Detection of Malicious DNS-over-HTTPS Traffic: An Anomaly Detection Approach using Autoencoders

Sergio A. Salinas Monroy, Aman Kumar Gupta, and Garrett Wahlstedt School of Computing Wichita State University

Abstract—To maintain the privacy of users' web browsing history, popular browsers encrypt their DNS traffic using the DNSover-HTTPS (DoH) protocol. Unfortunately, encrypting DNS packets prevents many existing intrusion detection systems from using plaintext domain names to detect malicious traffic. In this paper, we design an autoencoder that is capable of detecting malicious DNS traffic by only observing the encrypted DoH traffic. Compared to previous works, the proposed autoencoder looks for anomalies in DoH traffic, and thus can detect malicious traffic that has not been previously observed, i.e., zero-day attacks. We run extensive experiments to evaluate the performance of our proposed autoencoder and compare it to that of other anomaly detection algorithms, namely, local outlier factor, oneclass support vector machine, isolation forest, and variational autoencoders. We find that our proposed autoencoder achieves the highest detection performance, with a median F-1 score of 99% over several types of malicious traffic.

I. INTRODUCTION

The domain name system (DNS) is the Internet's core function that allows users to find web servers using domain names instead of IP addresses. Although DNS offers many advantages, it allows eavesdroppers to observe users' browsing history in plaintext. This is a critical issue because browsing history contains sensitive information that users would prefer to keep private [1].

To protect the privacy of users, most major browsers, including Mozilla's Firefox and Google's Chrome, [2], [3], have adopted DNS-over-HTTPS (DoH). DoH encrypts the DNS traffic by first establishing a TLS connection between the user's web browser and one of multiple available DoH public servers [4]. The browser then exchanges DNS packets encoded as HTTP requests with the DoH server. Since the encrypted packets are only accessible to the user and the DoH server, DoH prevents network administrators, ISPs, as well as malicious actors on the Internet from accessing the user's DNS traffic, thus protecting their privacy.

However, by encrypting the DNS packet payload, DoH interferes with the ability of existing intrusion detection techniques that rely on observing domain names [5]–[17]. For example, intrusion detection systems (IDS) implemented on devices from Palo Alto Networks, Cisco, and Trellix inspect the requested domains in plaintext to determine if they are likely to

This work has been submitted to the IEEE for possible publication. Copyright may be transferred without notice, after which this version may no longer be accessible

have been generated by a domain generation algorithm [18]–[20]. In addition, since DoH traffic is now part of a network's typical traffic, malicious DoH traffic generated by malware can appear normal, further complicating its detection. Thus, it is important to develop privacy-preserving intrusion detection systems that can detect malicious DoH traffic without the need to access plaintext information in the DNS packets.

There have been some efforts to classify malicious and normal DoH traffic using machine learning classification approaches [21]–[29]. However, since these works require both malicious and normal DoH traffic samples to be available during their training phase, they are unable to detect previously-unseen malicious DoH traffic, i.e., zero-day attacks. Furthermore, they only focus on detecting DNS tunneling traffic over DoH, ignoring other types of DNS abuse such as domain generation algorithms (DGA) that are used by bots to find their command and control servers and can be used as an earlier indication of compromised devices.

In this paper, we propose an autoencoder that can effectively detect zero-day malicious DoH traffic. The autoencoder is a type of deep learning model that can be trained to take normal DoH traffic as input and accurately recreate it as its output. When presented with malicious traffic, the autoencoder will recreate it with a large error. The difference in the autoencoder's recreation error allows us to identify malicious DoH traffic, even if it has not been observed during training. The autoencoder preserves the privacy of the users' browsing history since it only uses statistical information about the network flows and operates without accessing the plaintext DNS data in the DoH packets.

To evaluate the detection performance of the proposed autoencoder, we first collect realistic normal and malicious DoH traffic. Our normal traffic dataset is generated by automating a browser to visit a list of common websites while using different public DoH servers. The malicious traffic dataset contains both DGA and tunneling traffic. The DGA traffic was obtained by implementing several DGA algorithms and then automating their DoH requests. To simulate a DGA that attempts to avoid detection, we generated automated DoH requests with random waiting times between them. The DoH requests where sent both over independent and shared TLS connections. These datasets are available through our Kaggle repository [30]. We obtain DoH tunneling data from the datasets in MontazerShatoori et al. [21], which include traffic

generated by multiple tunneling protocols.

We train multiple autoencoder architectures with the normal DoH traffic dataset and identify the one that best recreates it. The best autoencoder architecture achieves up to a median F-1 score of 99% in identifying malicious DoH traffic. Compared to other anomaly detection algorithms (i.e., isolation forests, local outlier factor, one-class support vector machine, and variational autoencoders), the proposed autoencoder achieves a higher median F1-score, accuracy, area under the curve (AUC), precision and recall across all types of malicious and benign traffic.

We summarize our contributions as follows.

- We design a privacy-preserving anomaly detection autoencoder to detect malicious DoH traffic. To the best of our knowledge, this is the first work to investigate anomaly detection-based methods to identify malicious DoH traffic, and thus the first one that is designed to detect zero-day attacks.
- We create a new DoH traffic dataset that includes both benign and DGA-generated malicious traffic. The dataset is publicly available [30].
- We conduct extensive evaluations of our proposed autoencoder and compare its results to that of existing anomaly-detection methods that could potentially be used to detect malicious DoH traffic, including local outlier factor, isolation forest, one-class support vector machine, and variational autoencoders with various hyperparameter configurations.
- Our results show that our autoencoder can detect malicious DoH traffic with up to a median F1-score of 99%, which is higher than that of the other evaluated anomaly detection models.

The rest of the paper is organized as follows. Section II describes the differences between related works and this paper. Section III explains the differences between DNS and DoH. In Section IV, we describe the assumptions about the DGA and tunneling attacks considered in this paper. Section V presents our autoencoder architecture and anomaly detection approach. Section VI describes the data used to train and test the autoencoder. Section VII presents the results of our experiments. We conclude the paper in Section VIII.

II. RELATED WORKS

Detecting malicious DNS traffic has been studied in several works. However, there are only a few works that investigate detection of malicious DoH traffic, and, to the best of our knowledge, none that use anomaly detection autoencoders to address this problem. In this section, we provide an overview of existing works on malicious traffic detection methods, and highlight their differences with respect to this paper.

A. Detection of malicious DNS traffic

There is a rich literature on malicious DNS traffic detection using the domain names that appear in DNS packets. We present a few of the most relevant ones. Antonakakis et al. [7] design an approach that learns the textual characteristics

of the DGA-generated domain names obtained from non-existent domain DNS packets (NXDOMAIN). Schuppen et al. [31] develop FANCI, which is one of the best performing algorithms in the literature. FANCI uses the domain names in NXDOMAIN packets as input to both random forests and support vector machines. After training, FANCI can detect malicious traffic in real-time by analyzing DNS packets as they arrive to the network.

