Safeguarding Network Intrusion Detection Models from Zero-day Attacks and Concept Drift

Brian Matejek[1], Ashish Gehani[1], Nathaniel D. Bastian[2], Daniel Clouse[3], Bradford Kline[3], Susmit Jha[1]

[1]Computer Science Laboratory, SRI International
[2]Army Cyber Institute, United States Military Academy
[3]Laboratory for Advanced Cybersecurity Research, Department of Defense
brian.matejek@sri.com

Abstract

Machine learning models are increasingly being adopted to monitor network traffic and detect network intrusion. In this paper, we develop a deep learning architecture for traffic monitoring at the packet level. Despite high accuracy on inputs from the training distribution, these ML models fail to generalize to novel inputs, which limits their efficacy in the real world, where the network traffic is continuously evolving, and novel threats routinely emerge. Our deep learning framework introduces a safeguard that quantifies uncertainty in the decision made by the classification model. Our generative models learn class-conditional representations of the internal features of the DNNs. We demonstrate the effectiveness of our approach using packet-level CIC-IDS-2017 and UNSW-NB15 network intrusion datasets. We withhold from the training data certain attack categories to simulate zero-day attacks. Our encoder-only transformer model, which achieves an accuracy of over 99% when detecting known attacks, can only classify 1% of the novel attacks. Our proposed model safeguards that uses normalizing flows can achieve an AU ROC of over 0.97 in detecting these novel inputs.

Introduction

An increasing amount of new malware originates every day with both targeted and random attacks occurring against individuals, businesses, and government entities at large scales. These attacks often exploit known vulnerabilities or attempt to overwhelm system capacity to steal privileged information, disrupt legitimate traffic, and cause economic hardship to the victims. With an ever-increasing number of connected devices, manual monitoring of network traffic to identify malicious behavior is simply too expensive and infeasible. This has motivated increased adoption of machine learning (ML) (Leevy and Khoshgoftaar 2020), in particular deep learning (DL) (Ferrag et al. 2020), to aid and increase the bandwidth of cybersecurity professionals.

Traditionally, ML models for network traffic operate at the *flow-level* (Sarhan et al. 2020; Wang 2015; Kim, Shin, and Choi 2019). Network flow represents a continued connection between two devices parameterized by the five-tuple: (src_ip, src_port, dest_ip, dest_port, protocol). Flow summary tools such as CICFlowMeter (Draper-Gil et al. 2016; Lashkari et al. 2017) and Zeek produce features from the bidirectional flows on which an ML model can be trained. These features include flow duration, bytes per second, and flag settings. However, the features themselves are highly variable to the flow capture mechanism and its setup (Sarhan et al. 2020). For these reasons, we focus on the *packet-level* classification pipeline that labels traffic into benign and different attack categories at a per-packet granularity. Expanding on existing work of packet-level classification (Bierbrauer et al. 2022; De Lucia et al. 2021; Shenfield, Day, and Ayesh 2018), we include header context in inputs to our models to leverage additional information alongside the raw payload bytes.

While DNNs achieve high accuracy on inputs from the training distribution, they fail to generalize to novel inputs outside this distribution. Thus, their success on existing datasets does not entail their effectiveness in a real-world deployment where the network traffic is continuously evolving, and novel threats emerge routinely. We address this challenge by proposing a model-agnostic safeguard for DNNs that quantifies the uncertainty in the decision of any discriminative classification model by assigning a confidence score to each decision made by the DNN. The novel inputs are assigned low confidence and for such inputs, the decision by the machine learning model cannot be trusted. A high rate of detection of novel inputs by the monitor ensures wrong decisions by the DNN are avoided, and a low false positive rate ensures that the needed bandwidth of intervention (such as human experts) remains low. Thus, the proposed model safeguard improves the robustness of the DNN network intrusion detection models. This approach differs from unsupervised anomaly detection algorithms which look for concept drift before inference (Han et al. 2023; Rabanser, Günnemann, and Lipton 2019). We note that this is not an either-or situation. One could first detect anomalous inputs in an unsupervised manner before using a discriminative classifier. However, the classifier would still be susceptible to out-of-distribution inputs, requiring a safeguard.

Without such robust models, system operators relying on DNN models may have overconfidence in their network security. For example, incorrectly labeled benign network traffic may contain malicious payloads. In particular, DNN models need to remain robust in the presence of novel inputs such as zero-day exploits and distribution shift as network traffic and payload distribution evolves. Although we cannot currently expect models to generalize to correctly classify all packets from zero-day exploits without additional

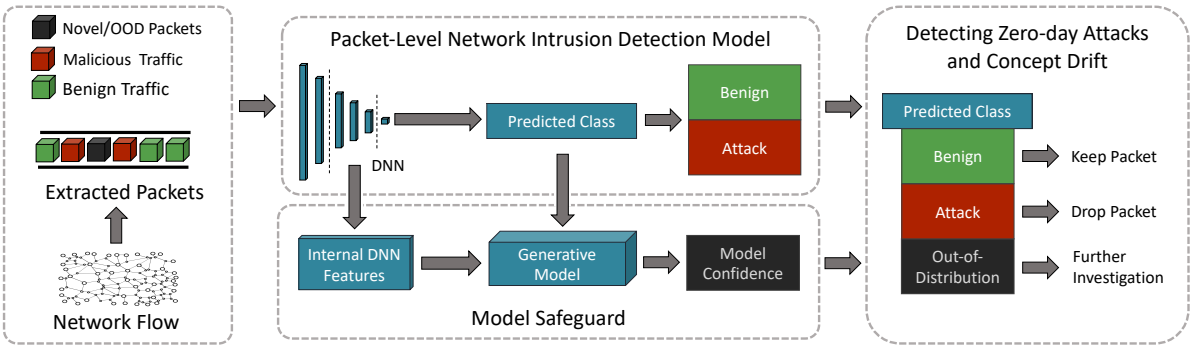

Figure 1: *Packet-level network intrusion detection system with model safeguards.* Our architecture ingests raw packet data and extracted information from the internet (IPv4) and transport layer (TCP/UDP) headers. We exclude information that is specific to the configuration used in data collection and not generalizable to the real-world. The predicted class and the internal features of the packet-level network intrusion detection model are used by the model safeguard. The monitor learns a class-condition distribution of the internal features and quantifies the uncertainty in prediction as a confidence score (that reflects the distance of internal features from the distribution of the features of the predicted class for any new input). The novel and out-of-distribution inputs are assigned low confidence score and can, hence, be detected for futher investigation.

finetuning, it is critical to recognize these novel inputs as out-of-distribution (Lee et al. 2018; Geng, Huang, and Chen 2020; Shen et al. 2021; Yang et al. 2021; Bulusu et al. 2020). By adding model safeguards to the DNN models, we can quantify uncertainty in the predictions from the models as a confidence score. Previous research has shown DNN models make overconfident wrong predictions on novel out-of-distribution inputs when considering softmax output (Guo et al. 2017) as the confidence score.

