matrix S denotes the per-packet features of all packets, where s_{ik} is defined as i^{th} packet's k^{th} property:

$$\mathrm{S} = [s^{(1)}, \ldots, s^{(i)}, \ldots, s^{(M)}] = \begin{bmatrix} s_{11} & \cdots & s_{1M} \\ \vdots & \ddots & \vdots \\ s_{N1} & \cdots & s_{NM} \end{bmatrix}. \tag{1}$$

Packet Feature Encoding. We perform a linear transformation w on S to encode the features of a packet to a real number v_i. v denotes the vector representation of traffic:

$$v = \mathrm{S}w = [v_1, \ldots, v_i, \ldots, v_N]^{\mathrm{T}}, \quad v_i = \sum_{k=1}^{M} s_{ik} w_k. \tag{2}$$

The feature encoding reduces the scale of features, which significantly reduces the processing overhead of Whisper. In Section 4.2, we will describe how Whisper automatically selects parameters for the encoding vector w.

Vector Framing. Now we segment the vector representation with the step length of W_{seg}. The goal of segmentation is to reduce the complexity of the frequency domain features by constraining the long-term dependence between packets. If the frames are excessively long, the frequency domain features will become too complex to learn in the statistical learning module. N_f denotes the number of the frames. We obtain the following equations:

$$f_i = v[\![(i-1) \times W_{\text{seg}} : i \times W_{\text{seg}}]\!] \qquad (1 \le i \le N_f), \tag{3}$$

$$N_f = \left\lfloor \frac{N}{W_{\text{seg}}} \right\rfloor. \tag{4}$$

Discrete Fourier Transformation. In the next step, we perform the Discrete Fourier Transformation (DFT) on each frame f_i to extract the sequential information via frequency domain and reduce the information loss incurred by the flow-level methods. We can acquire the frequency features of each frame as follows:[1]

$$F_i = \mathscr{F}(f_i) \quad (1 \le i \le N_f), \tag{5}$$

$$F_{ik} = \sum_{n=1}^{W_{\text{seg}}} f_{in} e^{-j\frac{2\pi(n-1)(k-1)}{W_{\text{seg}}}} \qquad (1 \le k \le W_{\text{seg}}), \tag{6}$$

where F_{ik} is a frequency component of i^{th} frame with the frequency of $2\pi(k-1)/W_{\text{seg}}$. Note that, all frequency features output by DFT are vectors with complex numbers, which cannot be used directly as the input for machine learning algorithms.

Calculating the Modulus of Complex Numbers. We transform the complex numbers to real numbers by calculating the modulus for the frequency domain representation. For simplicity, we transform F_{ik} to a coordinate plane representation:

$$F_{ik} = a_{ik} + jb_{ik}, \tag{7}$$

$$\begin{cases} a_{ik} &= \sum\limits_{n=1}^{W_{\text{seg}}} f_{in} \cos\frac{2\pi(n-1)(k-1)}{W_{\text{seg}}} \\ b_{ik} &= \sum\limits_{n=1}^{W_{\text{seg}}} -f_{in} \sin\frac{2\pi(n-1)(k-1)}{W_{\text{seg}}}. \end{cases} \tag{8}$$

We calculate the modulus for F_{ik} as p_{ik} in (9). For the i^{th} frame, we select the first half of the modulus as vector P_i. Because the transformation results of DFT are conjugate, the first half and the second half are symmetrical. Thus, we can obtain:

$$p_{ik} = a_{ik}^2 + b_{ik}^2 \quad (1 \le k \le W_{\text{seg}}), \tag{9}$$

$$P_i = [p_{i1}, \ldots, p_{iK_f}]^{\mathrm{T}} \quad (K_f = \left\lfloor \frac{W_{\text{seg}}}{2} \right\rfloor + 1), \tag{10}$$

$$F_{ik} = F^*_{i(W_{\text{seg}}-k)} \quad \Rightarrow \quad p_{ik} = p_{i(W_{\text{seg}}-k)}. \tag{11}$$

Logarithmic Transformation. To make the frequency domain features to be numerically stable [23] and prevent float point overflow during the machine learning model training, we perform a logarithmic transformation on P_i, and use constant C to adjust the range of the frequency domain features:

$$R_i = \frac{\ln(P_i + 1)}{C} \quad (1 \le i \le N_f), \tag{12}$$

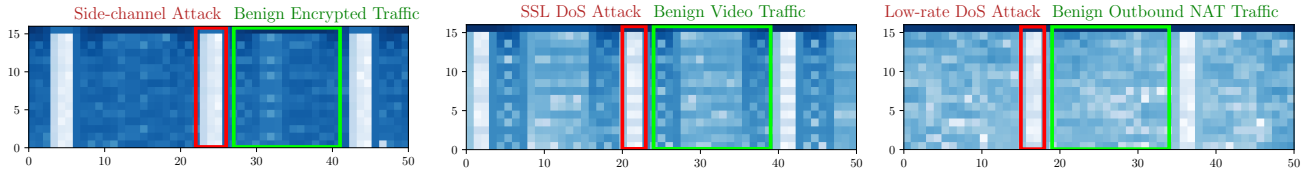
$$\mathrm{R}_{K_f \times N_f} = [R_1, \ldots, R_i, \ldots, R_{N_f}]. \tag{13}$$

As the output of the features extraction module, the i^{th} column component of R is the frequency domain features of the i^{th} frame. Matrix R is the input for the statistical clustering module.

Take an example, we collect three types of benign traffic (90%) mixed with the malicious traffic (10%) in Wide Area Network (WAN). We select 1500 continuous packets ($N = 1500$) from each type of traffic and extract three per-packet features ($M = 3$) including the packet length, the protocol type, and the arriving time interval. We fix the framing length $W_{\text{seg}} = 30$. Therefore, $N_f = 50$ and $K_f = 16$. Then we perform a min-max normalization operation on the frequency domain features R and map the results to the RGB space. We visualize the frequency domain features that are similar to the Spectrogram in speech recognition [1]. As shown in Figure 2,

[1] j denotes an imaginary number.

https://t.me/learningnets



(a) Benign TLS traffic and side-channel attack traffic (b) Benign UDP traffic and SSL DoS traffic (c) Outbound NAT traffic and LowRate TCP DoS traffic

Figure 2: We map the frequency domain features, which are extracted from the traffic with three types of typical attacks, to the RGB space, and observe that a small number of malicious packets incur significant changes in the frequency domain features.

we observe that the area associated with the frequency domain features of the malicious traffic is significantly lighter than that of the benign traffic.

4.2 Automatic Parameters Selection Module

Now we determine the encoding vector w for the feature extraction module that uses w to encode the per-packet feature sequences and acquires the vector representation of the traffic. In general, we formulate the encoding vector selection problem as a constrained optimization problem, and transform the original problem into an equivalent SMT problem. We approximate the optimal solution of the original problem through solving the SMT problem.

We assume that we can find a set of continuous functions to describe the changes of each kind of the per-packet feature $s^{(i)}$. Thus, we consider all obtained per-packet features are the samples of the continuous functions, which are denoted as $h_i(t)$ $(1 \le i \le M)$. We need to find a vector w to amplify and superpose all these functions. Our key optimization objective is to minimize mutual interference and bound the overall range when superposing the functions. We can first bound the range of encoding vector w and the range of the superposition function in the following:

$$W_{min} \le w_i \le W_{max} \quad (1 \le i \le M), \tag{14}$$

$$\sum_{i=1}^{M} w_i h_i(t) \le B, \tag{15}$$

where W_{min}, W_{max}, B are constants. We constrain the order preserving properties of the functions to ensure that different types of per-packet features do not interfere with each other when the feature extraction module performs packet encoding:

$$w_i h_i(t) \le w_{i+1} h_{i+1}(t) \quad (1 \le i \le M-1). \tag{16}$$

Second, we optimize w to maximize the distances between the functions so that we can minimize the mutual interference of the per-packet features and bound the ranges of all the functions. Therefore, under the constrains of (14) (15) (16), we obtain the optimization object:

$$\hat{w} = \arg\max \int_0^{+\infty} w_M h_M(t) - w_1 h_1(t) \mathrm{d}t \quad - \sum_{i=2}^{M-1} \int_0^{+\infty} |2w_i h_i(t) - w_{i+1} h_{i+1}(t) - w_{i-1} h_{i-1}(t)| \mathrm{d}t. \tag{17}$$

In practice, we cannot determine the convexity of the optimization object because the closed-form representations of $h_i(t)$ are not available. Thus, we reform the origin constrained optimization problem to a Satisfiability Modulo Theories (SMT) problem (19) with optimization object (18) to approximate the optimal solution of (17). For the i^{th} per-packet feature, we perform a min-max normalization on s_i and use n_i to indicate the normalized vector. We list constrains (19). And we obtain the satisfied (SAT) solutions of the SMT problem and maximize the following objective:

$$\widetilde{w} = \arg\max \sum_{k=1}^{N} w_M n_{Mk} - w_1 n_{1k} \quad - \sum_{i=2}^{M-1} 2w_i n_{ik} - w_{i-1} n_{(i-1)k} - w_{i+1} n_{(i+1)k}, \tag{18}$$

subjects to:

$$\begin{cases} w_i & \in & [W_{min}, W_{max}] \\ \sum_{i=1}^{M} w_i n_{ik} & \le & B \\ w_i n_{ik} & \le & w_{i+1} n_{(i+1)k} \\ 2w_i n_{ik} & \le & w_{i-1} n_{(i-1)k} + w_{i+1} n_{(i+1)k}. \end{cases} \tag{19}$$

Note that, we reform the absolute value operation in the optimization object (17) into constrains (19) because most SMT solvers do not support absolute value operations.

4.3 Statistical Clustering Module

Now we utilize the statistical clustering algorithm to learn the patterns of the frequency domain features obtained from the feature extraction module with the selected parameters. We train the statistical clustering algorithm with only benign traffic. In the training phase, this module calculates the clustering centers of the frequency domain features and the averaged training loss. In order to improve the robustness of Whisper and reduce false positive caused by the extreme values, we segment the frequency domain feature matrix R with a sampling window of length W_{win}. We use N_t to denote the number of samples and l to denote the start points. We average the sampling window on the dimension of the feature sequence and use r_i to indicate the input of the clustering algorithm. We can obtain:

$$l = iW_{win} \quad (0 \le i < N_t), \quad N_t = \left\lfloor \frac{N_f}{W_{win}} \right\rfloor, \tag{20}$$

$$r_i = \mathrm{mean}(\mathrm{R}[\![l : l + W_{win}]\!]). \tag{21}$$

We perform the statistical clustering algorithm and acquire all clustering centers to represent the benign traffic patterns. We use C_k to denote the K_C clustering centers, where $(1 \le k \le K_C)$, and then calculate the averaged training loss. For each r_i, we find the closest clustering center as $\hat{C}_i$ and we take averaged L2-norm as

https://t.me/learningnets

the training loss:

$$\hat{C}_i = \arg\min_{C_k} \|C_k - r_i\|_2 \quad (1 \le i \le N_t), \tag{22}$$

$$\text{train\_loss} = \frac{1}{N_t} \sum_{i=1}^{N_t} \left\| r_i - \hat{C}_i \right\|_2 . \tag{23}$$

In the detection phase, this module calculates the distances between the frequency domain features of traffic and the clustering centers. For each given frequency domain feature, we sample N_t segments on R with length W_{win}, which is the same as the training phase. We can find the closest clustering center $\hat{C}_i$ as an estimate of r_i. We calculate the L2-norm as the estimation error:

$$\text{loss}_i = \min(\|r_i - C_k\|_2) \quad (1 \le k \le K_C). \tag{24}$$

If the estimation error $\text{loss}_i \ge (\phi \times \text{train\_loss})$, we can conclude that the statistical clustering algorithm cannot understand the frequency domain features of the traffic, which means the traffic is malicious.

