
# You Do (Not) Belong Here: Detecting DPI Evasion Attacks with Context Learning

Shitong Zhu
University of California, Riverside
shitong.zhu@email.ucr.edu

Shasha Li
University of California, Riverside
sli057@ucr.edu

Zhongjie Wang
University of California, Riverside
zwang048@ucr.edu

Xun Chen
Samsung Research America
xun.chen@samsung.com

Zhiyun Qian
University of California, Riverside
zhiyunq@cs.ucr.edu

Srikanth V. Krishnamurthy
University of California, Riverside
krish@cs.ucr.edu

Kevin S. Chan
US Army Research Laboratory
kevin.s.chan.civ@mail.mil

Ananthram Swami
US Army Research Laboratory
ananthram.swami.civ@mail.mil

## ABSTRACT

As Deep Packet Inspection (DPI) middleboxes become increasingly popular, a spectrum of adversarial attacks have emerged with the goal of evading such middleboxes. Many of these attacks exploit discrepancies between the middlebox network protocol implementations, and the more rigorous/complete versions implemented at end hosts. These evasion attacks largely involve subtle manipulations of packets to cause different behaviours at DPI and end hosts, to cloak malicious network traffic that is otherwise detectable. With recent automated discovery, it has become prohibitively challenging to manually curate rules for detecting these manipulations. In this work, we propose CLAP, the first fully-automated, unsupervised ML solution to accurately detect and localize DPI evasion attacks. By learning what we call the *packet context*, which essentially captures inter-relationships across both (1) different packets in a connection; and (2) different header fields within each packet, from benign traffic traces only, CLAP can detect and pinpoint packets that violate the benign packet contexts (which are the ones that are specially crafted for evasion purposes). Our evaluations with 73 state-of-the-art DPI evasion attacks show that CLAP achieves an Area Under the Receiver Operating Characteristic Curve (AUC-ROC) of **0.963**, an Equal Error Rate (EER) of only **0.061** in detection, and an accuracy of **94.6%** in localization. These results suggest that CLAP can be a promising tool for thwarting DPI evasion attacks.

## CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**; • **Computing methodologies → Neural networks**.

**ACM Reference Format:**
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.



*CoNEXT '20, December 1–4, 2020, Barcelona, Spain*

ACM ISBN 978-1-4503-7948-9/20/12.
https://doi.org/10.1145/3386367.3431311

In *The 16th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '20), December 1–4, 2020, Barcelona, Spain.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3386367.3431311

## 1 INTRODUCTION

Deep Packet Inspection (DPI) middleboxes are widely deployed as part of modern network security infrastructures [25]. They are stateful , i.e., they not only inspect individual packets, but also reassemble them to form stateful connections defined by a network protocol (e.g., TCP), based on a predefined state machine. To do so, they (for ease of exposition we refer to these as DPIs) need to include a custom network protocol implementation that is often simplified compared to its counterpart on end point platforms (e.g. the OS kernel) due to scarce computation capability, prohibitive overhead and sometimes the need to provide generality in the presence of ambiguous network protocol specifications.

**Adversarial Packets**. Since the advent of DPI, there have been multiple attacks to circumvent them. These attacks exploit the discrepancies between the DPI's and the OS-level network protocol implementations, to craft subtle yet powerful packets which trigger completely different behaviours on the DPI and at the endpoint (e.g., server). For example, such a packet may be crafted so as to cause the DPI to ignore it, while it still reaches and is accepted by the server. Such packets can potentially contain malicious payloads, and yet successfully bypass the detection of the DPI (because their contents would not be inspected at all). Even worse, prior work shows that in some cases, using only one such packet could cause the DPI to disengage from monitoring of the associated connection, thereby allowing follow-up packets in the connection to pass through without triggering alarms.

**Automated Discovery of Evasion Packets**. In recent years, as network protocol stacks have become increasingly complex with newly added features [19], a growing number of vendor-specific implementations, and continually evolving specifications, there is a surge in research [4, 10, 23] on automating the discovery of *adversarial packets* as described above, to evade DPIs. By applying principled search [10], genetic mutation [4], or symbolic execution [23], a vast assortment of adversarial packets can be found with little to no manual intervention. While this works towards understanding

attackers, it poses a hard challenge from the defense perspective. As protocol stacks evolve and become more complex, the potential discrepancies that may be discovered with automation can grow in theory, and it becomes prohibitive to manually analyze these subtle implementation issues and patch all of them in the code base; in addition it is hard to generate new hardcoded DPI policies to keep up with new discrepancies that may arise given the pace of rapidly evolving implementations and protocol standards.

**Existing Defenses**. Given its difficulty, only a few limited countermeasures have been proposed against adversarial packets. Notably, [9], Kreibich et al., apply *traffic normalization* to mitigate the threats of adversarial packets. Specifically, a so-called traffic normalizer is proposed, which acts as a network forwarding element preceding DPI middleboxes, and alters/drops the packets going through the latter as per a predefined set of rules. These rules are manually curated to describe the "normal traffic" (e.g., the IP header length field must never be smaller or greater than the actual header length). This countermeasure unfortunately, cannot scale in presence of automation – the achievable search space of all possible adversarial packets can become large enough to make timely manual curation prohibitive. Moreover, the ever-evolving protocol standards introduce false alarms – a previously correct rule can later cause incorrect decisions (e.g. normal packets being dropped) because of updated implementations. It is also worth noting that as a normalizer, or more generally a *traffic shaper*, [9] provides no detection ability; rather, it blindly alters the traffic stream.

**Our Approach**. Our goal in this paper is to design a practical defense to effectively detect evasion attempts on DPI middleboxes. Instead of relying on significant manual curation/analysis, we propose a novel, fully-automated (with minimum feature engineering) Machine Learning (ML)-based approach called CLAP (**C**ontext **L**earning based **A**dversarial **P**rotection). CLAP learns the benign (what we call) *context* from only normal traffic traces (i.e., unsupervised), and uses this learning to detect adversarial packets. In other words, CLAP asserts whether the context of the unseen packets "fit" in the associated connection or not. Specifically, the context of a packet is composed of two types of sub-contexts to describe the aforementioned "fitness" of a given packet:

- **Inter-packet context**, which captures the inter-relationships among different packets in the connection in terms of how their header fields change/evolve over the trace; these changes generally relate to the transitioning of states for a stateful network protocol (e.g., TCP);
- **Intra-packet context**, which captures the inter-relationships among different header fields in a given packet (in terms of the combinations of their values).

By learning the joint distribution (i.e., the *packet context*) of the two (sub-)contexts from benign traffic, CLAP automatically finds violations thereof, caused by adversarial packets.

**Motivating Example**. To showcase the intuition behind how CLAP works, we present a concrete attack and its detection with CLAP. Bad-Checksum-RST is an attack that has been reported in [4, 23] and shown to be effective against Great Fire Wall (GFW), a state-of-the-art DPI-based censorship system. It injects an ill-formed RST packet with a garbled TCP checksum value after the three-way handshake. Since common endhost TCP implementations perform a rigorous checksum verification but the GFW does not, this injected RST packet is dropped by the endhost but not GFW. As a result, upon seeing a RST packet, GFW would disengage its monitoring of the connection, while the communications between two endhosts would be allowed to continue.

By learning the benign context from a large set of clean network traces, CLAP knows what requirements (inter- and intra-packet contexts) must be met by a packet at its position in the connection (e.g., can it be a RST and if so, should its checksum be correct?); then, CLAP checks whether the packet conforms (fits in) to a benign packet context. In this case, the RST packet with a bad checksum value that appears after three-way handshake is asserted as violating both the inter- (RST should not take place at this point) and intra-packet (checksum of RST packet should be correct) contexts (based on training) and thus, the evasion attempt is detected.

**Contributions**. In brief, our contributions in this paper are:

(1) We are the first to propose a fully-automated unsupervised learning approach to detect adversarial packets that are crafted to elude DPI middleboxes.
(2) Our evaluations on 73 state-of-the-art DPI evasion attacks over realistic backbone traffic captures, show that by only learning from benign traffic, CLAP achieves an overall detection AUC-ROC of over **0.963**, with an average EER of only **0.061** (the two most commonly used evaluation metrics for ML-based IDSs [2, 13, 14, 17]).
(3) Our evaluations show that beyond detection, CLAP achieves an average Top-5 localization accuracy (identifying a train of five packets most likely to contain the attack vector) of **94.6%** (Top-3 of **91.0%**).
(4) Our performance analysis shows that our pipeline can process over 2,100 packets per second on a single CPU core, with linear scalability.
(5) We will open source both our implementation of CLAP, and the used datasets, at https://github.com/seclab-ucr/CLAP to allow reproducibility and future extensions.

## 2 RELATED WORK

**ML-based Intrusion Detection System (IDS)**. As a critical and commonly deployed network infrastructure, intrusion detection systems (IDSs) are designed to detect suspicious traffic and flag them accordingly. Traditionally, IDSs are signature-based and catch malicious traffic that violates a predefined set of rules. This type of IDSs face challenges because their manually curated signatures cannot keep up with emerging threats. In contrast, anomaly-based IDSs do not rely on priori-created signatures; an anomaly-based IDS inspects and classifies traffic by asserting whether it aligns with patterns observed in normal traffic. More recently, ML techniques have been used for anomaly detection, as they are efficient in learning patterns from benign traffic and then distinguishing between normal and suspicious instances on unseen traffic [2, 13, 14, 17]. While with a similar general goal (detecting suspicious traffic), CLAP is fundamentally different because it (1) spots attacks that specifically seek to evade DPI detection, not for general malicious purposes (e.g. DDoS); and (2) uniquely considers packet context that is critical for

detecting DPI evasion attacks, while context-agnostic ML-based IDS is unable to do so as will be shown in our evaluations.