A central challenge in machine learning approaches for malicious DNS traffic detection is feature selection. To tackle this issue, deep learning, which can take as input the raw domain names, has been proposed as an efficient alternative. [11], [12], [14]–[16], [32], [33].

However, since the above works depend on accessing the domain names in the DNS packets, they cannot be used to detect malicious DoH traffic. The reason is that the domain names are part of the DNS packet payload that is encrypted by the DoH protocol.

B. Classification of Encrypted Traffic

Researchers have also studied classification of encrypted network traffic [34]–[37]. This research mainly focuses on classifying packet flows according to their application using deep learning approaches. However, since they only attempt to classify traffic according to its application without looking for malicious traffic, these works would simply classify malicious DoH traffic as an additional DoH packet flow. Nonetheless, the success of these works in classifying encrypted flows motivate the use of deep learning to detect malicious DoH traffic.

C. Detection of Malicious Traffic using an Autoencoder

In addition to classification of network traffic, deep learning has been successfully used to detect malicious traffic in various network environments. Meidan et al. [38] and Hwang et al. [39] train autoencoders that can recreate the normal network traffic of IoT devices. The autoencoders can detect when the device traffic deviates from the normal one. Dargenio et al. [40] use autoencoders to identify the most relevant features for malicious traffic detection. Kim et al. [41] find infected devices with variational autoencoders that analyze aggregate packet statistics in time windows. Zavrak and İskefiyeli [42] propose both an autoencoder and a variational autoencoder to detect various types of malicious network flows using plaintext traffic datasets.

Our work is the first one to design an autoencoder to detect not only malicious DoH traffic but any type of malicious DNS traffic. Specifically, the autoencoders proposed by the previous works [38]–[42] are trained using datasets that lack both benign and malicious DoH traffic. Thus existing autoencoders are not well-suited to detect malicious DoH traffic.

D. Classification of Malicious DoH Traffic

There a few works that propose methods to classify malicious DoH traffic. MontazeriShatoori et al. [21] evaluate both machine learning and deep learning algorithms to classify DoH tunneling data. They propose a two-step approach. In the

first step, their detector separates DoH traffic from the overall HTTPS traffic by analyzing a set of hand-crafted features from the HTTPS traffic. The second step classifies the DoH traffic into malicious and normal. Singh et al. [22] and Zhan et al. [24] employ several classification models, including naive Bayes, logistic regression, and random forest to detect DoH tunnels for data exfiltration. Kwan et al. [23] explore how to design DoH tunnels that are difficult to detect in a censored network. They propose a threshold detection mechanism over a set of hand-crafted features obtained from the overall HTTPS traffic in a network. Alenezu et al. [25] use recurrent neural networks to classify different types of malicious DoH traffic used for data exfiltration. Vekshin et al. [43] propose a machine learning approach to classify benign DoH traffic from other types of HTTPS traffic.

Several enhancements have been proposed to the above malicious DoH traffic classification approaches. Zebin et al. [26] propose an explainable random forest approach that finds the most important features for classification. Al-Haija et al. [27] design a two-stage approach with minimum number of features. Moure-Garrido et al. [28] use statistical analysis to identify the features that have the largest influence. Wang et al. [29] use convolutional neural networks to classify traffic using fused features as input.

Existing works on detecting malicious DoH traffic use supervised learning, i.e., they classify traffic into malicious and benign. This approach suffers from the following short-comings. First, the detection performance of this approach heavily depends on the ability of the defender to collect and maintain an up-to-date database of malicious traffic samples, which usually requires a large amount of resources to collect. Second, when the available malicious samples are out of date, supervised learning is unable to accurately classify traffic generated by new malware, i.e., zero-day attacks. The main reason is that supervised models assign malicious traffic from a previously unseen class to one of the traffic classes present in the training dataset, possibly the normal traffic class, which leads to an unpredictable and inaccurate classification [44]–[47].

To the best of our knowledge, this is the first work that can detect multiple types of malicious DoH traffic, even those that are unknown during the training phase.

III. BACKGROUND

In this section, we describe the DNS and DoH protocols, and highlight their differences.

A. DNS

DNS allows users to find the IP address of servers using human-readable domain names. To find the IP address for a given domain name, the DNS protocol first queries the local device's cache. If the requested IP address is missing from the local cache, it sends a query to the local DNS server. A local DNS server is usually provided by the local network or the ISP, but users can choose to use a public DNS server on the Internet. The DNS server then returns the IP address if it exists in its local cache. Otherwise, it recursively

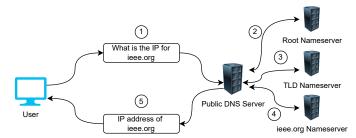


Fig. 1: The DNS protocol.

queries nameservers for each level in the domain. For example, consider www.ieee.org. The local DNS server will get the IP address of the nameserver in charge of the top-level domain .org from a known root nameserver. The .org nameserver will provide the IP address of the nameserver in charge of .ieee, and so on. Eventually, the name server controlling the sub-domain www.ieee.org returns the requested IP address to the local DNS server, which then sends the response to the original host. We show the DNS query process with a public DNS server in Fig. 1.

Since DNS uses UDP (or less commonly TCP) as its transport layer protocols, both queries and responses are sent in plaintext. This allows network administrators and ISPs to apply the various approaches described in Section II that use DNS packet contents for malicious traffic detection. Unfortunately, the ability to access plaintext DNS packets compromises the privacy of users because it reveals the websites that they visit as well as long term Internet usage patterns.

B. DNS-over-HTTPs

Aiming to protect user privacy, DoH uses HTTPS to encrypt DNS traffic [48]. In contrast to DNS, where the hosts send unencrypted packets to the local DNS server over UDP, DoH hosts first establish a TLS connection with a public DoH server on the Internet. The IP address of the DoH server can be obtained from a hard-coded file at the host, or by performing a traditional unencrypted DNS query. After the HTTPS connection is established with the DoH server, the host encodes its DNS request into either an HTTP GET or POST request, and transmits it over the TLS connection to the DoH server. After the DoH server resolves the query using the traditional DNS protocol, it replies to the host with an HTTP response message where the body contains the IP address of the domain, or any of the other DNS protocol messages. The HTTP reply from the DoH server is also sent over the TLS connection.

Although DoH protects the privacy of the users, it prevents intrusion detection systems from accessing plaintext domain names and other DNS packet data that they need to operate.

IV. THREAT MODEL

In this section, we first describe in details our assumptions about how malware uses DoH, and then formulate the problem of detecting malicious DoH traffic.

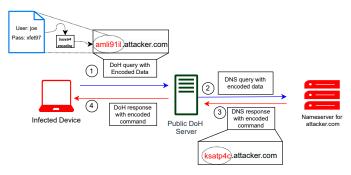


Fig. 2: Data exfiltration with DoH tunneling.

A. Finding Command and Control Servers with Domain Generation Algorithms

Malware usually communicates with a command and control (C2) server to send exfiltrated data and receive commands. But before it can do so, it needs to find out the C2 server's IP address. Unlike legitimate servers that can be easily found by making a conventional DNS query, finding a C2 requires additional steps due to the precautions that attackers take to avoid being blocked by ISPs or found by law enforcement.

