Deep Learning Based DNS Tunneling Detection and Blocking System

Mehmet Ali ALTUNCU, Fidan KAYA GÜLAĞIZ, Hikmetcan ÖZCAN, Ömer Faruk BAYIR,
Alperen GEZGİN, Ata NİYAZOV, Mehmet Ali ÇAVUŞLU, Suhap ŞAHİN
Computer Engineering, Kocaeli University, Kocaeli, 41001, Turkey
hikmetcan.ozcan@kocaeli.edu.tr

***Abstract*—The main purpose of DNS is to convert domain names into IPs. Due to the inadequate precautions taken for the security of DNS, it is used for malicious communication or data leakage. Within the scope of this study, a real-time deep network-based system is proposed on live networks to prevent the common DNS tunneling threats over DNS. The decision-making capability of the proposed system at the instant of threat on a live system is the particular feature of the study. Networks trained with various deep network topologies by using the data from Alexa top 1 million sites were tested on a live network. The system was integrated to the network during the tests to prevent threats in real-time. The result of the tests reveals that the threats were blocked with success rate of 99.91%. Obtained results confirm that we can block almost all tunnel attacks over DNS protocol. In addition, the average time to block each tunneled package was calculated to be 0.923 ms. This time clearly demonstrates that the network flow will not be affected, and no delay will be experienced in the operation of our system in real-time.**

***Index Terms*—artificial neural networks, computer networks, intrusion detection, Domain Name System, machine learning.**

I. INTRODUCTION

DNS (Domain Name System) is one of the important protocols and services used on the Internet. DNS consists of a collection of servers that allow quick response not to be affected by the increase in domain names and number of users and can be accessed from anywhere in the world [1]. Although it has multiple purposes, the most important function of DNS is to convert domain names and IP addresses both ways. For example, when the user enters the address www.kocaeli.edu.tr in the browser, the IP address corresponding to this address is obtained through the following steps (Fig. 1).

1. The resolver looks up in its own cache for the IP address corresponding to the address. If it is stored in its cache, it responds to the query and provides access to the corresponding web page.
2. If the IP address is not stored in the cache of the resolver, the resolver will query Root Name Servers for the address to be accessed. Root servers send the incoming requests to TLD (Top-Level Domain) server addresses known to them in the next level of the hierarchy. The operations performed in this step are shown by arrows 1, 2, 3 and 4 respectively, in Fig. 1.
3. The TLD server is where the top-level domain address information (such as .com, .org., .net) is stored and the workload distribution is initially performed. For example, in the www.kocaeli.edu.tr address, .edu addresses an educational institution and .tr addresses the country domain code for Turkey. After the TLD server, the query is transferred to Authoritative Name Server (ANS), which is the last step in the hierarchy. The operations performed in this step are shown by arrows 5 and 6 respectively, in Fig. 1.
4. To ensure efficient address resolution, an authorized ANS is associated with each geographic region. As the name ANS suggests, they are the servers that send the IP addresses in DNS queries to the resolver. The IP address received by the resolver is then sent to the computer requesting the DNS, which enables access to the associated web page [2]. The operations performed in this step are shown by arrows 7 and 8 respectively, in Fig. 1.

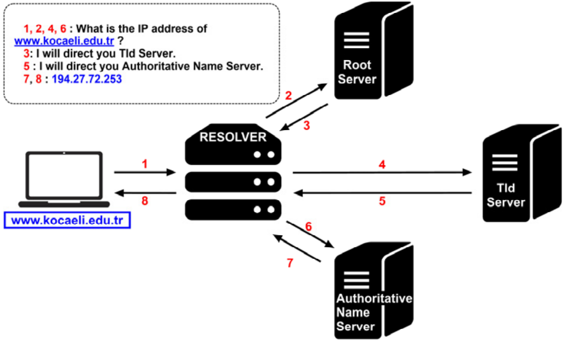
1, 2, 4, 6 : What is the IP address of www.kocaeli.edu.tr ?
3: I will direct you Tld Server.
5 : I will direct you Authoritative Name Server.
7, 8 : 194.27.72.253
Root Server
RESOLVER
Tld Server
Authoritative Name Server
www.kocaeli.edu.tr
1
2
3
4
5
6
7
8

Figure 1. DNS protocol working scheme

Although a typical DNS query uses the UDP/53 port, DNS queries with a response greater than 512 bytes use the TCP/53 port. Since the main purpose of DNS is not to convey data, DNS is generally not perceived as a threat for malicious communications or data leakage. Therefore, these ports (port 53) are usually enabled on firewalls. Having these ports enabled constitutes a security flaw with regards to various types of attacks, such as the file transfer system in the DNS protocol. Tunneling method is generally used in these attacks that take advantage of this vulnerability. The transfer of data belonging to one protocol over another protocol is called tunneling, whereas the transfer of the data via DNS packets is called DNS tunneling [3].

It will become clearer to explain what can be achieved through DNS tunneling over a scenario. For example, the following steps are carried out respectively when querying the address abc.tunnel.mali.com from a system on the local network (Fig. 2):

1. The local DNS server checks the records in its cache for the received query and if there is a record, it returns a response.

Digital Object Identifier 10.4316/AECE.2021.03005

2. If there are no records in its cache, it first finds out the DNS server associated with mali.com.
3. After finding the DNS server associated with mali.com, it queries the server that is associated with the tunnel.mali.com subdomain and queries the address abc.tunnel.mali.com on that server.

Thus, any DNS request from the local network reaches the DNS server associated with abc.tunnel.mali.com (let's assume that it is netsec.mali.com). If we assume that a malicious user creates a special DNS request and placed the data he wants (xyz byte instead of abc) in the field outside the query section, this request will be received at port UDP/53 of netsec.mali.com without any changes. By running a special application at netsec.mali.com, data from a specific client can be interpreted and thus data is leaked via DNS tunneling [4]. Briefly, DNS tunneling consists of a client and a server that will recognize the packets generated by this client. It runs on the DNS port of the server, making it look like a real DNS server.

