6520 Digital Forensics: Domain Name Service Attacks Defense and Prevention

Edward Crowder,
Ahmad Chaiban

ABSTRACT

Data exfiltration and its prevention is an essential aspect of the Digital Forensics field. Many solutions have been attempted in order to prevent these types of attacks over networks. The solution proposed in this research study makes use of four Machine Learning (ML) models in order to classify malicious and benign DoH network traffic. Data on realistic DoH network traffic was retrieved from an online data repository provided by the CIC (Canadian Institute for Cybersecurity). The ML models performed optimistically, and some performed well on practical datasets. The training of these ML models was completed through undersampling the data with cluster centroids, and the evaluation of these models was completed using the recall method, ROC curves, and testing on extremely noisly malicious samples. It was then concluded that in order to enhance the prevention of exfiltration, IDS/IPS rules, using services like YARA, SNORT and SURICATA, would have to be set up against domain names when the ML detection would have a high enough confidence.

1 INTRODUCTION

Domain Name Service (DNS) is arguably the back bone to today's internet. The goal of domain names is to provide a mechanism for naming resources in such a way that the names are usable in different hosts, networks, protocol families, Internets, and administrative organizations [4]. With this simple fact, it's imperative we are able to distinguish between malicious and benign name services with ease. How ever this is not always the case. Most of what makes a domain name malicious from sight comes from an analysts intuition based on past experience, analyst bias, geological context, political views, and much more. Using anomaly detection, we aim to group similar normal data and build a normal model so that we can identify outlier's.

2 RELATED WORK

To fully understand how to profile DNS traffic, we must first break down the options adversaries have to abuse DNS and what Expected DNS Traffic should be. Expected DNS Traffic will discuss some common types, architectures, and routes DNS is expected to travel. DNS network anomaly traffic describes the requirements to capture the data required to make an informed decision. Once we have the data, we can break it down into categories of malicious and non-malicious. Under malicious, we would expect it to match sections 2.1 and 2.2. Under malicious, we can expect traffic closer to sections 2.3 - 2.6.

2.1 Expected DNS Traffic Behaviours

To fully understand what can classify as malicious we must keep in mind the standard expected DNS packet. Typically we expect DNS queries to utilize UDP for a majority of traffic and TCP for zone transfer requests. A DNS zone transfer request transmits its domain database over TCP because TCP connections offer the consistency and reliability required to ensure data is transferred correctly. It was observed that the average normal DNS traffic often has longer TTL values, and generally are not very large packet size. Known ports are also notable features, because DNS traffic should not traverse unknown ports.

2.2 DNS Network Anomaly detection

Network anomaly detection detects something that deviates from what is standard, normal, or expected. Satam et al. describe one possible avenue to capture and train a model based on an enterprise network traffic baseline in only a few days [8]. Network baselines identify anomalous traffic that exceeds the expected thresholds. Monitoring an enterprise network based on expected network traffic thresholds provides insights into the infrastructure that allow informed decisions on issues as they arise. Threat detection is made possible by investigating obvious network outliers that would have usually gone unnoticed. However, detecting attacks with time-series-based anomaly detection is not all-encompassing—sophisticated attacks such as Botnet C2 communication, DNS Tunneling, and Typo-squatting can be exceptions. Preston noted that network traffic baselining can be avoided if the adversary modifies their techniques and tactics [7].

2.3 Botnet C2 over DNS

The Domain Generation Algorithms (DGAs) to dynamically generate the domain names that resolve to their command and control (C2) centers [3]. C2 communications are often detected as outlier's or during frequency analysis, due to their rapid introduction of new domains to the server cache. This can viewed as an anomaly in an enterprise that tends to have predictable web service use.

2.4 DDoS Attacks detection

A distributed denial-of-service (DDoS) is a malicious attack that threatens to disrupt the regular traffic of targeted servers, services or networks. It attempts to overwhelm the target, and in some cases, the surrounding infrastructure through flooding internet traffic.

To classify this specific attack, Wang et al. divised a novel method that stems from characteristics of attack traffics (CAT) time series [6]. This series is generated then transformed into a multidimensional feature space that is used to train a Support Vector Machine for classifying the abnormal flow of DDoS attacks. In order to create this CAT time series, Wang et al. extracted and combined the ratio of failed resolutions to successful resolutions, and the ratio of UDP responses to UDP requests. Then, Wang et al. used the Recursive Least Square (RLS) algorithm in order to generate their weight vectors and train their SVM classifier. Their results were optimistic,

with less than 5 percent false positive rates and an over 90 percent overall accuracy.