5 THEORETICAL ANALYSIS

In this section, we conduct a theoretical analysis to prove that Whisper achieves lower information loss in feature extraction than the packet-level and the traditional flow-level methods, which ensures that Whisper extracts traffic features accurately. Due to the page limitations, all proofs can be found in Appendix A - D. Moreover, we analyze the scale of the frequency domain features and the algorithmic complexity of Whisper.

5.1 Information Loss in Whisper

Traffic Feature Differential Entropy Model. First, we develop the traffic feature differential entropy model, a theoretical analysis framework that *evaluates the efficiency of traffic features by analyzing the information loss incurred by feature extractions* from an information theory perspective [39]. The framework aims to (i) model an observable packet-level feature as a stochastic process and observed features extracted from ongoing packets as the state random variables of the process; (ii) model feature extraction methods as algebraic transformations of the state random variables; (iii) evaluate the efficiency of the features by measuring the information loss during the transformations.

We model a particular type of packet-level feature (e.g., the packet length, and the time interval) as a discrete time stochastic process $\mathcal{S}$, which is used to model traffic feature extraction by different detection methods. We use a random variable vector $\vec{s} = [s_1, s_2, \ldots, s_N]$ to denote a packet-level feature sequence extracted from N continuous packets, i.e., N random variables from $\mathcal{S}$. f indicates a feature extraction function that transforms the original features $\vec{s}$ for the input of machine learning algorithms. According to Table 1, in the packet-level methods, f outputs the per-packet features sequence $\vec{s}$ directly. In the traditional flow-level methods, f calculates a statistic of $\vec{s}$. In Whisper, f calculates the frequency domain features of $\vec{s}$. We assume that $\mathcal{S}$ is a discrete time Gaussian process, i.e., $\mathcal{S} \sim \text{GP}(u(i), \Sigma(i, j))$. For simplicity, we mark $\Sigma(i, i)$ as $\sigma(i)$. We assume $\mathcal{S}$ is an independent process and then we can obtain the covariance function of $\mathcal{S}$, i.e., $\kappa(x_i, x_j) = \sigma(i)\delta(i, j)$. p_i denotes the probability density function of s_i. We use differential entropy [39] to measure the information in the features using the unit of *nat*:

$$\mathcal{H}(s_i) = -\int_{-\infty}^{+\infty} p_i(s) \ln p_i(s) \mathrm{d}s = \ln K\sigma(i), \tag{25}$$

where $K = \sqrt{2\pi e}$. We assume that the variance of each s_i is large enough to ensure the significant change because a kind of stable packet-level feature is meaningless to be extracted and analyzed. Thus, we establish non-negative differential entropy assumption, i.e., $\sigma(i) \ge K^{-1}$ to ensure $\mathcal{H}(s_i) \ge 0$.

Analysis of Traditional Flow-level Detection Methods. We analyze the information loss in the feature extraction of the traditional flow-level methods. We consider three types of widely used statistical features in the traditional flow-level methods [5, 24, 37, 43, 77]: (i) min-max features, the feature extraction function f outputs the maximum or minimum value of $\vec{s}$ to extract flow-level features of traffic and produces the output for machine learning algorithms. (ii) average features, f calculates the average number of $\vec{s}$ to obtain the flow-level features. (iii) variance features, f calculates the variance of $\vec{s}$ for machine learning algorithms. We analyze the information loss when performing the statistical feature extraction function f. Based on the probability distribution of the state random variables and Equation (25), we obtain the information loss of flow-level statistical features in the traditional flow-level detection over the packet-level detection and have the following properties of the features above.

Theorem 1. *(The Lower Bound for Expected Information Loss of the Min-Max Features).* For the min-max statistical features, the lower bound of expected information loss is:

$$\mathrm{E}[\Delta\mathcal{H}_{\text{flow-minmax}}] \ge (N-1) \ln K \mathrm{E}[\sigma]. \tag{26}$$

Theorem 2. *(The Lower Bound for Expected Information Loss of the Average Features).* The lower bound for the expectation of information loss in the average features is:

$$\mathrm{E}[\Delta\mathcal{H}_{\text{flow-avg}}] \ge \ln \sqrt{N} K^{N-1} \mathrm{E}[\sigma]^{N-1}. \tag{27}$$

We can obtain that the equality of Theorem 1 and Theorem 2 holds iff the stochastic process $\mathcal{S}$ is strictly stationary.

Theorem 3. *(The Lower Bound and Upper Bound for the Information Loss of the Average Features).* For the average features, the upper and lower bounds of the information loss in the metric of differential entropy is:

$$\ln N \le \Delta\mathcal{H}_{\text{flow-avg}} \le \ln \sqrt{N} K^{N-1} Q(\sigma)^{N-1}, \tag{28}$$

where $Q(\sigma)$ is the square mean of the variances of the per-packet features sequence $\vec{s}$.

Theorem 4. *(The Information Loss of the Variance Features).* When the Gaussian process $\mathcal{S}$ is strictly stationary with zero mean, i.e., $u(i) = 0$ and $\sigma(i) = \sigma$, for the variance features, an estimate of the information loss is:

$$\Delta\mathcal{H}_{\text{flow-var}} = N \ln K\sigma - \ln \frac{\sqrt{4\pi N^3}}{\sigma^2}. \tag{29}$$

According to the theorems above, we can conclude that the information loss in the traditional flow-level detection methods increases approximately linearly with the length of per-packet feature sequences. Thus, comparing with the packet-level methods, the traditional flow-level methods cannot effectively extract the

https://t.me/learningnets

features of traffic. Although the traditional flow-level methods can adopt multiple statistical features [4, 76], the number of packets in the feature extraction (N) is significantly larger than the number of features. In Section 6.3, we will use experiments to show that the traditional flow-level methods achieve low detection accuracy.

Analysis of Whisper. Different from the traditional flow-level methods, Whisper encodes per-packet features as vectors and performs DFT on the vectors to extract the frequency domain features of the traffic. We prove the low information loss property of Whisper by comparing with the packet-level methods (see Theorem 5) and the traditional flow-level methods (see Theorem 6) by leveraging the bounds of the information loss in Theorem 1 - 4.

Theorem 5. *(An Estimation of the Information Loss of Whisper over the Packet-level Methods).* When the Gaussian process $\mathcal{S}$ is strictly stationary with zero mean, i.e., $u(i) = 0$ and $\sigma(i) = \sigma$, we can acquire an estimate of the information loss in Whisper when ignoring the logarithmic transformation using:

$$\Delta\mathcal{H}_{\text{Whisper}} = N \ln \frac{\sigma}{w_i^2}\sqrt{\frac{\pi}{2e}} - N \ln N, \tag{30}$$

where w_i is the i^{th} element of the encoding vector w.

Theorem 6. *(An Estimation of the Information Loss Reduction of Whisper over the Traditional Flow-level Methods).* With the same assumption in Theorem 5, compared with the traditional flow-level methods that extract the average features, Whisper reduces the information loss with an estimation:

$$\Delta\mathcal{H}_{\text{Whisper-avg}} = \Delta\mathcal{H}_{\text{flow-avg}} - \Delta\mathcal{H}_{\text{Whisper}} \tag{31}$$

$$= N \ln 2ew_i^2 N + \ln \frac{\sqrt{N}}{K\sigma}. \tag{32}$$

Similarly, Whisper reduces the information loss in the flow-level methods that use min-max features and variance features. We present the estimations of reduced information loss in the metric of differential entropy as follows:

$$\Delta\mathcal{H}_{\text{Whisper-minmax}} = N \ln 2ew_i^2 N - \ln K\sigma, \tag{33}$$

$$\Delta\mathcal{H}_{\text{Whisper-var}} = N \ln 2ew_i^2 N - \ln \frac{\sqrt{4\pi N^3}}{\sigma^2}. \tag{34}$$

According to Theorem 5, by using the packet-level methods as a benchmark, we conclude that Whisper almost has no information loss when the number of packets involved in feature extraction is large. Thus, the feature efficiency of Whisper is not worse than the packet-level methods. Moreover, the packet-level methods have a large feature scale that results in high overhead for machine learning (proof in Section 5.2).

Based on Theorem 6, we conclude that the reduction of the information loss in the traditional flow-level methods increases more than linearly. Thus, by reducing the information loss in the traditional flow-level methods, Whisper can extract features from ongoing traffic more effectively than the traditional flow-level methods. In Section 6.3, we will measure the detection accuracy improvement of Whisper by using experiments.

5.2 Analysis of Scalability and Overhead

Feature Scale Reduction of Whisper. Original per-packet features are compressed in Whisper. Whisper reduces the input data scale and the processing overhead in machine learning algorithms. The compressed frequency domain features allow us to apply the machine learning algorithm in high throughput networks in practice. Compared with the packet-level methods, Whisper achieves high compression ratio C_r with a theoretical lower bound:

$$C_r = \frac{\text{size(R)}}{\text{size(S)}} = \frac{K_f N_f}{MN} \geq \frac{(\frac{N}{W_{\text{seg}}})(\frac{W_{\text{seg}}}{2}+1)}{MN} \geq \frac{1}{2M}. \tag{35}$$

By reducing the feature scale, Whisper significantly reduces the processing overhead in the packet-level methods and achieves high throughput. In Section 6.5, we will show the experimental results of Whisper to validate the analysis results.

Overhead of Feature Extraction in Whisper. Whisper incurs a low computational overhead of extracting the frequency domain features from traffic. Particularly, Whisper does not have an operation with high time or space complexity that is higher than quadratic terms. The time complexity and space complexity of Whisper are shown in Table 2.

Table 2: Complexity of the Feature Extraction Module

Steps	Time Complexity	Space Complexity
Packet Encoding	$O(MN)$	$O(MN)$
Vector Framing	$O(1)$	$O(1)$
DFT Transformation	$O(N \log W_{\text{seg}})$	$O(W_{\text{seg}})$
Calculating Modulus	$O(N/2)$	$O(N)$
Log Transformation	$O(N/2)$	$O(1)$
Total	$O(MN + N \log W_{\text{seg}})$	$O(MN + W_{\text{seg}})$

According to Table 2, the computational complexity of Whisper is proportional to the number of packets N. Most of the consumption is incurred by matrix multiplications in the packet encoding. Compared with the encoding, performing DFT on frames has relatively less computation overhead and consumes less memory space because of the high speed DFT operation, i.e., Fast Fourier Transformation (FFT). In Section 6.5, we will validate the complexity of Whisper by using the experimental results.