Specifically, to evade domain name denylists, attackers register their C2 under a domain name generated by a domain generation algorithm (DGA). The DGA pseudo-randomly generates a list of potential domain names for the C2. The attacker registers the C2 under one of these domain names. Then, to find the C2, infected devices run the DGA to generate the list of potential domain names. Since the infected devices are unaware of the domain name registered by the attacker, they iteratively send DNS queries for each potential domain name. Eventually, the infected device will query for the registered domain name and obtain its IP address. If the ISP adds the C2's registered domain name to the denylist, the attacker can simply register the C2 under different domain name from the list. The infected devices can follow the same procedure to find the C2 under the new domain name. We assume infected devices will use a DGA to find their C2 servers.

We further assume that the infected devices will attempt to avoid intrusion detection systems that rely on plaintext DNS queries. Specifically, by employing DoH instead of DNS to find their C2 server, infected devices can hide their requested domains from the intrusion detection system. This is a reasonable assumption since security researchers have recently observed many malware samples using DoH to avoid detection, for example [49]–[52].

B. Data Exfiltration with DNS

Besides finding the IP address of their command and control server, malware often uses the DNS protocol to exfiltrate data in a stealthy manner. The main idea is to use the second-level domain of a subdomain owned by the attacker to encode the exfiltrated data. When the infected devices send a DNS query for this domain name, the DNS query will eventually be forwarded to the subdomain's nameserver, which is controlled by the attacker. Although the second-level domains will not exist, the adversary's nameserver can

decode the data in the second-level domain and reply to the infected device with a DNS packet. For example, suppose the subdomain attacker.com is owned by the adversary and that the data that an infected device wants to exfiltrate is user:joe and password: xfet97. Then, the infected device would encode the data and make a DNS request for the second-level domain, e.g., amli91il.attacker.org, where amli91il is the encoded data. The local DNS server eventually sends the query to the attacker-controlled nameserver for attacker.com. The nameserver decodes the subdomain amli91il to obtain the exfiltrated data. We show an example of this process in Fig. 2. We assume the infected devices will exfiltrate data using DoH instead of DNS. This is a reasonable assumption due to recently observed malware that uses DoH for stealthy data exfiltration, e.g., Qakbot [53].

C. Problem Formulation

In this work, we consider the problem of detecting malicious DoH traffic generated by infected devices in a local network. Specifically, we define a local network as the set of devices that are owned by a single person (e.g., home LAN), or organization (e.g., enterprise network). A network manager sets the list of trusted DoH servers for the network devices, and obtains the statistical features of network flows by analyzing the packets traversing the gateway devices. Furthermore, we assume there are only a few approved DoH servers in the network. This is justified by the fact that commonly used browsers and operating systems only provide one to three approved DoH servers by default [54], [55].

Since legitimate devices only use the approved DoH servers, it is easy for the network administrator to differentiate between DoH traffic from other HTTPS traffic, i.e., web browsing, based on the destination IP address of outgoing packets. Hence, we focus on designing a detection approach that uses the encrypted DoH packets as observed by a local network's gateway device.

Moreover, we assume the malware uses one of the approved public DoH servers to find its C2 server and to exfiltrate data. Consequently, the infected device cannot be easily detected by inspecting the destination IP address or port number of its outgoing packets since legitimate users will be using the same server and port number. Note that malware that uses a non-approved DoH server looses the advantage of communicating with an approved IP address by the network administrator, and thus its packets can be easily detected and blocked based on the destination IP address.

V. AN AUTOENCODER ARCHITECTURE FOR DETECTION OF MALICIOUS DOH TRAFFIC

In this section, we explain our malicious DoH traffic detection approach and the architecture of the autoencoder in details.

A. Overall Approach

The main objective of the autoencoder is to recreate statistical features from observed legitimate DoH traffic with high

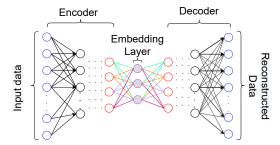


Fig. 3: An autoencoder architecture.

accuracy. If the input to the autoencoder is a legitimate DoH traffic flow, the reconstruction error will be small. Otherwise, the reconstruction error will be large. Thus, we use the magnitude of the reconstruction errors to determine if the observed DoH traffic is malicious.

Moreover, since we assume that there are only a few DoH servers approved by the local networks, it is feasible to train an autoencoder for each allowed server. Therefore, to detect malicious traffic, we filter the DoH traffic by DoH server and then use the corresponding autoencoder to evaluate it.

B. Autoencoder Architecture

Our autoencoder is formed by two main parts: 1) an encoder neural network that takes as input the feature vector and outputs a low-dimensional vector, called the embedding; and 2) a decoder neural network that takes the embedding as input and outputs a reconstruction of the original feature vector. We illustrate the architecture of the autoencoder in Fig. 3.

Formally, let $\mathbf{x} \in \mathbb{R}^n$ be the original statistical feature vector obtained from the DoH traffic flow, $\mathbf{e} \in \mathbb{R}^p$ be the embedding, and $\hat{\mathbf{x}} \in \mathbb{R}^n$ be the reconstructed statistical feature vector. Then, the encoder function is given by $E(\mathbf{x}) = \mathbf{e}$ and the decoder function by $D(\mathbf{e}) = \hat{\mathbf{x}}$. The complete autoencoder network function $A : \mathbb{R}^n \to \mathbb{R}^n$ is defined as $A(\mathbf{x}) = D(E(\mathbf{x}))$.

To train the autoencoder for malicious DoH traffic detection, we train autoencoder A as a single neural network that aims to output a vector $\hat{\mathbf{x}}$ that reconstructs the input \mathbf{x} . Consequently, the encoder network $E(\mathbf{x})$ learns to create an embedding with dimension p << n that contains enough information for the decoder $D(\mathbf{e})$ to output an accurate reconstruction $\hat{\mathbf{x}}$.

C. The layers of the autoencoder

The layers of the encoder E and decoder D networks are composed by a series of functions called neurons. Neurons take as input the output from the previous layer and their output is used as input by the next one. Neurons process their inputs using an activation function parametrized by weights and biases.

Specifically, let $\mathbf{h}_E^k \in \mathbb{R}^{I_E^k}$ be the output of the kth layer of encoder E, where I_E^k is the number of neurons. Let $\mathbf{W}_E^k \in \mathbb{R}^{I_E^k \times I_E^{k-1}}$ and $\mathbf{b}_E^k \in \mathbb{R}^{I_E^k}$ be matrices denoting the weights and biases of the kth layer, respectively. The total number of

layers is denoted by K. Then, the output of the kth layer is given by:

$$\mathbf{h}_E^k = f(\mathbf{W}_E^k \cdot \mathbf{h}_E^{k-1} + \mathbf{b}_E^k), \ \forall k = 1 \dots, K,$$

where f is the activation function. The input layer is given by the vector of statistical features for a packet flow, i.e., $\mathbf{h}_E^0 = \mathbf{x}$.