It should also be noted that the method proposed by Wang et al. could be quite robust and informative in enterprise settings. The time series itself can be monitored with the possible usage of certain pattern and time series analysis algorithms which could aid in detecting patterns and plans of malicious attack.

2.5 DNS Tunneling detection

DNS Tunneling attacks involve encoding the data of certain programs/protocols into DNS queries and responses. DNS tunneling may include data payloads built to attack DNS servers and possibly control remote servers and applications.

In order to detect this type of attack, Preston extracted certain features from DNS data and trained six machine learning models [7]. A Random Forest classifier, Gradient Boosting classifier, AdaBoost, Bagging, Support Vector Machine, and Stochastic Gradient Descent classifier. The features selected for training by Preston included character entropy, the alphanumeric content ratio, unique query volume, unique query ratio, and other representative characteristics of DNS tunneling data. Moreover, after amassing approximately 80,000 benign domain feature data points for training, Preston developed a DNS tunneling tool in Node.js in order to simulate and monitor adversarial behavior. After training these six selected classifiers, Preston performed an evaluation on the models, presenting highly optimistic results. However, Preston suggests that his method is effective and practical to a certain degree. If a malicious actor were to modify their behavior or find workarounds to the deployed model that runs over specific past intervals of time. Perhaps a more robust method is required in order to detect several variations of malicious behavior regarding this type of attack.

2.6 Typo-squatting

Typo-squatting is defined as the registration and deployment of a domain name that is similar to that of existing popular brands or websites with the intention of redirecting users to those websites that may contain malicious activity and content.

Moubayed et al. employed the usage of ensemble-based feature selection and bagging classification models in order to detect Typo-squatting [5]. Their proposed solution makes use of several mathematical characteristics such as the length of domain name, ratio of letters to domain length, and other similar NLP-based extractions as features for the classification model. Moubayed et al. designed a methodology of feature extraction and methodology that seemed to provide optimistic results in the detections made. The data was first transformed and the required features extracted, then a three-level feature selection system was implemented for optimal feature selection, and finally, Moubayed et al. applied feature reduction before training the bagging ensemble models for classification.

A performance evaluation by Moubayed et al. indicated an overall accuracy above 85%, and varying recall and F-scores. However, Moubayed et al. showed that although the training feature set was significantly reduced, the K-NN and decision-tree bagging ensemble classifiers maintained high accuracy, precision and F-score values. This is a significant step forward in terms of discovering models that are able to train on smaller amounts of DNS data and still perform respectably.

3 DESIGN GOALS

The design goals are broken up into two major sections, (3.1) Detection and (3.2) Prevention.

3.1 Detection

The primary goal of the project is to detect and classifying DNS traffic at the network level accurately. Classification of benign or malicious traffic through analysis of network datagram features is critical in protecting today's enterprises.

User transparency was also a key design goal we had in mind. The binary classification system is placed similarly to an IPS, directly behind the network's egress network traffic. The idea is to detect and capture outgoing DNS packets and create blocking rules for the IDS system before the packets leave our network via a network tap. This active monitoring avoids collecting any information regarding the source devices, and the intern only contains features that can not violate the end users' privacy.

3.2 Prevention

The secondary goal of the project is to generate static rules for popular open source security products. Suricata, Yara, Snort, and TAXII/STIX have been chosen to achieve this task. It was essential in our selection of defensive mechanisms that they are (a) open source and (b) widely used in the enterprise environment. For these two reasons, the firewall rules serve as a compliment to the ML pipeline due to their openness and wide variety of public data in their formats.

4 ARCHITECTURE AND IMPLEMENTATION

4.1 DoH Traffic Data

The data supplied for the training of the Machine Learning (ML) Model was obtained from an open-source provided by the Canadian Institute for Cybersecurity (CIC). The data repository, titled "CIRA-CIC-DoHBrw-2020", was collected using a double-layered method that tracks and captures benign and malicious DoH and Non-DoH network traffic. In order to generate proper representative data, benign DoH and non-DoH traffic is generated through accessing the top 10k Alexa websites, with the usage of browsers and DNS tunnleing tools that support the DoH protocol. At the first layer of the collection method, captured traffic is labelled as either DoH or non-DoH, and at the second layer, the data is labelled as benign or malicious DoH [1].