The overall uncertainty-quantified and robust deep learning architecture proposed in this paper is described in Figure 1. We complement the DNN model used for detecting attacks with a model safeguard that learns the class-conditional distribution of the internal features of the DNN model over the training data. For any new input at inference time, the predicted class and the internal features from the DNN model are used to query the learned distribution model for the likelihood of the input being in-distribution. The confidence of the prediction is high for inputs which have higher likelihood. The model safeguard-based architecture proposed in this paper applies to any DNN model and architecture. We observe very high in-distribution accuracy for the network intrusion datasets for relatively simple DNN architectures such as feed-forward neural networks and convolution neural networks, as well as more complex architectures such as transformers. Our model safeguard and uncertainty quantification approach is agnostic to the DNN architecture, and could even be used with non deep learning approaches for latency restricted applications.

We make the follow contributions in this paper. First, an encoder-only transformer architecture for the sequence to binary classification task that takes as input raw payload bytes and output a benign and malicious labeling. Second, a thorough analysis of the failures of this model in the presence of novel attacks and concept drift. Third, a model safeguard using normalizing flows that provides an uncertainty quantification score for our model. We show that our model safeguard can accurately model the class-conditional distributions of our in-distribution data and provides high AU ROC scores on the OOD detection problem.

Related Works

A significant body of research concerns the identification of malware at different points of the attack pipeline. Some ML models consider detecting malicious software at the earliest possible point by analyzing raw binary files. Early methods focused on identifying structural patterns within attributes from the PE headers in Windows executables (Anderson and Roth 2018; Saxe and Berlin 2015). More recent strategies attempt to exploit the advancements in computer vision by converting the raw binaries into images using N-grams (Mohammed et al. 2021). As an alternative, Ling *et al.* construct control graphs by unpacking and disassembling binaries (Ling et al. 2022). Although these methods achieve high classification accuracy, they are limited in scope as many frequent attacks exploit buffer overflows by tailoring inputs to otherwise benign programs (Butt et al. 2022). Furthermore, some disruption-oriented attacks such as the many variants of denial-of-service exploit the very processes that enable devices to interact like the SYN-ACK attack (Schuba et al. 1997). Network intrusion detection systems (IDS) classify network traffic as benign or malicious (Sharafaldin, Lashkari, and Ghorbani 2018; Moustafa and Slay 2015). These setups can identify ongoing denial-of-service attacks as well as advanced persistent threats (APTs) that have already infiltrated a compromised system.

Most existing efforts on learning-based network intrusion focus on classifying traffic flows with several available datasets (Sharafaldin, Lashkari, and Ghorbani 2018; Moustafa and Slay 2015). However, there are several difficulties in creating accurate training datasets for network intrusion detection such as adequately anonymizing data and correctly identifying malicious flows, among other prob-

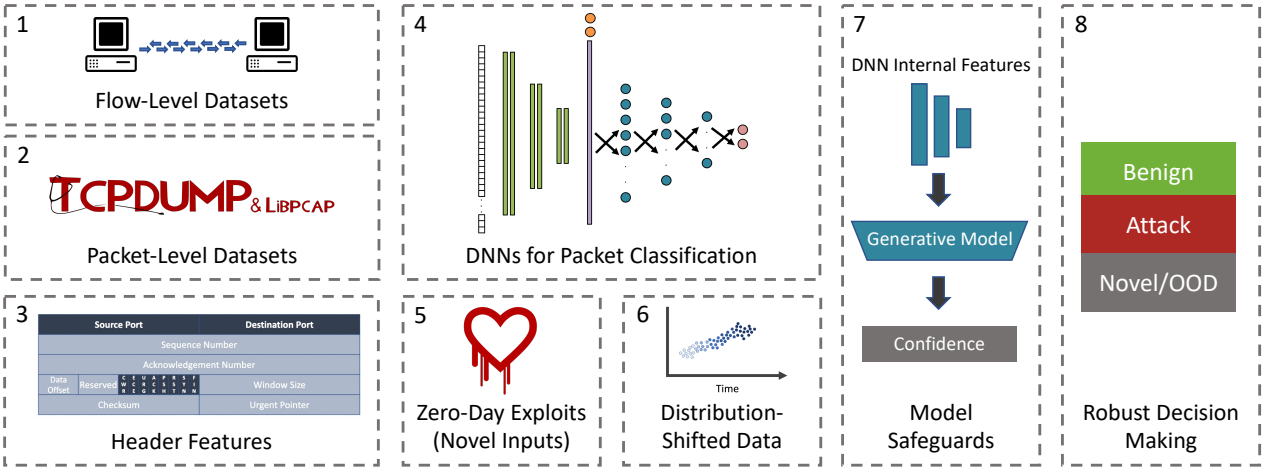

Figure 2: *System architecture overview.* We first take existing labeled flow-level datasets (box 1) and convert them into packet-level datasets by matching packets in a pcap file to their corresponding flows (box 2). We extract header features from the packets to augment our ML feature space, taking care to exclude features like source and destination IP that are network dependent (box 3). A DNN takes as input the payloads and header features to produce a binary classification of packets as benign or malicious (box 4). Despite high accuracy on the binary classification task, our model fails when given novel inputs, such as zero-day exploits (box 5) and inputs from a different time (box 6). We propose model safeguards that takes the internal features from the DNNs, models their distribution, and produces confidence in the predicted class (box 7). This enables us to have a robust decision-making process where we can classify inputs as benign, attack, or novel (box 8).

lems. Therefore, the Canadian Institute for Cybersecurity (CIC) and Intelligent Security Group (ISG) at the University of New South Wales (UNSW) produced several datasets that capture both benign and malicious network traffic in small, simulated environments (Moustafa and Slay 2015; Sharafaldin, Lashkari, and Ghorbani 2018; Koroniotis et al. 2019; Moustafa 2021). These datasets typically provide raw pcap files summarizing all of the packets that pass through a victim network. Frequently, the authors provide bidirectional flow-level summaries produced by tools such as CICFlowMeter (Draper-Gil et al. 2016; Lashkari et al. 2017) or Zeek. These summaries include features such as flow duration, number of forward and backward packets, and number of flags set, among others.