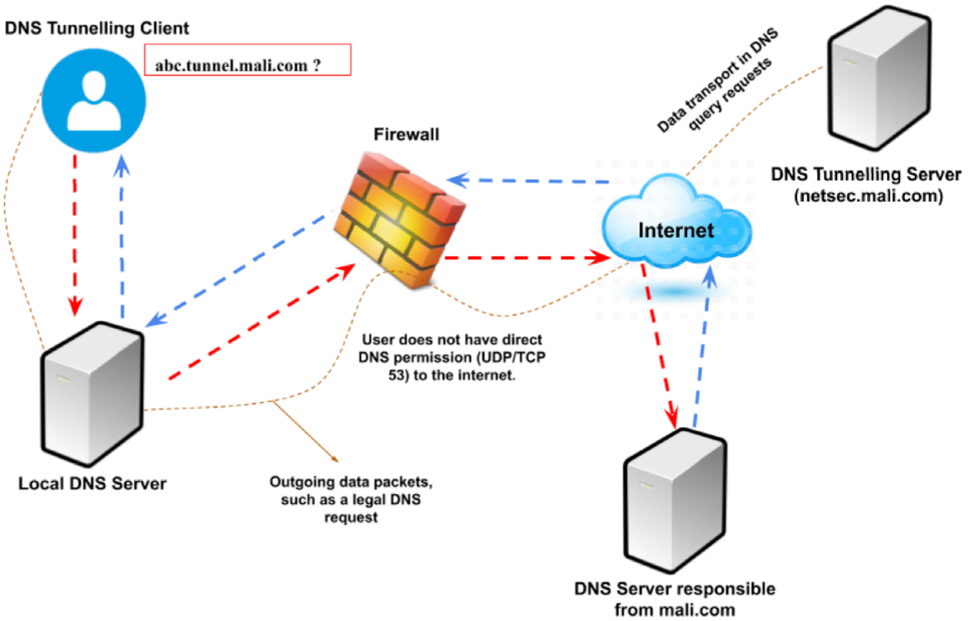
DNS Tunnelling Client
abc.tunnel.mali.com ?
Firewall
Data transport in DNS query requests
DNS Tunnelling Server (netsec.mali.com)
Internet
User does not have direct DNS permission (UDP/TCP 53) to the internet.
Local DNS Server
Outgoing data packets, such as a legal DNS request
DNS Server responsible from mali.com

Figure 2. A general scheme of DNS tunneling

There are several DNS tunneling tools that allow data to be injected into DNS queries and responses. The most popular among these tools are Iodine [5], DNSCAT2 [6] and DNS2TCP [7] due to ease of installation or platform independence. DNS Tunneling tool is responsible for exchanging the data that it previously placed in the appropriate section of DNS packets while sending DNS queries and responses between the tunnel client and the server. This way, the DNS tunneling server can perform the data leak operation by transmitting the received data to a target client [8]. Although their purposes are the same, each tool has its own unique architecture for creating a tunnel between the DNS tunneling client and the server.

In this study, a deep learning-based system is proposed that detects and prevents DNS tunneling attacks in real-time. The proposed system has three main contributions. Our first contribution is to obtain a comprehensive analysis data according to the real DNS traffic and to extract the IP length, query name length and query name entropy properties that will distinguish between legal queries and DNS tunneling attacks from the obtained DNS messages. Our second contribution is to develop and train a deep learning-based algorithm that will detect DNS tunneling attacks using the attributes listed above. Our last and most important contribution is that the obtained tunneling data is injected into an academic network in real-time, showing that the proposed deep learning-based system detects tunneling attacks with high accuracy and prevents them in real-time.

The rest of the study is organized as follows. Section II provides information on the studies in the literature and mentions the incentives behind the study. Section III describes the steps for the implementation of the decision mechanism that detects DNS tunneling in detail. Section IV provides information on the operations performed to block tunneled DNS queries in real-time. The configuration implemented for testing the suggested system, success rates and real-time system performances are provided in Section V. We complete this paper with the conclusion.

II. RELATED WORK

In general, two features of network packets are used in the detection of DNS Tunneling. These are load and traffic analysis. In the load analysis, DNS tunneling is detected by analyzing some properties of each DNS request and response one by one, such as domain length, number of bytes and content. Whereas in traffic analysis, DNS tunneling is detected by using statistical features of traffic in general (volume of DNS traffic, number of computer names per domain, geographic location, etc.) and domain history [9].

After the properties of the network packets are determined, the method to be used for DNS tunneling detection should be specified. When the studies in the literature are examined, classification-based machine learning techniques are generally used in DNS Tunneling detection. Aiello et al. [10] performed real-time DNS tunneling detection using one of the supervised learning techniques, the Bayesian classification method. Statistical properties of DNS queries and responses were used in the study.

In another study, Aiello et al. [11] performed real-time DNS tunneling detection with a different method by using statistical features such as arrival times between packets and packet sizes of DNS data. The results of machine learning techniques for DNS tunneling detection were compared in the study and it was concluded that the classification-based methods yielded more successful results.

Sammour et al. [3] used classification-based machine learning techniques for DNS Tunneling detection. In this study, DNS request length, IP packet sender length, IP packet response length, encoded DNS query name length, request application layer entropy, IP packet entropy and query name entropy properties of each DNS packet were used. Support Vector Machine (SVM), Naive Bayes (NB) and J48 algorithms were used as the methods and comparative analysis results of these algorithms were presented. When the results were analyzed, it was concluded that SVM was more successful than other methods with an f-measure ratio of 83%.

Almusawi and Amintoosi [12] suggested a multi-label SVM to distinguish between tunnel types along with DNS tunnel detection. A tunnel data set consisting of 4 different protocols (FTP, HTTP, HTTPS, and POP3) and 530 samples were used in the study. To distinguish between the tunnel types, 7 different properties were used, including DNS length, IP length and entropy information. Additionally, the suggested multi-label SVM method was compared with the multilabel Bayesian classifier, and it was observed that the suggested method yielded more successful results with an f-measure ratio of 80%.

Liu et al. [13] also implemented a classification-based DNS tunnel detection mechanism. The implemented system performed the detection process by taking the time intervals, DNS record types and DNS query lengths into consideration along with the differences in character distribution between the normal data and tunneled data. SVM, Decision Tree and Logistical Regression were used as classification methods in the study, and SVM was concluded to yield better results.

Buczak et al. [14] also suggested a classifier-based method for DNS Tunneling detection. In the study, tunnel dataset and legal DNS data were collected via Iodine DNSCat2 and Cobalt Strike. Then, 16 properties of the network data were addressed and subjected to the Random Forest method. In the study, it was stated that the tunnel types collected via each tool were detected by 95%.

Cambiaso et al. [15] proposed a new DNS tunneling system to examine the internal structure of the DNS tunneling system as opposed to detection. The approach they suggested was based on the extraction of statistical properties, such as average of network traffic, standard deviation, and arrival times of DNS queries and responses. For this aim, they used the Principal Component Analysis and Mutual Information methods.

Homem et al. [16] designed a system that aimed to estimate the protocol of the data tunneled by DNS traffic. In this study, the internal structure of DNS tunneling techniques were analyzed and the protocol type was estimated using the entropy of packet bytes. The data set that consisted of 20 samples was collected via Iodine in the study. As a result of the study, a success rate of 75% was achieved.

Nadler et al. [17] proposed a machine learning-based method to detect low-efficiency data leakage in addition to DNS Tunneling detection. In the study, first, the DNS traffic was collected and transformed into a feature vector corresponding to the domain. Then, a pre-trained one-class classifier was used to identify the domains that perform DNS Tunnel detection. In the study, the domain names that perform tunnel detection were blacklisted. The study was evaluated, and its accuracy was tested by using two DNS Tunneling tools (Iodine and Dns2tcp).