In order to generate malicious DoH traffic data, DNS tunneling tools such as dns2tcp, DNSCat2, and Iodine were utilized. These tools send TCP traffic embedded in DNS queries. These tools create tunnels of encrypted data, and therefore, DNS queries are sent with TLS-encrypted HTTPS requests to certain DoH servers.

For the purposes of this study, only malicious and benign DoH samples from this data repository were used in order to build the required classifier, as this study primarily tackles the classification between benign and malicious traffic.

The reason this dataset was selected for this research study was its strong representation of noisy and practical real world DNS Traffic data. It is essential that any ML-based model train on a realistic dataset in order to replicate predictions that would occur in a real-world setting.

4.2 DoH Data Preprocessing

Upon first importing the data, there were 35 features given by the original dataset. They span from Source IP Port, Destination IP Port, to flow bytes sent, received, rate and packet data length, and various response time metrics.

At first, the benign and malicious DoH datasets were given separately by the CIC data source repository, and therefore, for the purposes of this research study, the data was merged, and any null values were dropped, in order to begin building a dataset for binary classification.

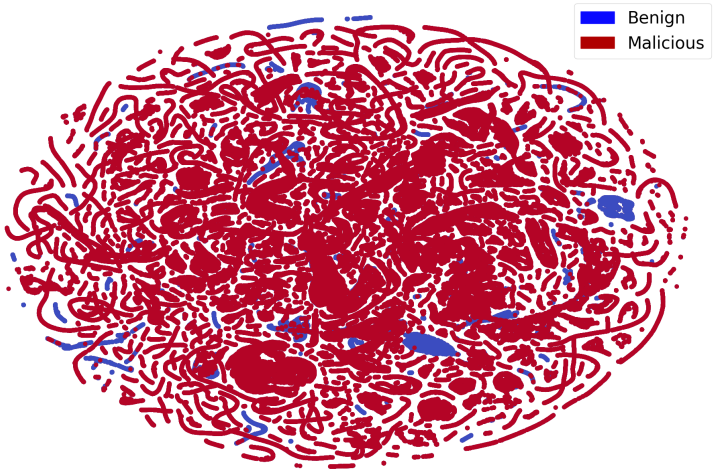
After dropping null values, the dataset contained 249, 836 malicious samples and 19, 746 benign samples, which implies that the data is quite imbalanced. A t-SNE visualization of this imbalanced dataset can be seen in figure 1a. Clearly, there is no discernible pattern or separation between the malicious and benign data points, which is mainly due to the extremely large sample of malicious data.

In order to solve this issue, the challenge was to find a method that reduces the number of malicious samples without compromising the integrity of the data. The method selected for this task was undersampling with cluster centroids. This method undersamples the majority class of the data (in this case the malicious samples) by replacing a cluster of those majority samples by the cluster centroids obtained by fitting the K-means unsupervised learning method. An N number of clusters is selected for that one majority class (the malicious samples), which in this case is the number of benign samples (19, 746), and those majority samples are replaced by the selected N centroids found.

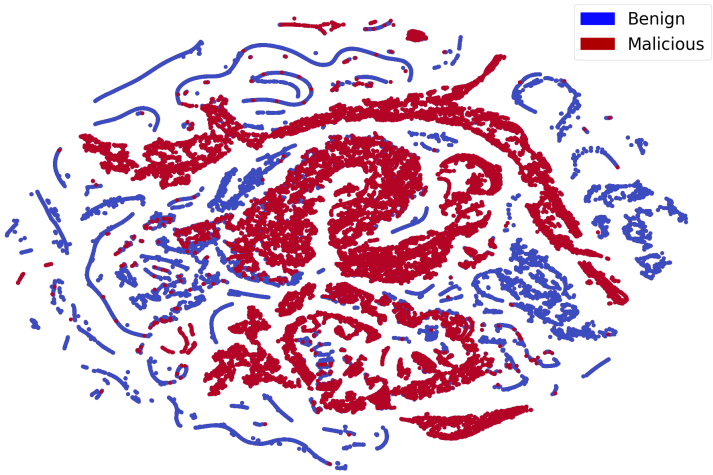
This method proved to effectively undersample the data, allowing for a discernible separation in the data, and its results can be seen in figure 1b. There is still indeed a significant loss of data with this method, and will be discussed at length in the limitations section. However, for training purposes, the assumption made, through the usage of cluster centroids, is that this new set of malicious data does still statistically represent the same distribution of data as the original dataset.