6 EXPERIMENTAL EVALUATION

In this section, we prototype Whisper and evaluate its performance by using 42 real-world attacks. In particular, the experiments will answer the three questions:

(1) If Whisper achieves higher detection accuracy than the state-of-the-art method? (Section 6.3)
(2) If Whisper is robust to detect attacks even if an attackers try to evade the detection of Whisper by leveraging the benign traffic? (Section 6.4)
(3) If Whisper achieves high detection throughput and low detection latency? (Section 6.5)

6.1 Implementation

We prototype Whisper using C/C++ (GCC version 5.4.0) and Python (version 3.8.0) with more than 3,500 lines of code (LOC). The source code of Whisper can be found in [21].

High Speed Packet Parser Module. We leverage Intel Data Plane Development Kit (DPDK) version 18.11.10 LTS [26] to implement

https://t.me/learningnets

Table 3: Recommended Hyper-parameter Configurations

Hyper-Parameters	Description	Value
W_{seg}	Framing length	50
W_{win}	Sampling window length	100
C	Adjusting frequency domain features	10
K_C	Number of clustering centers	10
$[W_{min}, W_{max}]$	Range of the encoding vector	$[10, 10^3]$
B	Upper bound of the encoded features	10^5

the data plane functions and ensure high performance packet parsing in high throughput networks. We bind the threads of Whisper on physical cores using DPDK APIs to reduce the cost of context switching in CPUs. As discussed in Section 4.1, we parse the three per-packet features, i.e., lengths, timestamps, and protocol types.

Frequency Domain Feature Extraction Module. We leverage PyTorch [52] (version 1.6.0) to implement matrix transforms (e.g., encoding and Discrete Fourier Transformation) of origin per-packet features and auto-encoders in baseline methods.

Statistical Clustering Module. We leverage K-Means as the clustering algorithm with the mlpack implementation (version 3.4.0) [44] to cluster the frequency domain features.

Automatic Parameter Selection. We use Z3 SMT solver (version 4.5.1) [40] to solve the SMT problem in Section 4.2, i.e., determining the encoding vector in Whisper.

Moreover, we implement a traffic generating tool using Intel DPDK to replay malicious traffic and benign traffic simultaneously. The hyper-parameters used in Whisper are shown in Table 3.

6.2 Experiment Setup

Baselines. To measure the improvements achieved by Whisper, we establish three baselines:

- *Packet-level Detection.* We use the state-of-the-art machine learning based detection method, Kitsune [42]. It extracts per-packet features via flow state variables and feeds the features to auto-encoders. We use the open source Kitsune implementation [41] and run the system with the same hardware as Whisper.
- *Flow-level Statistics Clustering (FSC).* As far as we know, there is no flow-level malicious traffic detection method that achieves task agnostic detection. Thus, we establish 17 flow-level statistics according to the existing studies [4, 5, 30, 37, 43, 77] including the maximum, minimum, variance, mean, range of the per-packet features in Whisper, flow durations, and flow byte counts. We perform a normalization for the flow-level statistics. For a fair comparison, we use the same clustering algorithm to Whisper.
- *Flow-level Frequency Domain Features with Auto-Encoder (FAE).* We use the same frequency domain features as Whisper and an auto-encoder model with 128 hidden states and Sigmoid activation function, which is similar to the auto-encoder model used in Kitsune. For the training of the auto-encoder, we use the Adam optimizer and set the batch size as 128, the training epoch as 200, the learning rate as 0.01.

Testbed. We conduct the Whisper, FSC, and FAE experiments on a testbed built on a DELL server with two Intel Xeon E5645 CPUs (2 × 12 cores), Ubuntu 16.04 (Linux 4.15.0 LTS), 24GB memory, one Intel 10 Gbps NIC with two ports that supports DPDK, and Intel 850nm SFP+ laser ports for optical fiber connections. We configure 8GB huge page memory for DPDK (4GB/NUMA Node). We bind 8 physical cores for 8 NIC RX queues to extract per-packet features and the other 8 cores for Whisper analysis threads, which extract the frequency domain features of traffic and perform statistical clustering. In summary, we use 17 of 24 cores to enable Whisper. Note that, since Kitsune cannot handle high-rate traffic, we evaluate it with offline experiments on the same testbed.

We deploy DPDK traffic generators on the other two servers with similar configurations. The reason why we use two traffic generators is that the throughput of Whisper exceeds the physical limit of 10 Gbps NIC, i.e., 13.22 Gbps. We connect two flow generators with optical fibers to generate high speed traffic.

Datasets. The datasets used in our experiments are shown in Table 4. We use three recent datasets from the WIDE MAWI Gigabit backbone network [69]. In the training phase, we use 20% benign traffic to train the machine learning algorithms. We use the first 20% packets in MAWI 2020.06.10 dataset to calculate the encoding vector via solving the SMT problem (see Section 4.2). Meanwhile, we replay four groups of malicious traffic combined with the benign traffic on the testbed:

- *Traditional DoS and Scanning Attacks.* We select five active attacks from the Kitsune [2] [42] and a UDP DoS attack trace [7] to measure the accuracy of detecting high-rate malicious flow. To further evaluate Whisper, we collect new malicious traffic datasets on WAN including Multi-Stage TCP Attacks, Stealthy TCP Attacks, and Evasion Attacks.
- *Multi-Stage TCP Attacks.* TCP side-channel attacks exploit the protocol implementations and hijack TCP connections by generating forged probing packets. Normally, TCP side-channel attacks have several stages, e.g., active connection finding, sequence number guessing, and acknowledgement number guessing. We implement two recent TCP side-channel attacks [10, 17], which have different numbers of attack stages. Moreover, we collect another multi-stage attack, i.e., TLS padding oracle attack [67].
- *Stealthy TCP Attacks.* The low-rate TCP DoS attacks generate low-rate burst traffic to trick TCP congestion control algorithms and slow down their sending rates [25, 32, 33]. Low-rate TCP DoS attacks are more stealthy than flooding based DoS attacks. We construct the low-rate TCP DoS attacks with different sending rates. Moreover, we replay other low-rate attacks, e.g., stealthy vulnerabilities scanning [38].
- *Evasion Attacks.* We use evasion attack datasets to evaluate the robustness of Whisper. Attackers can inject noise packets (i.e., benign packets of network applications) into malicious traffic to evade detection [19]. For example, an attacker can generate benign TLS traffic so that the attacker sends malicious SSL renegotiation messages and the benign TLS packets simultaneously. Basing on the typical attacks above, we adjust the ratio of malicious packets and benign packets, i.e., the ratio of 1:1, 1:2, 1:4, and 1:8, and the types

[2]We exclude passive attack datasets without malicious flow but only victim flow. Note that, in our threat model we do not consider attacks without malicious packets.

https://t.me/learningnets

Table 4: Attack Dataset Configurations

Group	Label	Attack Description	Benign Traffic[1]	Benign Flow Rate	Malicious Flow Rate	Ratio of Malicious[2]
Traditional Attacks	SYN DoS	TCP SYN flooding Deny-of-Service attack.	2020.6.10	5.276 Gbps	23.04 Mbps	0.0858
	Fuzz Scan	Scanning for vulnerabilities in protocols.	2020.6.10	5.276 Gbps	27.92 Mbps	0.0089
	OS Scan	Scanning for active hosts with vulnerable operating systems.	2019.1.2	4.827 Gbps	0.960 Mbps	0.0045
	SSL DoS	SSL renegotiation messages flooding Deny-of-Service attack.	2020.1.1	7.666 Gbps	21.60 Mbps	0.0128
	SSDP DoS	SSDP flooding Deny-of-Service attack.	2020.1.1	7.666 Gbps	27.20 Mbps	0.0321
	UDP DoS	High-rate UDP traffic blocks bottleneck links.	2019.1.2	4.827 Gbps	2.422 Gbps	0.4712
Multi-stage TCP Attacks	IPID SC	Side-channel attack via IPID assignments, disclosed in 2020 [17].	2020.6.10	5.276 Gbps	0.138 Mbps	0.0007
	ACK SC	ACK rate limit side-channel attack, disclosed in 2016 [10].	2019.1.2	4.827 Gbps	1.728 Mbps	0.0091
	TLS Oracle	TLS padding oracle attack [67].	2020.1.1	7.666 Gbps	1.626 Mbps	0.0031
Stealthy TCP Attacks	LRDoS 0.2	UDP burst triggers TCP retransmissions (burst interval 0.2s).	2019.1.2	4.827 Gbps	0.115 Gbps	0.0228
	LRDoS 0.5	UDP burst triggers TCP retransmissions (burst interval 0.5s).	2019.1.2	4.827 Gbps	0.046 Gbps	0.0112
	LRDoS 1.0	UDP burst triggers TCP retransmissions (burst interval 1.0s).	2019.1.2	4.827 Gbps	0.023 Gbps	0.0055
	IPID Scan	Prerequisite scanning of the IPID side-channel attack [17].	2020.6.10	5.276 Gbps	0.214 Mbps	0.0010
	TLS Scan	TLS vulnerabilities scanning [38].	2020.6.10	5.276 Gbps	0.046 Gbps	0.0071

[1] The Benign Traffic column shows the identifier (date) of WIDE MAWI traffic datasets [69].
[2] The Ratio of Malicious column shows the packet number ratio of benign and malicious traffic.

of benign traffic to generate 28 datasets. For comparison, we replay the evasion attack datasets with the same background traffic in Table 4.

Metrics. We use the following metrics to evaluate the detection accuracy: (i) true-positive rates (TPR), (ii) false-positive rates (FPR), (iii) the area under ROC curve (AUC), (vi) equal error rates (EER). Moreover, we measure the throughput and processing latency to demonstrate that Whisper achieves realtime detection.

6.3 Detection Accuracy

In this experiment, we evaluate the detection accuracy of different systems by measuring TPR, FPR, AUC, and EER. Table 5 illustrates the results. We find that Whisper can detect all 14 attacks with AUC ranging between 0.932 and 0.999 and EER within 0.201. Figure 3 shows the scatter plots of clustering results. For simplicity, we select two datasets with 2,000 benign and 2,000 malicious frequency domain features and choose two dimensions of the frequency domain features randomly. We observe that the malicious traffic has frequency domain features far from the clustering centers. We present the ROC curves of two datasets in Figure 4. We find that, by leveraging the frequency domain features, detectors can detect low-rate malicious traffic in high throughput networks, e.g., Whisper and FAE detect 138 Kbps IPID side-channel malicious traffic with 0.932 and 0.973 AUC under the 5.276 Gbps backbone network traffic, respectively. The increment of burst intervals in low-rate TCP DoS attacks causes 9.0%, 7.0%, 0.10%, and 0.06% AUC decrease for Kitsune, FSC, FAE, and Whisper, respectively. Thus, compared with the packet-level and the traditional flow-level detection, burst intervals in the low-rate TCP DoS attacks have a negligible effect on the detection accuracy of Whisper and FAE. However, FAE cannot effectively detect some sophisticated attacks, e.g., the ACK throttling side-channel attack and the TLS padding oracle attack, and only achieves only 39.09% AUC of Whisper. Note that, Whisper accurately identifies 2.4 Gbps high-rate malicious flows among 4.8 Gbps traffic online.