The size of the layers in encoder E decreases monotonically from the size of the input layer n to the size of the embedding layer p. Thus, we have that $I_E^{k+1} < I_E^k$ for $0 \le k \le K-1$.

Decoder D is similarly defined. The main difference is that D has one fewer layer than E (i.e., no embedding layer), and has layers with increasing sizes. In particular, decoder D's input layer size I_D^0 is given by the size of the embedding layer p, while its output layer I_D^{K-1} is equal to n. The hidden layers of D have monotonically increasing sizes, i.e., $I_D^{k+1} > I_D^k$ for $0 \ge k \ge K - 2$.

The exact size of each of the layers and the total number of layers of the encoder E and decoder D are design parameters that we explore in Section VII.

D. Malicious DoH Traffic Detection with an Autoencoder

We use the autoencoder's reconstruction error to determine if the observed DoH traffic correspond is normal or malicious. Since the autoencoder is trained with normal DoH traffic, we expect the reconstruction error for malicious traffic samples to be larger than the one obtained for normal traffic samples.

The reconstruction error is calculated as the mean squared error between the input and output features, i.e.,

$$MSE = \frac{1}{n}(x_i - \hat{x})^2 \tag{2}$$

where x_i and \hat{x}_i are the *i*th element of \mathbf{x} and $\hat{\mathbf{x}}$, respectively. If a sample has an MSE larger than a threshold t, we determine it is malicious. Otherwise, we determine it is normal.

The threshold t can be calculated in two ways depending on the availability of malicious samples. If there are no available malicious samples during training, then t can be calculated based on the distribution of MSEs for normal samples, i.e.,

$$t = \mu + s * \sigma \tag{3}$$

where μ is the mean of the MSEs for normal samples obtained during training, s is a positive integer, and σ is the standard deviation of the MSEs.

If malicious samples are available, then we can test several values for the threshold t and choose the value that offers the best trade-off between the false positives rate and the true positive rate. This can be achieved by plotting the receiver operating curve (ROC) and calculating the area under the curve (AUC). Alternatively, the threshold t can be chosen as the value that offers the best trade-off between precision and recall. In our experiments, we use the ROC curve to fully explore the detection capabilities of the autoencoder.

VI. DATASETS

Our dataset includes various types of normal and malicious DoH traffic. We use this dataset in Section VII to train and evaluate the autoencoder. The rest of this subsection describes the dataset in details.

A. Normal DoH Traffic Data

We collect normal DoH traffic by visiting popular websites using an automated browser. Specifically, we program a browser to visit a list of the 1000 most popular websites [56] using Python's Selenium library. The browser visited each website multiple times. Each time using a different public DoH server from either AdGuard, Google, Quad9, or Cloudflare. The browser ran inside a virtual machine and all of its network traffic was collected using Wireshark. Using the known IP address of the public DoH servers, we filtered the DoH traffic from the rest of the browser's traffic. This dataset is publicly available [30].

We complement our normal DoH traffic data set with traffic collected by Vekshin et al. [43] using a DoH proxy server. Their proxy server translated DNS requests from users in a small office network into DoH requests. They used the public DoH server from Cloudflare.

We use Python's NFStream library [57] to group the raw packets into flows, where each flow corresponds to a single TCP connection. Then, for each flow, we calculate the average, median, variance, minimum, and maximum of the number of incoming and outgoing packets, packet sizes in bytes, connection duration, and inter-packet delays. In total, we calculate 16 features for each flow. All features are scaled between zero and one. We later use these features as the input x for the autoencoder.

B. Malicious DoH Traffic Data

Our malicious DoH traffic data set is formed by encoding DNS queries from DGAs and tunneling tools into DoH packets. Specifically, we run 20 real-world DGAs that have been reversed engineered [58]. For each domain generated by a DGA, we form a DNS query, encode it into a DoH query, and then send it to a public DoH server. We stop generating DNS queries at a randomly chosen domain to simulate an infected device finding the correct domain for its C2 server. We use Cloudflare's public DoH server to generate the DGA traffic. This data set is available publicly available [30].

To simulate malware that attempts to shape its DoH traffic to avoid detection, we generate malicious DoH requests with different combinations of transmission frequency and number of TLS connections. In particular, we generate DoH requests for each DGA with the following characteristics:

- *Multiple Connections (MC)*: For each DoH query, we establish a new TLS connection to the DoH server, resolve the query, and then close the connection.
- Single Connection (SC): We establish a single TLS connection for all DoH queries generated by a DGA. The queries are sent immediately after receiving the reply to the previous query.
- Single Connection with Random Wait (SC-RW): Operates in the same way as SC, except that it waits a random amount of time between 0 and 2 seconds after receiving the reply to the previous query before sending the next one.

We also consider malware that uses DoH for data exfiltration. In particular, we incorporate the malicious DoH traffic samples from MontazeriShatoori et al. [21]. Their CIRA-CIC-DoHBrw-2020 data set provides DoH traffic generated by various DNS tunneling tools, namely iodine, dns2tcp, and dnscat2. These tools establish a TCP connection with an attacker-controlled nameserver by employing the subdomain part of the requested domain as their communications channel.

Similarly to normal DoH traffic, we use NFStream to group malicious DoH packets into flows and calculate their features.

C. Training and Evaluation Datasets

We build datasets for each DoH server in our normal traffic dataset. To ensure fair comparisons in Section VII, the size of the datasets is set to 2,350 flows to match the server with fewer flows, which in this case is the DoH proxy server [43]. These datasets were divided into k=5 folds for k-fold cross-validation. Each fold has 1880 training flows and 470 flows reserved for testing the reconstruction accuracy of the autoencoder.

We also form an evaluation dataset for each combination of DoH server and type of malicious DoH traffic. We later use these datasets to evaluate the detection performance of the autoencoders. The datasets are formed by combining the flows reserved for testing in each of the training folds described above with randomly selected samples from the malicious datasets. Each testing fold has 30% malicious flows. In total, we form 30 data sets, one for each pair of DoH server and malicious traffic type. We summarize our testing datasets in Table I.

VII. EXPERIMENTAL EVALUATION

In this section, we first describe our autoencoder implementation and training parameters. We then evaluate its performance under varying hyperparameter combinations, i.e., number of layers and number of neurons per layer. We also compare the autoencoder's performance to that of local outlier factor (LOF), one-class support vector machine (SVM), isolation forests (IF), and variational autoencoders.

A. Autoencoder Implementation and Training Parameters

We use the TensorFlow library [59] to implement the autoencoder described in Section V, as well as variational autoencoders for comparison. We use the Scikit-learn [60] library to implement the LOF, SVM, and IF models with default settings. The autoencoder training and inference are carried out on a High Performance Computing cluster using one Nvidia V100 Tesla GPU along, 8 CPU cores, and 16 GB of RAM.