At this stage, after merging and undersampling the data, the data was then normalized in order to prepare it for feature selection and training the selected classifiers.

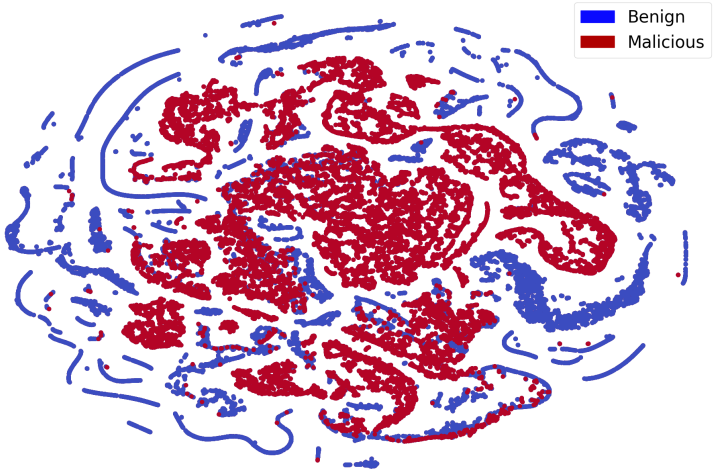
It is also worth mentioning that in practice, a network tap on LAN egress would allow the data to be processed upon arrival so that our prevention mechanisms, which will be discussed in a later section, could update the required systems before traffic the external DNS server replies on LAN ingress. Although this part of the system was not implemented because the network architecture is out of the scope of the paper, it would be fairly trivial to route the flow of information to our system once it has been captured, and build a feature extraction pipeline.



(a) t-SNE plot before preprocessing



(b) t-SNE plot after undersampling



(c) t-SNE plot after undersampling and feature selection

Figure 1: t-SNE plot at different stages of preprocessing

4.3 Feature Selection

In order to discover the features most suitable for training, two algorithms were utilized. The first was to select the top 10 correlated features based on a Pearson correlation matrix, and the second was to select the 10 best features using the chi squared method. Both methods displayed similar features, and therefore, the 10 features decided by the Pearson correlation method were the ones finally chosen for training. Table 1 shows the features selected in order.

Finally, a t-SNE visualization was produced, and showed a slightly stronger discernible separation in the data. The results can be seen in figure 1c.

No.	Feature
1	FlowSentRate
2	FlowReceivedRate
3	PacketTimeCoefficientOfVariation
4	PacketTimeSkewFromMode
5	PacketLengthMean
6	FlowBytesReceived
7	PacketLengthMedian
8	PacketLengthMode
9	SourcePort
10	FlowBytesSent

Table 1: Highly correlated DNS packet features extracted using a Pearson correlation matrix

4.4 Machine Learning

Four models were selected for training on the preprocessed data. A Support Vector Machine (SVM), a Random Forest Classifier (RFC), a Long-Short-Term Memory model (LSTM), and an Artificial Neural Network (ANN). The thought process for selecting these models was first to select two standard models in order to benchmark against more advanced models. These were the SVM and RFC. Next, a more robust classifier more suitable for this type of task, the LSTM, was selected. LSTM have the ability to study and train on sequences of data, which is considered vital for this type of problem since most of the data being used in this study comes from specific sources and are sequences of traffic from those same sources. Therefore, it was selected and deemed suitable for testing on this data and this type of problem. The final model was a Neural Network built with standard dense layers.

4.5 Model Training and Hyper-parameters

The ML training process began with a train-test-validation split of 67%, 22% and 11% respectively. The SVM and RFC were trained with the parameters found in tables 2 and 3 respectively. The choice of parameters depended heavily on multiple training and evaluation tests with various feature plot diagrams to assess the hyper-parameters and the ML training diagnostics. In other words, the parameters were selected through various experimentation tactics on the dataset.

Regarding the two selected Neural Networks, the LSTM and ANN parameters can be found in figures 4a and 4b respectively.

Parameter	Value
C	1.0
γ	1e4
Kernel	RBF (Radial Basis Function)

Table 2: SVM Training Parameters

Parameter	Value
No. of Estimators	50

Table 3: RFC Training Parameters

These parameters were selected based on several factors. The skeleton of the LSTM network was selected through the usage of certain research papers that used similar architectures for various classification problems. Elsayed et al. describe their LSTM's architecture as having 128, 64, 32 and 16 dense layers, in that order, which is where our model began and then evolved through experimentation on the training/test data [2].