Kitsune cannot effectively detect the side-channel attacks because it is unable to maintain enough states for the traffic. We find that Kitsune's offline processing speeds in the datasets are less than 4000 packets per second (PPS), and the expected time to complete the detection is more than 2 hours. The side-channel attacks trick Kitsune to maintain massive flow states by sending a larger number of probing packets. Different from using flow states to preserve the flow context information in Kitsune, Whisper preserves the flow-level context information via the frequency domain analysis, which ensures the ability to detect such attacks.

We observe that, with the same ML algorithm, i.e., auto-encoder, the frequency domain features achieve higher accuracy (at most 15.72% AUC improvements and 95.79% EER improvements) than the state-of-the-art packet-level features and can detect more stealthy attacks. Under the five types of stealthy TCP attacks, Kitsune achieves 0.837 - 0.920 AUC and cannot detect the low-rate scanning of the side-channel attack. Moreover, compared with FSC, Whisper achieves at most 65.26% AUC improvements and 98.80% EER improvements. Thus, we can conclude that the frequency domain features allow Whisper to achieve higher detection accuracy and outperform the packet-level methods and the traditional flow-level methods.

Moreover, we study the impact of the automatic parameter selection on the detection accuracy. We manually set encoding vectors to compare the results with automatically selected parameters. We use six attacks as validation sets for the manually selected encoding vector, and use 13 attacks to test the generalization of the manually selected parameters. Figure 5 shows the detection accuracy in terms of parameter settings. We observe that the automatic parameter selection module achieves 9.99% AUC improvements and 99.55% EER improvements compared with manual parameter selection.

6.4 Robustness of Detection

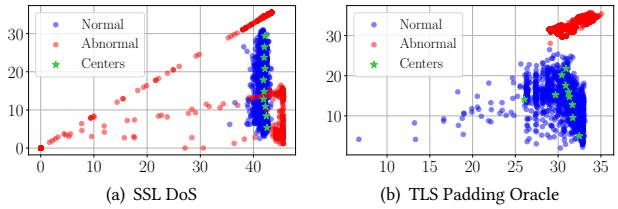
In order to validate the robustness of Whisper, we assume that attackers know the existence of malicious traffic detection. Attackers can construct evasion attacks, i.e., injecting various benign traffic, to evade the detection. In the experiments, for simplicity, we assume that attackers inject benign TLS traffic and UDP video traffic

https://t.me/learningnets

Table 5: Detection Accuracy of Whisper and Baselines on 14 Attacks

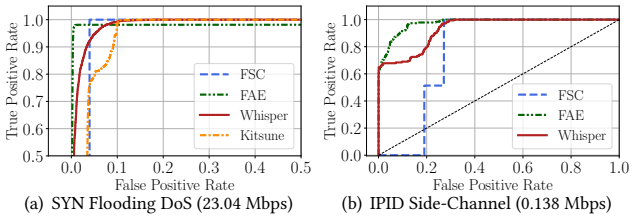
Methods	Kitsune				FSC				FAE				Whisper			
Metrics	TPR	FPR	AUC	EER	TPR	FPR	AUC	EER	TPR	FPR	AUC	EER	TPR	FPR	AUC	EER
SYN DoS	0.9801	0.0910	0.9562	0.0919	0.9999	0.0396	0.9603	0.0396	0.9813	0.0033	0.9840	0.0186	0.9924	0.0329	0.9870	0.0512
Fuzz Scan	0.9982	0.0015	0.9978	0.0336	0.0000	0.4007	0.6028	0.3964	0.0000	0.4111	0.6134	0.3954	0.9999	0.0046	0.9962	0.0047
OS Scan	0.9997	0.0786	0.9615	0.0800	0.0000	0.1114	0.8885	0.1114	0.9999	0.0069	0.9907	0.0075	0.9999	0.0106	0.9951	0.0111
SSL DoS	0.9417	0.0035	0.9781	0.0574	0.9992	0.0519	0.9732	0.0519	0.0000	0.1271	0.8774	0.1271	0.9699	0.0796	0.9391	0.0798
SSDP DoS	0.9901	0.0132	0.9955	0.0168	0.9999	0.0014	0.9986	0.0014	0.0003	0.1233	0.8770	0.1233	0.9969	0.0117	0.9902	0.0172
UDP DoS	0.4485	0.1811	0.8993	0.1433	0.9999	0.0173	0.9826	0.0173	0.9999	0.0068	0.9942	0.0071	0.9999	0.0083	0.9922	0.0093
IPID SC	/	/	/	/	0.0000	0.2716	0.7702	0.2716	0.8913	0.1001	0.9739	0.1001	0.6900	0.2324	0.9322	0.2014
ACK SC	/	/	/	/	0.0000	0.3090	0.6909	0.3090	-	-	-	-	0.9999	0.0001	0.9999	0.0001
TLS Oracle	0.9973	0.0335	0.9722	0.0392	-	-	-	-	-	-	-	-	0.9999	0.0121	0.9885	0.0124
LRDoS 0.2	0.6397	0.1270	0.9202	0.1239	0.9999	0.0254	0.9740	0.0254	0.9999	0.0254	0.9925	0.0088	0.9999	0.0109	0.9915	0.0123
LRDoS 0.5	0.0208	0.1882	0.8480	0.1835	0.9999	0.0551	0.9448	0.0551	0.9999	0.0078	0.9925	0.0081	0.9999	0.0101	0.9916	0.0114
LRDoS 1.0	0.0015	0.1774	0.8373	0.1758	0.9999	0.0940	0.9059	0.0940	0.9999	0.0074	0.9935	0.0074	0.9999	0.0115	0.9910	0.0122
IPID Scan	-	-	-	-	0.9999	0.0801	0.9255	0.0801	0.9999	0.0155	0.9934	0.0179	0.7964	0.1601	0.9579	0.1259
TLS Scan	-	-	-	-	-	-	-	-	0.0000	0.4014	0.6033	0.3973	0.9999	0.0091	0.9905	0.0095

[1] We highlight the best in ● and the worst in ● and we mark - when AUC < 0.5 (meaningless, no better than random guess).
[2] We mark / when Kitsune cannot finish the detection in 2 hours due to a large number of maintained flow state variables (process speed < 4×10^3 PPS).



(a) SSL DoS (b) TLS Padding Oracle

Figure 3: Frequency domain features clustering results of Whisper.



(a) SYN Flooding DoS (23.04 Mbps) (b) IPID Side-Channel (0.138 Mbps)

Figure 4: ROC of high-rate attack: SYN DoS and low-rate attack: IPID side-channel attack.

into the malicious traffic and disguise it as benign traffic for evasion. The reason why we use TLS and UDP video traffic is that it contributes to a high proportion of the benign traffic datasets, i.e., around 35% and 13%, respectively. Injecting the traffic can significantly interfere with traditional detection (see Figure 6). We select and replay 7 malicious traffic patterns and mix them into different ratio of benign traffic, i.e., the ratio of malicious traffic to the benign traffic ranging between 1:1 and 1:8. We do not inject the benign traffic with more ratio because the effectiveness of attacks is already low at the ratio of 1:8. We average the detection results with different ratio. Figure 6 shows the averaged detection accuracy on different attacks. The detailed detection accuracy results can be found in Appendix E (see Figure 9). We observe that the evasion attacks with high benign traffic mix ratio are prone to evade the detection. According to figure 6, we conclude that attackers cannot evade Whisper by injecting benign traffic into the malicious traffic. However, the attackers evade other detection systems.

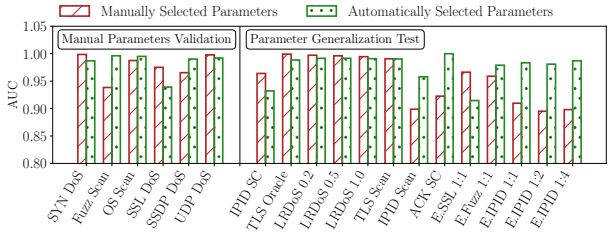
For instance, Whisper has at most 10.46% AUC decrease and 1.87 times EER increase under the evasion attacks. However, Kitsune has at most 35.4% AUC decrease and 7.98 times EER increase. Similarly, attackers can effectively evade the detection of the traditional flow-level detection system, especially injecting more benign traffic with higher ratio. The evasion attacks, e.g, evasion OS scan and evasion TLS vulnerabilities scan, lead to at most 11.59 times EER increase under the flow-level methods (AUC $\leq$ 0.5). Thus, we can conclude that the existing flow-level and packet-level detection systems are not robust to the evasion attacks. Whisper has stable detection accuracy at different ratio, e.g., the averaged AUC decrease is bounded by 3.0%, which is robust for evasion attacks. Moreover, We use other evading strategies to validate the robustness of Whisper (see Appendix E), e.g., injecting benign DNS, ICMP traffic and manipulating packet size and rate.

In summary, Whisper can achieve robust detection because the used frequency domain features represent robust fine-grained sequential information of traffic. Malicious traffic disguised as benign traffic do not incur significant changes in the flow-level statistics. Thus, the features of the malicious traffic in the flow-level methods are the same to the benign flows. As a result, due to the invariant features, packet-level and traditional flow-level detection is unable to capture such attacks. For example, the packet-level methods (e.g., Kitsune) use the statistics as the context information. However, the sequential information of the malicious traffic extracted by Whisper are significantly different from the benign traffic. Thus, to our best knowledge, Whisper is the first machine learning based method that achieves robust detection under evasion attacks.

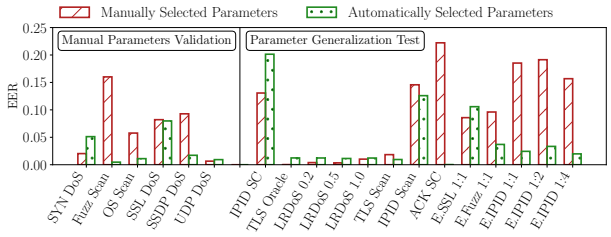
6.5 Detection Latency and Throughput

Detection Latency. To measure the latency, we replay the backbone network traffic datasets with different traffic rates (see Table 4). For simplicity, we use the low-rate TCP DoS attack with a 0.5s burst interval as a typical attack and measure the overall

https://t.me/learningnets

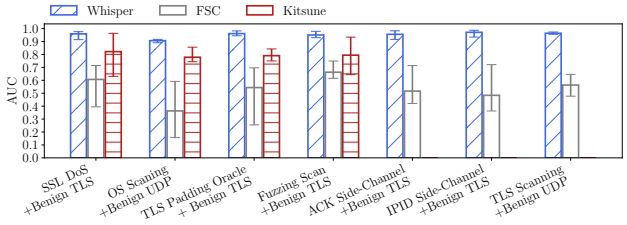


(a) AUC comparison (higher is better)

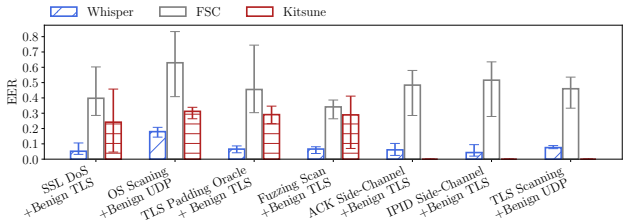


(b) EER comparison (lower is better)

Figure 5: Performance of the automatic parameter selection in comparison with manually selected parameters.