A significant amount of research considers the problem of manually engineering features to capture traffic flows and using these features for the detection of malicious flows. Several different neural network architectures have been explored (Maxwell, Alhajjar, and Bastian 2019; Pelletier and Abualkibash 2020; Wankhede and Kshirsagar 2018; Vinayakumar et al. 2019) in addition to the traditional machine learning models such as AdaBoost (Yulianto, Sukarno, and Suwastika 2019) or random forests (Wankhede and Kshirsagar 2018). Kim *et al.* train a convolutional neural network (CNN) classifier on these features by transforming the 78-dimensional feature space into images (Kim, Shin, and Choi 2019). Since flow-level detection is ex post facto and real-time detection requires packet-level classification, real-time detection requires packet-level detection. There has been some work on packet-level detection of malicious traffic (Wang 2015; Bierbrauer et al. 2022; De Lucia et al. 2021; Shenfield, Day, and Ayesh 2018) wherein, the model takes as input the raw payload data, typically from the transport layer. These previous approaches have generally focused on the discriminative models, with some analysis on transfer learning in the face of concept drift (Bierbrauer et al. 2022). However, they have not considered the problem of detecting inputs outside of the training distribution.

The problem of overconfident incorrect prediction by deep learning models on novel inputs that are out-of-distribution has been previously reported in domains such as computer vision (Guo et al. 2017) and several methods to compute confidence of deep learning models in these domains have been proposed. Broadly, these methods can be categorized into two classes. The first are the supervised techniques (Lee et al. 2017; Hendrycks, Mazeika, and Dietterich 2019; Meinke and Hein 2019; Kaur et al. 2021) that require some exposure to the out-of-distribution inputs. This is impractical for cybersecurity where we cannot assume even a small number of packets corresponding to novel attack classes would be available at training time. Unsupervised approaches use only the in-distribution data. A direct approach is to use the softmax score of the classifier as a confidence metric (Hendrycks and Gimpel 2016). ODIN (Liang, Li, and Srikant 2017) enhances the softmax score by adding perturbations to the input and using temperature scaling to the classifier's confidence. Recently, (Macedo et al. 2021) proposed replacing softmax scores with isomax scores and entropy maximization for detection. Other unsupervised detection techniques based on energy scores (Liu et al. 2020), trust scores (Jiang et al. 2018), and likelihood ratio (Ren et al. 2019) between the

in-distribution and OOD data points have been proposed for detection. In contrast, we use a normalizing flow generative model for learning the distribution of internal features. These uncertainty-quantification and out-of-distribution detection approaches rely on learning the manifold or distribution of training data and are known to be susceptible to adversarial attacks (Jang, Jha, and Jha 2020).

Overview

We provide an overview of our system architecture in Figure 2. We take as input pre-existing flow-level IDS datasets that also provide raw pcap files. Typically, these flow-level datasets use a flow summary tool to create a set of labeled features such as flow duration, and the number of forward and backward packets, among others. The flow labels come from existing knowledge of the test-bed architectures where set IPs and timestamps correspond to malicious activity of a certain type. Since the pcap files provided with the flow-level datasets are not labeled, we process the data to create packet-level datasets. We convert the packets into input features for our neural networks by first taking the first 1,500 bytes of the payload, and second, extracting relevant features. We need to take care to not extract features that are trivially correlated with benign and malicious behavior, such as source or destination IP. We consider an encoder-only transformer architecture for the sequence to binary classification task. Although our DNNs perform with high accuracies ($> 99\%$) on in-distribution data, we find that the networks perform poorly when we mimic novel inputs (e.g., zero-day exploits), or when the data consists of packets from a different timeframe and network. We extract internal features from the DNN and model their distributions through two methods to produce a confidence (or novelty) score to produce during inference. This end-to-end system enables us to produce packet-level predictions with an uncertainty-quantification.

Technical Approach

Throughout our work, we use the following terminology: a *packet label* refers to the binary classification of a packet (i.e., benign or malicious), whereas a *packet category* refers to the multiclass classification of a packet that further granularlizes attack types (i.e., benign, denial-of-service, heartbleed, fuzzers, etc.).

Packet-Level Capture

Most existing network IDS datasets provide flow summaries with accompanying benign and attack category labels (Moustafa and Slay 2015; Sharafaldin, Lashkari, and Ghorbani 2018). We instead focus on classifying packets as benign and malicious for a couple of reasons.

First, flow capture devices extract different features from pcap files which creates difficulties when designing ML solutions that must work across a wide range of differently configured networks. Sarhan *et al.* use NetFlow (Claise 2004) to extract features from four publicly available IDS datasets (Sarhan et al. 2020). NetFlow is easy to configure and generates features using almost solely packet headers; this ease of use, however, leads to a less expressive feature space compared to hand-tailored flow capture systems. In contrast, different packet capture technology running on different operating systems produce standardized PCAP files.

Second, learning benign and attack characteristics at the flow level is inherently a retroactive process. One has to wait for a flow to conclude before extracting features such as the number of forward and backward packets sent, mean payload length, and average time between packets, amongst others. Although this can be very useful in postmortem contexts where a network administrator wants to analyze fault points in a system's security after an attack concludes, it is a less viable paradigm for identifying ongoing and incoming attacks. By classifying packets as they arrive as benign or malicious, one can actively reject traffic before the payload arrives at its intended destination.

Extracting Header Features

We extract information from the packet headers to augment our input feature space. However, we cannot simply input the entire header as our models will learn the setup configuration of the testbed architectures. For example, an ML model could trivially learn the source and destination IP addresses that correspond to devices in the attack network. Therefore, we carefully consider the header fields that provide useful contextual information that is transferable to other network configurations.

We only make use of the "Total Length" field in IPv4 headers. We divide this field by 65,536 to standardize the value between 0 and 1. Notably, we do not consider information in the "TTL", "Protocol", or "Address" fields. Although time-to-live might be useful in real-world applications with diverse network topologies, the publicly available datasets have very few plausible attack paths because of the limited number of victim and attack devices. We found that one of the most cited IDS datasets (Sharafaldin, Lashkari, and Ghorbani 2018) almost exclusively uses the TCP transport protocol for attacks. Therefore, we exclude the protocol field since the great imbalance is not indicative of the real world where some attacks, such as UDP Flood, use the UDP protocol (Dittrich 1999). Lastly, most network IDS datasets have set IP addresses for the victim and attack devices. Training data with those fields would produce a model unable to generalize to other configuration settings.

