Peek Inside the Closed World: Evaluating Autoencoder-Based Detection of DDoS to Cloud

Hang Guo
hangguo@isi.edu
USC/ISI

Xun Fan
xufan@microsoft.com
Microsoft

Anh Cao
anhcao@microsoft.com
Microsoft

Geoff Outhred
geoffo@microsoft.com
Microsoft

John Heidemann
johnh@isi.edu
USC/ISI

ABSTRACT

Machine-learning-based anomaly detection (ML-based AD) has been successful at detecting DDoS events in the lab. However published evaluations of ML-based AD have only had limited data and have not provided insight into *why* it works. To address limited evaluation against real-world data, we apply autoencoder, an existing ML-AD model, to 57 DDoS attack events captured at 5 cloud IPs from a major cloud provider. To improve our understanding for why ML-based AD works or not works, we interpret this data with feature attribution and counterfactual explanation. We show that our version of autoencoders work well overall: our models capture nearly all malicious flows to 2 of the 4 cloud IPs under attacks (at least 99.99%) but generate a few false negatives (5% and 9%) for the remaining 2 IPs. We show that our models maintain near-zero false positives on benign flows to all 5 IPs. Our interpretation of results shows that our models identify almost all malicious flows with non-whitelisted (non-WL) destination ports (99.92%) by learning the full list of benign destination ports from training data (the *normality*). Interpretation shows that although our models learn incomplete normality for protocols and source ports, they still identify most malicious flows with non-WL protocols and blacklisted (BL) source ports (100.0% and 97.5%) but risk false positives. Interpretation also shows that our models only detect a few malicious flows with BL packet sizes (8.5%) by incorrectly inferring these BL sizes as normal based on incomplete normality learned. We find our models still detect a quarter of flows (24.7%) with abnormal payload contents even when they do not see payload by combining anomalies from multiple flow features. Lastly, we summarize the implications of what we learn on applying autoencoder-based AD in production.

1 INTRODUCTION

Anomaly detection (AD), as known as one-class classification, is a popular strategy in detecting DDoS attacks and other types of network intrusion, enabling responses such as filtering. AD identifies malicious network traffic by profiling benign traffic and flagging traffic deviating from these benign profiles as malicious. AD thus implicitly assumes that one could profile all benign traffic patterns and infer the rest as malicious (closed world assumption [44]). Comparing to binary classification, another popular strategy in DDoS detection that profiles both benign and malicious traffic and look for traffic similar to these known malicious profiles, AD could identify both known and potentially unknown malicious traffic.

Machine learning (ML) techniques lead to a new class of DDoS detection study using ML-AD models such as one-class SVM ([5, 38, 45]) and neural networks ([10, 17, 20]). However, these studies usually suffer from two major limitations. First, they evaluate their methods with limited datasets, often using simulated traffic, or traffic from universities or laboratories traffic, or two public DDoS datasets (DARPA/MIT [14] and KDD Cup [41]). It is thus unclear how well their methods could detect *real-world* DDoS attacks in operational networks. Prior work has suggested that conclusion based on traffic from simulation and small environments do not generalize to real-world environments at larger scales [34]. The widely used DARPA/MIT ([14]) and KDD CUP datasets ([41]) are synthetic, 2-decades-old and with known problems, making them inadequate for contemporary research [7, 34]. Second, these studies usually do not interpret their models' detection and explain *why* their models work or not work. Without interpretation on why detection works it is difficult to understand the strengths and limitations of ML-based AD in DDoS detection and how one could make the best use of ML-based AD in production environment. Moreover, operators look for interpretation to gain confidence in and understand the limits of ML-based AD approaches [34].

Our paper acts as the first step towards addressing these two limitations in prior DDoS study using ML-based AD.

Our first contribution is to evaluate the detection accuracy of autoencoder, an existing ML-AD model, with real-world DDoS traffic from a large commercial cloud platform (§2.1). Specifically, we apply our models to 57 DDoS attack events captured from 5 cloud IPs of this platform between late-May and early-July 2019 (§2.2). Detection results show that our models detects almost all malicious attack flows to 2 of these 4 cloud IPs under attacks (at least 99.99%) but generates a few false negatives (5% and 9%) for the remaining 2 IPs (§3.1). We show that our models maintain near-zero false positives on benign traffic flows to all 5 IPs (§3.2).

Our second contribution is to interpret our detection results with feature attribution (§2.4.1) and counterfactual explanation (§2.4.2) and show why our models work on certain malicious flows but not the rest (§3.4). Specifically, we show that our models identify almost all malicious flows with non-whitelisted (non-WL) destination ports (99.92% of 1M) by learning the full list of benign destination ports from training data (the *normality*). We shows that although our models learn incomplete normality for protocols and source ports, they still identify most, if not all, malicious flows with non-WL

protocols (100.0% of 15k) and with blacklisted (BL) source ports (97.5% of 5k) but risk false positives. We also show that our models only detect a few malicious flows with BL packet sizes (8.5% out of 3k) by incorrectly inferring these BL sizes as normal based on incomplete normality learned. Lastly, we shows our models detect a quarter of flows with abnormal payload contents (24.7% of 8k) even when they do not see payload contents by combining anomalies from multiple flow features.

Out last contribution is to summarize the implications of what we learn on using autoencoder-based AD in production (§4). We show that autoencoder-based AD are better at detecting some anomalies than others, and that autoencoder-based AD works best with certain classes of anomalies (§4.1). We then show that noise-free training data is not always necessary for AD (§4.2). We lastly show that autoencoder-based AD and heuristic-based filters each has its own strengths and could be used jointly for the best of both worlds (§4.3).

2 DATASETS AND METHODOLOGY

We examine real-world DDoS attacks and interpret our ML-AD model in §3. Our data is based on a large commercial cloud platform (§2.1) from which we gather 57 DDoS attack events (§2.2). We then describe our specific ML-AD model (§2.3) and how we interpret it (§2.4).

2.1 Cloud Platform Overview

We study a large commercial cloud platform that is made up of millions of servers from over 100 data centers across 140 countries worldwide. This cloud platform hosts a wide range of services, from traditional websites to managed Internet-of-Thing infrastructure. Each of these cloud services is assigned one or more public virtual IPs (VIP). Global-facing services usually deploy multiple VIPs and use each VIP to serve a different geological regions.

This cloud platform has seen increasing DDoS attacks over the past years and deploys "in-house" DDoS detection and mitigation.

In-house detection begins by detecting *DDoS events* based on comparison of aggregate inbound traffic to an VIP to a threshold. Thresholds are either supplied by the owners of cloud services or decided by the system automatically from historical traffic patterns.

In-house mitigation employs filtering and rate limiting. After a DDoS event has been detected, each inbound packet to that VIP is checked and possibly dropped based on a series of heuristics. These heuristics are filters designed by domain experts to identify and filter known DDoS attacks. Remaining packets are rate limited, with any that pass the rate limiter passed to the VIP.

The in-house methods consider the end of a VIP's DDoS event as when inbound traffic rate to this VIP goes under DDoS threshold for a certain amount of time. The duration depends on the attack type; here we simplify it to 15 minutes. In-house mitigation is only applied when there is an ongoing DDoS event (called *war time*) and are not otherwise applied (during *peace time*).

2.2 Cloud DDoS Data

To evaluate and interpret ML-based AD, we obtain peace and war-time traffic packet captures (pcaps) from this cloud platform and extract benign customer traffic and malicious DDoS attack from these pcaps.

Traffic Pcaps: We obtain over 100 hours of inbound traffic pcaps to each of 5 VIPs we study. Each VIP's pcaps include all war-time traffic and partial peace-time traffic, observed at this VIP in a 8-day period between late-May and early-July 2019. Table 1 shows anonymized VIPs and specific times. The pcaps are sampled, retaining 1 in every 1000 packets. We use only partial peace-time traffic because we find adding more traffic does not increase our models' detection accuracy. Note that we observe SR3VP1 for extended 180 hours because this VIP receives much less traffic than the other VIPs.

The 5 VIPs we study come from 3 different cloud services, with three instances of service SR1 (SR1VP1 to SR1VP3), each in a different data center and physical location, one instance of service SR2 (SR2VP1) and one instance of service SR3 (SR3VP1).

Different VIPs see DDoS events (detected as described in §2.1) of different durations (Figure 1). Specifically, SR1VP3 sees a large number of mostly short DDoS events, with about 71% of its 49 events being 1 second or less (see red crosses at 1-second duration in Figure 1). The cloud platform's DDoS team suggests these very brief DDoS events are likely botnets randomly probing IPs. In comparison, SR1VP1 and SR1VP2 see smaller numbers of longer DDoS events, with a median duration for their 20 and 27 events of 121 and 140 seconds, see Figure 1. SR2VP1 is frequently attacked, with about 59 hours of war time, and sees DDoS events of broad range of durations (from 1 second to more than 14 hours). The cloud's DDoS team reports that this VIP is hosting a critical service, so long attacks are likely attempts to gain media attention. SR3VP1 reports zero attack events since service SR3 is rarely attacked. We thus use SR3VP1 to evaluate false positives with our detection methods.