The final training cycle of the LSTM included training the model with a batch size of 128, and eventually 1000 epochs, all the while using the validation data. Binary Crossentropy was used for loss and the optimizer selected was the Adam optimizer. The activation function on the final output layer was the Sigmoid activation function. The entire training history of the model can be found under figure 2.

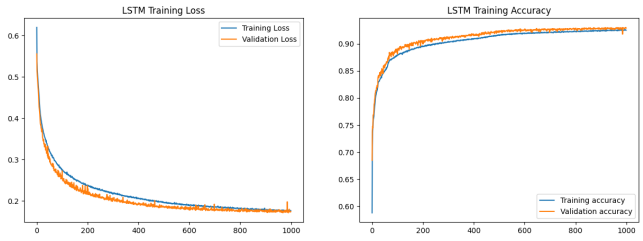


Figure 2: LSTM training accuracy and loss VS No. of training iterations

The ANN followed a similar pattern. The main idea for this model however was to find a relatively shallow Neural Network and benchmark it against the power LSTM, SVM and RFC. The layers were built from the ground up and purely through experimentation on the dataset. The input and output layers of the ANN utilized the Sigmoid activation function, the model was also trained on 1000 epochs with a batch size of 32, the loss and optimizer used were also Binary Crossentropy and Adam respectively. Moreover, the training history of the shallow ANN can be found under figure 3

4.6 ML Evaluation

To evaluate the ML models, several metrics, plots, and methods were used. The first of these was the most intuitive, using each model to predict on the test data and then using the accuracy score metric. Table 4 outlines these accuracies.

The next step was to evaluate the models using the recall method. This firstly involves finding the confusion matrices, which can be

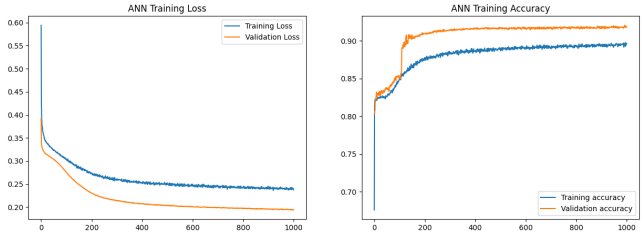


Figure 3: ANN training accuracy and loss VS No. of training iterations

```
Model: "LSTM"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_63 (LSTM)               (None, 10, 64)            16896
_________________________________________________________________
dropout_53 (Dropout)         (None, 10, 64)            0
_________________________________________________________________
lstm_64 (LSTM)               (None, 10, 32)            12416
_________________________________________________________________
dense_23 (Dense)             (None, 10, 1)             33
=================================================================
Total params: 29,345
Trainable params: 29,345
Non-trainable params: 0
_________________________________________________________________
```

(a) LSTM architecture

```
Model: "Artificial Neural Network"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_20 (Dense)             (None, 50)                550
_________________________________________________________________
dropout_50 (Dropout)         (None, 50)                0
_________________________________________________________________
dropout_51 (Dropout)         (None, 50)                0
_________________________________________________________________
dense_21 (Dense)             (None, 1)                 51
=================================================================
Total params: 601
Trainable params: 601
Non-trainable params: 0
_________________________________________________________________
```

(b) ANN architecture

Figure 4: LSTM and ANN architectures

No.	Algo	Training Accuracy
1	SVM	93.25%
2	RFC	98.41%
3	LSTM	78.77%
4	ANN	91.50%

Table 4: ML test split results

found under figure 5. They seemed to indicate optimistic results on the test data, with low false positive and low false negative rates. The next step was to assess the true positive rate VS the false positive rate, or in other words, the ROC curves of each model. These results can be found under figure 6. These curves at first glance seem to indicate several points. The first is that either the SVM and RFC models are over-fitting on the data, or that they perform overly optimistically, however, further tests are required to make any further conclusions.