In another study on DNS tunneling detection, Aiello et al. [18] used K-Means and Logic Learning Machine (LLM) methods. In the study, first, a clustering method (K-Means) was used to reduce the frequency of anomaly. Then, the data obtained from the cluster, rule extraction from the decision tree and the classification method LLM algorithm were used to detect DNS Tunneling. The study concluded successful results even though the behavior of the test data used in the study differed from the training data.

Bubnov [19] performed DNS Tunneling detection by using a classifier-based artificial neural network. In the study, the type, count, name length, name entropy, data length, data entropy properties of the packets were analyzed and DNS Tunneling was detected through the three-layer feedforward neural network model. A success rate of 83% was achieved in the study.

Van Thuan Do et al. [20] suggested the One-Class Support Vector Machine (OCSVM) and K-Means algorithms for DNS tunneling detection on mobile networks. The proposed machine learning-based methods could detect DNS tunneling in the mobile network by 96% by using features such as time, target, protocol and length of the DNS query.

Ahmed et al. [21] developed a real-time system to detect data leakage and tunneling over DNS. In the study, attribute extraction was performed on the DNS data collected from two organizations, and then abnormalities in DNS queries were detected on a live network using the machine learning method.

Although there are studies in the literature that detect DNS tunneling in real-time, it is clear that there are no systems designed to block DNS tunneling in real-time. In order to prevent situations as the stealing of 25K credit card information of Sally Beauty customers and 56M credit card information of Home Depot customers mentioned in [21], it is of great importance not only to detect DNS tunneling but also to block DNS tunneling attacks in real-time. Our aim in the study is to develop a system that blocks the attacks over DNS traffic on a network in real-time by using a DFF-based decision mechanism. Developing such a system will eliminate the requirement for an operator to continuously monitor network traffic to prevent tunnel attacks over DNS and will automate this process. The DFF method used in the decision mechanism of the system achieved a higher success rate in blocking threats, compared to the studies proposed in the literature.

III. MATERIALS AND METHODS

In this section, the phases for the offline preparation of the decision mechanism will be described. First, the collection of the data for the decision mechanism, and then the process of converting these data into their features will be described. DFF-based architecture was used for decision making in the study. Detailed information on the structure of this architecture and training results will be provided.

A. Dataset Collection

Alexa top 1 million sites [22] list were used to obtain the legal DNS data. Access to these sites was provided by coding a web browser application, and the legal DNS data was obtained by monitoring the network by using the wireshark and tcpdump applications.

To obtain DNS tunnel data, Iodine, Dns2cat and Dns2tcp DNS tunneling tools were used. These tools were run on the Ubuntu operating system on the same computer and the same network. Again, as in the legal DNS data, tunneled DNS data is collected by sending a request to the Alexa top 1 million sites through a shell script and monitoring the network with the tunnel through the wireshark and tcpdump applications. The distribution of the legal and tunnel data is presented in Table I.

TABLE I. DISTRIBUTION OF THE DATASET

Legal DNS Data (Packets)	Tunnel DNS Data (Packets)		
	Iodine	DnsCat2	Dns2Tcp
	164692	127496	93921
Total : 412587	Total: 386109		

B. Preprocessing and Feature Extraction

The preprocessing phase of the data we collected must be completed before modeling the data with deep learning. Data preprocessing is one of the main stages that directly influence the learning ability of the model in deep learning. Increasing the quality of the data by preprocessing helps in reducing the training period and preventing poor classification performances [23]. The processes performed in the pre-processing stage of our study are described below:

- Each DNS packet was merged into a single line. Rows of legal and tunnel data were merged and distributed randomly, thus, a dataset comprised of both legal and tunneled data was generated.
- The features of the DNS data contained in a single row were parsed into columns.
- Non-numerical features and values in DNS data are encoded and labeled.
- After the encoding process, the properties that we did not use in tunnel detection were removed to get rid of noise in the data. Since one of the properties used in tunnel detection is the entropy information of the query name, these values were also calculated and included in the data set.
- In the final stage of the pre-processing, feature scaling was performed. Feature scaling is an important step in preprocessing to standardize the values that the features can take [24]. In our study, the feature scaling process was performed on the properties used for tunnel detection to determine the effectiveness of the change in the values of the properties. Thus, it was confirmed by feature scaling that the use of IP length, Query Name length and Query Name entropy information in DNS packets is suitable for the detection of DNS tunneling.

C. Deep Feedforward (DFF) Neural Network

Deep Feedforward (DFF) Neural Network, which is one of the most common supervised learning models, was used as the decision mechanism in the study. Several hidden layers are combined in a chain to achieve the desired output in DFF. These layers consist of neurons or units with an input vector, whose activation is calculated by the formula in (1).

$$a_{\theta}(x) = g(\theta^{T} x) \qquad (1)$$

In (1), θ is a vector of n weights and g activation function.

The activation of unit k in layer m when input n is given is calculated as shown in (2).

$$a_{k}^{m} = g(\Theta_{k0}^{m-1} a_{0}^{m-1} + \Theta_{k1}^{m-1} a_{1}^{m-1} + ... + \Theta_{kn}^{m-1} a_{n}^{m-1}) \qquad (2)$$

Weights are used to optimize a cost function that shows the similarity between the desired outputs and actual outputs. During the learning process in DFF, weights of each unit are updated using backpropagation [25].

Training was performed by using 3 different network architectures and their performances were tested within the scope of the study (Table II). In all architectures, the input layer was made up of of 3 cells: IP length (I1), Query Name length (I2) and Query Name entropy (I3) in DNS packets. Shannon equation was used for calculating the entropy value of the Query Name in (3).

$$H(X) = -\sum_{i=1}^{k} P(x_i)\log_2 P(x_i) \qquad (3)$$

TABLE II. DFF TOPOLOGIES USED IN THE STUDY

	Topology	Activation Function
Architecture-1	3-6-5-1	ReLU – ReLU-Sigmoid
Architecture-2	3-18-20-5-1	Leaky-ReLU- ReLU-ReLU-Sigmoid
Architecture-3	3-18-20-11-7-17-1	Leaky-ReLU- ReLU-ReLU- ReLU Sigmoid

Architecture-1 was made up of 2 hidden layers. The first hidden layer was comprised of 6 and the second hidden layer was comprised of 5 cells. ReLU activation function was preferred in the hidden layers. Architecture-2 was made up of 3 hidden layers. The first hidden layer was comprised of 18, the second hidden layer was comprised of 20 and the last hidden layer was comprised of 5 cells. Leaky ReLU, ReLU, ReLU activation functions were used respectively in the hidden layers.

Architecture-3 was made up of 5 hidden layers (Fig. 3). The first hidden layer was made up of 18, the second hidden layer was made up of 20, the third hidden layer was made up of 11, the fourth hidden layer was made up of 7 and the last hidden layer was made up of 17 cells. The number of cells being higher in the first two hidden layers, relatively less in the third and fourth hidden layers and higher again in the last hidden layer is named as dense-sparse-dense training flow. It is observed that the results obtained in classification problems are more successful when this method is used [27]. Leaky ReLU was preferred as the activation function for the first hidden layer, whereas ReLU was preferred for the other hidden layers. Leaky ReLU yields better results than ReLU, however the transaction load and time is much higher than ReLU. Therefore, using Leaky ReLU only in the first layer was observed to be the best option in terms of network optimization. In our DFF network, a dropout ratio of 25% was used in the second hidden layer and a dropout ratio of 20% was used in the fifth hidden layer. Dropout method enables a certain part of the learned weights to be forgotten and to be learned again in the next repetition. When used in hidden layers, there is a visible increase in the speed and accuracy of learning [28].