Benign and Malicious Traffic: We report peace-time traffic as "benign traffic". While there may be very small attacks in the peace-time traffic, the cloud platform considers any such events too small to impact the service and does not filter them. We considered additional filtering to remove such attacks, but left them in to evaluate our system on noisy, real-world traffic [7]. War-time traffic is also a mix of benign user traffic and malicious attack traffic. We only consider the fraction of war-time traffic dropped by heuristic-based filters from in-house mitigation as malicious (annotated as "malicious traffic" hereafter), recalling these heuristics identify known attacks (§2.1). We only use these malicious traffic to evaluate our methods and ignore the rest of war-time traffic (that either get rate limited or forwarded to VIPs) since we do not have perfect ground truth for them.

Benign and Malicious Flow Features: While in-house mitigation filters at the packet level, using only per-packet features, our models improve detection by using flow-level statistics such as packet counts and maximum flow packet size. We thus aggregate packets from benign and malicious traffic as 5-tuple flows. We then extract the 23 flow features shown in Table 2 from the first 10 seconds (an empirical threshold) of each 5-tuple flow. We extract flows and flow features using Argus [26]. Our features are the standard ones provided by Argus; inventing new feature is not the focus of this paper. We ignore source IPs out of concern

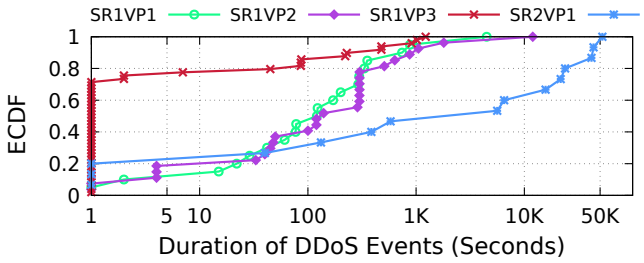


Figure 1: DDoS Events' Durations in Inbound Traffic Pcaps

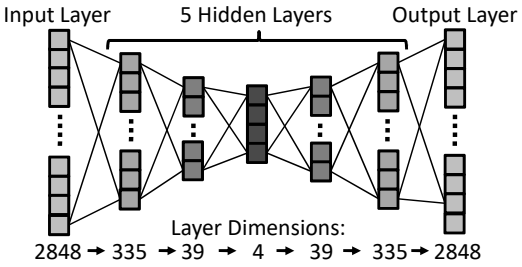


Figure 2: Architecture for Our Autoencoders

VIPs	Inbound Traffic Pcaps			Extracted Flows			Training	Threshold	Validation		Test	
	Peace Hrs	War Hrs	DDoS Evts	Benign	Malicious	DDoS Evts	Benign	Benign	Benign	Malicious	Benign	Malicious
SR1VP1	110.32	2.31	20	9,930k	119k	20	1,000k	59.5k	59.5k	59.5k	59.5k	59.5k
SR1VP2	96.96	5.44	27	13,107k	1,046k	20	1,000k	523k	523k	523k	523k	523k
SR1VP3	118.88	1.36	49	10,704k	90k	7	1,000k	45k	45k	45k	45k	45k
SR2VP1	57.73	58.89	15	5,469k	37k	10	1,000k	18.5k	18.5k	18.5k	18.5k	18.5k
SR3VP1	182.99	0	0	698k	0	0	548k	50k	50k	0	50k	0

Table 1: Summary of Inbound Traffic Pcaps and Extracted Traffic Flows Used in This Paper

that they may be spoofed. Three of our features (source port, destination port and protocol) have no ordering among their values, while the other features are ordered. Directly using unordered features would implicitly create an ordering among their values (for example, implying that protocol 5 is more similar to protocol 6 than protocol 4 is). We use one-hot encoding [8] to avoid this distortion. Specifically, we map protocol into 256 one-hot features (is_proto_0, is_proto_1, ... is_proto_255), each with a binary value. Similarly, we map ports into 1286 one-hot features, each representing a group of 51 adjacent ports (1 to 51, 52 to 102, ... 65485 to 65535), with port 0 used to indicate both illegal TCP/UDP port zero and non-existent port number in non-TCP-UDP flows. (We group every 51 ports because otherwise we will need 65536×2 one-hot features to represent source and destination ports, more than our machine can handle.) Grouping ports could cause false positives or negatives if two common ports appear in the same aggregate, we examined our data and found that all popular ports differ by at least 53 in the port space and we never group popular ports.

We summarize the number of extracted flows (both benign and malicious) and number of DDoS events in these malicious flows under "extracted flows" of Table 1. Note that since we extract malicious flows from a subset of war-time traffic that match the in-house mitigation's heuristics, the DDoS events in extracted flows are a subset of all DDoS events in inbound traffic pcaps (Table 1).

We note that a limitation of our data is that it is dominantly UDP (accounting for 99.87% of our 40M extracted flows in Table 1) likely due to the three cloud services we study mainly serve UDP traffic.

2.3 DDoS Detection Techniques

Having extracted flows, we describe the ML models we use and how we train, validate and test these models with these flows. We developed our specific ML-based AD techniques ourselves, but we follow the use of autoencoder like prior work [2, 4, 17, 18] and we specifically follow the idea of N-BaIoT of using reconstruction error to detect DDoS [17]. Our goal is not to show a new detection method, but to evaluate current state-of-the-art methods with real world data.

Model Overview: We use a type of neural-network ML model called autoencoder because it is widely used in AD (such as system monitoring [2], network intrusion detection [18] and outlier detection [4]) and has been shown to detect DDoS attacks accurately in lab environment ([17]). While other ML models are also used for AD, such as one-class SVM [5, 38, 45] and other neural networks [3, 10, 20, 21]. We currently focus on autoencoder and leave studying other models for future work.

Autoencoder is a symmetric neural network that reconstructs its input by first compressing the input to a smaller dimension and then expanding it back [37]. The aim of autoencoder is to minimize reconstruction error, the differences between input and output (the reconstructed input). We compute the difference between input and output vectors (F_{in} and F_{out}) as the mean of element-wise square error, as shown in Equation 1 where N is the number of elements in input (or output) vector and F^i_{in} and F^i_{out} are the i-th element in input and output vector.

$$E(F_{in}, F_{out}) = \frac{\sum_{i=1}^{N}(F^i_{in} - F^i_{out})^2}{N} \quad (1) \qquad T_{det} = \mu_{\mathbb{E}} + 3\sigma_{\mathbb{E}} \quad (2)$$

To detect DDoS events, we train an autoencoder with only benign traffic flows and identify malicious traffic flow by looking for large reconstruction errors. The rationale is the autoencoder learns to recognize useful patterns in the benign flows with effectively lossy compression. When it encounters statistically different flows like malicious traffic, it cannot compress this anomalous traffic efficiently and so the reconstruction results in relatively large reconstruction error, with the degree of error reflecting the deviation from normal of the anomaly.

We build a 6-layer neural network for each of our 5 VIPs, compressing a 2848-by-1 input vector (2×1286 one-hot features for ports, 256 one-hot features for protocols and the other 20 features

Sport	Dport	Proto	SrcPkts	SrcRate	SrcLoad	SIntPkt	sTtl	sMaxPktSz	sMinPktSz	SrcTCPBase
source port	dest port	protocol number	src-to-dst pkt count	src-to-dst pkt/s	src-to-dst bits/s	mean src-to-dst inter -pkt arrival time	TTL in last src -to dst pkt	src-to-dst max pkt size	src-to-dst min pkt size	src TCP base sequence
TcpOpt_{M, w, s, S, e, E, T, c, N, O, SS, D}										
the existence of certain TCP option: max segment size (M), window scale (w), selective ACK OK (s), selective ACK (S), TCP echo (e), TCP echo reply (E), TCP timestamp (T), TCP CC (c), TCP CC New (N), TCP CC Echo (O), TCP src congestion notification (SS), TCP dest congestion notification (D)										

Table 2: Our 23 Flow Features (Merging 12 Features About Existence of Certain TCP Option) Before One-hot Encoding

in Table 2) to a 4-by-1 vector and expand it back symmetrically (dimensions of each layer shown in Figure 2). As with many ML systems, the specific choices of 4-by-1 and 6 layers are empirical, although we also tried 8 layers without seeing much advantage. We use ReLu [19] as activation function, L2 regulation [25] and dropout [35] to prevent overfitting and mini-batch Adam gradient descent [13] for model optimization, all following standard best practices [36]. Our implementation uses the python library pyTorch [24].