For the transport layer headers, we produce a set of features based on the source and destination ports. We cannot simply produce a one-hot encoding for each port since many ports would indicate benign or attack based on the scripts that generate attacks on IDS datasets. After considering some commonly used network IDS datasets, we generate seven binary features for both the source and destination ports of every packet. These features are non-exclusive, i.e., at least one but perhaps two of the features can receive a value of one. We summarize the features in Table 1. Note that we group ports 443 and 444—the heartbleed attacks on the CIC-IDS-2017 dataset exploit the SSL vulnerability on port 444. Lastly, we include the following eight flags from the TCP header: CWR, ECE, URG, ACK, PSH, RST, SYN, and FIN. We set these features to zero for UDP packets.

Table 1: We construct seven indicator features from the source and destination ports. Features for sets that contain a given port number receive a value of one.

Port Number	Description
21	File Transfer Protocol (FTP)
22	Secure Shell (SSH)
80/8080	Hypertext Transfer Protocol (HTTP)
443/444	Hypertext Transfer Protocol Secure (HTTPS)
0 - 1,023	Well Known Ports
1,024 - 49,151	Registered Ports
49,152 - 65535	Dynamic/Private Ports

Deep Learning for Packet Data

We design a custom encoder-only transformer architecture for the sequence to classification problem of going from raw payload bytes to a binary classification of packets. We use a simple tokenization scheme to convert our payloads into vectors of tokens. We convert each byte in the payload into the corresponding ASCII number. Thus, we have a vocabulary size of 256 corresponding to the 256 character codes. We find existing subword tokenization strategies (Kudo and Richardson 2018; Song et al. 2020) provide sub-optimal results on payload data, in part because encryption and compression create high entropy payloads that do not correlate to the structured text in natural languages. Our encoded payloads are then input into a series of transformer encoder blocks to create an embedding for each token (input byte) (Vaswani et al. 2017). We then use mean pooling to convert the array of embeddings into a single sentence embedding (Reimers and Gurevych 2019). We then concatenate the header features to the sentence embedding and input the resultant vector into three hidden fully connected layers with 256, 128, and 64 units. Each of the hidden layers has a LeakyRELU activation with $\alpha = 0.01$ (Maas et al. 2013) and batch normalization (Ioffe and Szegedy 2015) and dropout regularization (Srivastava et al. 2014). The last layer uses a softmax activation function.

Handling Zero-day Exploits

Machine learning models generally perform well when given in-distribution testing data, i.e., data similar to the training data. However, real-world cybersecurity defense implementations will eventually receive data that falls out of the training distribution. For example, zero-day exploits cannot by definition exist in the training data. It is imperative that any machine learning systems do not fail when such exploits appear. Here, we focus on false negatives where our model classifies packets from previously unseen attack paradigms as benign.

We see similar characteristics between the zero-day exploit problem and the open-set recognition one in the broader machine learning community (Geng, Huang, and Chen 2020). In open-set recognition, a model trains on incomplete data that does not include all classes that exist during inference. We model this behavior by training ML models on the benign traffic and all but one type of attack. We repeat this procedure for each attack category, to create N trained models where each model corresponds to a single missing attack type. We consider two different metrics for success for our models: first, we can correctly classify the unseen attack types as malicious, and second, we can recognize that the attack types are unfamiliar and flag them as out-of-distribution. Correctly classifying unseen attacks as malicious indicates that our model is learning general attack patterns in the payloads themselves. For example, some of the brute force attack methods have similar characteristics and so removing one from training does not change the quality of inference. However, more often, when we remove an attack from the training data, we classify those packets on inference as benign. Thus, it is imperative to identify those packets as out-of-distribution.

Handling Distribution Shift

Internet most-common and best practices continually evolve, especially as privacy and security concerns become higher societal priorities. As the landscape morphs, the structure of the payloads changes, and previously trained models can become outdated. In recent years, an ever-increasing number of websites have transitioned from HTTP connections to HTTPS. Sometimes this evolution occurs organically as more websites adopt existing libraries to improve privacy. In such instances, we expect a moderate distribution shift over small stretches of time. We model the natural distribution shift of network traffic by training on a dataset from Q3 2017 and inferring on one from Q1 2015. This 2.5-year difference corresponds to a significant period of change from unencrypted (HTTP) to encrypted web traffic (HTTPS), from approximately 30% in Q1 2015 to 60% in Q2 2017, per the percentage of web pages loaded by Firefox using HTTPS (letsencrypt). However, occasionally targeted government mandates or external business pressures encourage the rapid adoption of a new framework or protocol, such as when Google Chrome began labeling all HTTP connections as "Not Secured" in 2018 (Robbins 2021). These "seismic" events can cause a significant break between the distributions of current network traffic from previous months (by Q2 2018, over 80% of web pages in the United States loaded by Firefox used HTTPS (letsencrypt)).

Out-of-Distribution Detection

We extract the outputs from three intermediate layers from our transformer neural network: that is, the first, second, and third fully-conencted layers after the transformer blocks and sentence embedding. We then model the distributions of these features for benign and malicious packets using two different methods.

We use normalizing flows for our model safeguard. In the normalizing flows paradigm, a model learns a series of bijective transformations that can perform a one-to-one mapping from a simple distribution, such as a multivariate Gaussian, into a complex target distribution (Papamakarios et al. 2021; Kobyzev, Prince, and Brubaker 2020). Since each transformation block is invertible, we can take a feature vector and easily determine its corresponding value in the probability density function of the complex distribution by applying a series of matrix multiplications followed by the inverse of

the simple activation functions. In this way, we extract features from the intermediate layers of our neural networks for the in-distribution training data and train normalizing flows that transform a multivariate Gaussian distribution into this complex space. During inference, we extract features from our network and transform the data into the multivariate Gaussian space. By looking at the loss (i.e., the probability that the vector belongs to the complex distribution), we can order our inputs by the probability that they belong to the training set distribution. Inputs with a lower negative log-likelihood loss are more likely to be in-distribution. We train two normalizing flows for each model, one each for benign and attack network traffic. We only consider the normalizing flow model that matches the output label from our NN classifier during inference.