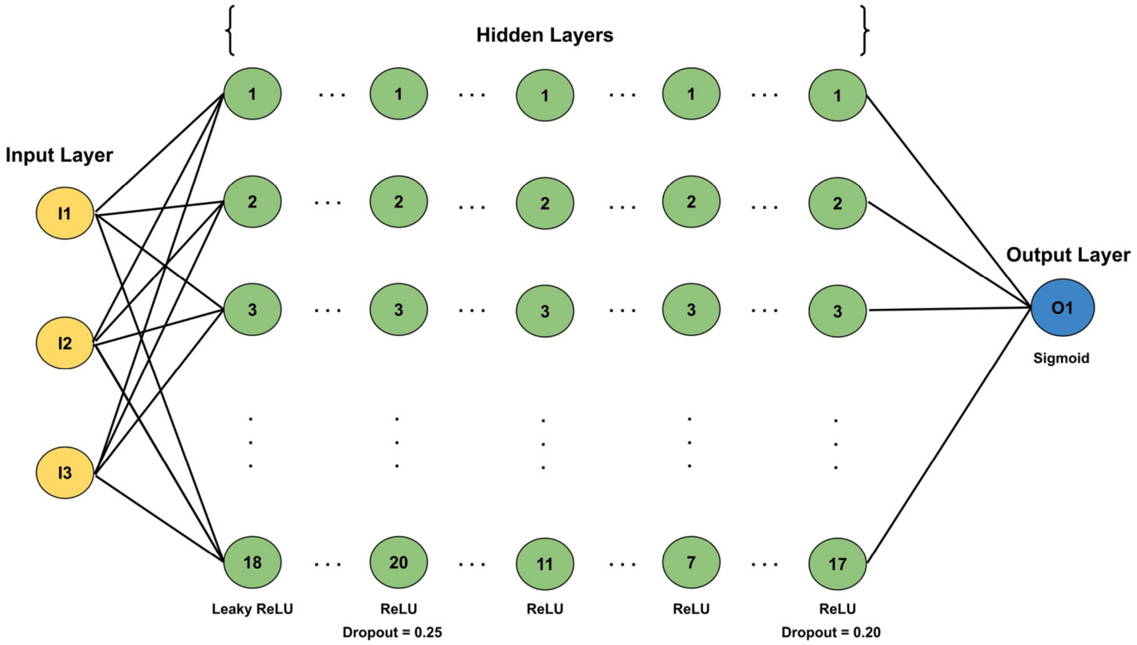

Figure 3. Proposed DFF neural network structure

Since DNS tunnel detection is a classification problem, the output layer consists of a single cell. (No tunnel = 0, Tunnel = 1). The sigmoid activation function takes a value between 0 - 1 depending on the input value. The output layer in our study also takes the values 0 and 1. The highest value in the ratio of success in this layer was obtained by the sigmoid activation function. For these reasons, the sigmoid function was found to be suitable for the output layer.

50,000 data randomly selected from each of the previously collected legal and tunneled DNS data were used in network training. The remaining data were used for real-time testing of the network after network training. The same training and test data were used for each architecture. Table III shows the average achievements after 50 training sessions. As shown in Table III, Architecture-3 made the right decisions for all data.

TABLE III. TRAINING AND TEST DATA IN DIFFERENT ARCHITECTURES AND DFF ACHIEVEMENTS

	Training (%)	Test (%)
Architecture-1	93,12	89,61
Architecture-2	96.44	94,98
Architecture-3	100	100

Optimization algorithms are used in deep learning to determine the training speed and final estimation performance of the models. The optimization algorithms used are Gradient Descent (1st Derivative) based methods that move step by step towards the minimum point. Picking a large step size (learning coefficient) may cause a disadvantage of not reaching the minimum point, whereas picking a small size may cause the disadvantage of taking too long to reach the minimum point in terms of time. Therefore, the selection of optimization algorithm is one of the important steps in deep learning [29-30]. The optimization algorithms in Keras library are compared for Architecture-3 in our study and the accuracy rates are given in Table IV. As seen in Table IV, although there was not a high difference in the success rates among optimization algorithms, the most successful results were obtained with Adamax. Batch size was determined to be 32 and epoch number was 50.

TABLE IV. ACCURACY RATES WITH DIFFERENT OPTIMIZERS

Optimizer	Accuracy (%)
Sgd	No learning
Rmsprop	No learning
Adadelta	95,9034
Nadam	97,8478
Adagrad	98,8958
Adam	99,1203
Adamax	100

IV. PRELIMINARY PREPARING FOR REAL-TIME TEST SETUP

The process of blocking DNS packets received via tunnel after the DNS tunneling detection phase is described in this section. Section A explains the process of capturing the DNS packets conveyed over the live network and transferring them to the decision mechanism described in Section 3. Section B provides the details of resolution and blocking of the incoming DNS packets in real time.

A. Capturing DNS Packets and Sending to System

In order for the system to operate in real-time, it is first necessary to capture DNS packets on a live network. Netfilter / Iptables framework [31] provided by the Linux kernel was used for packet capturing. Iptables consists of 3 basic structures: tables, chains and targets. Tables are the most important part of the packet processing system. It consists of 3 parts: Filter (Input, Output, Forward), Mangle (Prerouting, Postrouting, Input, Output, Forward) and NAT (Prerouting, Postrouting, Output). Filter is used if the packets will be processed in a standard manner and mangle is used if various headers such as the TCP header will be changed. NAT is used to rewrite the source or destination of the packets. Chains are a list of rules in a table and where traffic can be interrupted or triggered. The goals determine what happens to the packet in the chain when there is a match with one of the rules. Writing a rule that is suitable with a particular traffic and routing the packet to drop or accept destinations can be given as an example [32].

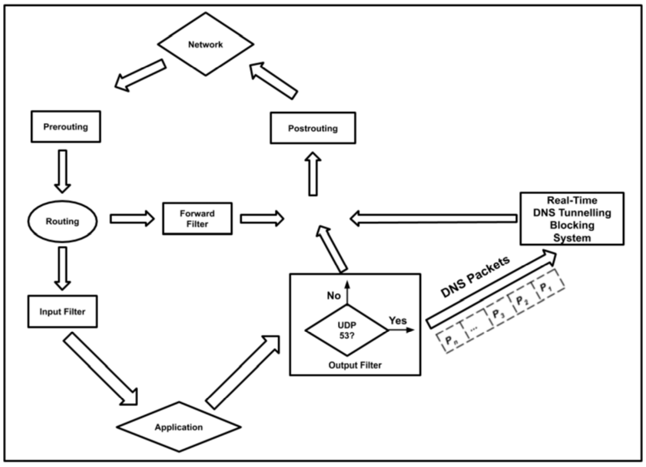

Figure 4. Capturing DNS packets and sending to system

Since the main purpose in our study is to capture DNS packets in network traffic, the process of routing all packets from port 53 of the UDP protocol Real Time DNS Tunneling Blocking System was added to the rules in the output chain of the filter table (Fig. 4).