We train our proposed autoencoders with different hyperparameter combinations for each training dataset described in Section VI. The number of hidden layers was varied from 2 to 6. The size of the embedding layer was varied from 3 to 9 neurons. The number of neurons of the encoder's hidden layers were varied from 14 to 110. The input layer was kept constant at 16 neurons to match the number of input features. At the encoder, the pattern was reversed.

Malicious Traffic Types DGA SC DGA SC-RW DGA MC dns2tcp dnscat2 iodine AdGuard (AG) AG-DGA SC AG-DGA SC-RW AG-DGA MC AG-dns2tcp AG-dnscat2 AG-iodine Google (G) G-DGA SC G-DGA SC-RW G-DGA MC G-dns2tcp G-dnscat2 G-iodine Q9-DGA MC Quad9 (Q9) Q9-DGA SC Q9-DGA SC-RW O9-dns2tcp O9-dnscat2 O9-iodine Cloudflare (C) C-DGA SC C-DGA SC-RW C-DGA MC C-dns2tcr C-iodine C-dnscat2 Proxy (P) P-DGA SC P-DGA SC-RW P-DGA MC P-dns2tcp P-dnscat2 P-iodine

TABLE I: The datasets used for evaluating the autoencoder performance.

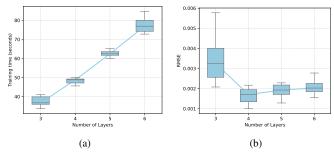


Fig. 4: Training results: (a) Autoencoder training time for increasing number of hidden layers; (b) Median RMSE across all architectures and training datasets.

We use the ADAM algorithm to train the autoencoder and use MSE as the loss function. Since the goal of the autoencoder training is to minimize the difference between its input and output, we set the sample labels \mathbf{x}' equal to the sample itself, i.e., $\mathbf{x} = \mathbf{x}'$. The batch size is set to 32 and the learning rate is kept at the default value of 0.001. The number of epochs is set to 30. To improve the numerical stability of the training algorithm and improve reconstruction accuracy, we add batch normalization layers to the autoencoder. All datasets are scaled to the range [0,1].

B. Autoencoder Training Results

DoH Servers

Fig. 4a shows the median training time of the autoencoder for an increasing number of layers across all training datasets. We observe that the training time increases with the number of layers. The training times range from 35s to 76s, which is low since the autoencoder would only need to be infrequently updated with new normal DoH traffic.

In Fig. 4b, we show the mean reconstruction median squared error (RMSE) for an increasing number of layers across all training datasets. We see that the median RMSE remains between 0.2300 and 0.2305, while the standard deviation is larger for models with 6 layers. This RMSE values are low, which shows that all the models are able to accurately reconstruct the normal DoH traffic from the multiple DoH servers.

C. Performance Comparison between Autoencoder Architectures

After training the autoencoders, we evaluate their ability to detect malicious DoH traffic in terms of their accuracy, area under the curve (AUC), accuracy, recall, precision, and F-1 score. We evaluate the median performance of the different hyperparameter combinations across the 30 normal and malicious DoH traffic datasets described in Table I. We summarize the evaluation results for the top-ten autoencoder models and rank them according to their F1-score in Table II. The **Architecture** column shows the number of layers in the encoder. The layers at the decoder have the same number of neurons but in reverse order

We see that all the top-ten autoencoder models achieve F1-scores, accuracy, AUC, precision and recall greater than 0.9800, which is high. The architecture [16,62,9] achieves a median F1-score of 0.99359 with a median training time of 38.8916 seconds and a median RMSE of 0.0021 during testing. Although it has a slightly higher F1-score variance than the next best performing architecture, its simpler architecture (i.e., only one hidden), and lower training time make it a more attractive choice. For this reason, we use the architecture [16,62,9] to compare our proposed autoencoder to existing anomaly detection approaches in the next section.

D. Performance Comparison with other Anomaly Detection Models

We compare the performance of our proposed autoencoder to other models that have successfully been used to perform anomaly detection in network traffic problems [42], [61]. We provide a brief description of these models next.

- 1) Variational Autoencoders (VAE).: VAEs are autoencoders with a probabilistic embedding layer. Instead of finding an embedding layer with deterministic outputs, it instead finds an embedding layer that defines the latent probability distribution. The decoder network of a VAE takes as input samples from the latent probability distribution and outputs the probability that the input was generated by the latent probability defined by the embedding layer. If the probability is low, the VAE declares the input as an anomaly. Anomaly detection VAEs are proposed by An and Cho [62]. We implemented VAEs with varying number of layers and neurons per layer. The results in this section correspond to the architecture [16, 50, 26, 3], which is the best performing one.
- 2) Isolation Forest (IF).: IF is an unsupervised method that builds isolation trees where leaves represent data points. IF assumes that there are significantly fewer anomalous samples in the data set and that these samples have drastically different features from the normal samples. Hence, anomalous samples will have a much shortest path to the root of the tree compared to the normal ones [63].

Architecture	F1-score	Accuracy	AUC	Precision	Recall
16, 62, 9	0.99359 ± 0.03494	0.99018 ± 0.04056	0.99564 ± 0.04657	0.99826 ± 0.00431	0.98825 ± 0.05210
16, 26, 17, 9	0.99359 ± 0.01912	0.99018 ± 0.02718	0.99491 ± 0.02731	0.99830 ± 0.00287	0.98936 ± 0.03484
16, 98, 9	0.99343 ± 0.04314	0.98990 ± 0.05014	0.99571 ± 0.05929	0.99787 ± 0.01436	0.98849 ± 0.06198
16, 86, 9	0.99325 ± 0.02001	0.98966 ± 0.02777	0.99542 ± 0.02812	0.99829 ± 0.00805	0.98846 ± 0.03410
16, 62, 35, 9	0.99287 ± 0.01568	0.98909 ± 0.02279	0.99506 ± 0.02294	0.99827 ± 0.00147	0.98738 ± 0.03002
16, 38, 23, 9	0.99283 ± 0.01647	0.98902 ± 0.02380	0.99499 ± 0.02381	0.99806 ± 0.00296	0.98797 ± 0.03101
16, 26, 20, 14, 9	0.99282 ± 0.02329	0.98901 ± 0.03291	0.99555 ± 0.03117	0.99828 ± 0.00485	0.98740 ± 0.04255
16, 74, 38, 3	0.99282 ± 0.01353	0.98902 ± 0.01962	0.99414 ± 0.01799	0.99856 ± 0.00211	0.98700 ± 0.02589
16, 50, 29, 9	0.99266 ± 0.02422	0.98878 ± 0.03343	0.99477 ± 0.03248	0.99809 ± 0.00203	0.98706 ± 0.04382
16, 110, 56, 3	0.99264 ± 0.01655	0.98874 ± 0.02376	0.99384 ± 0.02121	0.99831 ± 0.00156	0.98756 ± 0.03137

TABLE II: Metric scores ($median \pm std$) for the top-ten autoencoder models ordered by F-1 score.