Model Training: We train each VIP's autoencoder to accurately reconstruct benign flows from this VIP.

We first randomly draw 1 million benign flows from each VIP as their training dataset (see "training" column of Table 1.) SR3VP1 observes only 698k benign flows, even with extended observation, so there we train on 548k benign flows. (We experimented with additional training data but did not find it helped)

We then pre-process training dataset by normalizing training flows' feature values to approximate the same scale (about 0 to 1), following best practices [36]. The one-hot features are already normalized, but for a given other feature i of flow w in the training dataset (F^i_w in Equation 3), we normalize it with min-max normalization (Equation 3 where F^i_{tmax} and F^i_{tmin} are the maximum and minimum values for feature i in all training flows).

We initialize four hyper-parameters in our models: mini-batch size as 128, learning rate as 10^{-5}, drop-out ratio as 50% and weight decay for L2 regulation ([25]) as 10^{-5}. (We tune these values later during model validation if needed.)

Lastly, we train our models with normalized training data for 2 epochs. (Adding more epochs does not increase models' detection accuracy on validation datasets, and risks overfitting.)

Threshold Calculation: Detecting malicious flows from large reconstruction error requires a threshold to separate normal error from anomalies. We calculate this threshold by estimating the upper bound for benign flows' error. Specifically, we randomly draw benign flows from each VIP to form threshold datasets that are distinct from training, validation, and test datasets (see the "threshold" column of Table 1). We set the size of threshold dataset to match the size of validation and test dataset (described later this section). Similar to model training, we pre-process threshold data with min-max normalization (Equation 3) and maximum and minimum feature values extracted from training datasets (F^i_{tmax} and F^i_{tmin}). We then apply trained models to flows in threshold dataset and record their reconstruction errors as $\mathbb{E}$. We calculate detection threshold (T_{det}) with Equation 2 where $\mu_{\mathbb{E}}$ and $\sigma_{\mathbb{E}}$ are mean and standard deviation of $\mathbb{E}$.

$$\hat{F^i_w} = \frac{F^i_w - F^i_{tmax}}{F^i_{tmin} - F^i_{tmax}} \quad (3) \qquad A(j) = \frac{(F^j_{in} - F^j_{out})^2}{\sum_{i=1}^{N}(F^i_{in} - F^i_{out})^2} \quad (4)$$

Model Validation: We validate detection accuracy of trained models (with initial hyper-parameters) by applying them to detect benign and malicious flows in validation datasets. When we encounter poor accuracy in the validation data, we tune hyper-parameters of the models to improve validation accuracy.

To validate our model, we construct validation dataset for each VIP by randomly drawing half malicious flows from a VIP and equal amount of random benign flows from same VIP (shown under "validation" of Table 1). We pre-process validation dataset with min-max normalization and F^i_{tmax} and F^i_{tmin} (Equation 3). We apply trained models to detect benign and malicious flows in validation sets and check common accuracy metrics of detection results: mainly precision, recall and f1 score (Equation 5 where TP, FP and FN stands for true positives, false positives and false negatives in identifying malicious flows). Note that for SR3VP1 where we only have benign flows, we instead examine its true negative ratio (TNR, the fraction of benign flows that get correctly detected.)

If any detection metric for a per-VIP model goes under 99%, we tune this model's hyper-parameters with random search [1], by training multiple versions of this model, each with a set of randomly-chosen values for hyper-parameters. We then select as the final model the version that gets the highest f1 score against the validation dataset and use this final model for all subsequent detection. (We list hyperparamter values for our final models in §3.)

$$prec = \frac{TP}{TP + FP} \qquad rec = \frac{TP}{TP + FN} \qquad f1 = \frac{2 * prec * rec}{prec + rec} \quad (5)$$

Model Testing: Finally, we report detection accuracy for our trained and validated models by applying them to test datasets, consisting of the other half of malicious flows extracted from each VIP and equal amount of random benign flows from the same VIP (see "Test" of Table 1.) Specifically, we first pre-process test dataset with min-max normalization and F^i_{tmax} and F^i_{tmin} (Equation 3). We then report our models' detection precision, recall and f1 score on test dataset. (Similar to validation, we report TNR for SR3VP1.)

2.4 Interpreting the Results of Detection

While our models follow best practices, we are the first to evaluate such models with real-world data and interpret the results. We interpret our models' detection results with feature attribution (§2.4.1) and counterfactual explanations analysis (§2.4.2).

2.4.1 Feature Attribution. We use feature attribution analysis to understand the contribution from each feature to the detection of each flow instance. Prior work used feature attribution [30, 31, 46, 47].They either attribute feature importance by evaluating the difference in model output when perturbing each input feature

([46, 47]), or by taking the partial derivative of model output to each input feature ([30, 31]).

Since our models' detection is based on reconstruction error of input flow (Equation 1), which is the mean of per-feature errors from all flow features, we can measure a feature's contribution to detection by how much error it contributions to overall reconstruction error. We normalize per-feature error by dividing it by the sum of error from all features, as in Equation 4, and attribute that feature's contribution as this normalized per-feature error. (Prior work cannot use our simple form of feature attribution because they focus on models that output classification, rather than reconstruct the input.)

2.4.2 Counterfactual Explanations. Counterfactual explanations show how an input must change to significantly change its detection output, as advocated by prior work [16, 42].We use counterfactual explanations to understand the normality our models learn for each flow feature, which in turn implies what values the models consider anomalous.

Specifically, we first find a *base flow* that is detected as benign, then we repeatedly alter the target feature's value in this base flow while keeping other features unchanged. We feed these altered base flows into our model to observe how much the reconstruction error changes with each perturbation of target feature's value: an increase in errors suggests our models consider current feature value more abnormal than the previous value, and vice versa. We repeat this experiment on different base flows to see if our models consistently consider certain target feature values more normal than the other values, with relatively normal values suggesting normality our models learned.

3 RESULTS

To understand how well ML-based AD work in detecting real-world DDoS attacks. We train and validate an autoencoder model for each of our 5 VIPs using the training, threshold, validation, and test datasets (Table 1) as described in §2.3. Our final models for SR1VP1 and SR2VP1 use tuned hyperparameters values (mini-batch sizes 64 and 32, learning rates about 2×10^{-5} and 10^{-5}, drop-out ratio both 10% and weight decay about 10^{-6} and 2×10^{-6}). Our final models for the remaining 3 VIPs use initial hyperparamter values from §2.3.

With trained and validated models, we report detection accuracy on test datasets in §3.1 and examine false positive rates in §3.2. In §3.3, we evaluate our models on all extracted malicious flows (recall the test datasets contain only half extracted malicious flows), and we interpret *why* our models detect some malicious flows but miss others in §3.4.

3.1 Detection Accuracy on the Test Dataset

We evaluate accuracy by measuring precision, recall and the F1 score of our models' detection of test datasets in Table 4. (We report TNR for SR3VP1, recall §2.3.)

We first observe that our models almost never generate false positives: 2,556 false positives out of all 696,000 tests of benign flows, a false positive ratio of 0.36%. In fact, we later show that only 28 of these 2,556 are actual false positives in §3.2. Detection precision (Equation 5) and TNR for all 5 VIPs are high (at least 98.90% in Table 4), suggesting our models almost always correctly detecting benign traffic (high TNR) and are rarely generating false alerts (high precision).

Our second observation is that our models identify almost all malicious flows to 2 of the 4 VIPs under attack: detection recall is 99.99% for SR1VP2 and 100% for SR1VP3 (Table 4). Our models miss a small fraction of malicious flows for the other 2 VIPs: 5.25% for SR1VP1 and 8.63% SR2VP1 (Table 4). (We do not report recall on SR3VP1 since this VIP sees no attack events §2.2.)

We conclude that our models identify most, if not all, malicious flows to all 4 VIPs under attacks (recalls from 91.37% to 100%) while maintaining near-zero false detections for all 5 VIPs.

3.2 Examining False Positives on Test Datasets

Our models make 2,556 false positives against the test datasets (§3.1); we next compare these to in-house mitigation's heuristics such as whitelists of destination ports and protocols.

As shown in Table 3, we find most of our false positives (95.7%, 2,446 out of 2,556) are actually true positives (malicious flows that get correctly detected). Our training data is noisy and may contain malicious traffic (§2.2). Out of these 2,446 true-positive flows, most are malicious UDP flows with non-whitelisted (non-WL) destination ports (79.8% or 1,953) and malicious flows using ICMP, a non-WL protocol (19.9% or 487). We also find a very small fraction of true-positive flows using blacklisted (BL) source ports (0.2% or 4), and a few with at least one packet with bad payload content (that fail regular expressions required by in-house mitigation's heuristics) (0.1% or 2). (We show in §3.4.4 that although our models do not see packet payload, they still detect some malicious flows consisting of packets with bad payload content based on anomalies in flow features.)

