Information Processing and Management 59 (2022) 102930

Contents lists available at ScienceDirect

Information Processing and Management

journal homepage: www.elsevier.com/locate/ipm

DNS tunnels detection via DNS-images

Gianni D'Angelo, Arcangelo Castiglione *, Francesco Palmieri

Department of Computer Science, University of Salerno, Via Giovanni Paolo II, 132, I-84084, Fisciano (SA), Italy

ARTICLE INFO

Keywords:
DNS security
DNS tunneling
Data exfiltration
Anomaly detection
Classification
Convolutional neural network

ABSTRACT

DNS tunneling is a typical attack adopted by cyber-criminals to compromise victims' devices, steal sensitive data, or perform fraudulent actions against third parties without their knowledge. The fraudulent traffic is encapsulated into DNS queries to evade intrusion detection. Unfortunately, traditional defense systems based on Deep Packet Inspection cannot always detect such traffic. As a result, DNS tunneling is one problem that has worried the cybersecurity community over the past decade.

In this paper, we propose a robust and reliable Deep Learning-based DNS tunneling detection approach to mine valuable insight from DNS query payloads. More precisely, several features are first extracted by the DNS flow, and then they are arranged as bi-dimensional images. A *Convolutional Neural Network* is used to automatically and adaptively learn spatial hierarchies of features to be used in a fully connected neural network for traffic classification. The proposed approach may result in an extremely interesting task in predictive security approaches to attack detection.

The effectiveness of the proposal is evaluated in several experiments using a real-world traffic dataset. The obtained results show that our approach achieves 99.99% of accuracy and performs better than state-of-the-art solutions.

1. Introduction

The DNS (Domain Name System) service was originally born for translating human-friendly hostnames into IP addresses. Nevertheless, its underlying protocol can encapsulate and transfer any kind of data, such as other protocols or binary files. Indeed, a DNS query can be used as a container of data to be transferred, effectively creating a kind of invisible network traffic.

Cybercriminals know that DNS is a basic protocol that is widely trusted. On the other hand, many organizations do not monitor the DNS traffic for malicious activity. In addition, DNS is a must-pass infrastructural service that should traverse, almost unaffected, any kind of security enforcement point (screening routers, firewalls, etc.). As a result, many very effective DNS-based attacks can be carried out against company networks to compromise victims' devices, steal sensitive data, or perform fraudulent actions. For example, gaining free access to the Internet from private paid networks in airports, trains, or hotels by tunneling HTTP traffic into DNS queries. Also, DNS tunneling is typically used to create "Command & Control" infrastructures to carry out *distributed denial of service (DDoS)* attacks (Xu, Butler, Saha, & Yao, 2013), install malware (D'Angelo, Palmieri, Robustelli, & Castiglione, 2021), steal private data (Ogiela & Ogiela, 2020; Ogiela, Ogiela, & Ogiela, 2016), and more generally to perpetuate other fraudulent actions.

Although the performance of DNS-based overlay connections is very poor (about 1 Mbps and 150 ms of latency), they are very attractive for hackers because, as previously mentioned, such traffic is very often neither controlled nor filtered by firewalls, allowing it to pass freely (Schmid, 2021).

* Corresponding author.
E-mail addresses: giadangelo@unisa.it (G. D'Angelo), arcastiglione@unisa.it (A. Castiglione), fpalmieri@unisa.it (F. Palmieri).

https://doi.org/10.1016/j.ipm.2022.102930
Received 29 October 2021; Received in revised form 13 February 2022; Accepted 9 March 2022
Available online 25 March 2022

For these reasons, the automatic detection of anomalous DNS traffic is a hot and deeply-felt research field (Torabi, Boukhtouta, Assi, & Debbabi, 2018; Wang et al., 2021), mainly for implementing *predictive security* approaches aiming at countering stealth attacks and security breaches. We remark that *predictive intelligence* could play a key role in this context. In fact, since the behaviors and techniques used to encapsulate information within DNS queries change frequently, it is necessary to integrate methodologies and tools for monitoring, analyzing, and predicting these behaviors and techniques to adapt continuously to the detection model. Furthermore, predictive intelligence for the automatic detection of DNS tunnels is even more critical if we consider the large amount of data on which a detection algorithm operates. In fact, since these algorithms often work using a set of features that usually grows with the complexity of the data to be analyzed, predictive intelligence techniques can allow defining detection models able to operate more quickly and efficiently, taking advantage of a reduced number of features (Ogiela & Ogiela, 2021).