(a) Averaged AUC on different mix ratio (higher is better)



(b) Averaged EER on different mix ratio (lower is better)

Figure 6: Detection accuracy under attacks with various evading strategies.

detection latency, i.e., the time interval between sending the first malicious packet and detecting the traffic. The overall detection latency includes the transmitting latency, the queuing latency, and the processing latency. The cumulative distribution function (CDF) of the overall detection latency is shown in Figure 7(a). With four datasets, we find that the detection latency of Whisper is between 0.047 and 0.133 second, which shows that Whisper achieves real-time detection in high throughput networks. In order to accurately measure the processing latency incurred by Whisper, we replay the low-rate TCP DoS dataset with a 0.5s burst interval to construct a light load network scenario and measure the execution time of the four modules in Whisper. The CDF of the processing latency is shown in Figure 7(b). We observe that the processing latency of Whisper exhibits uniform distribution because most of the latency is incurred by polling per-packet features from the packet parser module in the light load situation. Thus, we can conclude that the averaged processing latency incurred by Whisper is only 0.0361 second, and the queuing latency raised by Whisper is the majority.

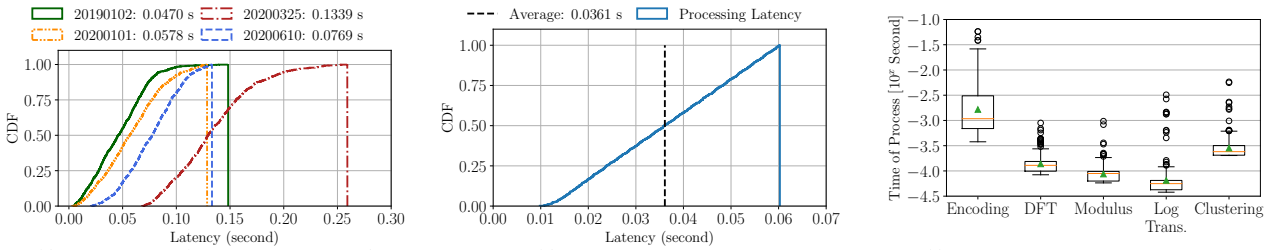
We also analyze the latency raised by each step of Whisper in Figure 7(c). We see that the measured latency in each step is consistent with the computational complexity analysis in Section 5.2. The DFT, Modulus Calculation, and Log Transformation have similar computational complexity and incur similar processing latency. The most latency is raised from the packet encoding (i.e., 5.20×10^{-3} second on average). The statistical clustering module has averaged processing latency of 1.30×10^{-4} second, which is significantly lower than the packet encoding. We find that most of the latency is incurred by the packet parsing module and the memory copy for parsing per-packet features incurs the most latency.

Throughput. We replay four MAWI [69] backbone network traffic datasets with the physical limit bandwidth of laser ports (20 Gbps) to measure the throughput. We measure the throughput of Whisper and FAE and validate that detection accuracy does not decrease when reaching the maximum throughput. We run Kitsune with the same hardware as Whisper and measure the offline processing speed, i.e., we ignore the packet parsing overhead in the online processing of Kitsune, because it cannot handle high speed traffic. The CDF of the throughput is shown in Figure 8. We find that Whisper achieves 11.35 Gbps to 13.22 Gbps average throughput, while Kitsune achieves 112.52 Mbps. Whisper achieves high throughput because it significantly reduces the processing overhead of the machine learning. FAE achieves the averaged throughput ranging between 11.28 Gbps and 13.18 Gbps, which is similar to Whisper. Note that, FAE uses a similar auto-encoder model in Kitsune and achieves 100 times higher throughput (though it has limited detection ability). We conclude that the frequency domain features used in Whisper enable higher throughput than the packet-level methods. In summary, Whisper and FAE achieve the most throughput, around 1.27 $\times 10^6$ PPS, compared with other detection systems.

7 RELATED WORK

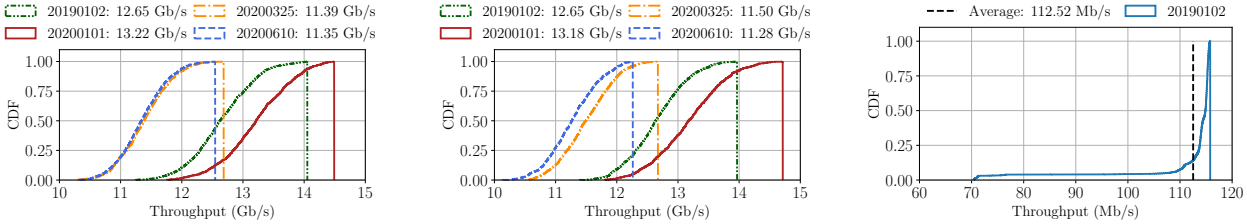
Machine Learning based NIDS. Machine learning based Network Intrusion Detection Systems (NIDSes) can achieve higher detection accuracy than the traditional signature based NIDSes [6, 35, 62, 64]. In particular, compared with the signature based NIDSes, they can detect zero-day attacks that have not been uncovered [12, 22]. For example, Antonakakis *et al.* [2], Nelms *et al.* [49], and Invernizzi *et al.* [28] detect malware traffic by using statistical machine learning approaches. Moreover, the specialized features of botnets have been used in botnet traffic detection [16, 20, 27, 30]. Different from these methods, Whisper detects various attack traffic including botnet traffic online. Bartos *et al.* [4] developed an invariant of statistical features based detection via matrix transformations, which is not scalable in large scale detection. Mirsky *et al.* [42] proposed Kitsune that leveraged lightweight deep neural networks, i.e., auto-encoders, to reduce the processing overhead. Whisper uses packet encoding and DFT to compress the original per-packet features for reducing feature redundancy. The compressed frequency domain features allow the machine learning to be readily deployable for high performance detection.

https://t.me/learningnets



(a) Overall latency (processing and queuing) (b) Pure processing latency (c) Processing latency of different steps

Figure 7: Detection latency of Whisper. We present the CDF of overall latency in (a), the CDF of pure processing latency in (b), the box plot of latency in different steps in (c).



(a) Whisper (b) FAE (c) Kitsune

Figure 8: CDF and the average number of throughput: Whisper, FAE, and Kitsune.

Traffic Classification. Machine learning algorithms are widely used in traffic classification [3, 9, 48, 56–59, 61, 66]. For example, web fingerprinting aims to invalidate the Tor anonymous services and infer the website that users are visiting by using the features of TLS encrypted traffic [55, 72, 73]. Similar to Web fingerprinting, Ede *et al.* [66] used semi-supervised learning to fingerprint mobile applications. Siby *et al.* [61] applied traffic analysis to classify encrypted DNS traffic and infer the activities of users. Bahramali *et al.* [3] analyzed the features of various realtime communication applications. Nasr *et al.* [48] compressed the statistical features of traffic, which achieved large scale traffic analysis. Zhang *et al.* [75] proposed a countermeasure against traffic analysis via adversarial examples. Although traffic classification achieves a different goal from malicious traffic detection and cannot be used in traffic detection, the extracted traffic features in Whisper, i.e., the frequency domain features, can be applied to perform traffic classifications.

Anomaly Detection with Data Augmentation. Data augmentation is recently developed efficiently model training for anomaly detection [18, 30, 60]. For example, Jan *et al.* [30] leveraged Generative Adversarial Network (GAN) to generate labeled datasets for botnet detection. Shetty *et al.* [60] generated paired data by using GAN to train a seq2seq model that aims to invalidate the anonymity of text. Fischer *et al.* [18] solved the dataset scalability problem to detect vulnerable code via Siamese Networks. In Whisper, we leverage the frequency domain features for efficient anomaly detection.

Throttling Malicious Traffic. IP blacklists have been widely used to throttle malicious traffic [36, 50]. For instance, Ramanathan *et al.* [54] proposed an IP blacklist aggregation method to locate attackers. Moreover, programmable data planes [34, 70, 71, 74, 77] have been recently leveraged to throttle various attack traffic, e.g., throttling different types of DoS flows and covert channels. All these defenses are orthogonal to our Whisper.

8 CONCLUSION

In this paper, we develop Whisper, a realtime malicious traffic detection system that utilizes sequential information of traffic via frequency domain analysis to enable robust attack detection. The frequency domain features with bounded information loss allow Whisper to achieve both high detection accuracy and high detection throughput. In particular, fine-grained frequency domain features represent the ordering information of packet sequences, which ensures robust detection and prevents attackers from evading detection. In order to extract the frequency domain features, Whisper encodes per-packet feature sequences as vectors and uses DFT to extract sequential information of traffic in the perspective of frequency domain, which enables efficient attack detection by utilizing a lightweight clustering algorithm. We prove that the frequency domain features have bounded information loss which is a prerequisite of accuracy and robustness. Extensive experiments show that Whisper can effectively detect various attacks in high throughput networks. It achieves 0.999 AUC accuracy within 0.06 second and around 13.22 Gbps throughput. Especially, even under sophisticated evasion attacks, Whisper can still detect malicious flows with high AUC ranging between 0.891 and 0.983.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful comments. This work was in part supported by the National Key R&D Program of China with No.2018YFB0803405, China National Funds for Distinguished Young Scientists with No.61825204, National Natural Science Foundation of China with No.61932016 and No.62132011, Beijing Outstanding Young Scientist Program with No.BJJWZYJH012019 10003011, BNRist with No.BNR2019RC01011. Ke Xu is the corresponding author of this paper.

https://t.me/learningnets

REFERENCES

[1] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, et al. 2016. Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin. In *ICML (JMLR Workshop and Conference Proceedings, Vol. 48)*. JMLR.org, 173–182.

[2] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. 2012. From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware. In *USENIX Security*. USENIX Association, 491–506.

[3] Alireza Bahramali, Amir Houmansadr, Ramin Soltani, Dennis Goeckel, and Don Towsley. 2020. Practical Traffic Analysis Attacks on Secure Messaging Applications. In *NDSS*. The Internet Society.

[4] Karel Bartos, Michal Sofka, and Vojtech Franc. 2016. Optimized Invariant Representation of Network Traffic for Detecting Unseen Malware Variants. In *USENIX Security*. USENIX Association, 807–822.