[8] is the only attempt towards using ML to discover DPI evasion attacks to our best knowledge. However, the approach proposed in [8] relies on training its model on both benign and malicious traffic traces, which renders a different threat model as compared to what we assume in this paper, and is therefore not comparable directly. In fact, one of the key attributes of CLAP is its ability to detect subtle evasion attacks without a priori knowing about them. For an apples-to-apples comparison, we use a state-of-the-art unsupervised IDS [17] as one of the baselines in Section 4, and show that the state of the art general-purpose IDS is ineffective/inaccurate in detecting DPI evasion attacks because it does not consider context.

**Context Learning**. The general notion of using context has been explored in computer vision for improving classifier performance [12, 21]. Context refers to co-occurrences of objects that commonly appear together in the same scene, and aids detection/segmentation tasks where certain objects lack inherent patterns to be recognized, but can be inferred by occurrences of other easily-recognizable objects. Note that while these approaches, by mainly characterizing spacial inter-relationships among different objects as context, are generally appropriate for vision applications (i.e., objects in same scene), it cannot be applied directly to network domain applications. Context inconsistency has also been very recently considered for thwarting adversarial examples in computer vision [11] but has never before considered in the network intrusion detection context. Remotely inspired by these efforts, for the first time, we define *packet context* for network traffic data, and propose CLAP to automatically learn and apply this to detect DPI evasion attacks. We consider the unique characteristics of network traffic, and design our system accordingly.

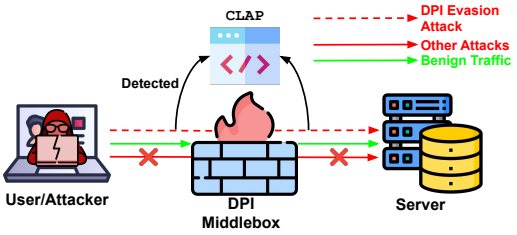


**Figure 1: Threat model of DPI with CLAP**

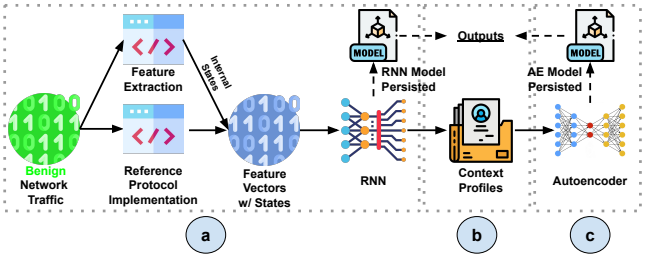


**Figure 2: Training phase of CLAP**

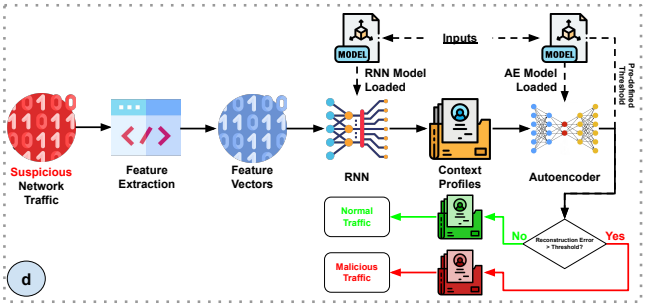


**Figure 3: Testing phase of CLAP**

## 3 SYSTEM DESIGN

### 3.1 Overview

As discussed earlier, CLAP draws on the fact that there are co-occurence relationships between packet fields (intra-packet), and across packets (inter-packet) in a single TCP flow. To re-iterate, these relationships form the packet context. An evasion attack is likely to tamper with these relationships to confuse the DPI middlebox and CLAP seeks to most efficiently learn and use the packet context to detect such tampering. Our design of CLAP consists of 4 stages:

- (a) Learning benign inter-packet context by training a RNN model whose task is to predict the transitions across a set of connection states (not only high-level TCP states but also subtle verdicts/states, as described later), on benign traffic traces;
- (b) Fusing/concatenating the benign inter-packet context (i.e., as discussed later gate weights from (a)'s RNN model represent inter-relationships among different packets) and intra-packet context (i.e., combinations of packet header fields) to generate benign *context profiles*;
- (c) Learning benign holistic packet context by characterizing the distribution of context profiles generated in (b);
- (d) Detecting DPI evasion attacks by verifying whether the context profile of unseen packet trains violates the distribution of benign context profiles as learned in (c).

We show diagrams that depict Stage (a)/(b)/(c) of CLAP in Figure 2, and (d) in Figure 3.

### 3.2 Threat Model

Before diving into the technical details of the design of CLAP, we establish the threat model we consider. As previously mentioned, CLAP is designed to protect stateful DPI middleboxes against evasion attacks. We capture the threat model associated with common DPI middleboxes relating to our work, and the role of CLAP in Figure 1. We assume both the DPI middlebox and CLAP are located in between the client and server, and capable of reading all packets going through. Commonly, DPI inspects payloads of these packets and detects malicious contents in those payloads (i.e., "Other Attacks" in Figure 1). When the attacker launches the DPI evasion attack from client side, he/she injects specially crafted packets (i.e., adversarial packets) to cause discrepancies in behaviours between

the DPI and the server. The discrepancies cause the follow-up malicious traffic to not be inspected by the DPI (but can still reach the server). CLAP is designed to specifically detect such adversarial packets. Note that we limit the scope of our target evasion attacks to those that only involve header fields manipulations, due to (1) scarcity of payload-related evasion strategies; and (2) prohibitive overheads in training for benign payloads. We only focus on the TCP protocol in this work since it is arguably the most popular transport layer protocol and is most commonly targeted in DPI evasion attacks. Since CLAP does not depend on the DPI itself, it can function interdependently (for forensic analysis) and does not rely on anything besides raw traffic that traverses the DPI middlebox (i.e., PCAP captures). Hence, CLAP can not only be deployed as an online detector complementing existing DPIs to detect evasion attacks (with affordable overhead as shown in Section 4.4), but also be used as a forensic tool to analyze the traffic captures offline.

## 3.3 CLAP Design

Next, we describe the different components in CLAP that help achieve its overarching goal.

**(a) Learning Inter-packet Context**.

*Goal.* The first stage of CLAP involves training a RNN model to drive it towards learning the inter-packet context as defined previously. As established earlier, the inter-packet context essentially captures the relationships among the header fields (co-occurrence) in a train of packets. We reiterate that this context strictly describes the relations across *different* packets, and therefore does not concern itself with any relations/combinations of header fields within the same packet.

In order to capture these inter-packet relationships with a ML model, we need to design a corresponding learning task. Since the transitions of a TCP state machine depend on such relationships across a sequence of given packets, we use a ML classifier that predicts these transitions; such a classifier will need to learn the temporal structure across packets. The best option among various neural network choices for doing so is a GRU-based RNN model. Based on this general architecture, we design a customized model that takes the header fields of each packet as inputs, and predicts the corresponding states to which the reference TCP state machine will transition as a result of this packet, as model output. Note that we are not interested in the classification result of the RNN, but rather require the distribution of the benign inter-packet contexts which we draw from the weights of the gates in the neural network architecture, as discussed in the next subsection. This requires exceptionally high performance for the task we design for RNN (connection state prediction), which from our evaluations (detailed in Table 5), is achieved with an overall prediction accuracy over 0.99).

*RNN/GRU.* RNNs are a class of neural networks that are widely used to model temporal data. In brief, they contain a recurrent cell that processes one element in the input sequence a time, considers both the current input and a memory state from the previous unit to output a new memory state. Beyond accomplishing classification tasks, the chaining of these repeated units in RNN models have also been highly successful [12, 21] in generally modeling and encoding the interrelationships across the input sequence. Among RNN cells, the Long Short-Term Memory Unit (LSTM) and the Gated Recurrent Unit (GRU) are considered the state-of-the-art architectures. Both of them consist of a "gating" mechanism, wherein gates that are in the cells update weights that represent the relationships across input sequence. We pick GRU in this work as it provides performance that is comparable with LSTM but incurs considerably lower overhead, and note that LSTM is also a viable option.

*Input Feature Set.* In order to cover (1) all header fields that influence the transitions of the TCP state machine and (2) header fields that can possibly be manipulated by DPI evasion strategies (i.e., all non-tuple-related, non-optional IP/TCP headers, and a few common TCP option headers, as specified in [18, 20]), we include 37 header field values as features (7 for IP and 25 for TCP, full list in Table 7) from the IP and TCP headers. To reiterate, we do not consider payload-related features here because (1) they are irrelevant to TCP state transitions; (2) there are only a very small fraction (7 out of 80 from [4, 10, 23]) of DPI evasion attacks that involve manipulating packet payloads; (3) it is prohibitively challenging to learn distributions of unstructured data such as benign payloads; we leave this as an open question to examine in the future; and (4) most public traffic archives (including what we use in the evaluations) strip payloads in the traces for privacy concerns. We alto follow the general principle of using these fields in the raw form to the extent possible to avoid heavy feature engineering (i.e., only need minimum pre-processing such as validating checksum, as detailed in Table 7), and show that the RNN is capable of inferring the connection states without requiring extensive domain knowledge.

*Labeling.* According to the IETF standard [20], TCP defines 11 different states (e.g. `SYN_SENT/SYN_RECV/ESTABLISHED`) that we refer to as *master TCP states*. In addition to these states, in order to enrich the learned packet context, especially in the rather coarse grained ESTABLISHED state, we also include the more subtle, in-/out-of-window states (i.e., whether an incoming packet is within the recipient's receive window) collected from the reference TCP implementation to be part of the label. Therefore, the label we use to train RNN is the concatenation of the master TCP state and the in-/out-of-window subtle *state* [1], resulting in a total of 11 * 2 = 22 potential classes (an in-/out-of-window possibility is included for each of the 11 master states); these are listed in Table 7. In order to collect reliable and accurate states for labeling the training data of our RNN, we resort to OS-level (e.g. Linux) TCP stack implementations and instrument their relevant modules to expose our desirable states (i.e., master TCP state and in/out-of-window subtle state). We replay the benign training traffic captured on the instrumented platform to harvest their corresponding states as labels (the details of the setup are in Section 4.1).