B.Real-Time DNS Tunneling Blocking System

Packets from Netfilter are in binary format. Since packet decoding is difficult and cumbersome, Scapy library [33] written in Python by Philippe Biondi was used for packet analysis. First, the packet from Netfilter was queued. In the second stage, the OSI network, transport and application layers of the data in Binary format were decomposed for use via Scapy. In the decomposing process, first the IP length data was obtained from the network layer; since the DNS data is transferred via UDP port 53, whether the incoming data is DNS data or not was obtained from the transport layer; and the Query Name was obtained from the application layer.

After obtaining the required information about the packet, the incoming packet was analyzed to check whether it was a DNS request packet or not. If it was not a request packet it was allowed to continue its process. If it is a request packet, the specified properties of the DNS packet were sent to the Deep Learning algorithm. Depending on the response of the algorithm, it was decided whether the packet is tunneled or not. If it was tunneled, it was blocked, if not, it continued its process. The flow chart for the Real-Time DNS Tunneling Blocking System is shown in Fig. 5.

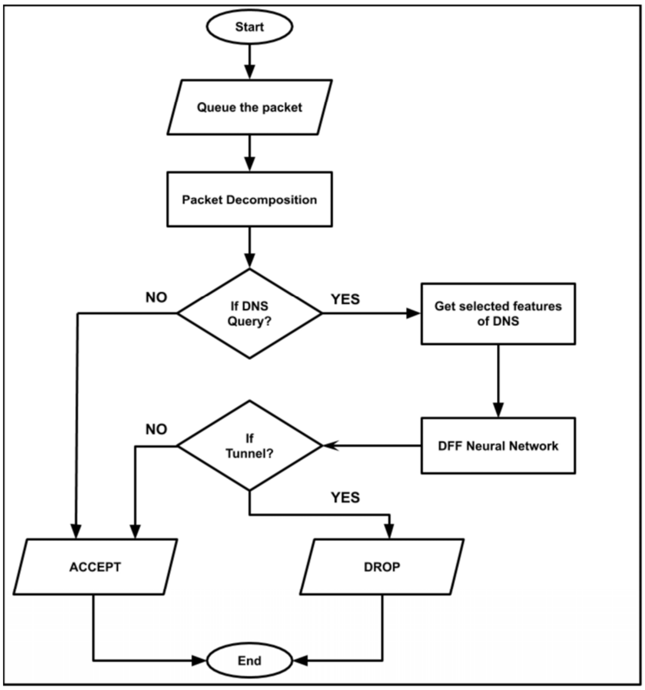

Figure 5. Flow chart of real-time DNS tunneling blocking system

V. EXPERIMENTAL RESULTS

Within the scope of the study, a functional real-time test configuration was implemented for the detection and blocking of DNS tunneling threats (Fig. 6). As seen in Fig. 6, a network was established for the test configuration. Among the 3 devices on the network, Computer-1 sends legal packets to Computer-3 and Computer-2 sends tunneled DNS packets to Computer-3. Computer-1 and Computer-2 are not directly connected to the internet. They access the internet via the network (hotspot) on Computer-3. While Computer-1 makes requests on the web in a standard manner, Computer-2 makes requests through an external tunnel server running over the web. When Computer-1 and Computer-2 start to make requests, the system suggested within the scope of the study installed on Computer-3 is initialized. This computer decides whether the packet is tunneled or not, according to the algorithm's response. If it is tunneled, it is blocked, if not, it continues to process. The properties of the DNS packets from Computer-1 and Computer-2 are shown in Table V. IP length, query name length and query name entropy values of 10% of DNS packets are shown in Fig. 7, Fig. 8 and Fig. 9.

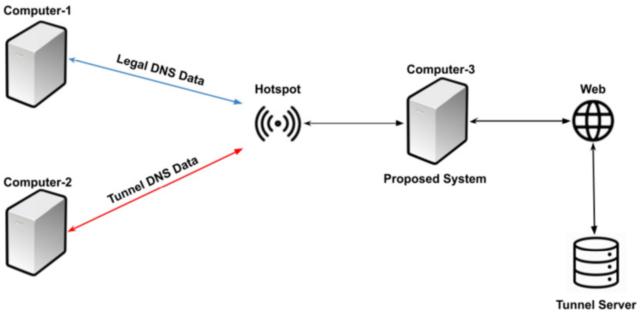

Figure 6. Test scheme

TABLE V. REAL-TIME TEST DATASET SUMMARY

Legal DNS packets	390992
Tunnel DNS packets	370548
DNS queries	412214
DNS responses	349326
Outgoing DNS queries	361747
Outgoing DNS queries (only qualified)	347925
Total DNS packets	761540