A few false positives (3.2%, 82 out of 2,446) are artifacts due to misdirected TCP flows. These misdirected flows appear to originate from our 5 VIPs, yet the pcaps we study contain only inbound packets to these VIPs (§2.2). These misdirected flows thus have wrong values (all zeros) for the 8 flow features counting source-to-destination traffic statistics such as packet count (feature SrcPkts in Table 2). Argus has a known limitation (confirmed with the author) where a missing TCP SYN and SYN/ACK results in Argus mislabeling the source and destination of a flow, so these flows actually have source ports that are well-known service ports (mostly 443). We believe these missing TCP SYN and SYN/ACK packets are likely dropped due to 1 in 1000 packet sampling (§2.2).

Lastly, we find the remaining 28 false positives are likely actual false positives. Each of these 28 flows (all TCPs) does not match any heuristics used by in-house mitigation.

We conclude that out of 2,556 false positives reported in §3.1, only a tiny fraction (1.1%, 28 out of 2,556) are actual false positives, suggesting the actual false positive rate is near zero (0.00%, 28 out of 696,000 test benign flows) (We explore potential causes for these 28 actual false-positive TCP flows in §3.4.1.)

3.3 Detection Accuracy On All Malicious Flows

Having shown our models identify most malicious flows in test dataset with near-zero false positives, we next explore how our models identify all extracted malicious flows. (The test datasets

total false positives	2,556	(100.0%)	
actual false positives	28	(1.1%)	
actual true positives	2,446	(95.7%)	(100.0%)
UDP flows w non-WL dst port	1,953	(76.5%)	(79.8%)
UDP flows w BL src port	4	(0.2%)	(0.2%)
UDP flows w bad payload content	2	(0.1%)	(0.1%)
flows w non-WL protocols	487	(19.1%)	(19.9%)
misdirected TCP flows	82	(3.2%)	

Table 3: False Positives on Test Dataset Breakdown

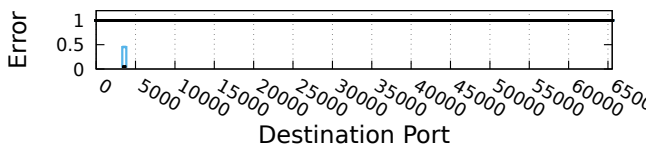


Figure 3: Normalized Reconstruction Errors for 100 Base Flows from SR1VP2 using Different Destination Ports

contain only half extracted malicious flows §2.3.) Specifically, we group malicious flows by their main anomalies, as determined by the in-house mitigation's heuristics, and show which anomalies are best detected by our models, and which are poorly detected.

As in Table 5, we show our models are near perfect at detecting anomalies on whitelisted (WL) features with only a few benign values that have unordered values (recall unordered features from §2.2): our models capture all malicious flows with non-WL protocol (100.00% of about 15k) and nearly all malicious UDP flows with non-WL destination ports (99.92% of about 1M). We show our models are reasonable at detecting anomalies in blacklisted (BL) features (those with only a few malicious values) with unordered values: our models identify nearly all UDP flow with BL source ports (97.5% of 5k) and most UDP flows with illegal port 0 (66%, 4 out of 6).

However we find our models are bad at detecting anomalies on BL features with ordered values (§2.2). Specifically, our models only detect a few malicious flows (8.5% of about 3k) consisting of at least one packet with too small payload (in-house mitigation drops UDP packets with payload smaller than a threshold). In §3.4.3, we show that although our models do not see packet payload size, they can infer if a UDP flow is consisting of packets with small payloads from the flow features of maximum and minimum flow packet size (Table 2).

Lastly, we show our models detect a quarter of UDP flows (24.7% of 8k) containing at least one packets with bad payload contents (that fail regular expressions required by in-house mitigation), despite our models do not see packet payloads (Table 5).

3.4 Interpreting Detection of Malicious Flows

We explore why our models are much better at at detecting anomalies on whitelisted (WL) and blacklisted (BL) unordered features (§3.4.1 and §3.4.2) than anomalies on BL ordered features (§3.4.3) and how our models detect some flows with anomalies in payload content (§3.4.4).

3.4.1 Whitelisted Unordered Features. We show the reason that our models are good at detecting anomalies in WL unordered features (recall we detect 99.92% and 100% of UDP flows with non-WL destination port and protocol in Table 5) is that our models could correctly learn these features' normalities. (We consider a feature WL if it has only a few benign values while the majority of its values are malicious, judged by in-house mitigation's heuristics.)

Learn Normality of Destination Ports: We show our models correctly learn the whitelisting of destination ports used by in-house mitigation with counterfactual explanation (§2.4.2). Specifically, we draw 100 random UDP flows (that are detected as benign) from each VIP's test datasets as base flows, alter these 500 base flows by enumerating their destination ports from 0 to 65535 with an step size of 51 (0, 51, 102 ... 65535) and feed altered base flows into models. (Note that having a sub-51 step size does not help because we group and one-hot encode every 51 adjacent ports in §2.2 and our models cannot distinguish the 51 ports from same group.) We then watch for how base flows' error change as destination ports change.

We find our models consistently consider all 500 base flows with non-WL ports much more abnormal than the same flows with WL ports. We use reconstruction errors from SR2VP1's 100 base flows as example (other VIPs are similar). Since we only care about how a base flow's error changes as its destination port changes (rather than the exact values of these errors) and want to compare these changes across all 100 base flows from this VIP, we normalize the set of errors resulted from one base flow using different destination ports to range [0, 1] by dividing these errors with the maximum error found among them. Figure 3 shows normalized errors of SR2VP1's 100 base flow with different destination ports. Specifically, we present the 100 normalized errors resulting from 100 base flows using a certain destination port as a blue box and whiskers in Figure 3 where the top and bottom of the whisker shows maximum and minimum of these errors, top and bottom of the box are 2 and 98 percentiles of these errors and the black line in middle of the box shows median. Figure 3 shows that all non-WL ports lead to similarly high reconstruction errors and this pattern is very consistent across all 100 base flows from SR2VP1 (shown as the horizontal black line at normalized error of 1 in Figure 3, which is caused by the blue boxes and whiskers for all non-WL ports collapsing with their black lines for medians). We also find consistently small reconstruction error at the one WL port for SR2VP1 (shown as the blue boxes near port 5000 in Figure 3, with median about 0.05).

We find for 298 of these 500 base flows (100 from SR1VP1, 29 from SR1VP2, 43 from SR1VP3, 100 from SR2VP1 and 27 from SR3VP1), our models consider base flows malicious when they use non-WL destination ports, suggesting for these base flows, anomaly from non-WL destination port alone could trigger models' detections (recalling base flows are detected as benign in the first place).

We conclude that our models correctly learn normality of destination ports by consistently consider base flows with non-WL port more abnormal (for all 500 base flows) and even malicious (for 298 of 500 base flows).

Detect Anomalies on Destination Ports: We next show how our models use the learned normality to detect almost all malicious flows with non-WL destination ports (99.92%; 1,260,943 out of 1,261,951 , from Table 5) with feature attribution analysis (§2.4.1).