There are several normalizing flow blocks common in the literature (Dinh, Sohl-Dickstein, and Bengio 2016; Dinh, Krueger, and Bengio 2014; Kingma and Dhariwal 2018; Sorrenson, Rother, and Köthe 2020), with each offering advantages and trade-offs during training and forward and reverse inference. For our purposes, we use RealNVP blocks (Dinh, Sohl-Dickstein, and Bengio 2016) that we can parameterize with the following equation:

$$y = R\Psi(s_{\text{global}}) \odot \text{Coupling}\Big(R^{-1}x\Big) + t_{\text{global}} \tag{1}$$

where R is a (deterministic) permutation matrix that allows each feature to influence the others, $\odot$ is the Hadamard Product (element-wise multiplication), x is the input vector for each block, s_{global} and t_{global} are learnable parameters, and Coupling is the following function that first evenly divides the input vector into halves x_1 and x_2:

$$u = \text{concat}(u_1, u_2) \tag{2}$$

$$u_1 = x_1 \odot \exp\left(\alpha \tanh\left(s(x_2)\right)\right) + t(x_2) \tag{3}$$

$$u_2 = x_2 \tag{4}$$

α is a clamping value that restricts the range of possible values in the exponent, and s and t are learnable parameters. Note that in this coupling block, features in x_2 can influence outputs in u_1 but not vice versa. The permutation matrix allows each feature to influence the others when stacking multiple blocks during training.

To avoid overfitting to our training distribution, we add a small amount of Gaussian noise to our features during training and inference with the following equation:

$$X = X + \mathcal{N}(0, 0.05) \tag{5}$$

This improves the stability of the training procedure by washing out any features with no discernible signal.

Experiments

Datasets

We evaluate our system using two different network intrusion detection datasets that publish both raw pcap data and corresponding hand-labeled flow data. Both datasets configure a testbed infrastructure with an attack and a victim network comprised of multiple connected devices. The attack networks initiate a series of different attack profiles throughout data collection.

CIC-IDS-2017. The CIC-IDS-2017 dataset (Sharafaldin, Lashkari, and Ghorbani 2018) contains raw pcap data captured over a week-long period in July 2017. The devices on the victim network run different versions of the three most common operating systems (Windows, Mac, and Linux). Three of the attacking PCs have the Windows 8.1 operating system and the fourth has Kali Linux. The dataset authors configured one of the ports of the main switch as a mirror port that completely captures all traffic traversing into and out of the victim network. A B-Profile system profiles the abstract (benign) behavior of 25 different users and an automated agent derived from these profiles generates realistic benign events for packet capture (Sharafaldin et al. 2018). The CIC-IDS-2017 dataset contains fourteen attack types from seven common attack families: brute force, heartbleed, botnet, denial-of-service, distributed denial-of-service, web, and infiltration.

We note that others (Liu et al. 2022; Engelen, Rimmer, and Joosen 2021) have found issues with this particular dataset caused in part by bugs in the CICFlowMeter (Draper-Gil et al. 2016; Lashkari et al. 2017) tool. We avoid many of the troublesome artifacts by approaching the problem at a packet-level. For example, we drop any packets that have a "TCP appendix" error simply since we do not consider empty payloads. The other main artifacts come from attempted attacks that fail to deliver their malicious payloads. We still include these packets in our analysis, as watching an attack unfold remains a valuable goal, even before the malicious content arrives.

UNSW-NB15. The UNSW-NB15 dataset (Moustafa and Slay 2015) captures raw network traffic over two full days (fifteen and sixteen hours) in January and February 2015. In the testbed architecture, an IXIA traffic generator uses three virtual servers, two that spread benign network traffic and one that forms malicious activity. The servers pass all traffic through two routers connected to a firewall that allows all packets to pass through. The dataset authors installed tcpdump (Jacobson 1989) on one of the routers to capture all packet data. On the first day, the IXIA tool generates one attack and ten attacks per second until fifty gigabytes of data are collected on each day, respectively. The IXIA tool simulated nine different attack categories including analysis, backdoor, denial-of-service, exploits, fuzzers, generic, reconnaissance, shellcode, and worms.

Data Division

Both datasets have highly imbalanced classes with 20 to 25 times as many benign packets as attack ones. Therefore, when constructing our data sets for training and inference, we sample from the benign data to produce two balanced classes of benign and malicious traffic. We split this data evenly into training and testing sets and stratify by packet category to divide each attack type equally. We repeat this process (including the benign packet sampling) ten times to produce ten unique training and testing datasets. We run every experiment on each train/test split and provide both the

means and standard deviations. When training our classification models, we preserve 25% of the training data for validation of model parameters.

Training Parameters

We use an embedding size of 384, used previously in the sentence transformer sequence to classification tasks (Reimers and Gurevych 2019). We stack two transformer blocks, and each block has six self-attention heads for 64-dimensional key, value, and query vectors (Vaswani et al. 2017). We use batch normalization (Ioffe and Szegedy 2015) and dropout ($p = 0.2$) (Srivastava et al. 2014) regularization techniques for our fully connected layers after the transformer block. We use the AMSGrad variant of the Adam optimizer (Kingma and Ba 2014; Reddi, Kale, and Kumar 2019) with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and a learning rate of $3\text{e}{-4}$. We use the binary cross-entropy loss function and train each network for six epochs.

For our normalizing flow models, we stack 20 RealNVP blocks with an affine clamping $\alpha = 2$. We learn our parameters for s and t using a simple fully connected network with two hidden layers with 128 features each and LeakyRELU activation with $\alpha = 0.01$. The input and output dimensions of these learnable blocks are dependent on the size of the extracted NN layers (either 256, 128, or 64 dimensions). We use the Adam optimizer (Kingma and Ba 2014) with $\beta_1 = 0.8$, $\beta_2 = 0.9$, a learning rate of $1\text{e}{-4}$, and weight decay of $2\text{e}{-5}$. We train each normalizing flow model for 512 epochs and use 25% of in-distribution data for validation.

Implementation

We implement our system in Python using PayloadByte (Farrukh et al. 2022) for extracting and labeling packet capture (PCAP) files of modern network intrusion detection system datasets, Pytorch (Paszke et al. 2019) for our neural networks and the Framework for Easily Invertible Architectures (FrEIA) (Ardizzone et al. 2018-2022) for our normalizing flows. Our code is freely available.[1].

Results

Packet-level Classification Accuracy

With our transformer architecture, we achieve an average accuracy, AU ROC, and F1-Score of 99.40%, 0.9997, and 0.9940, respectively. For these results, we train ten models with half of the CIC-IDS-2017 data stratified by packet category. The training and testing data includes attacks of every possible type. In real-world networks, the number of benign packets far exceeds the number of attack packets. We find that when introducting all of the possible testing benign packets, our model accuracy increases to 99.63%.