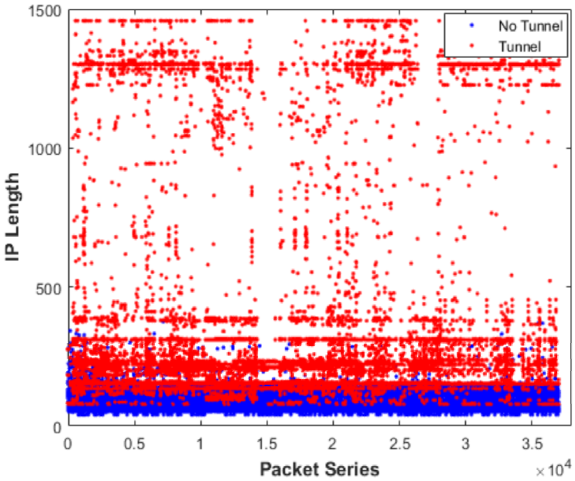

Figure 7. IP length of each DNS packet sent to the live network

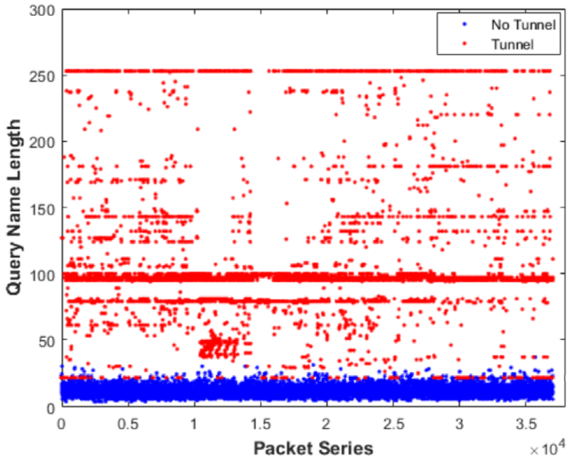

Figure 8. Query name length of each DNS packet sent to the live network

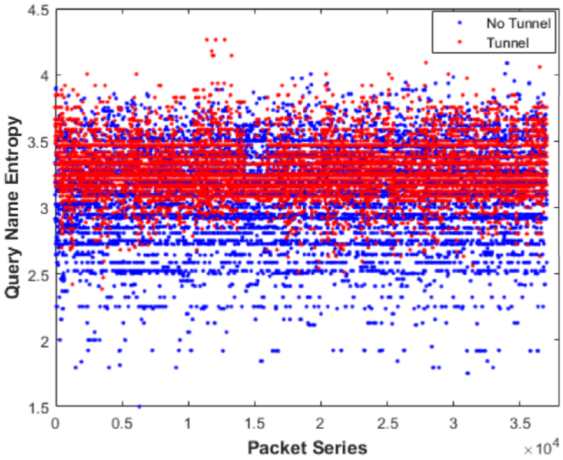

Figure 9. Query name entropy of each DNS packet sent to the live network

In Table VI, the average and standard deviation values of the dataset obtained within the the study's scope are given. When the table is examined, it is seen that the average and standard deviation values of legal data and tunnel data are significantly different. Thus, it is understood that tunnel data can be separated from legal data by using the selected features.

TABLE VI. STATISTICAL ANALYSIS OF FEATURES IN REAL-TIME TEST DATASET

	IP Length		**Query Name Length**		**Query Name Entropy**	
	Legal Data	Tunnel Data	Legal Data	Tunnel Data	Legal Data	Tunnel Data
Average	80.15	416.24	12.05	112.66	3.08	3.31
Standard Deviation	90.94	450.89	3.79	59.14	0.43	0.16

Performance metric values of the proposed system were calculated by in (4-7) and shown in Table VII.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

$$Precision = \frac{TP}{TP + FP} \tag{5}$$

$$Recall = \frac{TP}{TP + FN} \tag{6}$$

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{7}$$

TABLE VII. EVALUATION METRICS

Metric	**Value**
Accuracy	0,9991
Precision	0,9998
Recall	0,9984
F1 Score	0,9991

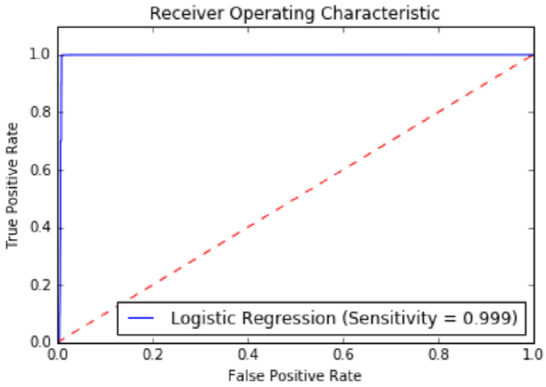

Figure 10. ROC curve of proposed DNS blocking system

Accuracy in Table VII refers to the ratio of the total number correctly detected legal and tunneled DNS packets, to the total number of DNS packets. Precision is obtained by the ratio of the correctly detected number of tunneled packets, to the number of correctly and incorrectly detected tunneled packets. Recall refers to the ratio of correctly detected tunneled packets to the sum of correctly detected tunneled and incorrectly detected legal packets. Whereas F1 Score is calculated by taking the harmonic average of the precision and recall values, to be able to evaluate these two values as a single value. Besides, ROC curve of the proposed system is shown in Fig. 10. It has been stated in the literature that the ROC curve of a near-perfect system should have a curve from vertical (0,0) to (0,1) and horizontally (1,1) [34]. As shown from the figure, the proposed system has a curve very close to the upper left corner. It is clear from the ROC curve that the system has a high accuracy rate.

TABLE VIII. ACCURACY VALUES OF THE PROPOSED METHOD DEPENDING ON THE |NUMBER OF FEATURES

	7 Feature	**3 Feature**
Feature Name	ip length	ip_length
	query_name_length	query_name_length
	query_name_entropy	query_name_entropy
	ip_flag_mf	
	ip_flag_rb	
	ip_flag_z	
	ip_header_length	
Accuracy	99.32%	99.91%
Training Time	≅ 7200 sec.	108.07 sec.

For the proposed method, in Table VIII, the accuracy values calculated separately using 7 features and 3 features in the dataset are shown. The 7 features used in the first stage were selected among the features commonly used in DNS tunneling. The 3 features used in the other stage represent the properties obtained after the feature reduction process specific on the dataset. As can be seen from the results, the most distinguishing features on the dataset were obtained with the feature reduction process. Thus, the model has been effectively trained with both higher accuracy values and shorter processing times. It is understood from the results in the table that the use of low-impact features in the dataset does not have a positive effect on the success of the method.

The average time to determine whether a DNS query is tunneled and to block the tunneled packet is shown in Table IX. The values in Table IX were calculated on a computer configured with 4 CPU cores and 8 GB of memory.

TABLE IX. AVERAGE TIME COMPLEXITY OF OUR SYSTEM

DNS tunneling detection	0.614 ms
DNS tunneling blocking	0.309 ms
Total time per each DNS packet	0.923 ms