Actually, there are two main DNS tunneling detection techniques: *payload analysis*, where the DNS query content is analyzed, and *traffic analysis*, where network traffic is inspected. However, conventional detection mechanisms, based purely on a statistical analysis of network traffic or packet content (Dusi, Crotti, Gringoli, & Salgarelli, 2009) are not always effective and can be easily circumvented.

In the last decade, a valid alternative to the statistical-based approaches for DNS tunneling detection is represented by the usage of techniques based on Deep Learning (Zhang, Yang, Yu, & Ma, 2019), which has proven to be very effective for the classification of DNS queries in a more accurate and precise manner. Deep Learning, Machine Learning, and more in general Artificial Intelligence-based techniques have been increasingly applied in many heterogeneous contexts (D'Angelo & Palmieri, 2020a, 2020b, 2021; D'Angelo, Tipaldi, Palmieri, & Glielmo, 2019), such as image recognition, sentiment analysis, product and service recommendation systems, spam filtering, robot control systems, and human language processing. Again, recently such techniques are being applied more and more to the medical and industrial fields. Their popularity is also increasing thanks to their ability to elicit useful knowledge from the large amounts of data currently produced by many consumer electronic devices.

Contribution. In this paper, we propose a robust and reliable Deep Learning-based DNS tunneling detection framework able to mine valuable insight from DNS query payloads. More precisely, a number of features are first extracted by the DNS flow. Then they are arranged as bi-dimensional images. A *Convolutional Neural Network* that is a very successful tool for image classification and recognition, by progressively extracting higher and higher-level representations of the image content in terms of textures, edges, shapes, is used to automatically and adaptively learn spatial hierarchies of features to be used in a fully connected neural network for traffic classification.

The resulting framework turns out to be particularly useful in the field of network forensics. Forensic investigators often carry out their activity by focusing on reactive strategies to discover malicious traffic for law enforcement or intrusion detection purposes (Khan, Gani, Wahab, Shiraz, & Ahmad, 2016). These strategies typically concern capturing and storing huge amounts of network traffic as well as manually analyzing such traffic (Sikos, 2020). As we can imagine, these activities, due to the highly volatile and dynamic nature of network traffic, in addition to being time-consuming, require skills and storage resources, so more proactive strategies are needed (Berg & Forsberg, 2019; Homem & Papapetrou, 2017). In particular, the predictive model underlying our framework can identify DNS-tunneled traffic in real-time and with high accuracy by constructing (and periodically refining) knowledge about DNS tunneling techniques from the analysis of traffic data through a sophisticated training process.

Another advantage of our solution is that once the classification model training phase is finished, the classification process can be carried out in real-time on multiple tunneling instances. We emphasize that training is not required for each new instance but can occur through a specific process outside the classification process that runs periodically. Furthermore, due to the modularity of the proposed solution and because many ML predictive models are adaptive, information brought from a never-seen-before sample can be helpful to train the model for classifying future samples.

Through a quite realistic experimental analysis, we have shown that the proposed solution outperforms, in terms of performance, well-established predictive machine learning techniques commonly used in the state-of-the-art for the detection of data sent through DNS tunnels. Therefore, the proposed solution can effectively support the forensic investigation of DNS tunnels, especially in its initial stages.

Organization. The remaining of the paper is organized as follows. In Section 2, we provide the necessary background information about DNS tunneling techniques and convolutional neural networks (CNNs). In Section 3, we explore the state-of-the-art concerning DNS traffic detection techniques. In Section 4, we show the proposed framework, at first by describing the solution as a whole and then highlighting the details of each component of such solution. In Section 5, we show the performance achieved by evaluating the proposed solution. Finally, in Section 6, we draw conclusions and future research directions.