VIP	Precision	Recall	F1-Score	TNR
SR1VP1	98.90%	94.75%	96.78%	–
SR1VP2	99.69%	99.99%	99.83%	–
SR1VP3	99.81%	100.0%	99.90%	–
SR2VP1	99.50%	91.37%	95.26%	–
SR3VP1	–	–	–	99.68%

Table 4: Detection to Test Dataset

	Total Flows by Main Anomalies	Detected Flows (TP)		If Only Main Anom	
Main Anomaly	Count	Count	Frac of Total	Count	Frac of TP
Flows w Non-WL Protocol	15,206	15,206	100.00%	15,206	100.00%
UDP Flows w Non-WL Dst Port	1,261,951	1,260,943	99.92%	18,279	1.45%
UDP Flows w BL Src Port	5,334	5,201	97.5%	11	0.21%
UDP Flows w Invalid Port Zero	6	4	66%	2	50%
UDP Flows w Too Small Payload	2,522	215	8.5%	0	0%
UDP Flows w Bad Payload Contents	8,229	2,036	24.7%	0	0.0%

Table 5: Detection to All Malicious Flows Breakdown by Main Anomalies

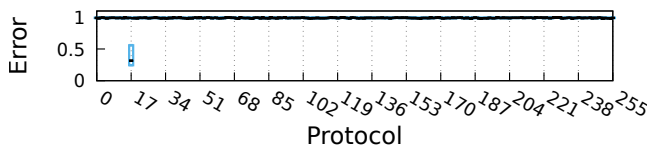


Figure 4: Normalized Reconstruction Errors for 100 Base Flows from SR3VP1 using Different Protocols

We find that when our models detect these malicious flows, anomalies from destination ports usually provide most reconstruction error that causes detection (on average 0.80× threshold of errors in 98.55% of true-positive detections). Our models thus rely on the help from anomalies in other features for these detections, mainly Sport, sMaxPktSz, sMinPktSz and SIntPkt (at least 10% attributions in 100.00%, 84.3%, 84.0% and 3.53% of these detections, recalling features from Table 2). We also find that anomalies from the destination port alone will only trigger a small fraction (1.45%; right-most two columns of Table 5) of these true-positive detections, which is not consistent with our counterfactual result earlier where anomalies from non-WL destination ports alone triggers detections for 298 of our 500 base flows (about 60%). One possible explanation is sampling error: non-destination-port features in our 500 base flows happen to provide more per-feature errors than their counterparts in these malicious flows, requiring less per-feature errors from destination port in base flows to trigger detection. (In comparison, we observe consistent counterfactual results and actual detection results for flows with non-WL protocols later in this section.)

We find all of our models' false negatives (0.08%, 1,008 out of 1,261,951 Table 5) are artifacts of our one-hot encoding of destination port. Specifically, since we encode every adjacent 51 destination ports as one one-hot feature (§2.2), our models can not distinguish among these ports. Our models miss these 1,008 false-negative flows because our models consider their destination ports the same as the WL ports, because their ports are close to WL ports in values (within 51).

We conclude that our models correctly learns the normality of destination port and detects almost all malicious UDP flows with non-WL destination ports, mostly by combining anomalies from destination port and other features.

Learn Normality of Protocols: Despite identifying all malicious flows with non-WL protocol (Table 5) we show our models actually fail to learn the complete normality of protocols. Per heuristics from in-house mitigation, UDP, TCP and 3 other protocols (omitted for security) are all WL for some cloud service: UDP for all three services we study, TCP for both SR1 and SR3 and 3 other protocols for SR3. By applying counterfactual explanation to same 500 base flows from destination port analysis and varying their protocols from 0 to 255, we find our models consistently consider all 500 base flows with non-UDP protocols more abnormal (than the same base flows but with UDP) and consider 298 of these 500 base flows malicious when using non-UDP protocols. As an example, we show normalized errors from 100 base flows of SR3VP1 in Figure 4. (Other 4 VIPs are similar.) From Figure 4, we find all protocols except UDP (17) consistently lead to similarly high errors, suggesting our models only learn the whitelisting of UDP. We conclude that our models learn incomplete normality for protocols for all our 5 VIPs by only considering one of 5 WL protocols relatively normal.

We believe the reasons our models fail to learn non-UDP WL protocols is that they are under-representing in training data. Specifically, while UDP accounts for almost all training data for our 5 VIPs (99.87% of 4.5M), TCP accounts for only a tiny fraction (0.01% of 4.5M) and the 3 other WL protocols are completely missing. We note that TCP show up even less than noises (non-WL protocols, showing up in 0.11% of 4.5M) in training data, suggesting that it is actually reasonable for our models to not learn infrequent protocol values like TCP otherwise it risks learning noises.

Detect Anomaly on Protocols: Having learned incomplete normality of protocols, our models risk false positive by considering benign flow with WL non-UDP protocol abnormal but should still be able to detect malicious flows with non-WL protocols. To support our argument, we first show that our models detect all 15,206 malicious flows using non-WL protocols (Table 5) and that all these detections could be triggered by anomaly from protocol alone. We argue the 28 false positive our models made on test dataset (§3.2) are likely due to our models consider TCP protocol used by these 28 flows abnormal based on the incomplete normality learned. We support our hypothesis by showing that protocol is the highest-attributing feature in 27 of these 28 detection false positives (second highest for the rest 1 false positive), providing in average about 0.85× threshold of errors. (Anomalies from protocol alone could trigger for 3 of these 28 detection false positives.)

3.4.2 Blacklisted Unordered Features. We next show that while our models fail to learn complete anomaly of BL source ports, they still detect most malicious flows with BL source ports (97.5%, or 5,201 out of 5,334) by considering all source ports (including BL ones), except the ones frequently seen in training data, as equally abnormal. (We consider a feature BL if it has only a small number of malicious values while the rest majority of its values are benign, judging from heuristics used by in-house mitigation.)

Learn Normality of Source Ports: Similar to our analysis of destination port (§3.4.1), we explore what source ports do our models consider relatively abnormal by altering source port from the same 500 base flows from 0 to 65535 with an step size of 51 and watch for change in reconstruction errors of base flows.

We find our models consistently consider base flows using one to two source ports frequently seen in training data (called "frequent training ports" for simplicity) much more normal than same flows using the rest source ports. We show normalized reconstruction errors of SR2VP1's 100 base flows in Figure 5 as example. (We summarize normalized errors for other 4 VIPs at the end of this paragraph.) We find one non-BL source port 3111 with consistently low error (shown as blue box and whisker on bottom left of Figure 5, with median about 0.003) likely due to it corresponds to the most frequent training source port 3074 (in 75.31% of 998k training UDP flows of SR2VP1). (Ports in range 3061 to 3111, including 3074, look the same to our models due to we group and one-hot encode adjacent 51 ports.) We find all the rest source ports, including the BL ones, consistently get high errors (shown as the horizontal black line around error of 1 in Figure 5). We report similar trend in normalized reconstruction errors for other 4 VIPs: they consistently give the most-frequent training ports (all non-BL) similarly low errors and the rest source ports (including BL ones) with similarly high errors. The only exception is that we find SR1VP1 also consistently gives a BL source port (omitted for security concern) low error because this source port is the second most-frequent training port (in 1.21% of 999k UDP training flows from SR1VP1). Lastly, we find our models never consider any of these 500 base flows malicious regardless of their source ports, suggesting the anomaly from source ports alone cannot trigger detection. We conclude that our models learn incomplete normality of source ports for 4 VIPS (all except SR1VP1) by only considering some non-BL source ports (frequent training ports) instead of all non-BL ports as relatively normal. Our models learns incorrect normality of source ports for SR1VP1 by considering both one non-BL port (frequent training port) and one BL port relatively normal, due to noises (flows with this BL port) in training data.

Detect Anomaly on Source Ports: We show the incomplete normalities our models learn still enable detecting most malicious flows with BL source ports (5,201 out of 5,334, 97.5%, recall Table 5) due to our models consider all except frequent training ports (including BL ones) relatively abnormal. Similar to destination port, we find anomaly from source ports usually provides most but not all reconstructing errors in true-positive detections (providing in average about 0.79× threshold of errors in 99.79% of these true-positive detection). Our models thus relies on anomaly from additional features to trigger these detections, mainly sMaxPktSz, sMinPktSz, SIntPkt, SrcPkts, TcpOpt_M and sTtl (recalling features from Table 2), each with at least 10% attribution in a good fraction of (from 85.63% to 8.17%) of these detections. (We find our models could only detects about 0.21% true positive solely base on anomaly from source port, as in Table 5, consistent with our counterfactual results where no base flow get considered malicious due to its source port.)

We believe the 133 false negatives of our models result from inability to enough anomalies from other features in these flows. To support this hypothesis, we show that in these false negatives, source ports typically provide about 0.64× the threshold of errors, similar to what source ports provide in most true positives (0.79× threshold). We also show that in these false-negative detections, in average 92% of the flow reconstruction errors come from source ports, suggesting our models cannot find too much anomaly from other features. (We see no false negative caused by our models incorrectly consider one BL source port from SR1VP1 relatively normal.)