A final necessary evaluation was to test wheter the models can recognize samples outside of the ones trained on, tested on and curated through undersampling. Therefore, the original dataset, with the full 249, 836 malicious samples, was reused for prediction. The data was normalized and underwent feature selection without undersampling. And at this point, after prediction, some interesting results emerged. Table 5 outlines how accurate the models performed on the this full dataset. Morever a significant different in the confusion matrices was discovered, which can be seen in figure 7.

$$\begin{bmatrix} 3877 & 494 \\ 86 & 4145 \end{bmatrix}$$

(a) SVM Confusion Matrix

$$\begin{bmatrix} 4283 & 88 \\ 49 & 4182 \end{bmatrix}$$

(b) RFC Confusion Matrix

$$\begin{bmatrix} 3217 & 1154 \\ 672 & 3559 \end{bmatrix}$$

(c) LSTM Confusion Matrix

$$\begin{bmatrix} 3800 & 571 \\ 160 & 4071 \end{bmatrix}$$

(d) ANN Confusion Matrix

Figure 5: Confusion Matrices on the test set

No.	Algo	Testing Accuracy
1	SVM	94.09%
2	RFC	98.67%
3	LSTM	47.72%
4	ANN	76.54%

Table 5: ML full data results

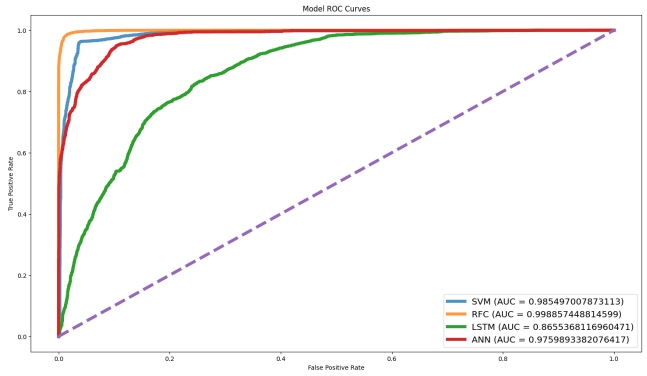


Figure 6: Receiver Operating Characteristic Curves (ROC) for all four models. True Positive Rate VS False Positive Rate

$$\begin{bmatrix} 18735 & 1011 \\ 14898 & 234655 \end{bmatrix}$$

(a) SVM Confusion Matrix

$$\begin{bmatrix} 19612 & 134 \\ 3446 & 246107 \end{bmatrix}$$

(b) RFC Confusion Matrix

$$\begin{bmatrix} 14461 & 5285 \\ 135495 & 114058 \end{bmatrix}$$

(c) LSTM Confusion Matrix

$$\begin{bmatrix} 17089 & 2657 \\ 60521 & 189032 \end{bmatrix}$$

(d) ANN Confusion Matrix

Figure 7: Confusion Matrices on the full original dataset

drop dns any any -> any any (msg:"6520 AI Malciious Detection"; content:"http://www.google.ca"; classtype:policy-violation; sid:5599962; rev:1;)
drop tls any any -> any any (msg:"6520 AI Malciious Detection"; tls.subject:"http://www.google.ca"; classtype:policy-violation; sid:5599962; rev:1;)

Table 6: Example Suricata TLD DNS Block Rules

4.7 Malicious DNS Prevention

Preventing malicious network traffic is a trivial task with many modern solutions available. Open source projects such as Snort and Suricata provide the ability to write static rules to alert or block detection near real-time. Yara and Stix/TAXII offer frameworks to enable threat hunting and information sharing, respectively. We chose to implement the aforementioned software due to its maturity and use in most common enterprises. Table 6 provides an example of a block rule generated by our prevention engine. We use a static drop rule to parse incoming traffic containing DNS or TLS traffic to detect both DoH and standard traffic to the parent domain. The SID's must be unique for this software, and so we had chosen a random number between 5,000,000 and 6,000,00 arbitrary to keep track of the rules generated by our prevention rule.

5 DISCUSSION

Given the several evaluation metrics collected, the SVM and RFC seem to have performed with extremely optimistic results. When looking at the first few evaluation metrics, the test scores and ROC curves, the models seem to be over-fitting. However, a strong argument can be made that suggests both of these models are quite powerful in their detection. The final scores and confusion matrices seen in figure 7, after evaluating on the large dataset that was not undersampled, suggest that these two models were able to identify over 230, 000 samples they did not train or validate on. This seems to put the models in a high tier level where futher tests will need to be conducted in the future.

Regarding the Neural Networks however, they seemed to, after many rounds of training and hyper-parametric tuning, lag behind when attempting to predict samples outside of the processed undersampled ones. Their accuracies and confusion matrix scores are quite lacking, and it is either the case that the model architectures and training need improvement, or that the models themselves may not be suitable for this type of problem.