[5] Leyla Bilge, Davide Balzarotti, William K. Robertson, Engin Kirda, and Christopher Kruegel. 2012. Disclosure: detecting botnet command and control servers through large-scale NetFlow analysis. In *ACSAC*. ACM, 129–138.

[6] Kevin Borders, Jonathan Springer, and Matthew Burnside. 2012. Chimera: A Declarative Language for Streaming Network Traffic Analysis. In *USENIX Security*. USENIX Association, 365–379.

[7] University Of New Brunswick. Accessed January 2021. A realistic cyber defense dataset. https://www.unb.ca/cic/datasets/ids-2018.html.

[8] Anna L. Buczak and Erhan Guven. 2016. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Commun. Surv. Tutorials* 18, 2 (2016), 1153–1176.

[9] Jiahao Cao, Zijie Yang, Kun Sun, Qi Li, Mingwei Xu, and Peiyi Han. 2019. Fingerprinting SDN Applications via Encrypted Control Traffic. In *RAID*. USENIX Association, 501–515.

[10] Yue Cao, Zhiyun Qian, Zhongjie Wang, Tuan Dao, Srikanth V. Krishnamurthy, and Lisa M. Marvel. 2016. Off-Path TCP Exploits: Global Rate Limit Considered Dangerous. In *USENIX Security*. USENIX Association, 209–225.

[11] Yue Cao, Zhiyun Qian, Zhongjie Wang, Tuan Dao, Srikanth V. Krishnamurthy, and Lisa M. Marvel. 2018. Off-Path TCP Exploits of the Challenge ACK Global Rate Limit. *IEEE/ACM Trans. Netw.* 26, 2 (2018), 765–778.

[12] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly Detection: A Survey. *ACM Comput. Surv.* 41, 3, Article 15 (July 2009), 58 pages.

[13] Cisco. Accessed January 2021. Cisco SPAN. https://www.cisco.com/c/en/us/support/docs/switches/catalyst-6500-series-switches/10570-41.html.

[14] Holger Dreger, Anja Feldmann, Vern Paxson, and Robin Sommer. 2004. Operational experiences with high-volume network intrusion detection. In *CCS*. ACM, 2–11.

[15] Min Du, Zhi Chen, Chang Liu, Rajvardhan Oak, and Dawn Song. 2019. Lifelong Anomaly Detection Through Unlearning. In *CCS*. ACM, 1283–1297.

[16] Juan Echeverría, Emiliano De Cristofaro, Nicolas Kourtellis, Ilias Leontiadis, Gianluca Stringhini, and Shi Zhou. 2018. LOBO: Evaluation of Generalization Deficiencies in Twitter Bot Classifiers. In *ACSAC*. ACM, 137–146.

[17] Xuewei Feng, Chuanpu Fu, Qi Li, Kun Sun, and Ke Xu. 2020. Off-Path TCP Exploits of the Mixed IPID Assignment. In *CCS*. ACM, 1323–1335.

[18] Felix Fischer, Huang Xiao, Ching-yu Kao, Yannick Stachelscheid, Benjamin Johnson, Danial Razar, Paul Fawkesley, Nat Buckley, Konstantin Böttinger, Paul Muntean, et al. 2019. Stack Overflow Considered Helpful! Deep Learning Security Nudges Towards Stronger Cryptography. In *USENIX Security*. USENIX Association, 339–356.

[19] Prahlad Fogla and Wenke Lee. 2006. Evading network anomaly detection systems: formal reasoning and practical techniques. In *CCS*. ACM, 59–68.

[20] David Freeman, Sakshi Jain, Markus Dürmuth, Battista Biggio, and Giorgio Giacinto. 2016. Who Are You? A Statistical Approach to Measuring User Authenticity. In *NDSS*. The Internet Society.

[21] Chuanpu Fu. Accessed January 2021. The source code of Whisper. https://github.com/fuchuanpu/Whisper.

[22] Pedro Garcia-Teodoro, Jesús Esteban Díaz Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. 2009. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Comput. Secur.* 28, 1-2 (2009), 18–28.

[23] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.

[24] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. 2008. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In *USENIX Security*. USENIX Association, 139–154.

[25] Amir Herzberg and Haya Shulman. 2010. Stealth DoS Attacks on Secure Channels. In *NDSS*. The Internet Society.

[26] Intel. Accessed January 2021. Data Plane Development Kit. https://www.dpdk.org/.

[27] Luca Invernizzi and Paolo Milani Comparetti. 2012. EvilSeed: A Guided Approach to Finding Malicious Web Pages. In *SP*. IEEE Computer Society, 428–442.

[28] Luca Invernizzi, Stanislav Miskovic, Ruben Torres, Christopher Kruegel, Sabyasachi Saha, Giovanni Vigna, Sung-Ju Lee, and Marco Mellia. 2014. Nazca: Detecting Malware Distribution in Large-Scale Networks. In *NDSS*. The Internet Society.

[29] Muhammad Asim Jamshed, Jihyung Lee, Sangwoo Moon, Insu Yun, Deokjin Kim, Sungryoul Lee, Yung Yi, and KyoungSoo Park. 2012. Kargus: a highly-scalable software-based intrusion detection system. In *CCS*. ACM, 317–328.

[30] Steve T. K. Jan, Qingying Hao, Tianrui Hu, Jiameng Pu, Sonal Oswal, Gang Wang, and Bimal Viswanath. 2020. Throwing Darts in the Dark? Detecting Bots with Limited Data using Neural Data Augmentation. In *SP*. IEEE, 1190–1206.

[31] Samuel Jero, Md. Endadul Hoque, David R. Choffnes, Alan Mislove, and Cristina Nita-Rotaru. 2018. Automated Attack Discovery in TCP Congestion Control Using a Model-guided Approach. In *NDSS*. The Internet Society.

[32] Aleksandar Kuzmanovic and Edward W. Knightly. 2003. Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants. In *SIGCOMM*. ACM, 75–86.

[33] Aleksandar Kuzmanovic and Edward W. Knightly. 2006. Low-rate TCP-targeted denial of service attacks and counter strategies. *IEEE/ACM Trans. Netw.* 14, 4 (2006), 683–696.

[34] Guanyu Li, Menghao Zhang, Shicheng Wang, Chang Liu, Mingwei Xu, Ang Chen, Hongxin Hu, Guofei Gu, Qi Li, and Jianping Wu. 2021. Enabling Performant, Flexible and Cost-Efficient DDoS Defense With Programmable Switches. *IEEE/ACM Trans. Netw.* 29, 4 (2021), 1509–1526.

[35] Hongda Li, Hongxin Hu, Guofei Gu, Gail-Joon Ahn, and Fuqiang Zhang. 2018. vNIDS: Towards Elastic Security with Safe and Efficient Virtualization of Network Intrusion Detection Systems. In *CCS*. ACM, 17–34.

[36] Vector Guo Li, Matthew Dunn, Paul Pearce, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. 2019. Reading the Tea leaves: A Comparative Analysis of Threat Intelligence. In *USENIX Security*. USENIX Association, 851–867.

[37] Chih-Yuan Lin and Simin Nadjm-Tehrani. 2019. Timing Patterns and Correlations in Spontaneous SCADA Traffic for Anomaly Detection. In *RAID*. USENIX Association, 73–88.

[38] Robert Merget, Juraj Somorovsky, Nimrod Aviram, Craig Young, Janis Fliegenschmidt, Jörg Schwenk, and Yuval Shavitt. 2019. Scalable Scanning and Automatic Classification of TLS Padding Oracle Vulnerabilities. In *USENIX Security*. USENIX Association, 1029–1046.

[39] Joseph Victor Michalowicz, Jonathan M Nichols, and Frank Bucholtz. 2013. *Handbook of differential entropy*. Crc Press.

[40] Microsoft. Accessed January 2021. A theorem prover from Microsoft Research. https://github.com/Z3Prover/z3.

[41] Yisroel Mirsky. Accessed January 2021. The source code of Kitsune. https://github.com/ymirsky/Kitsune-py.

[42] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. 2018. Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. In *NDSS*. The Internet Society.

[43] Yisroel Mirsky, Naor Kalbo, Yuval Elovici, and Asaf Shabtai. 2019. Vesper: Using Echo Analysis to Detect Man-in-the-Middle Attacks in LANs. *IEEE Trans. Inf. Forensics Secur.* 14, 6 (2019), 1638–1653.

[44] mlpack. Accessed January 2021. mlpack: open source machine learning library and community. https://www.mlpack.org/.

[45] Gabi Nakibly, Alex Kirshon, Dima Gonikman, and Dan Boneh. 2012. Persistent OSPF Attacks. In *NDSS*. The Internet Society.

[46] Gabi Nakibly, Adi Sosnovich, Eitan Menahem, Ariel Waizel, and Yuval Elovici. 2014. OSPF vulnerability to persistent poisoning attacks: a systematic analysis. In *ACSAC*. ACM, 336–345.

[47] Jaehyun Nam, Muhammad Jamshed, Byungkwon Choi, Dongsu Han, and Kyoungsoo Park. 2015. Haetae: Scaling the Performance of Network Intrusion Detection with Many-Core Processors. In *RAID (Lecture Notes in Computer Science, Vol. 9404)*. Springer, 89–110.

[48] Milad Nasr, Amir Houmansadr, and Arya Mazumdar. 2017. Compressive Traffic Analysis: A New Paradigm for Scalable Traffic Analysis. In *CCS*. ACM, 2053–2069.

[49] Terry Nelms, Roberto Perdisci, Manos Antonakakis, and Mustaque Ahamad. 2015. WebWitness: Investigating, Categorizing, and Mitigating Malware Download Paths. In *USENIX Security*. USENIX Association, 1025–1040.

[50] Arman Noroozian, Jan Koenders, Eelco van Veldhuizen, Carlos Hernandez Gañán, Sumayah A. Alrwais, Damon McCoy, and Michel van Eeten. 2019. Platforms in Everything: Analyzing Ground-Truth Data on the Anatomy and Economics of Bullet-Proof Hosting. In *USENIX Security*. USENIX Association, 1341–1356.

[51] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. 2007. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press.

[52] Pytorch. Accessed January 2021. An open source deep learning framework. https://pytorch.org/.

[53] Benjamin J. Radford, Leonardo M. Apolonio, Antonio J. Trias, and Jim A. Simpson. 2018. Network Traffic Anomaly Detection Using Recurrent Neural Networks. *CoRR* abs/1803.10769 (2018).