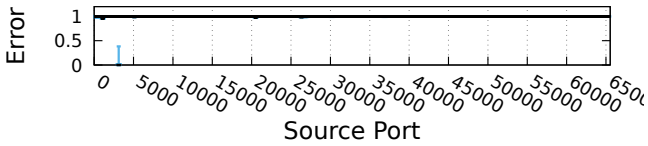


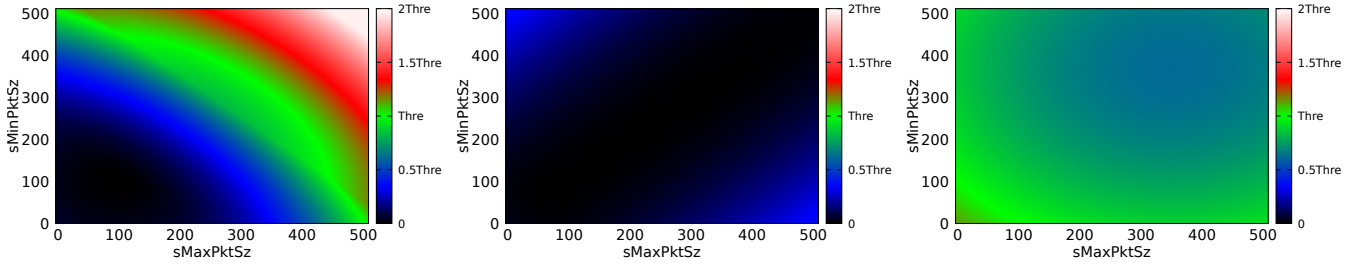
Figure 5: Normalized Reconstruction Errors for 100 Base Flows from SR2VP1 using Different Source Ports

By considering all except frequent training source ports relatively abnormal, our models risk generating false positive by considering non-BL source port of an benign flow abnormal. Luckily, we find no such false positives in test data (§3.2).

Detect UDP Flows with Invalid Port Zero: Our models also missed 2 out of 6 malicious DNS flows with either source or destination port as 0 (Table 5). We believe these misses are caused by two reasons. First, our models have no way to infer port 0 is special (invalid) because there are a tiny fraction of UDP flows with 0 as source or destination port in training dataset as noise (0.16% of 4.5M training UDP flows), Second, since our models do not know port 0 is special and invalid, they simply consider port 0 as a regular non-WL destination port or non-frequent-training source port. As a result, flows with destination or source port 0 share the same chance to be missed as flows with any non-WL destination port or with any non-frequent-training source port. (Recall from Table 5 that we missed 133 malicious flows with BL source ports, which are also not frequently seen in training data.)

3.4.3 Blacklisted Ordered Features. We next explore why our model for SR2VP1 is bad at detecting malicious UDP flows consisting of at least one UDP packets with too small payloads (smaller than a threshold, omitted for security). (Other VIPs do not filter on small packet payload.) Note that although our model does not see packet payload sizes, it could still detect these flows based on features sMaxPktSz and sMinPktSz (the maximum and minimum packet size in flows, recalling Table 2). The rationale is that since we find all too-small-payload UDP packets in our data are either 56 or 60 bytes and all malicious UDP flows consisting of these packets are either made of all 56 or all 60-byte packets, these malicious flows have only two possible sMaxPktSz and sMinPktSz combinations: both 56 or both 60. Since UDP training flows for SR2VP1 rarely have these two sMaxPktSz and sMinPktSz combinations (0.01% of 998M, not bad comparing to, for example, 0.46% noises for WL destination port in §3.4.1), detecting flows consisting of too-small-payload packets is equivalent to detecting flows with two BL sMaxPktSz and sMinPktSz combinations (both 56 and both 60).

Learn Normality of Packet Sizes: We use counterfactual explanation to understand what sMaxPktSz and sMinPktSz combinations are considered relatively abnormal. We draw 10 random base



(a) SR1VP1 (abnormal when both large) **(b) SR2VP1 (abnormal when either large)** **(c) SR3VP1 (abnormal when both small)**

Figure 6: Reconstruction Errors for 1 Base Flow from 3 VIPs with Varying Packet Sizes

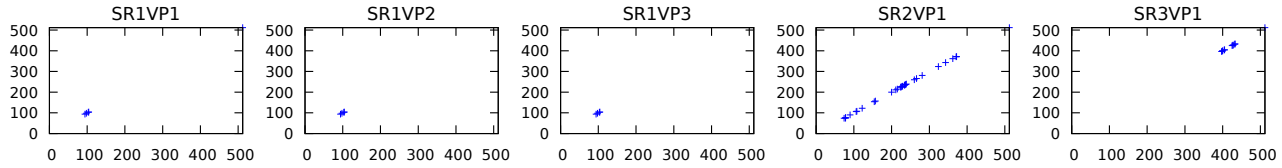


Figure 7: Frequent Combinations of sMaxPktSz (X axis) and sMinPktSz (Y axis) in Training UDP Flows

flows from test dataset of 5 VIPs, vary sMaxPktSz and sMinPktSz in base flows from 0 to 512 bytes (the largest packet size we find in all training, threshold, validation and test flows in Table 1) with a step size of 1 and watch for the change in base flows' errors.

We find our models for the three VIPs from SR1 consistently consider base flows more abnormal when both sMaxPktSz and sMinPktSz are larger. We show reconstruction errors from one base flow from SR1VP1 as an example (Figure 6a): this flow' reconstruction errors increases when sMaxPktSz and sMinPktSz both increase from 0 to 512, shown as the black and blue (indicating errors from near zero to about 0.5× threshold) on left bottom gradually becoming red and white on top right (indicating errors from about 1.5× threshold to about 2×threshold). We find our model for SR2VP1 also consistently considers larger sMaxPktSz and sMinPktSz abnormal but not when they are both large (see Figure 6b for reconstruction errors for one of its base flow). We show our model for SR3VP1 consistently considers base flows more abnormal when sMaxPktSz and sMinPktSz are both small (see Figure 6c for reconstruction errors of one of its base flows).

We show the reason our models consider different sMaxPktSz and sMinPktSz combinations abnormal is that they see different benign combinations in training flows and consider combinations *more different* from those in training flows as *more abnormal*. We plot frequent sMaxPktSz and sMinPktSz combinations (at least 1000 occurrence) in each VIP's training UDP flows in Figure 7. We find three VIPs from SR1 consider both sMaxPktSz and sMinPktSz being large as abnormal because they mostly see flows with sMaxPktSz and sMinPktSz both small in training data (both from 94 to 104 bytes). The only exception is that we find a few of SR1VP1' training UDP flows (about 1.42% of 999M) have sMaxPktSz and sMinPktSz of both 512 bytes (see blue pluses on right top corner of SR1VP1's chart in Figure 7). However we find our model for SR1VP1 still considers sMaxPktSz and sMinPktSz of both 512 relatively abnormal (see the white top right corner of Figure 6a), likely due to our models treat these both-512 combinations in training data as noises. We find SR2VP1 consider flows with big sMaxPktSz and small sMinPktSz (or the other way around) abnormal because it have seen sMaxPktSz and sMinPktSz being both small and both big in its training data (both from 74 to 512 bytes) . We find SR3VP1 consider sMaxPktSz and sMinPktSz being both small abnormal because it mostly see them being both big in its training data (both from 397 to 512 bytes).

We conclude our models learn incomplete normality of sMaxPktSz and sMinPktSz combinations for all 5 VIPs by considering some non-BL combinations (the frequent training ones), instead of all non-BL combinations, as relatively normal. (Recall that SR2VP1 BL sMaxPktSz and sMinPktSz combination of both 56 and both 60 while other 4 VIPs BL no combinations.)

Detect Anomalies in Packet Sizes: We show that by learning incomplete normality, our model for SR2VP1 completely fails to infer the BL sMaxPktSz and sMinPktSz combinations are relatively abnormal. In Figure 6b, we find the two BL combinations (both 56 and both 60) are getting the lowest errors (shown as the black bottom left corner of Figure 6b), likely due to they are too similar to some frequent combinations in SR2VP1's training data (such as both 74 bytes, as shown in SR2VP1's chart in Figure 7) to be considered abnormal by our model.

Since our models fail to learn BL sMaxPktSz and sMinPktSz combinations are abnormal, it is not surprising that our models only detect a few malicious flows with these BL combinations (8.5%, 215 out of 2,522 Table 5). We find sMaxPktSz and sMinPktSz only contribute in average about 0.002× threshold of reconstruction error in detection of these 2,522 malicious flows with BL combinations (including the 215 true-positive detections), suggesting these detections almost completely depend on anomaly from other features.

3.4.4 Combining Anomalies from Multiple Features. Recall that by combining anomalies from multiple features, our models still detect malicious flows when anomalies from source (§3.4.2) or destination port (§3.4.1) alone are not enough to trigger detection

and when our models fail to infer BL sMaxPktSz and sMinPktSz combinations are abnormal (§3.4.3). Similarly, our models detect a quarter malicious UDP flows (24.7%, 2,036 out of 8,229 Table 5) whose main anomaly is in packet payload content (that fail regular expressions required by in-house mitigation) and is not visible to our models.

To understand how good our models are at finding anomalies from multiple features, We breakdown number of features with significant attributions (at least 10%) in all detected malicious flows (recall Table 5). (We calculate source port's attribution as the sum of attributions from its 1286 one-hot features. Similarly, we calculate attributions for destination port and protocol.) We find our models uses multiple significantly-attributing features in nearly all detections (99.90% of 1.2M) and uses 4 in most detections (79.16% of 1.2M).