*Training.* To put things together, we now formally describe how we train the RNN model. Consider a benign network connection consisting of $n$ packets $P_{1...n}$, where $P_i = [F^1_{IP}, ..., F^8_{IP}, F^1_{TCP}, ..., F^{29}_{TCP}]$ (i.e., $F^i_{IP}$ represent the features from the corresponding packet's IP header fields, and $F^i_{TCP}$, those from the TCP header fields). We

[1] Although, we are slightly misusing the term, when we refer to state from hereon, we not only include the traditionally defined TCP states, but also a packet classification/verdict (in-window or out-of-window) that strongly relates to inter-packet relationships, to better drive the RNN model to learn bengin inter-packet contexts.

train a RNN model $M_{GRU}$, with GRU as its cell architecture, by feeding $P_{1...n}$ and their corresponding ground-truth labels (i.e., states from TCP state machine) $L_{1...n}$, and executing standard error back-propagation [6]. The standard multi-class cross entropy loss function in Equation 1, where $L_i$ and $\hat{L}_i$ are the probabilities relating to class $i$ of the ground-truth label vector and the predicted label vector, respectively, is used as the loss function for training the RNN. It measures the error between the current RNN model output (i.e., a vector of probabilities) and the ground-truth label (i.e., a vector of values of 0 except that the index of expected/ground-truth state is 1). The error is propagated back to the prior layers of the RNN to update its parameters until the model produces the correct output. In the next subsection, we describe how the intermediate/latent gate states of the training GRU are used to build context profiles.

$$Loss_{CrossEntropy}(L, \hat{L}) = -\sum_{i} L_i log(\hat{L}_i), \quad \text{where } \hat{L} = M_{GRU}(P) \tag{1}$$

**(b) Fusing Inter- and Intra-packet Contexts**.

*Goal.* Once the RNN model is trained, its gates contain the inter-packet context learned from benign traffic traces (described in more detail in the next paragraph). Next, we need to extract the intra-packet context and fuse it with the inter-packet context to form the *context profiles* of a packet. Recall that the intra-packet context is defined by the relationship/combinations across header field features within the packet. We fuse the two contexts by simply concatenating them (i.e., gate weights and header field values) into a unified feature vector which is referred to as the context profile for that packet. This fusion strategy enables the autoencoder in Stage (c) (to be described) to learn the joint distribution of both contexts. By examining this joint distribution, CLAP is capable of exposing attacks that violate (1) only the inter-packet context; (2) only the intra-packet context; and (3) both sub-contexts simultaneously.

*Gate Weights.* Figure 4 provides a closer look at the internals of GRU cells. For one such cell, besides the input (i.e., $x_t$ in Figure 4) and the output state (i.e., $h_t$ in Figure 4), there are also "gates" for optionally letting information through [6] (i.e., reset and update gates as marked in Figure 4). In order words, these gates explicitly control whether the output classification (i.e., the TCP state described earlier) of a given packet strongly relates to its previous packets, or not. Specifically, if at the current time step $t1$ the gate weights with respect to a previous packet $P_{t2}$ at time step $t2$ are large, it means the features of $P_{t2}$ contribute greatly to the classification of the next output at $t1$ (and vice versa). As one can see, the gate weights are ideal means to characterize the dependencies or inter-relationships across the different packets in a connection, or in other words, capture the inter-packet context.

*Chain Graph.* To visualize the context profile of a packet, we can represent the same as a graph-based model wherein the packet header features are the graph nodes, and gate weights are the edges connecting packets (nodes). Thus, the train of packets can be represented as a chain-shaped graph. The packets header field features are the node features, and the edges between two adjacent nodes (consecutive packets) are the GRU gates that describe the inter-relationship between the two packets. The resulting connected

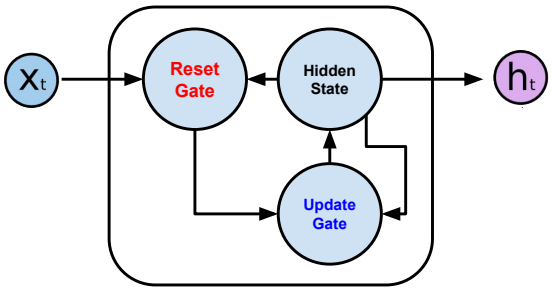


**Figure 4: Internals of GRU cell**

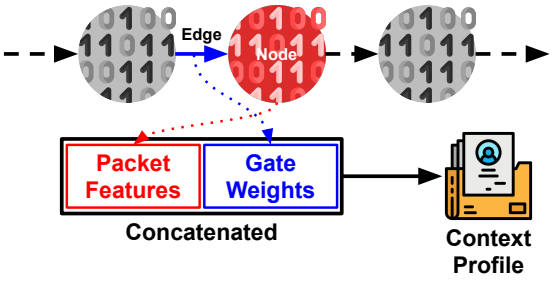


**Figure 5: Representation of context profile as chain graph**

chain graph, models the network trace. The gate weights (edges) in the graph, (which are learnt during training) control the information propagation between the adjacent packets (nodes). We depict this chain like graph model in Figure 5.

*Amplification Features.* While we have included all necessary packet header fields as the feature set for learning the intra-packet context, we discover that some subtle context violations are challenging to capture in practice with just these features. These relate to extremely small perturbations in terms of the associated changes of certain features; these help in successfully evading the DPI, but without amplification seem too insignificant for the ML classifier (specifically the autoencoder) to recognize. To address this, we augment our feature set with two types of *amplification features*, that are incorporated into the context profile; these features, crafted based on domain knowledge, amplify the aforementioned subtle context violations such that the distribution discrepancy they cause are easily captured by the autoencoder that is described later when we discuss Stage (c). In particular, the amplification features that we design are: (1) *out-of-range features*, which indicate whether the packet's associated numerical header field value is out of the range of what has been observed in the benign training traces; and the (2) *equivalence relation feature*, which indicates whether an expected equivalence relationship with respect to certain header field values is maintained (e.g., TCP payload length = IP total length - IP header length - TCP data offset) or not. We provide the full list of the 15 amplification features (all belonging to the two types) with their semantics in Table 7. We refer to the concatenation of the raw header field value features and the amplification features as *packet features*.

*Concatenation.* We next provide details on "how exactly" we generate the context profile. Consider a GRU taking $n$ input features with its hidden state size also being $n$-dimensional; in such a case, the size of its gates should be equal to the hidden state (i.e., $n$) [6]. As previously discussed, the context profile is the concatenation of the packet features and the gate weights, as defined in Equation 2. We provide a full list of all packet features in Table 7. Upon generating the context profiles for all the packets in a train, we can concatenate consecutive packet profiles to form a "stacked profile." This design aggregates multiple context profiles from consecutive packets, and therefore explicitly embeds inter-packet relationships into the stacked profile (in addition to existing gate weights that are also designed to capture inter-packet context); this in turn provides profiles with richer inter-packet context encoded. As one might expect, we find that this helps improve performance and eventually use it in our evaluations.

$$\begin{aligned} CxtProf &= [P_{IP}, P_{TCP}, P_{amp}, G_{forget}, G_{update}] \\ StackedCxtProf &= [..., CxtProf_{t-1}, CxtProf_t, CxtProf_{t+1}, ...] \end{aligned} \tag{2}$$

**(c) Learning the Joint (Context) Distribution**.

*Goal.* Once the benign context profiles that contain both inter- and intra-packet contexts are generated as described in (b), CLAP needs to learn their joint distribution so that it can later detect suspicious packets that violate the benign context (i.e., joint distribution).

*Autoencoder.* Antoencoders are a class of neural networks that characterize the distribution of given training samples by forcing the networks to reproduce the inputs themselves as model outputs. In other words, they are tasked to encode a given input to a compressed latent space (bottleneck layer), and decode it to recover the input as much as possible. During this process, the autoencoder learns the compressed representation from the training data (i.e., benign context profiles in our case) distribution as the features of its bottleneck layer, and the reconstruction error between the real input and recovered input (model output) is considered an ideal metric to characterize an input (context profiles) in terms of how close it is to the learned distribution. If a context profile traversing an autoencoder trained on benign context profiles, exhibits a high reconstruction error, we infer that it deviates from, or violates the benign context. Autoencoders are considered as the state-of-the-art means for anomaly-detection tasks [26]. We use an autoencoder here to learn the distribution of benign context profiles, and then determine if the context of unseen packets is consistent with what is observed in benign packet traces in (d).

*Training.* We train the autoencoder with benign context profiles obtained. The L1 loss function is used to measure the reconstruction error and is the sum of the all the absolute differences between the true value (ground-truth context profile) and the predicted value (reconstructed context profile). During training, by minimizing the L1 loss, the autoencoder learns to characterize the distribution of benign context profiles. The choice of the L1 loss function also relates to its excellent performance in handling dense input data, meaning data wherein there is little to no sparsity (i.e., features with zeroes as values). This suits the properties of the context profiles generated in Stage (b), because both the packet features and the gate weights are dense and rarely have 0 as values.

$$\begin{aligned} Loss_{L1}(X_{input}, X_{output}) &= \frac{1}{n}\sum_{i=1}^{n} |X^i_{output} - X^i_{input}| \\ \text{where } \{X_{input}, X_{output}\} &\in \mathcal{R}^n \end{aligned} \tag{3}$$

**(d) Verification**.

*Goal.* Lastly, CLAP with its trained RNN (to generate gate weights for context profiles) and autoencoder (trained using those profiles), is to be deployed online (i.e., it encounters previously unseen packets) to detect possible DPI evasion attacks. Recall that the autoencoder from Stage (c) can only compute the reconstruction error with respect to each context profile. Given that the input to CLAP would be connections/sequences of packets and our goal is to provide a connection-level detection conclusion, we need to first determine a strategy for CLAP to compute a score (which captures the likelihood of whether there are evasion packets within the connection for a given connection that contains sequence of reconstruction errors produced by the trained autoencoder. Then, CLAP must analyze the context profiles for unseen packets (their conformance to benign profiles) and use the chosen score (discussed below) to assert if this connection is adversarial.