It is however important to discuss the reliability of ML models as a whole for this type of solution. ML models will always have a false positive and false negative rate. For this reason, automating the detection alone through ML may not suffice, and the IDS/IPS rules were selected as a compliment to the detection for this reason. Confidence scores of the detection can be taken and used in order to generate rules against certain URLS and behaviors, allowing for a more secure network environment if a URL ever gives a relatively notable confidence score.

6 LIMITATIONS AND ASSUMPTIONS

The use of publicly available datasets, rather than collecting and curating data that was more balanced, was due to the project's time constraint. A proof of concept was forged through the usage of CIC's publicly available dataset for training the ML models.

Although the SVM and RFC seem to have performed extremely well on these datasets, the main issue is how limited in scope the CIC data is compared to a real network environment, and how well these models would actually perform in these environments. For these reasons alone, the usage of CIC's publicly available dataset and the time constraint, the following are the limitations of this research study.

(1) It is firstly assumed that since Cluster Centroids was used to undersample the data, that the undersampled dataset is indeed still representative of the whole CIC dataset. The optimistic SVM and RFC results seem to indicate this, however, further tests are required.
(2) It may be the case that the CIC dataset may not wholly or accurately represent DoH traffic realistically, and therefore, further experimentation is required.
(3) Replicating an enterprise network and implementing a network tap to test on a realistic network environment with real DoH network traffic data was out of scope due to time.
(4) Prevention engine rule outputs are bare minimum due to time and complexity.
(5) The neural networks either require architecture overhauls, more training, or to be replaced entirely, as they currently do not seem to be very effective at solving the DoH binary classification problem.
(6) Lastly, the AI detection engines will always be limited by their input data, and therefore, testing the models rigorously is a necessary future step.

7 CONCLUSION

We have shown that the use of artificial neural networks for general DNS traffic detection has promising results. However, somewhat surprisingly, the SVM and RFC, that were originally intended to be baselines for a more complicated solution provided the highest accuracy. Overall, all 4 models successfully were able to identify most malicious from benign samples with high confidence. The prevention engine is capable of blocking any confident true positive

detections completely from the network while also generating rules for threat sharing and hunting. The generated output is useful in disrupting adversarial attacks in their track, while also promoting the digital forensic and incident response process.

8 FUTURE WORK

Dedicating time into reconstructing an enterprise environment in order to reliably collect and test with more realistic data is the next obvious step to further investigating the project's accuracy. Other major next steps are listed in 8 which solve many of the limitations laid out in section 6.

(1) Further test the ML models that performed well on more realistic and new datasets.
(2) Attempt to train multi-attention head transformers on URL detection.
(3) Adapt model to other forms of malicious attacks, or train new models.
(4) Source and retrain model with larger data sets.
(5) Enhance the prevention engine to support more dynamic rule output.
(6) Implement real-time prevention mechanism through host-based network monitors.

REFERENCES

[1] CIC dns over https dataset. https://www.unb.ca/cic/datasets/dohbrw-2020.html. Accessed: 2021-04-10.
[2] Mahmoud Elsayed, Nhien-An Le-Khac, Soumyabrata Dev, and Anca Jurcut. Network anomaly detection using lstm based autoencoder. 11 2020.
[3] A. Menon. Thwarting c2 communication of dga-based malware using process-level dns traffic tracking. In *2019 7th International Symposium on Digital Forensics and Security (ISDFS)*, pages 1–5, 2019.
[4] Paul Mockapetris. DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION. RFC 1035, RFC Editor, November 1987.
[5] Abdallah Moubayed, Emad Aqeeli, and Abdallah Shami. Ensemble-based feature selection and classification model for dns typo-squatting detection. In *2020 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–6. IEEE, 2020.
[6] Tongguang Ni, Xiaoqing Gu, and Hongyuan Wang. Detecting ddos attacks against dns servers using time series analysis. *TELKOMNIKA Indonesian Journal of Electrical Engineering*, 12(1):753–761, 2014.
[7] Richard Preston. Dns tunneling detection with supervised learning. In *2019 IEEE International Symposium on Technologies for Homeland Security (HST)*, pages 1–6. IEEE, 2019.
[8] Pratik Satam, Hamid Alipour, Youssif Al-Nashif, and Salim Hariri. Anomaly behavior analysis of dns protocol, 2015.