4 IMPLICATIONS

We next distill the detailed analysis of ML-based AD to three implications: leverage the strengths of anomaly detection, training with somewhat noisy data is possible, and combinations of AD and heuristics can help both.

4.1 Use Anomaly Detection to Its Strengths

Prior work has suggested that ML models are by nature better at finding similarity to training data (binary classification) than finding deviations from it (anomaly detection) [34, 44]. We argue that even for AD, our autoencoder models are better at detecting some anomalies than others, and that autoencoder-based AD works best with certain classes of anomalies. Specifically, we show that our models are better at detecting anomalies on whitelisted (WL) features than blacklisted (BL) features because they could learn correct normality for WL features (§4.1.1). We also show that our models are better at detecting anomalies in unordered features than in ordered features because even with incomplete normality, models could still detect anomalies in unordered feature with high recall (§4.1.2).

4.1.1 Learn Normality of Features. Since AD is about inferring values deviating from normality as abnormal, learning correct normality is key to reliable AD. Our models learn the most frequent feature values in training data as normality (§3.4.1, §3.4.2 and §3.4.3). For models to learn correct normality of a feature, all of its benign values need to be well-represented in training data. Since by definition WL features have only a few benign values, while the majority of values for BL features are benign, it is more likely for WL features to have all its benign values well-represented in training dataset. For example, our models correctly learn normality for WL destination port because the one WL port is the most frequent port in training data, resulting in perfect detection of malicious flow with non-WL destination ports (§3.4.1). In comparison, while there are thousands of non-BL source ports, our models only learn 1 of them (the most frequent one in training data) as relatively normal (§3.4.2). (Note that our models do not always learn correct normality form WL features. For example, our models only learn 1 out of 5 WL protocols as normal in §3.4.1 because the rest 4 protocols are either infrequent or non-existent in training data.)

4.1.2 Detect with Incomplete Normality of Features. When only part of a feature's benign values are well-represented in training data, as is often the case for BL features, our models learn an *incomplete normality*, capturing only some benign feature values as normal.

We show that with incomplete normality of unordered features, our models still detect malicious flows as anomalous based on unordered features with high recall (while risking false positives). Specifically, after learning incomplete normality for target unordered features,we find our models simply infer the rest values for target feature (including the rest benign values and all malicious values) as equally abnormal, enabling detection of malicious flows with malicious target feature values, and risking identifying benign flows with benign target feature values as abnormal. For example, by only learning 1 of 5 WL protocols (UDP) as normal in §3.4.1, our models identify all 15,206 malicious flows with non-WL protocol, but they generate 28 false positives using WL protocol TCP, since they incorrectly consider it abnormal (§3.4.1). Similarly, by only learning the most frequent non-BL source ports as normal and all other ports (including all BL ports) as relatively abnormal, our models identify almost all malicious UDP flows with these BL ports (97.5%, 5,201 out of 5,334) but risk false positives (although we do not find such false positive).

We show that, however, with incomplete normality of ordered features, our models risk low-recall detections. After learning partial benign values from target ordered features as normal, our models consider values more different from these benign values as more abnormal, risk incorrectly considering malicious values normal if they happen to be numerically close to these benign values. As a example, our models only detect a small fraction of malicious UDP flows with BL sMaxPktSz and sMinPktSz combinations (8.5%, 215 out of 2,522) almost entirely relying on anomaly from other features (§3.4.3), because our models do not consider these BL combinations abnormal since they are similar to some frequent benign combinations in training data (that our models consider normal).

Our observations support prior belief that perfect model of normality is required for reliable AD ([34]) while complementing it by showing that when normality is incomplete, our models could still reliably identify malicious flows with anomalies on unordered features.

4.2 Noise-Free Data is Not Always Necessary

Prior work suggests that one reason AD may not be applicable to network intrusion detection is that the attack-free training data that many AD study assumes does not exist outside simulation [7]. Our results supports their claim—we find some brief attacks in our training data. However, our results refutes the claim that noisy data makes AD impossible, since our AD system trains successfully with noisy data, provided training data is *representative*, with all benign values of target features are well-represented in that data.

Specifically, we show that, given representative training data, ML-based AD can learn normality in spite of noise. For example, our models correctly learn the normality of whitelisted (WL) destination port despite noise in the training data (0.46% of 4.5M UDP flows are sent to non-WL port) because the WL ports are the most frequent in training data (99.54% of 4.5M). We also show that when some benign

values of target features are under-represented in training data, noise be confused with normality, because both noise and under-represented benign values are infrequently seen. For example, in §3.4.2, our model for SR1VP1 learns a blacklisted (BL) source port (noise) as normal because this port is the second most frequent (in 1.21% of training UDP flows, more frequent than all under-represented non-BL ports). In §3.4.1, our models fail to learn under-represented WL protocol TCP (in 0.01% of training flows) as normal likely due to our models consider TCP as noises (considering actual noises show up in 0.22% of training flows, more frequent than TCP).

4.3 Combine AD and Heuristic-Based Filters

Finally, we show the potential for join use of heuristic-based filters like in-house mitigation and autoencoder-based AD, since each has its own strengths.

We find our models are very good at finding and using anomalies from multiple features (4 in detection to most malicious flows §3.4.4). ML-based AD is particularly important when the anomalies are not obvious to human perception, such as anomalies on flow packet count, flow packet TTLs, and flow inter-packet arrival time (recall our models use these features to detect malicious flows with BL source port in §3.4.2) However, we find our models are not very certain about each one of these anomaly (models would have missed 97.15% of all 1.2M detected malicious flows in Table 5 if only using the highest-attributing feature), and as a result it almost always detect malicious flow by combining multiple anomaly (in detecting 99.90% of all 1.2M malicious flows).

The heuristic-base filter, by relying on human expertise, is very good at detecting malicious flow based on single anomaly (consider the single main anomaly identified by in-house mitigation's heuristics in Table 5). For example, a flow with non-whitelisted (non-WL) destination port is certainly malicious because the server only serve WL ports. (Note that although in-house mitigation uses multiple heuristic-based filters, only one filter is used in detecting a given malicious flows: the highest-priority filter triggered by this flow.) However we argue that it is more challenging for heuristic-based filters to make use of more subtle features to indicate malice, such as flow inter-packet arrival time or packet TTLs. Our models are able to make use of these features (§3.4.2), and can combine multiple suggestive features.

We propose two possible strategies to combine heuristic-based filters and autoencoder-based AD. The first is to simply stack them: apply the heuristic filter first, to cover intuitive anomalies with great certainty. Then add ML-based AD to covering additional anomalies that are not obvious or require combinations of features. Our second strategy is to build new heuristics based on interpretations of what the autoencoder-based AD has discovered, as discussed in §3.4. Such "ML-discovered" filters could directly use the ML model, or we could extract the relevant features into a new implementation.

5 RELATED WORK

To the best of our knowledge, we are the first attempt to address the two limitations (limited evaluation dataset and no detection interpretations) in prior DDoS detection study using ML-based AD.

5.1 DDoS Study using ML-based AD

The most related class of prior work are those also detect DDoS attacks with ML-AD models.

Most prior work in this class train some form of ML-AD models, such as one-class SVM models ([5, 38, 45]) and neural network models (autoencoder [17], GRU network [20] and multiple models such as fuzzy ARTMAP [10]), with benign traffic and detect attacks by looking for deviations from these benign traffic. Since these prior work mostly test their models with limited datasets including simulation [5, 10], lab traffic [17, 20, 38, 45] and DARPA/MIT dataset [38], it is not clear how well their methods could work in real-world production environment ([7, 34]). Moreover, they usually do not interpret their models' detection decision nor explore why their models work or not work in detecting certain DDoS attacks. In comparison, we evaluate our models with real-world benign and attack traffic from a major cloud provider and show our models work well in production environment in general: capturing most, if not all, malicious flows to 4 VIPs under attack while maintaining near-zero false positives. We also interpret our detection results and show why our models work well on attacks of certain anomalies but not as well on the others.

Two prior work in this class uses clustering algorithms (K-mean [40] and single-linkage [23]) to separate benign and malicious traffic flows into different clusters. Although their detection results are intuitively interpretable (a flow is flagged as malicious since its features are qualitatively close to features of other flows in the "malicious cluster"), they rely on manual inspection to determine which clusters are malicious. They also evaluate their methods with limited datasets (lab data [40] and KDD datasets [23]). In comparison, we do not rely on manual inspection for our detection, and we test our methods on real-world traffic from a large cloud platform.

5.2 DDoS Study using Other Techniques

Many prior work detect DDoS attacks with other techniques. We classify them into following 3 classes.