2. Background

In this section, we provide some key concepts used throughout the paper. In particular, we focus on DNS tunneling techniques and Convolutional Neural Networks (CNNs).

2.1. DNS tunneling

As before mentioned, DNS tunneling is a technique used to embed (and eventually encrypt) in DNS queries and responses data from other protocols or programs. The concept of DNS tunneling was firstly created to bypass the *captive portals* of private network infrastructures, which require the creation of an account, typically based on an hourly cost, for surfing the Internet.

2.1.1. How DNS tunneling works

Taking advantage of DNS tunneling techniques typically does not require very complex operations or infrastructures. More precisely, to exploit these techniques, all an attacker needs are (1) a domain or a subdomain, (2) a DNS server that is authoritative for that domain, representing one end of the tunnel, (3) and a compromised client-side machine, representing the other end of the tunnel. We remark that these components are common to all DNS Tunneling tools. Indeed, such tools typically differ based on the resource record used or the type of data encoding only.

More precisely, DNS Tunneling tools make use of the following types of encoding:

- *Base32* for DNS queries. Indeed, since domain names are alphanumeric and case insensitive, we can take advantage of the 26 characters of the alphabet, numbers, and even the '-' symbol. Then, each possible 5 bit permutation is converted into one of the 32 (2^5) available values. The maximum length for each query (*request*) is 255 characters, that is 63 for each subdomain. For example:

label3.label2.label1.example.com

- *Base64* for DNS responses. Indeed, since the 'TXT' resource record (i.e., the most used record) allows us to send an arbitrary text string, we can take advantage of both upper and lower case letters, numbers, and some symbols such as '-' or '+'. Each possible 6 bit permutation is thus converted into one of the 64 available values.

To facilitate the understanding of DNS tunneling techniques, in the following, we provide a simple example:

1. The attacker registers a new domain, e.g., *tunnelDNS.com*;
2. The compromised client sends a DNS query. More precisely, in this query, the data to be sent is encoded in *Base32*. Then, using a resource record of type 'A', the encoded data is added in the hostname:

DNS Query: *gewsopos34f32fgqqe.tunnelDNS.com*

3. The authoritative DNS server for the *tunnelDNS.com* domain answers with:

DNS Response: *ZwsAq5sT43jgcDkhuH6rsp.example.com*

2.1.2. DNS tunneling tools

In the state-of-the-art, there are several tools to perform DNS tunneling. Among them, the following tools are the most popular and effective (Aiello, Merlo, & Papaleo, 2013):

- *Iodine:* released in 2006 and continuously updated until 2016 by Bjorn Andersson and Erik Ekman. It is written in *C* language and is compatible with *Microsoft Windows*, macOS, and Android OS. This tool uses the 'NULL' resource record and allows up to 16 simultaneous connections on the same server.
- *Dnscat:* developed by Ron Bowes. It is available for both Windows-based and Linux-based systems. This tool encodes the request query using *NetBIOS encoding*. Again, it can use resource records of type 'A', 'AAAA', 'CNAME', 'NS', 'TXT', and 'MX'.
- *Ozyman:* written in Perl by Dan Kaminsky and released in 2004. This tool is used to tunnel *SSH* traffic into DNS queries, mainly to transfer files. Queries are Base32-encoded, and the tool uses the Base64-encoded 'TXT' resource record for responses.
- *Heyoka:* it is a Proof of Concept that aims to create a tunnel using DNS queries to steal sensitive data. It allows the source of queries to be spoofed. In this way, it appears that these queries are coming from different servers, thus avoiding suspicion in the traffic observers.

2.1.3. DNS tunneling detection

There are mainly two techniques for finding tunnels that leverage DNS queries to communicate: *Payload Analysis Detection* and *Traffic Analysis Detection*.