Handling Zero-day Exploits

We reformulate the problem of classifying inputs from zero-day exploits into the more general open-set recognition task in machine learning. Using this strategy, we train models on all benign packets and all but one type of attack. We then can evaluate how well our model predicts packets from the withheld classes. In Table 2, the "In-Distribution" column shows the recall of predicting a packet from the category in the first column as an attack. For example, we classify 99.94% of FTP-Patator packets as malicious when we include FTP-Patator samples in the training data. The last "Out-of-Distribution" column shows the recall for a given attack when we exclude only that attack from training.

Table 2: The "In-Distribution" column refers to one model trained with all attack types and provides the recall for each attack on withheld testing data. The "Out-of-Distribution" column refers to fourteen models with the indicated attack excluded from training.

Attack Category	In-Distribution (↑)	Out-of-Distribution (↑)
Bot	94.24% (±2.70)	9.50% (±9.35)
DDoS	100.00% (±0.00)	21.45% (±2.58)
DoS GoldenEye	99.99% (±0.01)	19.95% (±4.26)
DoS Hulk	100.00% (±0.00)	75.29% (±16.26)
DoS Slowhttptest	99.80% (±0.25)	52.48% (±16.59)
DoS Slowloris	99.96% (±0.02)	17.37% (±28.59)
FTP-Patator	**99.94% (±0.02)**	**0.30% (±0.19)**
Heartbleed	44.18% (±6.60)	0.02% (±0.02)
Infiltration	**99.46% (±0.51)**	**7.80% (±3.12)**
Port Scan	98.56% (±0.71)	80.58% (±2.34)
SSH-Patator	**99.98% (±0.01)**	**0.27% (±0.26)**
Web Attack-Brute Force	99.97% (±0.01)	94.39% (±7.01)
Web Attack-SQL Injection	72.22% (±17.45)	46.78% (±13.82)
Web Attack-XSS	99.59% (±0.30)	63.05% (±17.17)

We highlight three specific attack categories where recalls fall precipitously when excluded from training: FTP-Patator, Infiltration, and SSH-Patator. When included in the training, we classify between 99.46–99.98% of these malicious payloads. However, our models achieve accuracies between 0.27–7.80% when excluding the various classes from training. Perhaps more surprisingly, we can correctly classify several attack categories as malicious despite removing them from training, suggesting a general hierarchy of attacks where some are merely derivative of others. The WebAttack-BruteForce, DoS Hulk, and Port Scan attacks see only limited degradation in recall. The results are illustrated in Table 2. We focus on the bolded attack categories when evaluating our model safeguard since those attack categories had a large drop in recall when in- and out-of-distribution.

Handling Distribution Shift

We note substantial declines in accuracy when we train on the CIC-IDS-2017 dataset and infer on the UNSW-NB15 dataset with an accuracy of 48.07%. We do not find these results surprising since distribution shift is a well-known phenomenon in the vision domain (Kulinski and Inouye 2022; Taori et al. 2020) with similar results in network traffic data (Bierbrauer et al. 2022). However, it does highlight the need for real-time monitoring of any cybersecurity ML models. We note that the benign accuracy on the UNSW-NB15 inference data is 92.16%, whereas most of the attack categories have accuracies less than 25%. Thus, our model nearly always classifies packets as benign in this scenario.

[1] github.com/SRI-CSL/trinity-packet

Table 3: We achieve high scores across all metrics using normalizing flows to identify OOD inputs.

Layer	AU ROC (↑)	TPR (TNR 95%) (↑)
FTP-Patator		
linear1	**0.9957 (±0.0008)**	**100.00% (±0.00%)**
linear2	0.9792 (±0.0060)	95.84% (±7.63%)
linear3	0.9644 (±0.0137)	77.51% (±19.94%)
Infiltration		
linear1	0.9614 (±0.0132)	70.60% (±21.01%)
linear2	**0.9742 (±0.0149)**	**86.69% (±15.44%)**
linear3	0.9528 (±0.0358)	66.26% (±28.85%)
SSH-Patator		
linear1	**0.9921 (±0.0014)**	**100.00% (±0.00%)**
linear2	0.9742 (±0.0055)	94.58% (±7.02%)
linear3	0.9513 (±0.0125)	64.12% (±15.48%)
UNSW-NB15		
linear1	0.9263 (±0.0071)	77.83% (±0.39%)
linear2	0.9536 (±0.0072)	**77.96% (±3.03%)**
linear3	**0.9583 (±0.0090)**	77.71% (±4.49%)

Detecting OOD Inputs

We evaluate our model safeguard with two metrics: area under the receiver operating characteristic (AU ROC) and true positive rate at a true negative rate of 95% (TPR (TNR 95%)) (Table 3). For both metrics, we first calculate the normalizing flow loss for each inference packet. Higher losses indicate an inference input that we deem farther from the target in-distribution data. For AU ROC, we order the inputs by increasing score (farther from the target distribution) with binary labels corresponding to the out-of-distribution classes. For the TPR (TNR) score, we adopt a hard threshold for what constitutes in- and out-of-distribution. Under these metrics, we allow ourselves to dispose of 5% of the in-distribution inference data (i.e., achieve a true negative rate of 95%). We then take the corresponding distance at those two thresholds and assign any inputs with scores greater than that as out-of-distribution. Our true positive rate thus refers to the proportion of out-of-distribution data that had a measured distance greater than those values. For both metrics, higher values are better. For some networks, losing 5% of in-distribution may not be tolerable. The AU ROC score in part shows the sensitivity to toggling that threshold.

Table 3 show the results for detecting out-of-distribution inputs using our normalizing flows model safeguard. We note high AU ROC scores with optimal values between 0.9742 and 0.9957 for each withheld attack class. However, we achieve high score regardless of which layer we extract features for conditional class modeling. The UNSW-NB15 section shows our results for detecting concept drift as OOD. We note lower AU ROC scores than for the withheld CIC attack classes. We attribute this, in part, to the higher accuracies on the UNSW-NB15 dataset. Although there was a distribution shift between the datasets, we achieve higher accuracies. We correctly classify many of these benign packets. Thus some of these packets may not truly be out-of-distribution.

Conclusion

There are several challenges to using deep learning methods for classifying packets as benign or malicious. We demonstrate that DNNs trained for network intrusion detection on these datasets achieve high accuracy (over 99%) on in-distribution inputs, but fail drastically (accuracy below 1%) in the presence of zero-day attacks, with severe degradation in the presence of concept drift. We address this robustness challenge by constructing a model safeguard that uses the prediction of the DNN classifier and its internal features for uncertainty quantification and to detect novel out-of-distribution inputs. We use an encoder-only transformer architecture for our discriminative classifier with a normalizing flows model for our safeguard. However, our overall framework could extend to other models based on latency requirements and compute resources.