*Adversarial Score.* Once the autoencoder is trained, it produces a numeric reconstruction error for a given (stacked or non-stacked) context profile. Consider a connection that consists of $n$ packets in total. Let us assume that stacked context profiles containing $t$ packets, are generated in a sliding-window fashion (i.e., concatenation of every $t$ consecutive single-packet profiles from the beginning to the end of a unidirectional traffic sequence; these profiles are overlapping) are obtained. Specifically, with this approach, CLAP generates $n - t + 1$ such stacked profiles (i.e., leading to $n - t + 1$ tests and thereby, reconstruction errors at the autoencoder stage) for the connection. This sliding window enumerates all possibilities of temporal contexts, and can be expected to provide the best coverage. CLAP now needs to capture a "summarized" value that characterizes the "overall profile" of the connection, by means of these enumerated profiles. There is a spectrum of approaches for characterizing a group of observations in general, ranging from basic statistical quantities such as maximum/minimum, variance, mean, median, to more advanced methods such as training a separate autoencoder for the summarization [15].

We examine the reconstruction error trends from adversarial connections, towards empirically choosing a proper metric. An example is shown in Figure 6, where an adversarial packet introduces a spike in the error value (around the time it is encountered) and then, the error level falls off to get closer to that with benign profiles. Based on this observation (which is expected since the sliding windows closest to the adversarial packets are likely to show the highest error), we propose a *localize-and-estimate* approach to choose a metric; this maximizes our odds of distinguishing adversarial from benign connections in the testing phase. Specifically, we first localize (identify the position within the sliding window) the profile with the maximum reconstruction error in the connection; and (2) sample the mean reconstruction error over the window (of 5 profiles in this work) by choosing the profile with the maximum

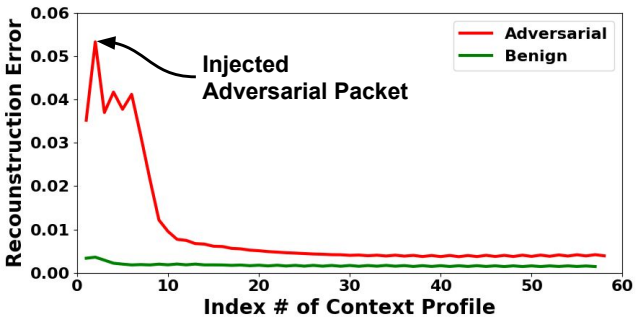


**Figure 6: Typical trend of reconstruction errors across a connection**

reconstruction error as center. We refer to this mean as the *adversarial score* for the connection. We believe this newly designed metric best captures the most distinguishing (the spike) part of the reconstruction error sequence in a connection, provides the best detection performance. In addition to the score, to make a Boolean classification (attack or no attack), we also need a threshold to determine at what level of the adversarial score, do we consider the connection to be an attack. The choice of this threshold will provide a trade-off between true and false positive rates. We leave the freedom of choosing the threshold to the deployer of CLAP towards achieving the appropriate trade-off; however, our results in Section 4.2 offer possible choices for these thresholds (by means of ROC curves).

*Deployment.* Lastly, with the 3-tuple $\{M_{GRU}, M_{AE}, TH_{Adv}\}$ (i.e., trained GRU-based RNN model, trained autoencoder model and selected threshold for adversarial score) generated in the previous steps and an incoming (suspicious) connection $C_{Sus}$, the deployer of CLAP first executes the pipeline in Figure 3 to compute the adversarial score for $C_{Sus}$, and then compares it with $TH_{Adv}$ to determine if the connection is adversarial or not. Note that beyond binary classification (i.e., adversarial or benign), CLAP can also pinpoint the most "suspicious" packets in a connection, by localizing the ones that bear the highest reconstruction error (i.e., the first step of "localize-and-estimate" approach). We will evaluate both the detection and localization accuracies in the next section.

## 4 EVALUATIONS

In this section, we provide comprehensive evaluation results with regards to different aspects of CLAP, viz.,: (1) effectiveness analysis in terms of how well it detects and localizes adversarial packets, compared to other baselines; (2) case studies with in-depth analysis on its performance on certain attacks; and (3) performance analysis in terms of how fast it can process network traffic in practice.

### 4.1 Setup

**Dataset**. We first describe the dataset used in our evaluations. For learning benign packet contexts that are as complete and sound as possible, the dataset of benign traffic traces must guarantee that it (1) only consists of benign (i.e., no DPI evasion attempts) flows; and (2) is adequately representative i.e., non-adversarial packets including those with uncommon but legitimate patterns are well-covered. Given these requirements, we select the MAWI Traffic Archive [1] that provides PCAP captures of a backbone network in Japan. MAWI traffic traces are considered one of the state-of-the-art long-term measurements on wide-area/global Internet and have been used in prior networking/security research [2, 5, 7]. Note that the payloads of MAWI traces (and most other public datasets) are all stripped (payloads are not visible) for privacy concerns, making it impossible to use for detecting payload-related attacks (e.g. segmentation-based attacks) – we adjust the corpus of evaluated attacks accordingly. Table 4 (in Appendix due to space limitation) lists basic statistics of the traffic capture we use.

Recall that Stage 1 of CLAP needs a reference TCP implementation to generate the internal states required for training the RNN model. We select the `conntrack-tools` module [3] in Linux's `Netfilter` sub-system that provides standard and reliable infrastructures for packet inspection. We instrument relevant code in Linux kernel version 5.6.3 to expose the required states, and use it as the traffic "replayer" for collecting labels for RNN training.

**Simulated Attacks**. For generating the the traffic that seeks to evade the DPI middlebox, and to include these to form the dataset, we adopt a simulation-based approach. We integrate these traffic flows into the benign MAWI traffic traces. Specifically, we thoroughly analyze the evaluated 73 DPI evasion attacks proposed in [4, 10, 23], and implement a simulator that injects the modifications incorporated in these to evade the DPI middlebox, into the MAWI traffic traces (i.e., PCAP files). We acknowledge that this PCAP-level simulation might potentially introduce some slight divergence in behaviors, compared to what happens in live DPI evasion attacks (e.g. the timings associated with the injected packets is estimated in our simulation, which could differ from the live case due to unpredictable congestion effects). However, we argue that (1) these differences do not disrupt the underlying mechanisms of DPI evasion attacks and thus, do not affect our evaluation results/conclusions; and (2) the open-source implementations released by [4, 10, 23] do not support a direct replay of the attack traces and, thus we are unable to verify those directly; however, the simulations are a faithful reproduction of those attack behaviors and we assume them to yield similar success against DPI middleboxes.

**Baselines**. We compare CLAP's performance with that of 2 baselines to showcase its effectiveness. Baseline #1 reuses the same pipeline as CLAP, but (1) removes all gate weight features from its context profiles, (2) limits the length of the profile to single packet. In other words, only intra-packet context features are considered to eliminate inter-packet context information. One can think of this as a "temporal context" agnostic version of CLAP. Baseline #2 is the faithful reproduction of a state-of-the-art anomaly-detection-based IDS [17]. This IDS also leverages autoencoders as its underlying model and the paper claims that it targets a broad spectrum of attacks. We will show the results with these baselines, in comparison with those with CLAP's in Figures 7, 8 and 9.

### 4.2 Effectiveness Analysis

*Takeaway*: CLAP achieves an average AUC-ROC score of **0.963** (vs. 0.846 [-12.1%] for Baseline #1 and 0.498 [-48.3%] for #2), Equal Error Rate (EER) of **0.061** (vs. 0.198 [+224.6%] for Baseline #1 and 0.502

[+723.0%] for #2) in detecting attacks; it also achieves a **94.6%** Top-5 (meaning the localized packet is within a window of five packets from the identified point), **91.0%** Top-3 (the localized packet is within a window of 3 packets), and **76.8%** Top-1 (exactly identify the position of the adversarial packet) accuracy in localizing the positions of injected adversarial packets.

**Evaluation Metrics**. For our evaluations of CLAP, we use the following, most commonly adopted metrics in the state-of-the-art ML-based networked systems research [2, 13, 14, 17]. "Area Under of the Receiver Operating Characteristic Curve" (AUC-ROC) is our first metric of interest; it captures the trade-off between the True Positive Rate (TPR) and the False Positive Rate (FPR) by considering a comprehensive set of reconstruction error thresholds, and is considered a comprehensive metric for evaluating binary classifiers. The higher the AUC-ROC, better the classifier. Complementary to AUC-ROC, EER is the point on ROC curve where the False Positive Rate is equal to the False Negative Rate; this metric is sometimes considered to be a more balanced metric to evaluate a classifier compared to AUC-ROC. The lower the ERR, the more accurate the detection. Lastly, *Top-N Hit Rate* is defined by the percentage of the connections where the context profiles with the $N$-highest reconstruction errors produced by CLAP, intersect with actual injected adversarial packets from among all tested connections. In other words, it measures the accuracy of the localization step in CLAP's "localize-and-estimate" adversarial score generation approach, in terms of how well top candidates marked by CLAP capture the real adversarial packets. With accurate localization, CLAP can pinpoint the exact positions of adversarial packets for the purposes of forensic analyses. The higher the hit rate, the more accurate the localization.