- 3) One-class Support Vector Machine (OC-SVM).: The One-class SVM is a semi-supervised method that aims to find a decision boundary on a hyperplane with maximum separation between the normal samples and the origin when only normal data is available, which is the case in our problem [64]. OC-SVM can be considered as a type of supervised learning that classifies normal network traffic as normal, and classifies all other types of traffic as anomalous.
- 4) Local Outlier Factor (LOF).: LOF is an unsupervised method that calculates an isolation factor for each sample in the data set. The isolation factor is calculated as the distance of the sample to the center of its nearest cluster. By calculating the factor relative to a cluster, the LOF can identify anomalous samples even when there are multiple clusters of normal data. [65].

Fig. 5 shows the performance metrics for our autoencoder model as well as for IF, One-class SVM, LOF, and VAE across all normal and malicious DoH traffic datasets. Specifically, the bar heights are the median values of the performance metrics across all datasets. The figure also shows the standard deviation values as error bars. We see that our proposed autoencoder achieves the highest median F1-score, accuracy, AUC, and recall. The standard deviation of these metrics is also smallest for the autoencoder. The median precision is comparable for all the models, with the standard deviation being the smallest for the autoencoder.

Next, we investigate the impact of training the autoencoder with normal traffic from different DoH servers. Fig. 6 compares the median detection performance of the models across multiple malicious DoH traffic datasets for a given normal DoH server traffic dataset. Overall, we see that our proposed autoencoder is the least affected model by the choice of training dataset. The autoencoder performs better than the other models for all datasets except for the Google DoH server dataset, where it performs similarly to LOF. In contrast, we see that the other models suffer from a significantly lower detection performance under at least one training dataset compared to the rest. That is, One-class SVM performs poorly under the AdGuard dataset, LOF under the Cloudflare and Proxy datasets, IF under the Proxy dataset, and VAE under the Proxy dataset. These results show that the autoencoder is

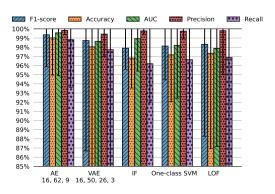


Fig. 5: Performance comparison between an autoencoder (AE) and other anomaly detection models. The bar heights are the median values and the error bars are the standard deviations across all benign and malicious DoH traffic datasets.

less sensitive to the choice of training data sets, and thus can be deployed more widely.

We now investigate the performance of the models in detecting the different types of malicious traffic. Fig. 7 shows the median performance metrics for a given malicious traffic type across the normal DoH traffic datasets. We see that the autoencoder achieves a performance equal or higher than the other detection models for all malicious DoH traffic types, except for DGA multiple connections where LOF has a better performance. The autoencoder achieves its highest performance when detecting DGA SC traffic, and its lowest one under DGA MC. The reason is that DGA SC traffic contains a high number of requests from each DGA which provides enough information to determine it is malicious. On the other hand, flows in the DGA MC traffic only contain information about a single DoH query, and thus are harder to detect.

Finally, we take a deeper look into the autoencoder's performance by considering one evaluation dataset at a time. Fig. 8a shows a heatmap of the F-1 scores achieved by the autoencoder when trained with the normal DoH traffic dataset of the corresponding row and tested using normal samples and the malicious traffic indicated in the corresponding column.

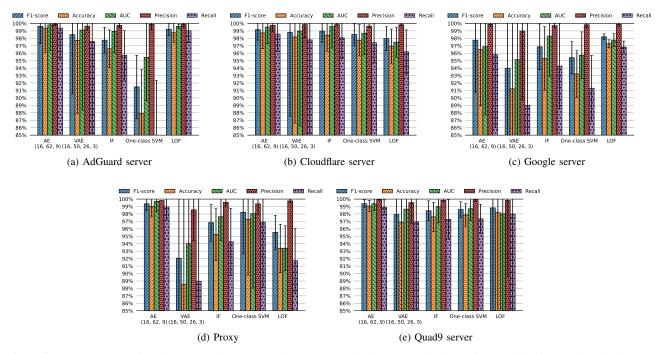


Fig. 6: Performance comparison between the autoencoder (AE) and other anomaly detection models for a given benign DoH traffic dataset. The bar heights are the median values and the error bars are the standard deviations across all malicious DoH traffic datasets.

Darker colors indicate a higher F-1 score, and thus a better overall detection performance for the traffic combination. Fig. 8b, Fig. 8c, and Fig. 8d show the F1-scores for LOF, IF, and One-class SVM, respectively.

Overall, the heatmaps confirm our observation that the autoencoder generally outperforms the other models, except for the DGA MC. Although the median F1-score of the autoencoder under the DGA MC dataset is lower than the score for LOF, we see from Figs. 8a and 8b that the performance of the models is very close, except for the Proxy server dataset, where the autoencoder performs worse. When the autoencoder trained with Google DoH server traffic attempts to detect dns2tcp traffic, it also achieves a relatively low F1-score. The VAE performs worse than all the other anomaly detection algorithms, thus it is omitted for brevity.

Note that to maintain a fair comparison, we leave out supervised learning approaches, e.g., traditional SVM, classification neural networks, random forest, etc., from our evaluation. The reason is that supervised learning approaches assume that malicious traffic observed during testing is generated by a class of malicious traffic available in the training dataset, which is not the case in our considered scenario. Moreover, we know from the literature [44]–[47], that supervised learning performs poorly when classifying traffic from previously unseen classes.

VIII. CONCLUSIONS

The DNS protocol compromises users' privacy by revealing their web browsing history to devices that handle their packets. To address this issue, popular web browsers have adopted DNS-over-HTTPS (DoH), an encrypted version of the DNS protocol that protect users' privacy. Unfortunately, the adoption of DoH prevents existing intrusion detection approaches from using the plaintext domain names in DNS query packets to detect malicious traffic. In this paper, we have designed an autoencoder that detects malicious traffic by only observing the encrypted DoH traffic. Since the autoencoder was trained to recognize normal DoH traffic, it can recognize previously unseen malicious traffic. We run extensive experiments with real-world and realistic simulated data to evaluate the performance of the autoencoder, and compare it to other anomaly detection algorithms, i.e., local outlier factor (LOF), oneclass support vector machine (SVM), isolation forests (IF), and variational autoencoders. We find that the autoencoder achieves the highest malicious DoH traffic detection performance, with a median F-1 score of 0.99359 across all training and evaluation datasets. Future work includes using the output of the embedding layer as a low dimensional representation of the DoH traffic to find anomaly detection models with lower computing requirements for both training and inference.

REFERENCES

- [1] D. J. Leith, "Web browser privacy: What do browsers say when they phone home?" *IEEE Access*, vol. 9, pp. 41615–41627, 2021.
- [2] Mozilla, "Firefox dns-over-https," https://support.mozilla.org/en-US/kb/firefox-dns-over-https, 2022.
- "Dns-over-https setting," https://support.google.com/chrome/a/ thread/10152459/dns-over-https-setting?hl=en, 2019.
- [4] Adguard, "Known dns provides," https://kb.adguard.com/en/general/dns-providers, 2022.
- [5] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "Exposure: Finding malicious domains using passive dns analysis." in Ndss, 2011, pp. 1–17.