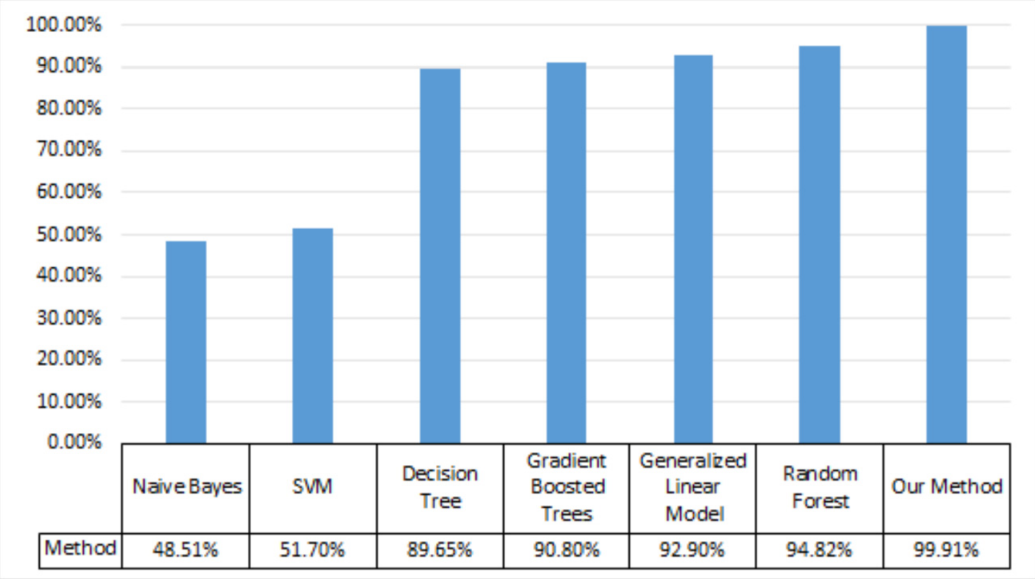

Figure 11. Comparison of the classification accuracy with other methods

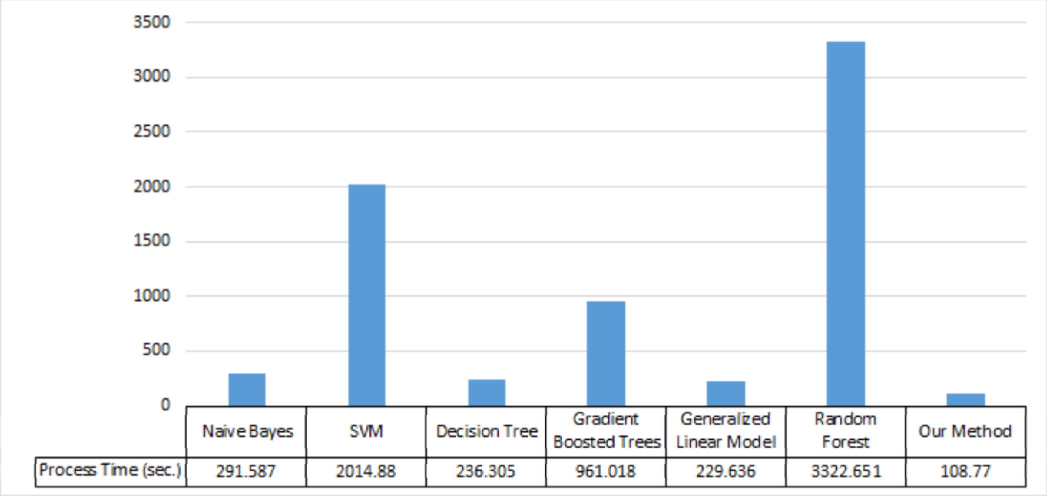

Figure 12. Comparison of the training time of proposed method with other methods

The comparison of the accuracy values with the methods generally used in DNS tunneling are shown in Fig. 11. The proposed method is compared with the Naive Bayes, SVM, Decision Tree (DT), Gradient Boosted Trees (GBM), Generalized Linear Model (GLM) and Random Forest methods. The most successful result among traditional machine learning methods is obtained by Random Forest (94.82%), while Naive Bayes obtained 48.51%, SVM obtained 51.70%, DT obtained 89.65%, GBT obtained 90.80% and GLM method obtained 92.90% accuracy. As can be seen from Table VIII, the 3 features selected from the dataset have a high effect on the classification process.

However, as can be seen in Fig. 11, even if operations are performed with these three features in classical machine learning methods, more successful results have been obtained with the proposed model.

In addition to the results in Fig. 11, the comparison of the proposed method with traditional machine learning methods in terms of processing time is given in Fig. 12. In general, it is seen that the process time is higher in methods which have a success rate more than 90 percent. When the proposed model is compared with the models which have high success, it is seen that our model has higher success and completed the training process in a shorter and reasonable time in terms of processing time.

TABLE X. COMPARISON OF RESULTS WITH STATE-OF-ART

Author	**Proposed Method**	**Accuracy (%)**	**F-Measure (%)**	**Real-Time Detection**	**Real-Time Blocking**
Buczak et. al. [14]	Random Forest	95.40	-	No	No
Homem et.al. [16]	Fuzzy C-Means	96.00		No	No
Ahmed et al. [21]	Isolation Forest Algorithm	98.90	-	Yes	No
Aiello et al. [18]	K-Means+ LLM	90.70	-	No	No
Bubnov [19]	Feed Forward Neural Network	83.00	-	No	No
Sammour et al. [3]	SVM	-	83.00	No	No
Almusawi and Amintoosi [12]	Kernel SVM	-	80.00	No	No

Table X shows the comparison results of the proposed model with similar studies in the literature. These studies obtained the DNS data via Iodine tool as in our study. As can be seen in the table, comparisons were made with the accuracy values for [14], [16], [18-19], and [21]. Also F-Measure values were used for [3] and [12] (because accuracy values are not given in the results of these studies). While the tunneling detection is done in real-time in study [21], it is seen that a system that will perform detection or prevention in real-time is not recommended in other studies. As can be seen in the table, with the proposed model, a high accuracy value has been obtained in blocking DNS tunneling in real time compared to the studies in the literature.

VI. CONCLUSION

In this study, a deep network-based system that blocks tunneling attacks over DNS in real-time is proposed. At the first phase of the study, legal and tunneled data were obtained by using the data from Alexa top 1 million sites. Then, networks with 3 different topologies were trained by using these data. At the training stage for the 3 different topologies, 89%, 95% and 100% success rates were obtained. After the topology with the highest success rate was integrated to our system, the performance of our system was tested on a live network. Throughout the tests, the system was integrated to the network to block incoming threats in real time. Statistics on the number of threats and the ratio of threats that were blocked by the system was produced by analyzes.

It can be seen in Table VII that DNS tunneling in a live network was detected with 99.91% Accuracy, 99.98% Precision, 99.84% Recall and 99.91% F1 Score. These rates confirm that our system has a high success rate in blocking tunneling threats in DNS traffic.

As seen in Table IX, the average time for our system to decide whether a packet is tunneled is 0.614 ms and the blocking time is 0.309 ms. Our system, which makes a decision in a total of 0.923 ms per DNS query, showed that we can respond to approximately 1080 DNS queries per second. When we consider that the sample campus network specified in study [21] responds to maximum 800 DNS queries per second, it can be seen that our system can respond sufficiently in networks with a DNS traffic similar to campus networks.

REFERENCES

[1] T. K. Skow, “Protection against DNS tunneling abuses on mobile networks,” MSc Thesis, Norwegian University of Science and Technology, 2016

[2] R. Chandramouli and S. Rose, “Secure domain name system (DNS) deployment guide,” National Institute of Standards and Technology Special Publication, 2013. doi:10.6028/NIST.SP.800-81-2

[3] M. Sammour, B. Hussin and F. I. Othman, “Comparative analysis for detecting dns tunneling using machine learning techniques,” International Journal of Applied Engineering Research, vol. 12, no. 22, pp. 12762-12766, 2017

[4] H. Önal, “DNS Tünelleme,” http://www.enderunix.org/docs/dns_tunelleme.pdf (Accessed on June 10, 2020)

[5] S. Hangal, S. Narayanan, N. Chandra and S. Chakravorty, “IODINE: a tool to automatically infer dynamic invariants for hardware designs,” in Proc. 42nd Design Automation Conference, 2005, Anaheim, CA, 2005, pp. 775-778. doi:10.1109/DAC.2005.193920

[6] S. Yassine, J. Khalife, M. Chamoun et al., “A Survey of DNS Tunnelling Detection Techniques Using Machine Learning,” in Proc. 1st International Conference on Big Data and Cyber-Security Intelligence, Hadath, Lebanon, 2018, pp. 63-66