**Detection Performance**. For evaluating CLAP's detection accuracy, we must first inject the attacks into the benign traffic dataset to form its adversarial counterpart for the testing samples. As previously discussed, DPI evasion attacks all function by either injecting new packets, or modifying existing packets in a connection. However, the exact injection and modification methods differ from attack to attack. We therefore provide a breakdown/taxonomy of the evaluated attacks and the projects they were discovered from.

Figure 7 shows the evaluation results of attacks presented in [23]. Since all of these attacks work at TCP layer by modifying the TCP header fields of injected or existing packets, they can be categorized based on (1) the type (i.e., TCP flags) of the packets that are injected or altered, and (2) the exact header modifications. In Figure 7, for each attack (i.e., the title of each bar plot), the first line is the key TCP flag (e.g., SYN) in the injected/altered packets, and the second line indicates how header fields are modified (e.g., Bad SEQ means changing the SEQ number to an invalid value). As shown in Table 1, CLAP outperforms both baselines in detecting evasion strategies from [23] by a significant margin.

Figure 8 shows the results relating to attacks proposed in [10]. Unlike [4, 23], these attacks target DPI-based traffic classification systems (e.g., is it YouTube traffic or not?) by manipulating header fields across the TCP/IP layers in certain packets in a connection; these classifiers make decisions by examining an arbitrarily long subset of data packets called the matching packets, which are transferred after the initial TCP handshake. To evade the classifier, evasion packets are inserted in front of all of these matching packets. However, without knowing what classification possibilities the attacker is trying to evade we cannot know the exact number of evasion packets that are inserted (i.e., they vary depending on the content that requires evasion). Hence, we simulate two extreme cases wherein there is (a) a single matching packet *min* and (b) where there are a $max = 5$ matching packets as considered in the [10] paper. Note that these packets are sent after the connection transitions into the `ESTABLISHED` state. With the above strategies we show the corresponding detection accuracies in Figure 8. Again, CLAP outperforms both baselines in detecting evasion strategies from [10] by a significant margin, as reported in Table 1

Lastly, Figure 7 reports the detection accuracies with respect to attacks from [4]. Unlike [23] and [10], these attacks (1) are executed/injected rather blindly and all data packets (i.e., packets transferred in `ESTABLISHED` TCP state after completing the handshake) are altered; and (2) consist up to 2 different modifications in one attack strategy. Therefore, in Figure 9, when labeling each attack, the first and second lines describe the first and second modification type ("/" means only one modification for that strategy), respectively. We again observe significant gains in terms of detection performance, with CLAP over the other baselines. Once again, CLAP wins over both baselines in terms of the detection accuracy against attack strategies from [4] by a considerable margin, as shown in Table 1.

*Analysis.* From the detection performance results, alluded to above, across different evasion strategies, we have the following observations: (1) CLAP consistently outperforms both Baselines #1 and #2, in terms of all the considered metrics and across all attack strategies; (2) all evaluation metrics indicate that CLAP tends to perform exceptionally well on all evasion strategies that involve inter-packet context violations (e.g., Pure RST strategy from [23] with an AUC-ROC of 0.999, positioned at row #2 column #7 in Figure 7) and most strategies that involve intra-packet context violations (e.g., Invalid Data Offset/Bad TCP Checksum from [4], positioned at row #1, column #1 in Figure 9); however, the performance is not as good on a small fraction of strategies involving intra-packet context violations (e.g., SYN w/ Payload attack from [23] with an AUC-ROC of 0.782, positioned at row #3, column #7 in Figure 7 and, Low TTL attack (min) from [10] with an AUC-ROC of 0.851, positioned at row #1, column #7 in Figure 8). We believe that this is because, for these intra-packet context violations, even with amplification features, the quantum of the adversarial perturbation/modification imposed onto the packet is still considered too insignificant by the autoencoder in CLAP, to be spotted. Note however that, even so, CLAP still (1) detects a large fraction of evasion attempts of these types (an AUC-ROC > 0.75); and (2) provides considerable improvements (> 30%) over both baselines.

**Localization Accuracy**. Next, we evaluate the accuracy of CLAP in localizing the injected adversarial packets; this is the first step of the "localize-and-estimate" approach in computing the adversarial score for a given connection (see Section 3.3). Here, our goal is: (1) evaluating the localization ability of CLAP, and (2) providing an analysis of the design choice made in CLAP in association with

| Results/ Approach | Mean AUC-ROC for [23] | Mean EER for [23] | Mean AUC-ROC for [10] | Mean EER for [10] | Mean AUC-ROC for [4] | Mean EER for [4] |
|---|---|---|---|---|---|---|
| CLAP | **0.953** | **0.072** | **0.952** | **0.082** | **0.988** | **0.024** |
| Baseline #1 | 0.829 (-13.0%) | 0.218 (+202.8%) | 0.805 (-15.4%) | 0.232 (+182.9%) | 0.913 (-5.9%) | 0.133 (+454.2%) |
| Baseline #2 | 0.501 (-47.4%) | 0.501 (+595.8%) | 0.500 (-47.5%) | 0.500 (+509.8%) | 0.491 (-50.3%) | 0.504 (+2000%) |

**Table 1: Breakdown of average detection performance for strategies in [4, 10, 23]**

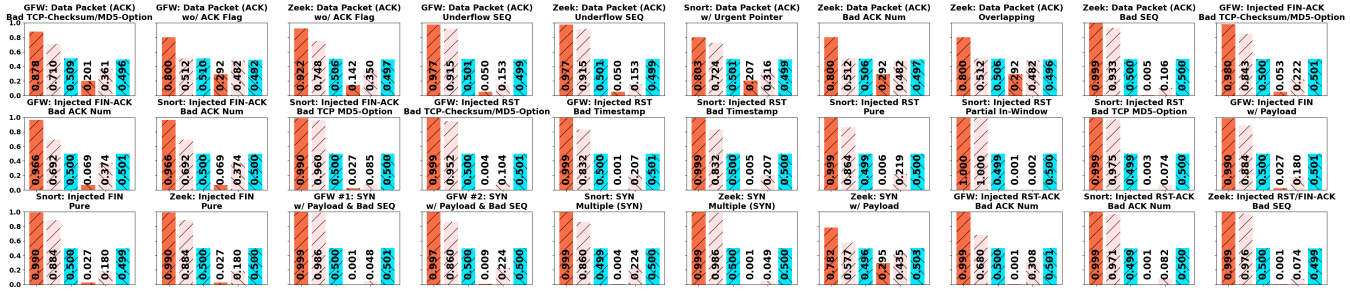


**Figure 7: Per-strategy detection accuracy of CLAP in detecting the different attacks (shown in title) from [23]**

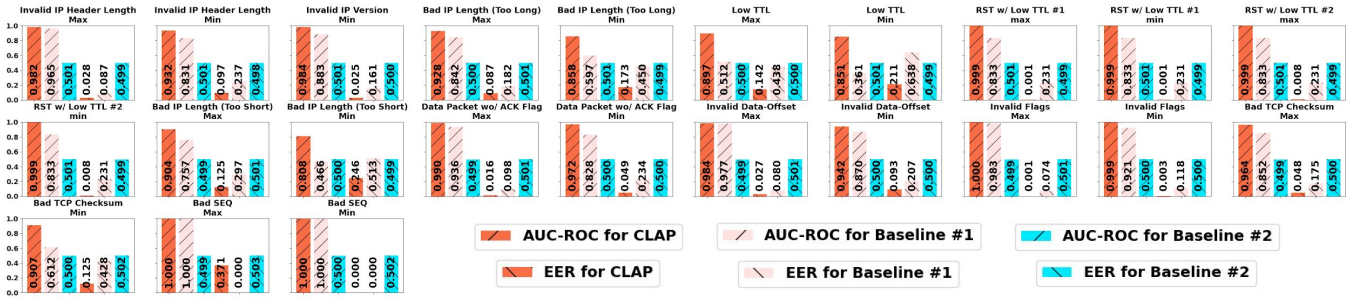


**Figure 8: Per-strategy detection accuracy of CLAP in detecting the different attacks (shown in title) from [10]**

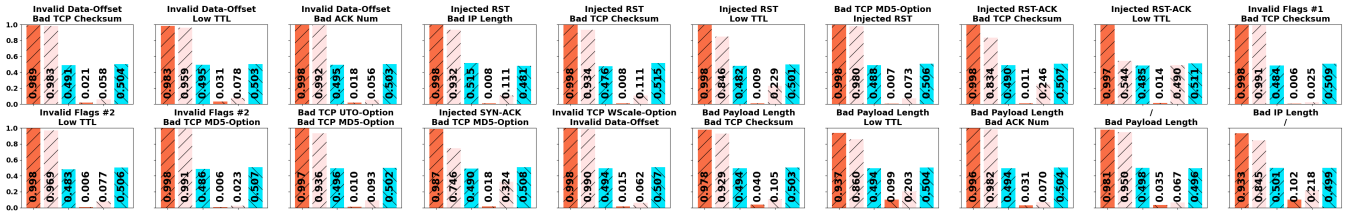


**Figure 9: Per-strategy detection accuracy of CLAP in detecting the different attacks (shown in title) from [4]**

the computation of adversarial score. We do not consider baseline approaches for localization evaluations, because neither of the baselines are sufficiently accurate in detecting adversarial packets in the first place and furthermore, do not consider the temporal dependencies across a train of packets (indicating their inadequacy in terms of localization abilities). As previously discussed, we use the Top-N hit rate as the metric to measure the localization accuracy. We report the Top-5, Top-3 and Top-1 rates in Figure 10, 11 and 12 with the same categorizations adopted in the detection accuracy evaluations. Specifically, the three bars in each plot from left to right correspond to Top-5, Top-3 and Top-1 rates, respectively.