References

Anderson, H. S.; and Roth, P. 2018. Ember: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637*.

Ardizzone, L.; Bungert, T.; Draxler, F.; Köthe, U.; Kruse, J.; Schmier, R.; and Sorrenson, P. 2018-2022. Framework for Easily Invertible Architectures (FrEIA).

Bierbrauer, D. A.; De Lucia, M.; Reddy, K.; Maxwell, P.; and Bastian, N. D. 2022. Transfer Learning for Raw Network Traffic Detection. *Expert Systems with Applications*, 211(118641): 1.

Bulusu, S.; Kailkhura, B.; Li, B.; Varshney, P. K.; and Song, D. 2020. Anomalous example detection in deep learning: A survey. *IEEE Access*, 8: 132330–132347.

Butt, M. A.; Ajmal, Z.; Khan, Z. I.; Idrees, M.; and Javed, Y. 2022. An In-Depth Survey of Bypassing Buffer Overflow Mitigation Techniques. *Applied Sciences*, 12(13): 6702.

Claise, B. 2004. Cisco systems netflow services export version 9. Technical report, Cisco.

De Lucia, M. J.; Maxwell, P. E.; Bastian, N. D.; Swami, A.; Jalaian, B.; and Leslie, N. 2021. Machine learning raw network traffic detection. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III*, volume 11746, 185–194. SPIE.

Dinh, L.; Krueger, D.; and Bengio, Y. 2014. Nice: Nonlinear independent components estimation. *arXiv preprint arXiv:1410.8516*.

Dinh, L.; Sohl-Dickstein, J.; and Bengio, S. 2016. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*.

Dittrich, D. 1999. The DoS Project's 'trinoo'distributed denial of service attack tool.

Draper-Gil, G.; Lashkari, A. H.; Mamun, M. S. I.; and Ghorbani, A. A. 2016. Characterization of encrypted and vpn traffic using time-related. In *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*, 407–414.

Engelen, G.; Rimmer, V.; and Joosen, W. 2021. Troubleshooting an intrusion detection dataset: the CICIDS2017 case study. In *2021 IEEE Security and Privacy Workshops (SPW)*, 7–12. IEEE.

Farrukh, Y. A.; Khan, I.; Wali, S.; Bierbrauer, D.; Pavlik, J. A.; and Bastian, N. D. 2022. Payload-Byte: A Tool for Extracting and Labeling Packet Capture Files of Modern Network Intrusion Detection Datasets. *Proceedings of the 9th IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT2022)*.

Ferrag, M. A.; Maglaras, L.; Moschoyiannis, S.; and Janicke, H. 2020. Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. *Journal of Information Security and Applications*, 50: 102419.

Geng, C.; Huang, S.-j.; and Chen, S. 2020. Recent advances in open set recognition: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 43(10): 3614–3631.

Guo, C.; Pleiss, G.; Sun, Y.; and Weinberger, K. Q. 2017. On calibration of modern neural networks. In *International conference on machine learning*, 1321–1330. PMLR.

Han, D.; Wang, Z.; Chen, W.; Wang, K.; Yu, R.; Wang, S.; Zhang, H.; Wang, Z.; Jin, M.; Yang, J.; et al. 2023. Anomaly Detection in the Open World: Normality Shift Detection, Explanation, and Adaptation. In *30th Annual Network and Distributed System Security Symposium (NDSS)*.

Hendrycks, D.; and Gimpel, K. 2016. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*.

Hendrycks, D.; Mazeika, M.; and Dietterich, T. 2019. Deep anomaly detection with outlier exposure. *In International Conference on Learning Representations*.

Ioffe, S.; and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, 448–456. PMLR.

Jacobson, V. 1989. Tcpdump. *ftp://ftp. ee. lbl. gov*.

Jang, U.; Jha, S.; and Jha, S. 2020. On the Need for Topology-Aware Generative Models for Manifold-Based Defenses. *8th International Conference on Learning Representations (ICLR) 2020*.

Jiang, H.; Kim, B.; Guan, M.; and Gupta, M. 2018. To trust or not to trust a classifier. In *Advances in neural information processing systems*, 5541–5552.

Kaur, R.; Jha, S.; Roy, A.; Park, S.; Sokolsky, O.; and Lee, I. 2021. Detecting oods as datapoints with high uncertainty. *arXiv preprint arXiv:2108.06380*.

Kim, J.; Shin, Y.; and Choi, E. 2019. An intrusion detection model based on a convolutional neural network. *Journal of Multimedia Information System*, 6(4): 165–172.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P.; and Dhariwal, P. 2018. Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems*, 31.

Kobyzev, I.; Prince, S. J.; and Brubaker, M. A. 2020. Normalizing flows: An introduction and review of current methods. *IEEE transactions on pattern analysis and machine intelligence*, 43(11): 3964–3979.

Koroniotis, N.; Moustafa, N.; Sitnikova, E.; and Turnbull, B. 2019. Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. *Future Generation Computer Systems*, 100: 779–796.

Kudo, T.; and Richardson, J. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*.

Kulinski, S.; and Inouye, D. I. 2022. Towards Explaining Image-Based Distribution Shifts. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4788–4792.

Lashkari, A. H.; Draper-Gil, G.; Mamun, M. S. I.; Ghorbani, A. A.; et al. 2017. Characterization of tor traffic using time based features. In *ICISSp*, 253–262.

Lee, K.; Lee, H.; Lee, K.; and Shin, J. 2017. Training confidence-calibrated classifiers for detecting out-of-distribution samples. *arXiv preprint arXiv:1711.09325*.

Lee, K.; Lee, K.; Lee, H.; and Shin, J. 2018. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. *Advances in neural information processing systems*, 31.

Leevy, J. L.; and Khoshgoftaar, T. M. 2020. A survey and analysis of intrusion detection models based on cse-cic-ids2018 big data. *Journal of Big Data*, 7(1): 1–19.

letsencrypt. 2023. Let's encrypt stats. https://letsencrypt.org/stats/. Accessed: November 29, 2023.

Liang, S.; Li, Y.; and Srikant, R. 2017. Enhancing the reliability of out-of-distribution image detection in neural networks. *arXiv preprint arXiv:1706.02690*.