ML-based binary classification: This class of papers train some form of ML binary classification models (such as KNN [6], decision tree [6, 32], two-class SVM [9, 11], random forest [6] and neural network models [27–29]) with both benign and attack traffic. These ML models thus identify attacks similar to the ones they have seen during training. In comparison, we focus on a different model (ML-AD model) and by training with only benign traffic and looking for deviations from these benign traffic, our models do not rely on on knowledge of known attacks and have the ability to identify potential unknown attacks.

Statistical AD: This class of papers applying statistical models (such as adaptive threshold [33], cumulative sum [22, 33], entropy-based analysis [15] and Bayesian theorem [12]) to identify abnormal traffic pattern that is significantly different from some or all of previously seen (benign) traffic pattern. These papers thus could also cover potentially unknown attacks. In comparison, we focus on AD based on ML models instead of statistical models.

Heuristic-based rule: This class of papers use heuristic-based rules to detect specific types of attacks matching their heuristics. For example, history-based IP filtering remembers frequent remote IPs during peace time and consider traffic from other IPs during

attack time as potential DDoS traffic [39]. Hop-count based filtering identifies spoofed DDoS packets by remembering peace-time IP to (estimated) hop count mapping and considering packets with unusual IP-to-hop-count mapping during attack time as spoofed DDoS packets [43]. In comparison, we use a different method (ML-based AD) and could cover many different types of attacks instead of only a specific type.

6 CONCLUSION

This paper addressed two limitations in prior studies of machine-learning-based anomaly detection: use of real-world data, and interpretation of why the models are successful. We applying autoencoder-based AD to 57 real-world DDoS events captured at 5 VIPs of a large commercial cloud provider. We used feature attribution and counterfactual techniques to explain when our models worked well and when they did not. Key results are that our models work well, detecting nearly all malicious flows to 2 of the 4 VIPs under attacks but with only a few percent false negatives for the other 2 VIPs, with near-zero false positives. Analysis of why our approach works on whitelisted destination ports and protocols and blacklisted source ports showed that our models learn correct normality for destination ports and could relatively accurately detect malicious flows with incomplete normality for protocols and source ports. Two key implications of our work are that we can successfully train ML-based AD even with imperfect training data, and that autoencoder-based AD and heuristic-based AD have complementary strengths.

ACKNOWLEDGMENTS

We thank Yaguang Li from Google, Wenjing Wang from Microsoft and Carter Bullard from QoSient for their comments on this paper.

This work was begun with the support of a summer internship by Microsoft.

Hang Guo and John Heidemann's work in this paper is based on research research sponsored by Air Force Research Laboratory under agreement number FA8750-17-2-0280. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

REFERENCES

[1] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 2012.
[2] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini. Anomaly detection using autoencoders in high performance computing systems. *CoRR*, abs/1811.05269, 2018.
[3] R. Chalapathy, A. K. Menon, and S. Chawla. Anomaly detection using one-class neural networks. *CoRR*, abs/1802.06360, 2018.
[4] J. Chen, S. Sathe, C. Aggarwal, and D. Turaga. Outlier detection with autoencoder ensembles. In *Proceedings of SIAM International Conference on Data Mining*, 2017.
[5] Cynthia Wagner, Jérôme François, Radu State, and Thomas Engel. Machine learning approach for IP-flow record anomaly detection. In *Proceedings of IFIP Networking Conference*, 2011.
[6] R. Doshi, N. Apthorpe, and N. Feamster. Machine learning ddos detection for consumer internet of things devices. *CoRR*, abs/1804.04159, 2018.
[7] C. Gates and C. Taylor. Challenging the anomaly detection paradigm: A provocative discussion. In *Workshop on New Security Paradigms*, 2007.
[8] GeeksforGeeks. One-hot encoding introduction. https://www.geeksforgeeks.org/ml-one-hot-encoding-of-datasets-in-python/.
[9] D. Hu, P. Hong, and Y. Chen. FADM: DDoS flooding attack detection and mitigation system in software-defined networking. In *IEEE GLOBECOM*, 2017.
[10] L. Jun, C. N. Manikopoulos, J. Jorgenson, and J. L. Ucles. HIDE: a hierarchical network intrusion detection system using statistical preprocessing and neural network classification. In *Workshop on Information Assurance and Security*, 2001.
[11] K. Kato and V. Klyuev. An intelligent ddos attack detection system using packet analysis and support vector machine. *Intelligent Computing Research*, 2014.
[12] Y. Kim, W. C. Lau, M. C. Chuah, and H. J. Chao. PacketScore: A statistics-based packet filtering scheme against distributed denial-of-service attacks. *IEEE Transactions on Dependable and Secure Computing*, 2006.
[13] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014.
[14] M. L. Lab. DARPA/MIT dataset. https://www.ll.mit.edu/r-d/datasets.
[15] X. Ma and Y. Chen. DDoS detection method based on chaos analysis of network traffic entropy. *IEEE Communications Letters*, 2014.
[16] D. Martens and F. Provost. Explaining data-driven document classifications. *MIS Quarterly*, 2014.
[17] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici. N-BaIoT—network-based detection of IoT botnet attacks using deep autoencoders. *IEEE Pervasive Computing*, 2018.
[18] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai. Kitsune: An ensemble of autoencoders for online network intrusion detection, 2018.
[19] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning*, 2010.
[20] T. D. Nguyen, S. Marchal, M. Miettinen, M. H. Dang, N. Asokan, and A. Sadeghi. DÏoT: A crowdsourced self-learning approach for detecting compromised IoT devices. *CoRR*, abs/1804.07474, 2018.
[21] P. Oza and V. M. Patel. One-class convolutional neural network. *CoRR*, abs/1901.08688, 2019.
[22] T. Peng, C. Leckie, and K. Ramamohanarao. Proactively detecting distributed denial of service attacks using source IP address monitoring. In *International Conference on Research in Networking*, 2004.
[23] L. Portnoy. Intrusion detection with unlabeled data using clustering. *Thesis*, 2010.
[24] PyTorch. PyTorch project front page. https://pytorch.org.
[25] PyTorch. Weight decay for Adam. https://pytorch.org/docs/stable/optim.html.
[26] Qosient. Argus- auditing network activity. https://qosient.com/argus/.
[27] A. Saied, R. E. Overill, and T. Radzik. Artificial neural networks in the detection of known and unknown DDoS attacks: Proof-of-concept. In *Highlights of Practical Applications of Heterogeneous Multi-Agent Systems. The PAAMS Collection*, 2014.
[28] S. Seufert and D. O'Brien. Machine learning for automatic defence against distributed denial of service attacks. In *IEEE ICC*, 2007.
[29] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi. A deep learning approach to network intrusion detection. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2018.
[30] A. Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje. Not just a black box: Learning important features through propagating activation differences, 2016.
[31] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2013.
[32] C. Sinclair, L. Pierce, and S. Matzner. An application of machine learning to network intrusion detection. In *Proceedings of ACSAC*, 1999.
[33] V. A. Siris and F. Papagalou. Application of anomaly detection algorithms for detecting syn flooding attacks. In *IEEE GLOBECOM*, Nov 2004.
[34] R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *Proceedings of IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2010.
[35] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 2014.
[36] Stanford. CS231n. http://cs231n.github.io/.
[37] Stanford. DL tutr. http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/.
[38] Taeshik Shon, Yongdae Kim, Cheolwon Lee, and Jongsub Moon. A machine learning framework for network anomaly detection using svm and ga. In *IEEE SMC Information Assurance Workshop*, 2005.
[39] Tao Peng, C. Leckie, and K. Ramamohanarao. Protection from distributed denial of service attacks using history-based ip filtering. In *IEEE ICC*, 2003.
[40] D. S. Terzi, R. Terzi, and S. Sagiroglu. Big data analytics for network anomaly detection from netflow data. In *UBMK*, 2017.
[41] UCI. KDD cup dataset. https://kdd.ics.uci.edu/databases/kddcup99/.
[42] S. Wachter, B. Mittelstadt, and C. Russell. Counterfactual explanations without opening the black box: Automated decisions and the gdpr, 2017.
[43] H. Wang, C. Jin, and K. G. Shin. Defense against spoofed ip traffic using hop-count filtering. *IEEE/ACM Transactions on Networking*, 2007.
[44] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 2005.
[45] J. Yu, H. Lee, M.-S. Kim, and D. Park. Traffic flooding attack detection with snmp mib using svm. *Computer Communications*, 2008.
[46] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks, 2013.
[47] L. M. Zintgraf, T. S. Cohen, T. Adel, and M. Welling. Visualizing deep neural network decisions: Prediction difference analysis, 2017.