*Analysis.* Our general observations with regards to the localization results are as follows: (1) as one might expect, the accuracy varies when we require different levels of localization; specifically, achieving Top-1 (with an average accuracy across all strategies of 76.8%) localization (as expected) is much more challenging than Top-5 (average accuracy 94.6%) and Top-3 (average accuracy 91.0%) localization. This in a way highlights the importance of our design choice with regards to using the "mean across a sliding window"

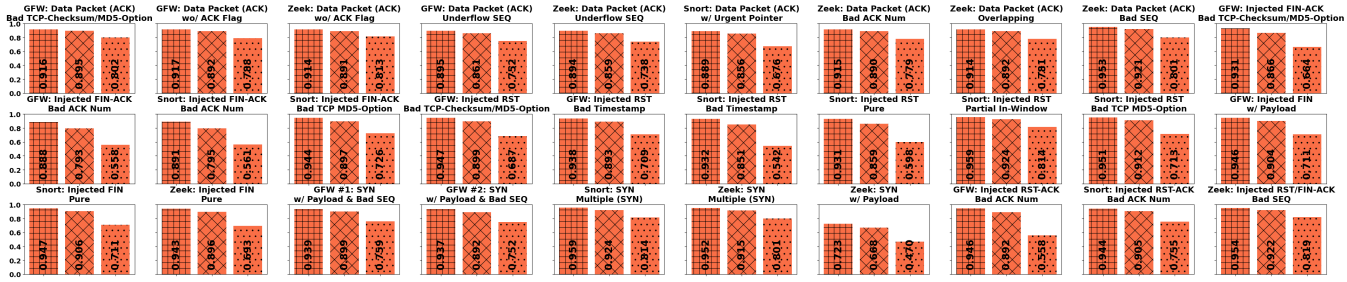


**Figure 10: Per-strategy localization accuracy of CLAP in detecting the different attacks (shown in title) from [23]**

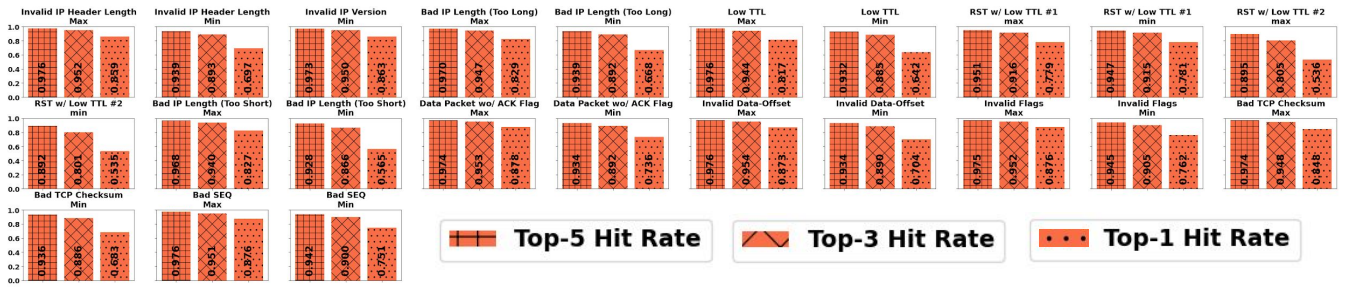


**Figure 11: Per-strategy localization accuracy of CLAP in detecting the different attacks from [10]**

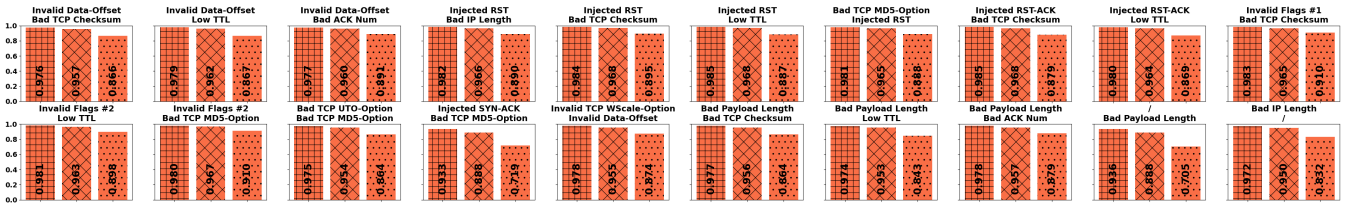


**Figure 12: Per-strategy localization accuracy of CLAP in detecting the different attacks from [4]**

surrounding the context profile with the maximum reconstruction error as the adversarial score described previously; (2) the better the localization, the better the detection performance, as more accurate localization yields richer and more directive context information for CLAP to utilize.

## 4.3 Case Studies

Having provided a holistic picture of the performance of CLAP across a large set of DPI evasion attacks, we now pick two concrete attacks to exemplify why CLAP can detect them accurately. Specifically, we pick an attack that is discovered due to violations of the inter-packet context and an attack that is caught for violating the intra-packet context. These case studies also serve to validate the core design of CLAP.

Note here that we have undertaken a careful analysis of all the attacks reported and are able to reason about them; we believe that the two attacks that we choose to magnify in this section are representative of the entire set. We also provide a summarized categorization of all 73 evaluated strategies from [4, 10, 23] into the two context violation types, in Appendix 8. Recall that our Baseline #1 approach does not account for inter-packet context since it only considers single-packet features (gate weights from RNN are ignored). In other words, it only captures intra-packet context towards detecting DPI evasion attacks; thus, any evasion strategy that exhibits disparity between CLAP and Baseline #1, is considered as one that invokes a inter-packet violation for evasion. Following this principle, if the AUC-ROC disparity between CLAP and Baseline #1 is greater than a chosen threshold $TH_{inter}$ (we pick 0.15 here), we categorize the attack as one invoking an inter-packet context violation; otherwise, the attack is considered as causing an intra-packet context violation. Based on this rule of thumb, we categorize 27 out of the 73 evasion attack strategies as primarily inter-packet violations, and 49 as primarily intra-packet violations. Note that this does not mean these two categories are strictly disjoint. While some strategies violate both contexts, we categorize them based on the main violation (manifested by the significant accuracy discrepancy).

**Inter-packet context violation**. Recall that the inter-packet context refers to the inter-relationships across different packets in a connection. Among all evaluated attacks, there are many that evade

| Category/ Metric | Inter-packet Context Violation (24 Strategies) | | Intra-packet Context Violation (49 Strategies) | |
|---|---|---|---|---|
| | CLAP | Basline #1 | CLAP | Baseline #1 |
| Mean AUC-ROC | **0.925 (+37.6%)** | 0.672 | **0.980 (+6.2%)** | 0.923 |
| Mean EER | **0.109 (-70.0%)** | 0.364 | **0.039 (-68.3%)** | 0.123 |

**Table 2: Breakdown of inter vs intra-packet violation detections**

DPI detection by injecting manipulated packets that cause a behavioural discrepancy between the DPI and endhost only when the TCP state machine is in a specific state. For example, the RST with Bad Timestamp strategy from [23] injects a RST packet with an invalid TCP timestamp option value, specifically when the target connection is in the `SYN_RECV` state. Since this evasion packet is then only dropped by endhost, but accepted by target DPIs (e.g. Zeek and GFW), it tricks the DPI into disengaging the monitoring subsequently. The end host, which is in exactly `SYN_RECV` TCP state, receives the follow-up packets that effectively bypass DPI detection. We determine that this strategy primarily violates the inter-packet context because it (1) it is successful only when the current TCP state is in a specific state (i.e., a Bad Timestamp RST packet would not cause behavioural discrepancy between DPI and endhost in `ESTABLISHED` state) and, (2) the validity of the TCP timestamp option is determined by the timestamps in previous packets. In other words, in order to detect this strategy accurately, CLAP must utilize the inter-packet context/relationship. It asserts if a given packet (1) bears a bad timestamp relative to previous packets, and (2) if the current TCP state is `SYN_RECV`. As expected, in this case, Figure 2 shows that CLAP performs much better than Baseline #1 (> 35%).

**Intra-packet context violation**. Recall that the intra-packet context captures the relationships across different header fields in the same packet. Several evaluated attacks inject shadow packets in front of legal data packets. In these shadow packets, certain TCP/IP header field values are altered such that they are rejected by end point's rigorous TCP implementation but accepted by DPI's simplified version. This way, the data packets that follow the injected shadow packets are cloaked and evade DPI detection. For example, the *Invalid IP Version* attack from [10] injects packets with an incorrect IP Version header value (e.g 5 as there is no IPv5) to trigger the discrepancy between the DPI and endhost. CLAP detects this attack by learning the intra-packet context i.e., legitimate packets should not have a IP Version of 5; it is thus able to flag any packet that violates this requirement. The aggregate accuracies in Table 2 show that CLAP performs considerably better than Baseline #1.

## 4.4 Runtime Overhead Analysis

Finally, we evaluate how efficiently CLAP processes network traffic streams with our current implementation. Given that the training phase of CLAP is completely offline, the critical overhead would be its runtime overhead, i.e., the model processing efficiency of CLAP's testing phase (Stage (d)). For reference, we also measure the runtime model inference overhead of the open-source version released by [17], which is considered to be the state-of-the-art autoencoder-based IDS, and show that CLAP achieves a significantly higher inference/processing speed. We use the default hyper-parameters to train the model from [17], as detailed in Table 6.

| Model/ Metric | CLAP | [17] |
|---|---|---|
| Packets/Second | **2,162.2 (+49.7%)** | 1,444.5 |
| Connections/Second | **97.0 (+49.7%)** | 64.8 |

**Table 3: Model processing throughput**

**Setup**. We run the pipelines of both CLAP and [17] on a desktop with Intel Xeon E3-1225 Processor (3.2Ghz, 4 cores), 20GB RAM and disabled GPU support. We constrain both pipelines to only use one logical core for fair comparison. We note that [17] claims a higher processing throughput using a C++ implementation, but find that its released Python version [16] is much slower. Furthermore, it is fair to compare our Python-based prototype implementation with its Python-based baseline counterpart – we leave the task of re-implementing CLAP in more efficient languages (e.g. C/C++) for improving its overhead performance as future work. Our test adversarial traffic corpus consists of 92,262 packets and 6,424 connections.