[54] Sivaramakrishnan Ramanathan, Jelena Mirkovic, and Minlan Yu. 2020. BLAG: Improving the Accuracy of Blacklists. In *NDSS*. The Internet Society.

https://t.me/learningnets

[55] Vera Rimmer, Davy Preuveneers, Marc Juárez, Tom van Goethem, and Wouter Joosen. 2018. Automated Website Fingerprinting through Deep Learning. In *NDSS*. The Internet Society.
[56] Meng Shen, Yiting Liu, Liehuang Zhu, Xiaojiang Du, and Jiankun Hu. 2021. Fine-Grained Webpage Fingerprinting Using Only Packet Length Information of Encrypted Traffic. *IEEE Trans. Inf. Forensics Secur.* 16 (2021), 2046–2059.
[57] Meng Shen, Yiting Liu, Liehuang Zhu, Ke Xu, Xiaojiang Du, and Nadra Guizani. 2020. Optimizing Feature Selection for Efficient Encrypted Traffic Classification: A Systematic Approach. *IEEE Netw.* 34, 4 (2020), 20–27.
[58] Meng Shen, Mingwei Wei, Liehuang Zhu, and Mingzhong Wang. 2017. Classification of Encrypted Traffic With Second-Order Markov Chains and Application Attribute Bigrams. *IEEE Trans. Inf. Forensics Secur.* 12, 8 (2017), 1830–1843.
[59] Meng Shen, Jinpeng Zhang, Liehuang Zhu, Ke Xu, and Xiaojiang Du. 2021. Accurate Decentralized Application Identification via Encrypted Traffic Analysis Using Graph Neural Networks. *IEEE Trans. Inf. Forensics Secur.* 16 (2021), 2367–2380.
[60] Rakshith Shetty, Bernt Schiele, and Mario Fritz. 2018. A4NT: Author Attribute Anonymity by Adversarial Training of Neural Machine Translation. In *USENIX Security*. USENIX Association, 1633–1650.
[61] Sandra Siby, Marc Juárez, Claudia Díaz, Narseo Vallina-Rodriguez, and Carmela Troncoso. 2020. Encrypted DNS -> Privacy? A Traffic Analysis Perspective. In *NDSS*. The Internet Society.
[62] Snort. Accessed January 2021. An open source network intrusion detection system. https://www.snort.org/.
[63] Robin Sommer and Vern Paxson. 2010. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *SP*. IEEE Computer Society, 305–316.
[64] Suricata. Accessed January 2021. An open source threat detection engine. https://suricata-ids.org/.
[65] Ruming Tang, Zheng Yang, Zeyan Li, Weibin Meng, Haixin Wang, Qi Li, Yongqian Sun, Dan Pei, Tao Wei, Yanfei Xu, et al. 2020. ZeroWall: Detecting Zero-Day Web Attacks through Encoder-Decoder Recurrent Neural Networks. In *INFOCOM*. IEEE, 2479–2488.
[66] Thijs van Ede, Riccardo Bortolameotti, Andrea Continella, Jingjing Ren, Daniel J. Dubois, Martina Lindorfer, David R. Choffnes, Maarten van Steen, and Andreas Peter. 2020. FlowPrint: Semi-Supervised Mobile-App Fingerprinting on Encrypted Network Traffic. In *NDSS*. The Internet Society.
[67] Serge Vaudenay. 2002. Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS .... In *EUROCRYPT (Lecture Notes in Computer Science, Vol. 2332)*. Springer, 534–546.
[68] Ke Wang and Salvatore J. Stolfo. 2004. Anomalous Payload-Based Network Intrusion Detection. In *RAID (Lecture Notes in Computer Science, Vol. 3224)*. Springer, 203–222.
[69] WIDE. Accessed January 2021. MAWI working group traffic archive. http://mawi.wide.ad.jp/mawi/.
[70] Jiarong Xing, Qiao Kang, and Ang Chen. 2020. NetWarden: Mitigating Network Covert Channels while Preserving Performance. In *USENIX Security*. USENIX Association, 2039–2056.
[71] Jiarong Xing, Wenqing Wu, and Ang Chen. 2021. Ripple: A Programmable, Decentralized Link-Flooding Defense Against Adaptive Adversaries. In *USENIX Security*. USENIX Association, 3865–3880.
[72] Yixiao Xu, Tao Wang, Qi Li, Qingyuan Gong, Yang Chen, and Yong Jiang. 2018. A Multi-tab Website Fingerprinting Attack. In *ACSAC*. ACM, 327–341.
[73] Qilei Yin, Zhuotao Liu, Qi Li, Tao Wang, Qian Wang, Chao Shen, and Yixiao Xu. 2021. Automated Multi-Tab Website Fingerprinting Attack. *IEEE Trans. Dependable Secur. Comput.* (2021).
[74] Menghao Zhang, Guanyu Li, Shicheng Wang, Chang Liu, Ang Chen, Hongxin Hu, Guofei Gu, Qi Li, Mingwei Xu, and Jianping Wu. 2020. Poseidon: Mitigating Volumetric DDoS Attacks with Programmable Switches. In *NDSS*. The Internet Society.
[75] Xiaokuan Zhang, Jihun Hamm, Michael K. Reiter, and Yinqian Zhang. 2019. Statistical Privacy for Streaming Traffic. In *NDSS*. The Internet Society.
[76] Peilin Zhao and Steven C. H. Hoi. 2013. Cost-sensitive online active learning with application to malicious URL detection. In *KDD*. ACM, 919–927.
[77] Jing Zheng, Qi Li, Guofei Gu, Jiahao Cao, David K. Y. Yau, and Jianping Wu. 2018. Realtime DDoS Defense Using COTS SDN Switches via Adaptive Correlation Analysis. *IEEE Trans. Inf. Forensics Secur.* 13, 7 (2018), 1838–1853.
[78] Shitong Zhu, Shasha Li, Zhongjie Wang, Xun Chen, Zhiyun Qian, Srikanth V. Krishnamurthy, Kevin S. Chan, and Ananthram Swami. 2020. You do (not) belong here: detecting DPI evasion attacks with context learning. In *CoNEXT*. ACM, 183–197.

APPENDIX

A PROOF OF THEOREM 1

$\mathcal{H}_{\text{packet}}$ denotes the overall differential entropy of the sampling sequence $\vec{s}$, i.e., the sum of the differential entropy of each random variable in $\vec{s}$: ($K = \sqrt{2\pi e}$)

$$\begin{aligned}\mathcal{H}_{\text{packet}} &= -\sum_{i=1}^{N}\int_{-\infty}^{+\infty} p_i(s)\ln p_i(s)\mathrm{d}s \\ &= \sum_{i=1}^{N}\ln\sigma(i)K \\ &= \ln K^N\prod_{i=1}^{N}\sigma(i).\end{aligned}$$

We assume that the statistical feature extraction function f calculates the minimum of $\vec{s}$ to acquire the flow-level features. I_{min} denotes the index of the sample with the minimum value. The differential entropy of the feature is $\mathcal{H}_{\text{flow-min}}$ that equals to the entropy of the random variable with the minimum value:

$$I_{\text{min}} = \arg\min_i s_i,$$

$$\begin{aligned}\mathcal{H}_{\text{flow-min}} &= -\int_{-\infty}^{+\infty} p_{I_{\text{min}}}(s)\ln p_{I_{\text{min}}}(s)\mathrm{d}s \\ &= \ln K\sigma(I_{\text{min}}).\end{aligned}$$

$\Delta\mathcal{H}_{\text{flow-min}}$ denotes the differential entropy loss of the minimum feature, i.e., the difference between the overall differential entropy and the differential entropy of the minimum feature:

$$\begin{aligned}\Delta\mathcal{H}_{\text{flow-min}} &= \mathcal{H}_{\text{packet}} - \mathcal{H}_{\text{flow-min}} \\ &= \ln K^{N-1}\prod_{i\neq I_{\text{min}}}\sigma(i).\end{aligned}$$

We focus on the expectation of the loss, and leverage Jensen inequation to get the lower bound of the information loss:

$$\begin{aligned}\mathrm{E}[\Delta\mathcal{H}_{\text{flow-min}}] &\geq \ln K^{N-1}E[\sigma^{N-1}] \\ &\geq (N-1)\ln K\mathrm{E}[\sigma].\end{aligned}$$

We conduct the same proof procedure for the features that calculate the maximum of the per-packet feature sequence and complete the proof of Theorem 1.

B PROOF OF THEOREM 2 AND THEOREM 3

We consider the situation that a flow-level feature extraction method calculates the average number of sampled per-packet features. We denote the average of $\vec{s}$ as a random variable f_m that obeys a Gaussian distribution:

$$f_m \sim \mathcal{N}(\frac{1}{N}\sum_{i=1}^{N}u(i), \frac{1}{N^2}\sum_{i=1}^{N}\sigma^2(i)).$$

p_m denotes the probability density function (PDF) of f_m. We use $\mathcal{H}_{\text{flow-avg}}$ and $\Delta\mathcal{H}_{\text{flow-avg}}$ to indicate the differential entropy of

https://t.me/learningnets

the average feature and the information loss, respectively:

$$\mathcal{H}_{\text{flow-avg}} = -\int_{-\infty}^{+\infty} p_m(s)\ln p_m(s)\mathrm{d}s$$
$$= \ln\frac{K}{N}\sqrt{\sum_{i=1}^{N}\sigma^2(i)},$$

$$\Delta\mathcal{H}_{\text{flow-avg}} = \mathcal{H}_{\text{packet}} - \mathcal{H}_{\text{flow-avg}}$$
$$= \ln NK^{N-1}\frac{\prod_{i=1}^{N}\sigma(i)}{\sqrt{\sum_{i=1}^{N}\sigma^2(i)}}.$$

To get the upper bound, we use Q to indicate the square mean of the variances of $\vec{s}$. According to the inequality of arithmetic and geometric means, the geometric mean is not bigger than the square mean. We get the upper bound of the differential entropy loss:

$$\Delta\mathcal{H}_{\text{flow-avg}} \le \ln NK^{N-1}\frac{Q^N}{\sqrt{\sum_{i=1}^{N}\sigma^2(i)}}$$
$$\le \ln\sqrt{N}K^{N-1}Q^{N-1}.$$

If and only if $\sigma(i)$ is a constant, the information loss $\Delta\mathcal{H}_{\text{flow-avg}}$ reaches its maximum. We use $\sigma_{\max}$ to indicate the maximum of the variances of $\vec{s}$, and get the lower bound of the information loss by leveraging the non-negative differential entropy assumption:

$$\sigma_{\max} = \max(\sigma(i)) \quad (1 \le i \le N),$$

$$\Delta\mathcal{H}_{\text{flow-avg}} \ge \ln\sqrt{N}K^{N-1}\frac{\prod_{i=1}^{N}\sigma(i)}{\sigma_{\max}}$$
$$\ge \ln\sqrt{N} \qquad (K\sigma(i) \ge 1).$$

The equality holds iff. $\sigma(i) = \frac{1}{K}$. When the equality holds, the upper bound equals the lower bound. Here we complete the proof of Theorem 3. Similar to the proof of Theorem 1, we apply Jensen inequation to get $\Delta\mathcal{H}_{\text{flow-avg}}$ and prove Theorem 2.