[7] M. Al-kasassbeh, T. Khairallah, “Winning tactics with DNS tunneling,” Network Security, vol. 2019, no. 12, pp. 12-19, 2019. doi:10.1016/S1353-4858(19)30144-8

[8] A. Merlo, G. Papaleo, S. Veneziano, et al., “Comparative performance evaluation of DNS tunneling tools,” in Proc. Computational Intelligence in Security for Information Systems, Torremolinos-Málaga, Spain, 2011, pp. 84-91. doi:10.1007/978-3-642-21323-6_11

[9] G. Farnham and A. Atlasis, “Detecting DNS tunneling. SANS Institute InfoSec Reading Room,” https://www.sans.org/reading-room/whitepapers/dns/detecting-dns-tunneling-34152 (Accessed on June 10, 2020)

[10] M. Aiello, M. Mongelli and G. Papaleo, “Basic classifiers for DNS tunneling detection,” in Proc. IEEE Symposium on Computers and Communications, Split, Croatia, 2013, pp. 880-885. doi:10.1109/ISCC.2013.6755060

[11] M. Aiello, M. Mongelli and G. Papaleo, “DNS tunneling detection through statistical fingerprints of protocol messages and machine learning,” International Journal of Communication Systems, vol. 28, no. 14, pp. 1987-2002, 2015. doi:10.1002/dac.2836

[12] A. Almusawi and H. Amintoosi, “DNS Tunneling detection method based on multilabel support vector machine,” Security and Communication Networks, vol. 2018, 2018. doi:10.1155/2018/6137098

[13] J. Liu, S. Li and Y. Zhang, et al., “Detecting DNS tunnel through binary-classification based on behavior features,” in Proc. IEEE Trustcom/BigDataSE/ICESS, Sydney, Australia, 2017, pp. 339-346. doi:10.1109/Trustcom/BigDataSE/ICESS.2017.256

[14] A. L. Buczak, P. A. Hanke, G. J. Cancro, et al., “Detection of tunnels in PCAP data by random forests,” in Proc. 11th Annual Cyber and

Information Security Research Conference, USA, 2016, pp. 1-4. doi:10.1145/2897795.2897804

[15] E. Cambiaso, M. Aiello, M. Mongelli, et al., “Feature transformation and Mutual Information for DNS tunneling analysis,” in Proc. Eighth International Conference on Ubiquitous and Future Networks, Vienna, Austria, 2016, pp. 957-959. doi:10.1109/ICUFN.2016.7536939

[16] I. Homem, P. Papapetrou and S. Dosis, “Entropy-based prediction of network protocols in the forensic analysis of dns tunnels,” arXiv, 2017. arXiv preprint arXiv:1709.06363

[17] A. Nadler, A. Aminov and A. Shabtai, “Detection of malicious and low throughput data exfiltration over the DNS protocol,” Computers & Security, vol. 80, pp. 36-53, 2019. doi:10.1016/j.cose.2018.09.006

[18] M. Aiello, M. Mongelli, M. Muselli et al., “Unsupervised learning and rule extraction for Domain Name Server tunneling detection,” Internet Technology Letters, vol. 2, no. 2, pp. 1-6, 2019. doi:10.1002/itl2.85

[19] Y. Bubnov, “DNS tunneling detection using feedforward neural network,” European Journal of Engineering Research and Science, vol. 3, no. 11, pp. 16-19, 2018. doi:10.24018/ejers.2018.3.11.963

[20] T. V. Thuan, P. Engelstad and B. Feng, “Detection of DNS tunneling in mobile networks using machine learning,” in Proc. International Conference on Information Science and Applications, Macau, China, 2017, pp. 221-230. doi:10.1007/978-981-10-4154-9_26

[21] J. Ahmed, H. Gharakheili, Q. Raza, et al., “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. doi:10.1109/TNSM.2019.2940735

[22] Alexa, “The top 500 sites on the web,” https://www.alexa.com/topsites (Accessed on February 17, 2019)

[23] J. Huang, Y. F. Li and M. Xie, “An empirical analysis of data preprocessing for machine learning-based software cost estimation,” Information and software Technology, vol. 67, pp. 108-127, 2015. doi:10.1016/j.infsof.2015.07.004

[24] D. Bollegala, “Dynamic feature scaling for online learning of binary classifiers,” Knowledge-Based Systems, vol. 129, pp. 97-105, 2017. doi:10.1016/j.knosys.2017.05.010

[25] A. Carrio, C. Sampedro, A. Rodriguez-Ramos, et al., “A review of deep learning methods and applications for unmanned aerial vehicles,” Journal of Sensors, vol. 2017, pp. 1-13, 2017. doi:10.1155/2017/3296874

[26] J. Lin, “Divergence measures based on the Shannon entropy,” IEEE Transactions on Information Theory, vol. 37, no. 1, pp. 145-151, 1991. doi:10.1109/18.61115

[27] S. Han, J. Pool, S. Narang, et al.,”Dsd: Dense-sparse-dense training for deep neural networks,” in Proc. International Conference on Learning Representations (ICLR), France, 2017, pp 1-13.

[28] G. E. Dahl, T. N. Sainath and G. E. Hinton, “Improving deep neural networks for LVCSR using rectified linear units and dropout,” in Proc. IEEE International Conference On Acoustics, Speech And Signal Processing, British Columbia, Canada, 2013, pp. 8609-8613. doi:10.1109/ICASSP.2013.6639346

[29] D. Choi, C. J. Shallue, Z. Nado, et al., “On empirical comparisons of optimizers for deep learning,” 2019. arXiv preprint:1910.05446.

[30] E. Seyyarer, T. Uçkan, C. Hark, et al., “Applications and comparisons of optimization algorithms used in convolutional neural networks,” in Proc. International Artificial Intelligence and Data Processing Symposium, Malatya, Turkey, 2019, pp. 1-6. doi:10.1109/IDAP.2019.8875929

[31] B. Wang, K. Lu and P. Chang, “Design and implementation of Linux firewall based on the frame of Netfilter/Iptable,” in Proc. 11th International Conference on Computer Science & Education, Japan, 2016, pp. 949-953. doi:10.1109/ICCSE.2016.7581711

[32] L. F. Xuan and P. F. Wu, “The optimization and implementation of iptables rules set on linux,” in Proc. 2nd International Conference on Information Science and Control Engineering, USA, 2015, pp. 988-991. doi:10.1109/ICISCE.2015.223

[33] R. Rohith, M. Moharir, and G. Shobha, “SCAPY-A powerful interactive packet manipulation program,” in Proc. International Conference on Networking, Embedded and Wireless Systems, India, 2018, pp. 1-5. doi:10.1109/ICNEWS.2018.8903954

[34] L. Tomak and Y. Bek, “İşlem karakteristik eğrisi analizi ve eğri altında kalan alanların karşılaştırılması,” Journal of Experimental and Clinical Medicine, vol. 27, no. 2, pp. 58-65, 2009. doi:10.5835/jecm.omu.27.02.008