**Throughput**. We compare the model processing throughputs of CLAP and [17] in Table 3. We see that CLAP outperforms [17] by a margin of $\approx 50\%$, while detecting DPI evasion attacks with much higher accuracies, as shown in previous subsections. However, we acknowledge that CLAP is not designed for detecting other network attacks, that are captured by [17]. We believe that the performance disparity is because [17] uses an ensemble (forest) of 10 small autoencoders to process subsets of different features pertaining to different attacks (details in Table 6); it merges the separate reconstruction errors into an aggregate error with another autoencoder, making it overall a heavy procedure i.e., it cannot process packets as fast as CLAP (with only one autoencoder).

# 5 DISCUSSION

**Ethical Implications**. In recent years, DPI evasion attacks have been considered as a means to bypass censorship systems [4, 10, 22, 23]. This is because, censors are essentially DPI middleboxes. We acknowledge that malicious DPI evasion attacks cannot be fully separated from censorship circumvention practices (since they rely on DPI evasion attacks). However, while CLAP can inherently enable censorship i.e., potentially expose censorship circumvention, we do not advocate censorship and our work is completely inspired by research questions.

**Feature Completeness**. Currently, CLAP covers all non-optional, non-tuple-related header fields. One might wonder if new evasion strategies that are outside our current feature set are viable. Since CLAP currently does not consider (1) uncommon IP and TCP option fields that are variable-sized (i.e., hard to encode as fix-sized features); and (2) payloads as they are unstructured data (i.e., with no fixed formats), we envision that these are the most likely surfaces

that future emerging evasions can manipulate. To tackle this challenge, we plan to explore the remaining headers (TCP/IP options) and payload contents, possibly via embedding techniques [24] that can map unstructured/variable-sized inputs to fix-sized vectors in future work.

**Generalizability**. In this work, we focus on evasions via attacks of TCP/IP protocols as these protocols are the most popular/common parts of most network protocol suites used today, and thus attract the vast majority of evasion efforts. However, we believe that the pipeline and core design of CLAP can be easily transferred and applied to other stateful protocols (e.g., HTTP) with minimum effort. As long as one can define the internal states inside the protocol implementation, CLAP is expected to learn packet contexts that help enable adversarial evasions.

**Deployability**. As shown in Section 4.4, the model inference time for CLAP is considerably faster than the prototype released by a state-of-the-art ML-based IDS, making CLAP potentially deployable on middleboxes on low-workload networks. We acknowledge that compared to signature-based general-purpose IDSs such as Zeek, our overall processing speed is considerably slower, but argue that (1) we target specifically DPI evasion attacks that cannot be captured by general-purpose IDS; (2) a re-implementation of CLAP in more efficient languages (e.g. C/C++) can greatly help reduce its overhead (this is left to future work); and (3) the rising trend of GPU-based ML acceleration can also be of great help in boosting the runtime performance of CLAP, which is again, part of our future plan.

We acknowledge that even though we have demonstrated that CLAP comes with very low error rates in both detection and localization tasks in Section 4, there can still be a number of false alarms that might be generated. However, we emphasize that CLAP can be tuned using a pre-defined adversarial score threshold (see Section 3.3), which would allow the deployers of CLAP the flexibility to freely choose the desired trade-off between true positive and false positive rates. In addition, one can down sample the alarms generated by CLAP, and only selectively inspect/analyze those packets that have the highest associated adversarial scores. We believe these two strategies can significantly control the number/ratio of false alarms.

**Attacker Adaptation**. We are aware of the emerging research that exploits inherent vulnerabilities against ML models such that specifically crafted inputs, known as adversarial examples, can bypass ML models with minimum differences compared to its benign counterparts. We envision the possibility that attackers could generate such examples (packets) to hide from CLAP's detection. However, we point out that most of the adversarial example attacks from the adversarial ML community, consider end-to-end models so that it is rather easy to obtain the gradients from such end-to-end models to guide the generation of adversarial examples. In CLAP our design choices of using a multi-step pipeline (i.e., RNN + autoencoder) makes it extremely challenging, if possible at all.

## 6 CONCLUSIONS

In this paper, we design and implement a framework (called CLAP) for detecting DPI evasion attacks that have emerged recently. The key observation that drives the design of CLAP is that these attacks often violate either legitimate relationships between the headers within a packet, or relationships across headers in a train of packets. We construct a packet context-profile that captures these relationships and train a set of appropriate ML models to learn the legitimate (benign) distributions of such profiles. During test time, CLAP checks if encountered packets conform with these distributions and flags those that do not as evasion attempts. Our comprehensive evaluations on a large variety of attacks from three different recent efforts show that CLAP (a) is extremely effective in detecting evasion patterns, and (b) significantly outperforms ML methods agnostic to packet context-profiles.

## ACKNOWLEDGEMENT

This research was sponsored by the U.S. Army Combat Capabilities Development Command Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Combat Capabilities Development Command Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on. We also thank the anonymous reviewers and our anonymous shepherd for their comments and suggestions that helped significantly improve our paper.

## REFERENCES

[1] [n.d.]. MAWI Working Group Traffic Archive. http://mawi.wide.ad.jp/mawi/. Accessed: 2020-06-26.
[2] Azeem Aqil, Karim Khalil, Ahmed OF Atya, Evangelos E Papalexakis, Srikanth V Krishnamurthy, Trent Jaeger, KK Ramakrishnan, Paul Yu, and Ananthram Swami. 2017. Jaal: Towards network intrusion detection at isp scale. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. 134–146.
[3] P Neira Ayuso. 2008. The conntrack-tools user manual.
[4] Kevin Bock, George Hughey, Xiao Qiang, and Dave Levin. 2019. Geneva: Evolving Censorship Evasion Strategies. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2199–2214.
[5] Kenjiro Cho. 2017. Recursive lattice search: hierarchical heavy hitters revisited. In *Proceedings of the 2017 Internet Measurement Conference*. 283–289.
[6] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 1724–1734.
[7] Romain Fontugne, Patrice Abry, Kensuke Fukuda, Darryl Veitch, Kenjiro Cho, Pierre Borgnat, and Herwig Wendt. 2017. Scaling in internet traffic: a 14 year and 3 day longitudinal study, with multiscale analyses and random projections. *IEEE/ACM Transactions on Networking* 25, 4 (2017), 2152–2165.
[8] Jia Jingping, Chen Kehua, Chen Jia, Zhou Dengwen, and Ma Wei. 2019. Detection and recognition of atomic evasions against network intrusion detection/prevention systems. *IEEE Access* 7 (2019), 87816–87826.
[9] Christian Kreibich, Mark Handley, and V Paxson. 2001. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *Proc. USENIX Security Symposium*, Vol. 2001.
[10] Fangfan Li, Abbas Razaghpanah, Arash Molavi Kakhki, Arian Akhavan Niaki, David Choffnes, Phillipa Gill, and Alan Mislove. 2017. lib• erate,(n) a library for exposing (traffic-classification) rules and avoiding them efficiently. In *Proceedings of the 2017 Internet Measurement Conference*. 128–141.
[11] Shasha Li, Shitong Zhu, Sudipta Paul, Amit Roy-Chowdhury, Chengyu Song, Srikanth Krishnamurthy, Ananthram Swami, and Kevin S Chan. 2020. Connecting the Dots: Detecting Adversarial Perturbations Using Context Inconsistency. *arXiv preprint arXiv:2007.09763* (2020).
[12] Yong Liu, Ruiping Wang, Shiguang Shan, and Xilin Chen. 2018. Structure inference net: Object detection using scene-level context and instance-level relationships. In *Proceedings of the IEEE conference on computer vision and pattern*

*recognition*. 6985–6994.

[13] Gonzalo Marín, Pedro Casas, and Germán Capdehourat. 2018. Rawpower: Deep learning based anomaly detection from raw network traffic measurements. In *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*. 75–77.

[14] Gonzalo Marín, Pedro Casas, and Germán Capdehourat. 2019. DeepSec meets RawPower-Deep Learning for Detection of Network Attacks Using Raw Representations. *ACM SIGMETRICS Performance Evaluation Review* 46, 3 (2019), 147–150.

[15] William Mendenhall, Robert J Beaver, and Barbara M Beaver. 2012. *Introduction to probability and statistics*. Cengage Learning.

[16] Yisroel Mirsky. 2020. Kitsune-py. https://github.com/ymirsky/Kitsune-py.

[17] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. 2018. Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society. http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018_03A-3_Mirsky_paper.pdf

[18] Jon Postel. 1981. RFC0791: Internet Protocol.

[19] Anantha Ramaiah, R Stewart, and Mitesh Dalal. 2010. Improving TCP's robustness to blind in-window attacks. *Internet Engineering Task Force, RFC* 5961 (2010).

[20] IETF RFC793. 1981. Transmission control protocol. *J. Postel. September* 981 (1981).

[21] Kaihua Tang, Hanwang Zhang, Baoyuan Wu, Wenhan Luo, and Wei Liu. 2019. Learning to compose dynamic tree structures for visual contexts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 6619–6628.

[22] Zhongjie Wang, Yue Cao, Zhiyun Qian, Chengyu Song, and Srikanth V Krishnamurthy. 2017. Your state is not mine: a closer look at evading stateful internet censorship. In *Proceedings of the 2017 Internet Measurement Conference*. 114–127.

[23] Zhongjie Wang, Shitong Zhu, Yue Cao, Zhiyun Qian, Chengyu Song, Srikanth V Krishnamurthy, Kevin S Chan, and Tracy D Braun. 2020. SYMTCP: Eluding Stateful Deep Packet Inspection with Automated Discrepancy Discovery. In *Network and Distributed System Security Symposium (NDSS)*.