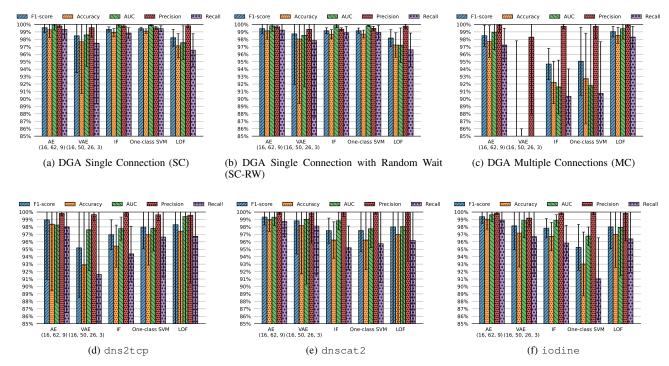


Fig. 7: Performance comparison between an autoencoder (AE) and other anomaly detection models. The bar heights are the median values and the error bars are the standard deviations across all training and testing files.

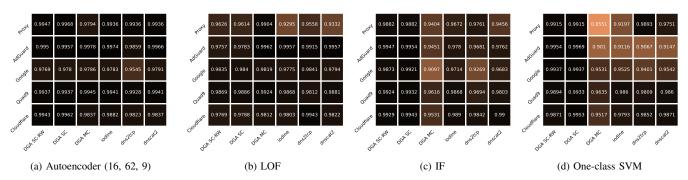


Fig. 8: F1-scores achieved when detecting malicious DoH traffic under different normal and malicious data sets.

- [6] Y. Shi, G. Chen, and J. Li, "Malicious domain name detection based on extreme machine learning," *Neural Processing Letters*, vol. 48, no. 3, pp. 1347–1357, 2018.
- [7] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon, "From throw-away traffic to bots: detecting the rise of dga-based malware," in 21st {USENIX} Security Symposium ({USENIX} Security 12), 2012, pp. 491–506.
- [8] T. Chin, K. Xiong, C. Hu, and Y. Li, "A machine learning framework for studying domain generation algorithm (dga)-based malware," in *International Conference on Security and Privacy in Communication* Systems. Springer, 2018, pp. 433–448.
- [9] J. Ahmed, H. H. Gharakheili, Q. Raza, C. Russell, and V. Sivaraman, "Monitoring enterprise dns queries for detecting data exfiltration from internal hosts," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 265–279, 2019.
- [10] M. Zago, M. G. Pérez, and G. M. Pérez, "Scalable detection of botnets based on dga," *Soft Computing*, vol. 24, no. 8, pp. 5517–5537, 2020.
- [11] J. Woodbridge, H. S. Anderson, A. Ahuja, and D. Grant, "Predicting domain generation algorithms with long short-term memory networks," arXiv preprint arXiv:1611.00791, 2016.
- [12] H. Mac, D. Tran, V. Tong, L. G. Nguyen, and H. A. Tran, "Dga botnet detection using supervised learning methods," in *Proceedings of the*

- Eighth International Symposium on Information and Communication Technology, 2017, pp. 211–218.
- [13] J. Spooren, D. Preuveneers, L. Desmet, P. Janssen, and W. Joosen, "Detection of algorithmically generated domain names used by botnets: a dual arms race," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, 2019, pp. 1916–1923.
- [14] D. Tran, H. Mac, V. Tong, H. A. Tran, and L. G. Nguyen, "An 1stm based framework for handling multiclass imbalance in dga botnet detection," *Neurocomputing*, vol. 275, pp. 2401–2413, 2018.
- [15] A. Drichel, U. Meyer, S. Schüppen, and D. Teubert, "Analyzing the real-world applicability of dga classifiers," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, 2020, pp. 1–11.
- [16] L. Sidi, Y. Mirsky, A. Nadler, Y. Elovici, and A. Shabtai, "Helix: Dga domain embeddings for tracking and exploring botnets," in *Proceedings* of the 29th ACM International Conference on Information & Knowledge Management, 2020, pp. 2741–2748.
- [17] Z. Liu, X. Yun, Y. Zhang, and Y. Wang, "Ccga: clustering and capturing group activities for dga-based botnets detection," in 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE). IEEE, 2019, pp. 136–

143.

- [18] Palo Alto Networks, "Domain generation algorithm (dga) detection," 2022. [Online]. Available: https://docs.paloaltonetworks.com/pan\-os/9-1/pan-os-admin/threat-prevention/\dns-security/domain-generation-algorithm-detection
- [19] CISCO, "Detecting algorithmically generated domains," 2015. [Online]. Available: https://blogs.cisco.com/security/talos/detecting-\dga
- [20] Trellix, "How network security platform protects against dga botnet?" 2020. [Online]. Available: https://docs.trellix.com/bundle/network-security-\platform-9.2.x-product-guide/page/\GUID-8D7E2D31-01B7-4724-A93D-5C660F138FF0.html
- [21] M. MontazeriShatoori, L. Davidson, G. Kaur, and A. H. Lashkari, "Detection of doh tunnels using time-series classification of encrypted traffic," in 2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech). IEEE, 2020, pp. 63–70.
- [22] S. K. Singh and P. K. Roy, "Detecting malicious dns over https traffic using machine learning," in 2020 International Conference on Innovation and Intelligence for Informatics, Computing and Technologies (3ICT). IEEE, 2020, pp. 1–6.
- [23] C. Kwan, P. Janiszewski, S. Qiu, C. Wang, and C. Bocovich, "Exploring simple detection techniques for dns-over-https tunnels," in *Proceedings* of the ACM SIGCOMM 2021 Workshop on Free and Open Communications on the Internet, 2021, pp. 37–42.
- [24] M. Zhan, Y. Li, G. Yu, B. Li, and W. Wang, "Detecting dns over https based data exfiltration," *Computer Networks*, vol. 209, p. 108919, 2022.
- [25] R. Alenezi and S. A. Ludwig, "Classifying dns tunneling tools for malicious doh traffic," in 2021 IEEE Symposium Series on Computational Intelligence (SSCI). IEEE, 2021, pp. 1–9.
- [26] T. Zebin, S. Rezvy, and Y. Luo, "An explainable ai-based intrusion detection system for dns over https (doh) attacks," *IEEE Transactions* on Information Forensics and Security, vol. 17, pp. 2339–2349, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID: 245783609
- [27] Q. A. Al-Haija, M. Alohaly, and A. J. Odeh, "A lightweight double-stage scheme to identify malicious dns over https traffic using a hybrid learning approach," *Sensors (Basel, Switzerland)*, vol. 23, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID: 257799594
- [28] M. Moure-Garrido, C. Campo, and C. García-Rubio, "Detecting malicious use of doh tunnels using statistical traffic analysis," Proceedings of the 19th ACM International Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID: 252995801
- [29] Y. Wang, C.-H. Shen, D. Hou, X. Xiong, and Y. Li, "Ff-mr: A doh-encrypted dns covert channel detection method based on feature fusion," *Applied Sciences*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:254625371
- [30] S. Salinas Monroy, "Dns-over-https traffic data," 2023.
- [31] S. Schüppen, D. Teubert, P. Herrmann, and U. Meyer, "{FANCI}: Feature-based automated nxdomain classification and intelligence," in 27th {USENIX} Security Symposium ({USENIX} Security 18), 2018, pp. 1165–1181.
- [32] Y. Li, K. Xiong, T. Chin, and C. Hu, "A machine learning framework for domain generation algorithm-based malware detection," *IEEE Access*, vol. 7, pp. 32765–32782, 2019.
- [33] R. Vinayakumar, M. Alazab, S. Srinivasan, Q.-V. Pham, S. K. Padannayil, and K. Simran, "A visualized botnet detection system based deep learning for the internet of things networks of smart cities," *IEEE Transactions on Industry Applications*, vol. 56, no. 4, pp. 4436–4456, 2020.
- [34] J. Yang, J. Narantuya, and H. Lim, "Bayesian neural network based encrypted traffic classification using initial handshake packets," in 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks–Supplemental Volume (DSN-S). IEEE, 2019, pp. 19–20.
- [35] R. Houser, Z. Li, C. Cotton, and H. Wang, "An investigation on information leakage of dns over tls," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, 2019, pp. 123–137.
- [36] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 445–458, 2019.