Ling, X.; Wu, L.; Deng, W.; Qu, Z.; Zhang, J.; Zhang, S.; Ma, T.; Wang, B.; Wu, C.; and Ji, S. 2022. MalGraph: Hierarchical Graph Neural Networks for Robust Windows Malware Detection. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, 1998–2007. IEEE.

Liu, L.; Engelen, G.; Lynar, T.; Essam, D.; and Joosen, W. 2022. Error Prevalence in NIDS datasets: A Case Study on CIC-IDS-2017 and CSE-CIC-IDS-2018. In *2022 IEEE Conference on Communications and Network Security (CNS)*, 254–262. IEEE.

Liu, W.; Wang, X.; Owens, J. D.; and Li, Y. 2020. Energy-based Out-of-distribution Detection. *arXiv preprint arXiv:2010.03759*.

Maas, A. L.; Hannun, A. Y.; Ng, A. Y.; et al. 2013. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, 3. Atlanta, Georgia, USA.

Macedo, D.; Ren, T. I.; Zanchettin, C.; Oliveira, A. L. I.; and Ludermir, T. 2021. Entropic Out-of-Distribution Detection. arXiv:1908.05569.

Maxwell, P.; Alhajjar, E.; and Bastian, N. D. 2019. Intelligent Feature Engineering for Cybersecurity. In *2019 IEEE International Conference on Big Data (Big Data)*, 5005–5011. IEEE.

Meinke, A.; and Hein, M. 2019. Towards neural networks that provably know when they don't know. *arXiv preprint arXiv:1909.12180*.

Mohammed, T. M.; Nataraj, L.; Chikkagoudar, S.; Chandrasekaran, S.; and Manjunath, B. 2021. Malware detection using frequency domain-based image visualization and deep learning. *arXiv preprint arXiv:2101.10578*.

Moustafa, N. 2021. A new distributed architecture for evaluating AI-based security systems at the edge: Network TON_IoT datasets. *Sustainable Cities and Society*, 72: 102994.

Moustafa, N.; and Slay, J. 2015. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *2015 military communications and information systems conference (MilCIS)*, 1–6. IEEE.

Papamakarios, G.; Nalisnick, E. T.; Rezende, D. J.; Mohamed, S.; and Lakshminarayanan, B. 2021. Normalizing Flows for Probabilistic Modeling and Inference. *J. Mach. Learn. Res.*, 22(57): 1–64.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Pelletier, Z.; and Abualkibash, M. 2020. Evaluating the CIC IDS-2017 dataset using machine learning methods and creating multiple predictive models in the statistical computing language R. *Science*, 5(2): 187–191.

Rabanser, S.; Günnemann, S.; and Lipton, Z. 2019. Failing loudly: An empirical study of methods for detecting dataset shift. *Advances in Neural Information Processing Systems*, 32.

Reddi, S. J.; Kale, S.; and Kumar, S. 2019. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*.

Reimers, N.; and Gurevych, I. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Ren, J.; Liu, P. J.; Fertig, E.; Snoek, J.; Poplin, R.; Depristo, M.; Dillon, J.; and Lakshminarayanan, B. 2019. Likelihood ratios for out-of-distribution detection. In *Advances in Neural Information Processing Systems*, 14707–14718.

Robbins, M. 2021. Effective July 2018, Google's chrome browser will mark non-HTTPS sites as 'not secure'. https://searchengineland.com/effective-july-2018-googles-chrome-browser-will-mark-non-https-sites-as-not-secure-291623. Accessed: December 2, 2022.

Sarhan, M.; Layeghy, S.; Moustafa, N.; and Portmann, M. 2020. Netflow datasets for machine learning-based network intrusion detection systems. In *Big Data Technologies and Applications*, 117–135. Springer.

Saxe, J.; and Berlin, K. 2015. Deep neural network based malware detection using two dimensional binary program features. In *2015 10th international conference on malicious and unwanted software (MALWARE)*, 11–20. IEEE.

Schuba, C. L.; Krsul, I. V.; Kuhn, M. G.; Spafford, E. H.; Sundaram, A.; and Zamboni, D. 1997. Analysis of a denial of service attack on TCP. In *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No. 97CB36097)*, 208–223. IEEE.

Sharafaldin, I.; Gharib, A.; Lashkari, A. H.; and Ghorbani, A. A. 2018. Towards a reliable intrusion detection benchmark dataset. *Software Networking*, 2018(1): 177–200.

Sharafaldin, I.; Lashkari, A. H.; and Ghorbani, A. A. 2018. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1: 108–116.

Shen, Z.; Liu, J.; He, Y.; Zhang, X.; Xu, R.; Yu, H.; and Cui, P. 2021. Towards out-of-distribution generalization: A survey. *arXiv preprint arXiv:2108.13624*.

Shenfield, A.; Day, D.; and Ayesh, A. 2018. Intelligent intrusion detection systems using artificial neural networks. *Ict Express*, 4(2): 95–99.

Song, X.; Salcianu, A.; Song, Y.; Dopson, D.; and Zhou, D. 2020. Fast wordpiece tokenization. *arXiv preprint arXiv:2012.15524*.

Sorrenson, P.; Rother, C.; and Köthe, U. 2020. Disentanglement by nonlinear ica with general incompressible-flow networks (gin). *arXiv preprint arXiv:2001.04872*.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1): 1929–1958.

Taori, R.; Dave, A.; Shankar, V.; Carlini, N.; Recht, B.; and Schmidt, L. 2020. Measuring robustness to natural distribution shifts in image classification. *Advances in Neural Information Processing Systems*, 33: 18583–18599.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Vinayakumar, R.; Alazab, M.; Soman, K.; Poornachandran, P.; Al-Nemrat, A.; and Venkatraman, S. 2019. Deep learning approach for intelligent intrusion detection system. *Ieee Access*, 7: 41525–41550.

Wang, Z. 2015. The applications of deep learning on traffic identification. *BlackHat USA*, 24(11): 1–10.

Wankhede, S.; and Kshirsagar, D. 2018. DoS attack detection using machine learning and neural network. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, 1–5. IEEE.

Yang, J.; Zhou, K.; Li, Y.; and Liu, Z. 2021. Generalized out-of-distribution detection: A survey. *arXiv preprint arXiv:2110.11334*.

Yulianto, A.; Sukarno, P.; and Suwastika, N. A. 2019. Improving adaboost-based intrusion detection system (IDS) performance on CIC IDS 2017 dataset. In *Journal of Physics: Conference Series*, volume 1192, 012018. IOP Publishing.