C PROOF OF THEOREM 4

We consider the situation that a flow-level feature extraction method calculates the variance of the sampling sequence to extract the features of traffic. Random variable V denotes the variance of $\vec{s}$:

$$V = \frac{\sum_{i=1}^{N}(s_i-u)^2}{N}, \quad u = \frac{\sum_{i=1}^{N}s_i}{N}.$$

The random variable V obeys general Chi-square distribution. We assume that the Gaussian process $\mathcal{S}$ is strictly stationary with zero mean, i.e., $u(i) = 0$ and $\sigma(i) = \sigma$. We present an estimate of differential entropy loss when N is large enough:

$$V = \frac{\sum_{i=1}^{N}s_i^2}{N} = \frac{\sigma^2}{N}\sum_{i=1}^{N}(\frac{s_i}{\sigma})^2, \quad \sum_{i=1}^{N}(\frac{s_i}{\sigma})^2 \sim \chi^2(N).$$

$\mathcal{H}_{\text{flow-var}}$ denotes the differential entropy of the variance feature:

$$\mathcal{H}_{\text{flow-var}} = \mathcal{H}[V] = \mathcal{H}[\frac{\sum_{i=1}^{N}s_i^2}{N}]$$
$$= \ln\frac{\sigma^2}{N} + \mathcal{H}[\sum_{i=1}^{N}(\frac{s_i}{\sigma})^2]$$
$$= \ln\frac{\sigma^2}{N} + \ln 2\Gamma(\frac{N}{2}) + (1-\frac{N}{2})\psi(\frac{N}{2}) + \frac{N}{2},$$

where Γ is Gamma function and ψ is Digamma function. When N is large enough we take the even number that is closest to N to approach the information loss: (γ is Euler–Mascheroni constant)

$$\begin{cases}\psi(x) &= \frac{\Gamma'(x)}{\Gamma(x)}\\ \Gamma(x) &= (x-1)!\\ \Gamma'(x) &= (x-1)!(-\gamma+\sum_{k=1}^{x-1}\frac{1}{k})\end{cases}$$
$$\Rightarrow \mathcal{H}_{\text{flow-var}} = \ln\frac{\sigma^2}{N} + \ln 2(\frac{N}{2})! - \frac{N}{2}(-\gamma + \sum_{k=1}^{\frac{N}{2}}\frac{1}{k}) + \frac{N}{2}.$$

Then we approach the Harmonic series as follows,

$$\sum_{k=1}^{N}\frac{1}{k} \approx \ln N + \gamma,$$
$$\Rightarrow \mathcal{H}_{\text{flow-var}} = \ln\frac{\sigma^2}{N} + \ln 2(\frac{N}{2})! - \frac{N}{2}\ln\frac{N}{2} + \frac{N}{2}.$$

Finally, we use $\Delta\mathcal{H}_{\text{flow-var}}$ to indicate the information loss and leverage Stirling's formula to approach the factorial.

$$\Delta\mathcal{H}_{\text{flow-var}} = \mathcal{H}_{\text{packet}} - \mathcal{H}_{\text{flow-var}}$$
$$= N\ln K\sigma - \ln\frac{\sigma^2}{N} - \frac{N}{2} - \ln 2(\frac{N}{2})! + \frac{N}{2}\ln\frac{N}{2}$$
$$(n! \approx \sqrt{2\pi n}(\frac{n}{e})^n)$$
$$= N\ln K\sigma - \ln\frac{\sigma^2}{N} - \frac{N}{2} + \frac{N}{2}\ln\frac{N}{2} - \ln 2\sqrt{\pi N}(\frac{N}{2e})^{(\frac{N}{2})}$$
$$= N\ln K\sigma - \ln\frac{\sqrt{4\pi N^3}}{\sigma^2}.$$

Here, we complete the proof of the Theorem 4.

D PROOF OF THEOREM 5 AND THEOREM 6

Without the loss of generality, we analyze i^{th} kind of per-packet features, and denote its sampling sequence as $\vec{s}$. Based on the original assumption, we assume that Gaussian process $\mathcal{S}$ is strictly stationary with zero mean, i.e., $u(i) = 0$ and $\sigma(i) = \sigma$. Whisper extracts the frequency domain features of the per-packet feature sampling sequence $\vec{s}$ with the following steps:

(1) Perform linear transformation by multiplying w_i on $\vec{s}$, for simplicity, we use w to indicate w_i.
(2) Perform DFT on $w\vec{s}$. We denote the result as $\vec{F} = \mathscr{F}(w\vec{s})$ and its i^{th} element as $\vec{F}_i = (a_i + jb_i)w$.
(3) Calculate modulus for the result of DFT. $\vec{P}$ denotes the result and $\vec{P}_i = (a_i^2 + b_i^2)w^2$ denotes its i^{th} element.
(4) Perform logarithmic transformation on $\vec{P}$. $\vec{R}$ denotes the extracted frequency domain features for $\vec{s}$ and $\vec{R}_i = \ln(\vec{P}_i + 1)/C$ denotes its i^{th} element.

The property of Discrete Fourier Transformation: $\mathscr{F}(w\vec{s}) = w\mathscr{F}(\vec{s})$, implies that:

$$b_i =_{st} a_i, \quad a_i \sim \mathcal{N}(0, N\sigma^2).$$

We estimate the overall differential entropy of the frequency domain features by ignoring the impact of the logarithmic transformation

https://t.me/learningnets

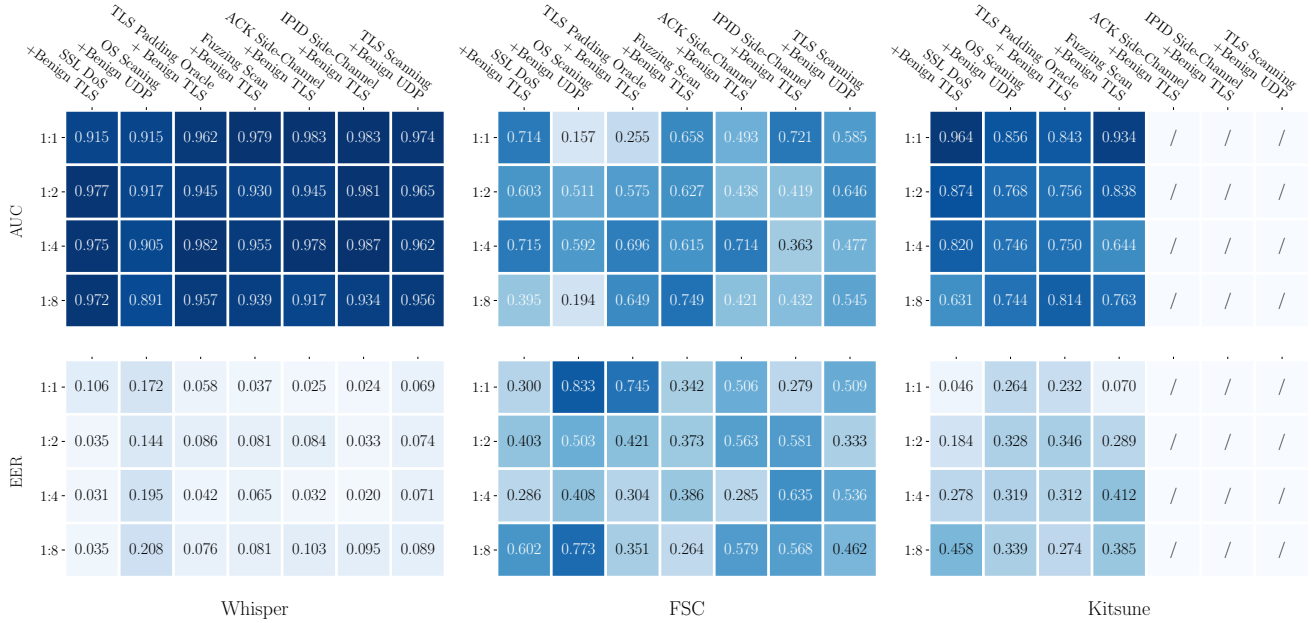


Figure 9: Detection accuracy under 28 evasion attacks. During the attacks, in order to evade the detection, the attackers use different strategies to inject benign traffic.

and obtain the entropy as $\mathcal{H}_{\text{Whisper}}$. According to the properties of differential entropy and several inequalities about differential entropy, we obtain an estimation for the differential entropy of the frequency domain features:

$$\begin{aligned}
\mathcal{H}_{\text{Whisper}} &= \mathcal{H}[\vec{P}] = \sum_{i=1}^{N} \mathcal{H}[P_i] \\
&= \sum_{i=1}^{N} \mathcal{H}[w^2(a_i^2 + b_i^2)] \\
&= N \ln w^2 + \sum_{i=1}^{N} \mathcal{H}[a_i^2 + b_i^2] \\
&= N \ln N w^2 + \sum_{i=1}^{N} \mathcal{H}[(\frac{a_i}{\sqrt{N}})^2 + (\frac{b_i}{\sqrt{N}})^2], \\
&(t_i = (\frac{a_i}{\sqrt{N}})^2 + (\frac{b_i}{\sqrt{N}})^2, \quad t_i \sim \chi^2(2)), \\
\mathcal{H}_{\text{Whisper}} &= N \ln N w^2 + \sum_{i=1}^{N} \mathcal{H}[t_i] \\
&= N \ln N w^2 + N(1 + \ln 2).
\end{aligned}$$

We use $\Delta\mathcal{H}_{\text{Whisper}}$ to indicate the information loss of Whisper and get an estimation of the differential entropy loss of Whisper:

$$\begin{aligned}
\Delta\mathcal{H}_{\text{Whisper}} &= \mathcal{H}_{\text{packet}} - \mathcal{H}_{\text{Whisper}} \\
&= N \ln \frac{\sigma}{w^2} \sqrt{\frac{\pi}{2e}} - N \ln N.
\end{aligned}$$

We complete the proof of Theorem 5. According to Theorem 1 - 4, we can obtain Theorem 6.

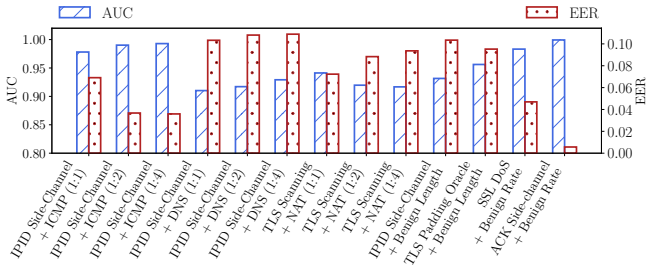


Figure 10: Detection accuracy under sophisticated evasion strategies.

E THE DETAILED RESULTS OF ROBUST EVALUATION

Figure 9 shows the detailed detection results under different evasion attacks, i.e., seven types of malicious traffic mixed with benign traffic with four types of inject ratio. We observe that the injected benign traffic has negligible effects on the detection accuracy of Whisper.

We also measure the effects of more sophisticated evasion strategies on the detection accuracy. The strategies include (i) injecting different types of benign traffic (i.e., ICMP, DNS, and outbound NAT traffic that includes various types of benign traffic), (ii) changing the rate of sending malicious packets according to the rate of benign TLS flows, (iii) manipulating the packet length in the malicious traffic according to the benign TLS packet length. Figure 10 shows that the detection accuracy is not significantly impacted by the attacks, which is consistent with the results shown in Figure 9.

https://t.me/learningnets