[24] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[25] Chengcheng Xu, Shuhui Chen, Jinshu Su, Siu-Ming Yiu, and Lucas CK Hui. 2016. A survey on regular expression matching for deep packet inspection: Applications, algorithms, and hardware platforms. *IEEE Communications Surveys & Tutorials* 18, 4 (2016), 2991–3029.

[26] Chong Zhou and Randy C Paffenroth. 2017. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 665–674.

# APPENDIX

## A MAWI TRAFFIC DATASET STATISTICS

As described in Section 4.1, here we list the basic statistics of the dataset used in our evaluations in Table 4.

## B RNN PREDICTION ACCURACY

We present the per-label accuracy breakdown along with the number of samples (packets) in Table 5. Overall, the RNN model in CLAP achieves an accuracy of 0.995 on its testing set.

## C FEATURE SET

Here we list the features we used in building context profiles in Table 7. Note that, as previously described, only TCP and IP layer header features (i.e., #1-#32) are used for training the RNN model.

## D PER-CONTEXT CATEGORIZATION OF EVASION STRATEGIES

We categorize the 73 evasion strategies that were evaluated in Section 4.2 in Table 8 according to the context they primarily violate, using the categorization scheme proposed previously.

## E MODEL HYPER-PARAMETERS

We summarize the hyper-parameters used in training the RNN, autoencooder models in CLAP, and the autoencoder models used in Baseline #1 and #2 for our evaluations, in Table 6.

| **Original Trafffc Archive** | | | |
|---|---|---|---|
| Capture Timestamp | 04/07/2020 14:00 JPT | # Packets | 111,851,572 |
| # TCP/IPv4 Packets | 51,692,562 | / | |
| **Sampled (Used) Dataset** | | | |
| # TCP/IPv4 Packets | 540,353 | # TCP/IPv4 Connections | 37,622 |
| # TCP/IPv4 Packets (Training) | 448,091 | TCP/IPv4 Connections (Training) | 31,198 |
| # TCP/IPv4 Packets (Testing) | 92,262 | # TCP/IPv4 Connections (Testing) | 6,424 |

**Table 4: Statistics of used MAWI dataset**

| TCP State/ Packet Window Classification | SYN_SENT | SYN_RECV | ESTABLISHED | FIN_WAIT | CLOSE_WAIT | LAST_ACK | TIME_WAIT | CLOSE |
|---|---|---|---|---|---|---|---|---|
| In-Window | 0.999678 (6166) | 1.0 (18906) | 0.994733 (47664) | 0.996628 (2966) | 0.988205 (3117) | 0.993641 (2988) | 0.992513 (2805) | 0.987467 (2314) |
| Out-of-Window | 0.0 (2) | / | 0.953246 (1155) | 0.911764 (34) | 0.694444 (36) | 0.814818 (27) | 0.95 (20) | 0.956521 (23) |

**Table 5: Per-label breakdown of RNN accuracy**

| | RNN (GRU-based) in CLAP | | | | Autoencoder in CLAP | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Model Parameter | # Layer(s) | Input Size | Hidden Size (Gate Size) | # Epochs | # Layer(s) | Input Size | Length of Context Profile Stacking | Bottleneck Layer Size | # Epochs |
| Value | 1 | 32 | 32 | 30 | 7 | 345 | 3 | 40 | 1,000 |
| | Autoencoder in Baseline #1 | | | | Ensembled Autoencoders in Baseline #2 | | | | |
| Model Parameter | # Layer(s) | Input Size | Bottleneck Layer Size | # Epochs | # Layer(s) | Ensemble Size (# Autoencoders) | Total Input Size | Average Input Size (Per Autoencoder) | Average Bottleneck Layer Size | # Epochs |
| Value | 3 | 51 | 5 | 1,000 | 1 | 16 | 100 | 6.25 | 4.68 | 1 |

**Table 6: Hyper-parameters used in the paper**

| Index | Type | Sementic | Index | Type | Sementic | Index | Type | Sementic | Index | Type | Sementic |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **TCP Layer Features** | | | 17 | Integer | Payload Length | **IP Layer Features** | | | **Amplification Features (not included for training RNN)** | | |
| 1 | Binary | Packet direction | 18 | Integer | Option: Maximum Segment Size | 26 | Integer | Length | 33-45 | Binary | Out-of-Range indicators for numeric TCP features |
| 2 | Integer | SEQ number (incremental) | 19 | Integer | Option: Timestamp Value (TSVal) | 27 | Integer | Time-To-Live | 46-50 | Binary | Out-of-Range indicators for numeric IP features |
| 3 | Integer | ACK number (incremental) | 20 | Integer | Option: Timestamp Echo Reply (TSecr) | 28 | Integer | Header Length | 51 | Binary | TCP Payload Length correctness (#17 = #26 - #28 - #4) |
| 4 | Integer | Data Offset | 21 | Integer | Option: Window Scale | 29 | Binary | Checksum validity | **Gate Weights from GRU** | | |
| 5-13 | Categorical | Flags (one-hot encoded) | 22 | Integer | Option: User Timeout | 30 | Integer | IP Version | 52-83 | Float | Update Gates |
| 14 | Integer | Window Size | 23 | Binary | Option: MD5 Header Validity | 31 | Integer | Type of Service | 84-115 | Float | Reset Gates |
| 15 | Binary | Checksum vadility | 24 | Integer | TCP Timestamp | 32 | Binary | Existence of non-standard IP options | | | |
| 16 | Integer | Urgent Pointer | 25 | Integer | Frame Timestamp | | | | | | |

**Table 7: List of features in context profile**

| From | Strategy Name | From | Strategy Name | From | Strategy Name |
|---|---|---|---|---|---|
| **Inter-packet Context Violation** | | | Zeek: Data Packet (ACK) Bad SEQ | | Bad SEQ (Min) |
| [23] | GFW: Data Packet (ACK) Bad TCP-Checksum/MD5-Option | [23] | GFW: Injected FIN-ACK Bad TCP-Checksum/MD5-Option | [10] | Data Packet wo/ ACK Flag (Max) |
| | GFW: Data Packet (ACK) wo/ ACK Flag | | Snort: Injected FIN-ACK Bad TCP MD5-Option | | Data Packet wo/ ACK Flag (Min) |
| | Zeek: Data Packet (ACK) wo/ ACK Flag | | GFW: Injected RST Bad TCP-Checksum/MD5-Option | | Invalid Data-Offset (Max) |
| | Zeek: Data Packet (ACK) Bad ACK Num | | Snort: Injected RST Pure | | Invalid Data-Offset (Min) |
| | Zeek: Data Packet (ACK) Overlapping | | Snort: Injected RST Partial In-Window | | Invalid Flags (Max) |
| | GFW: Injected FIN-ACK Bad ACK Num | | Snort: Injected RST Bad TCP MD5-Option | | Invalid Flags (Min) |
| | Snort: Injected FIN-ACK Bad ACK Num | | GFW: Injected FIN w/ Payload | | Bad TCP Checksum (Max) |
| | GFW: Injected RST Bad Timestamp | | Snort: Injected FIN Pure | | Bad SEQ (Max) |
| | Snort: Injected RST Bad Timestamp | | Zeek: Injected FIN Pure | | Bad SEQ (Min) |
| | Zeek: SYN w/ Payload | | GFW #1: SYN w/ Payload & Bad SEQ | | Invalid Data-Offset Bad TCP Checksum |
| | GFW: Injected RST-ACK Bad ACK Num | | GFW #2: SYN w/ Payload & Bad SEQ | | Invalid Data-Offset Low TTL |
| [10] | Bad IP Length (Too Long) (Min) | | Snort: SYN Multiple (SYN) | | Invalid Data-Offset Bad ACK Num |
| | Low TTL (Max) | | Zeek: SYN Multiple (SYN) | [4] | Injected RST Bad IP Length |
| | Low TTL (Min) | | Snort: Injected RST-ACK Bad ACK Num | | Injected RST Bad TCP Checksum |
| | RST w/ Low TTL #1 (Max) | | Zeek: Injected RST/FIN-ACK Bad SEQ | | Bad TCP MD5-Option Injected RST |
| | RST w/ Low TTL #1 (Min) | | Invalid IP Header Length (Max) | | Invalid Flags #1 Bad TCP Checksum |
| | RST w/ Low TTL #2 (Max) | | Invalid IP Header Length (Min) | | Invalid Flags #2 Low TTL |
| | RST w/ Low TTL #2 (Min) | [10] | Invalid IP Version (Min) | | Invalid Flags #2 Bad TCP MD5-Option |
| | Bad IP Length (Too Short) (Min) | | Bad IP Length (Too Long) (Max) | | Bad TCP UTO-Option Bad TCP MD5-Option |
| | Bad TCP Checksum (Min) | | Bad IP Length (Too Short) (Max) | | Invalid TCP WScale-Option Invalid Data-Offset |
| [4] | Injected RST Low TTL | | Data Packet wo/ ACK Flag (Max) | | Bad Payload Length Bad TCP Checksum |
| | Injected RST-ACK Bad TCP Checksum | | Data Packet wo/ ACK Flag (Min) | | Bad Payload Length Low TTL |
| | Injected RST-ACK Low TTL | | Invalid Data-Offset (Max) | | Bad Payload Length Bad ACK Num |
| | Injected SYN-ACK Bad TCP MD5-Option | | Invalid Data-Offset (Min) | | / Bad Payload Length |
| **Intra-packet Context Violation** | | | Invalid Flags (Max) | | Bad IP Length / |
| [23] | GFW: Data Packet (ACK) Underflow SEQ | | Invalid Flags (Min) | | |
| | Zeek: Data Packet (ACK) Underflow SEQ | | Bad TCP Checksum (Max) | | |
| | Snort: Data Packet (ACK) w/ Urgent Pointer | | Bad SEQ (Max) | | |

**Table 8: Per-context categorization of evasion strategies from [4, 10, 23] (with $TH_{inter} = 0.15$)**