- [37] Z. Zhang, C. Kang, G. Xiong, and Z. Li, "Deep forest with Irrs feature for fine-grained website fingerprinting with encrypted ssl/tls," in Proceedings of the 28th ACM International Conference on Information and Knowledge Management, 2019, pp. 851–860.
- [38] 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*, vol. 17, no. 3, pp. 12–22, 2018.
- [39] R.-H. Hwang, M.-C. Peng, and C.-W. Huang, "Detecting iot malicious traffic based on autoencoder and convolutional neural network," in 2019 IEEE Globecom Workshops (GC Wkshps). IEEE, 2019, pp. 1–6.
- [40] R. Dargenio, S. Srikant, E. Hemberg, and U.-M. O'Reilly, "Exploring the use of autoencoders for botnets traffic representation," in 2018 IEEE Security and Privacy Workshops (SPW). IEEE, 2018, pp. 57–62.
- [41] J. Kim, A. Sim, J. Kim, and K. Wu, "Botnet detection using recurrent variational autoencoder," in GLOBECOM 2020-2020 IEEE Global Communications Conference. IEEE, 2020, pp. 1–6.
- [42] S. Zavrak and M. İskefiyeli, "Anomaly-based intrusion detection from network flow features using variational autoencoder," *IEEE Access*, vol. 8, pp. 108 346–108 358, 2020.
- [43] D. Vekshin, K. Hynek, and T. Cejka, "Doh insight: Detecting dns over https by machine learning," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, 2020, pp. 1–8.
- [44] P. Laskov, P. Düssel, C. Schäfer, and K. Rieck, "Learning intrusion detection: supervised or unsupervised?" in *International Conference on Image Analysis and Processing*. Springer, 2005, pp. 50–57.
- [45] S. Zanero and S. M. Savaresi, "Unsupervised learning techniques for an intrusion detection system," in *Proceedings of the 2004 ACM symposium* on Applied computing, 2004, pp. 412–419.
- [46] M. Usama, J. Qadir, A. Raza, H. Arif, K.-L. A. Yau, Y. Elkhatib, A. Hussain, and A. Al-Fuqaha, "Unsupervised machine learning for networking: Techniques, applications and research challenges," *IEEE* access, vol. 7, pp. 65579–65615, 2019.
- [47] Y. Liu, Y. Gu, X. Shen, Q. Liao, and Q. Yu, "Msca: An unsupervised anomaly detection system for network security in backbone network," *IEEE Transactions on Network Science and Engineering*, 2022.
- [48] P. E. Hoffman and P. McManus, "DNS Queries over HTTPS (DoH)," RFC 8484, Oct. 2018. [Online]. Available: https://rfc-editor.org/rfc/ rfc8484.txt
- [49] C. Cimpanu, "First-ever malware strain protocol," doh (dns 2019. ing new over https) https://www.zdnet.com/article/first-ever-\ [Online]. Available: malware-strain-spotted-abusing-new-doh-dns-\over-https-protocol/
- Sharma, "Attackers abuse Α. google dns over download malware," 2020. https [Onhttps://www.bleepingcomputer.com/news/security/ line1. Available: \attackers-abuse-google-dns-over-https-to-\download-malware/
- [51] C. Osborne, "Psixbot upgraded malware with google sexploitation dns over https kit." 2022 [Onhttps://www.zdnet.com/article/psixbot-malware-\ Available: upgraded-with-google-dns-over-https-\sexploitation-kit/
- [52] K. J. Higgins, "3 ways attackers bypass cloud security," 2022. [Online]. Available: https://www.darkreading.com/cloud/3-ways\ -attackers-bypass-cloud-security
- [53] N. H. Ian Kenefick, Lucas Silva, "Black basta infiltrates networks via qakbot, brute-ratel, and coba," 2022. [Online]. Available: https://www.trendmicro.com/en_us/research/22/ j/\black-basta-infiltrates-networks-via-qakbot-\brute-ratel-and-coba. html
- [54] Mozilla, "Doh resolver policy-conforming resolvers," 2022, Mozilla Wiki, accessed 22 December 2022. [Online]. Available: https://wiki.mozilla.org/Security/DOH-resolver-policy
- [55] Microsoft, "Dns over https (doh) client support," 2022, Microsoft Learn, accessed 22 December 2022. [Online]. Available: https://learn.microsoft.com/en-us/windows-server/\networking/dns/doh-client-support
- [56] "Top websites," 2021. [Online]. Available: https://dataforseo.com/ top-1000-websites
- [57] "Nfstream: Flexible network data analysis framework," 2023.
- [58] J. Bader, "Domain generation algorithms," 2022. [Online]. Available: https://github.com/baderj/domain_generation_algorithms
- 59] Google, "Tensorflow," www.tensorflow.org, 2020.
- [60] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

- [61] F. Falcão, T. Zoppi, C. B. V. Silva, A. Santos, B. Fonseca, A. Ceccarelli, and A. Bondavalli, "Quantitative comparison of unsupervised anomaly detection algorithms for intrusion detection," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, 2019, pp. 318–327.
- [62] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:36663713
- [63] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in 2008 eighth ieee international conference on data mining. IEEE, 2008, pp. 413–422.
- [64] M. Amer, M. Goldstein, and S. Abdennadher, "Enhancing one-class support vector machines for unsupervised anomaly detection," in *Pro*ceedings of the ACM SIGKDD workshop on outlier detection and description, 2013, pp. 8–15.
- [65] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *Proceedings of the 2000 ACM SIGMOD* international conference on Management of data, 2000, pp. 93–104.