Payload Analysis Detection exploits the observation that the content of suspicious DNS queries has some peculiar characteristics compared to normal queries, such as:

- *Size of inbound and outbound queries:* Generally, DNS tunneling tools send as much data as possible both in the request and in the relative response, trying to minimize the number of queries sent. This behavior is caused by the fact that the reception speed is very low, and the latency is very high.
- *Hostname entropy:* Legitimate queries are generally directed towards domains whose name is made up of common language words or, more in general, towards something that does not have a high entropy level. On the other hand, an encoding operation on the query payload tends to increase entropy significantly.
- *Character frequency:* A high frequency of numbers or consecutive consonants can help us understand the potential maliciousness of a DNS query.
- *Use of uncommon resource records:* Paying attention to the use by the client of uncommon resource records, such as the 'TXT' record, can be beneficial for detecting tunneling.
- *Specific signatures:* In some cases, the presence of unusual information in the DNS query headers, added by some tools, may be detected.

Traffic Analysis Detection is based on the identification of anomalies in network traffic. In particular, the following factors can help identify such anomalies:

- *The volume of DNS traffic generated by an IP address:* It is necessary to send many queries to make real communication since the traffic that can be tunneled in every single query is quite limited.
- *The volume of DNS traffic sent towards a domain:* A high volume of traffic sent towards the same domain could be a very discriminating factor for detecting malicious DNS traffic.
- *The server's geographic location:* A large amount of DNS traffic sent towards a part of the world far from ours can help detection.
- *Domain history:* Checking how long the domain to which the queries are directed has been active could help identify tunneling.

2.2. Convolutional neural networks (CNN)

Convolutional neural networks (CNNs) arise from the study of the brain's visual cortex and have been used in image recognition since the 1980s. In recent years, due to the increase in computational power and the amount of training data available, CNNs have achieved high performance on complex visual processing tasks. Subsequently, CNNs have also been used, for voice recognition and *natural language processing (NLP)*, providing satisfactory results.

Hubel and Wiesel performed a series of experiments on cats in 1958 and 1959 (and a few years later on monkeys), providing crucial insights into the visual cortex structure. In particular, they showed that many neurons in the visual cortex have a small local receptive field, which means that these neurons only react to visual stimuli located in a limited region of the visual field. The receptive fields of different neurons can overlap, and together they generate the entire visual field. Furthermore, the authors showed that some neurons only react to images of horizontal lines. In contrast, others only react to lines with different orientations. That is, two neurons can have the same receptive field but react to different line orientations. Such researchers also noted that some neurons have larger receptive fields and react to more complex patterns, which are combinations of lower-level patterns.

The concepts mentioned above can be used to create a robust and quite general architecture, which can detect all kinds of complex patterns in any area of the visual field. More precisely, a CNN architecture consists of two components: a loosely connected component and a strongly connected component. The loosely connected component consists of a series of *convolution levels (or layers)* and *polling levels (or layers)*. In detail, convolutional layers apply a series of filters to the input image, called *convolutional filters*. Such filters highlight different characteristics of the image. In this way, *feature-maps* can be generated to represent the features extracted from the input image. More precisely, the extraction of features from the image takes place using a fixed dimension filter $N \times M$. For each region of $N \times M$ pixels of the image, the convolution operation calculates the scalar product between the image's intensity values and the weights defined in the filter.

The most important parameters that should be set for the convolutional levels are the following:

- *Size of filters*;
- *Number of filters*;
- *Stride*, which defines how the filter should move along the size of the image;
- *Padding*, which defines whether the padding should be added to keep the image dimensions unchanged;
- *Activation function*, which applies some operations to the input value to produce the output.

Pooling aims to reduce the number of parameters that the network has to learn. In a pooling step, the CNN performs downsampling on the convolved feature by saving computational time, thus reducing the dimensionality of the feature map while still preserving the information associated with the most critical features. In particular, *aggregation functions* are used to perform this action. These functions reduce the use of memory, the necessary computational power, and overfitting. In particular, the most used aggregation functions are the following:

- *Max-pooling function.* For each region below the filter, this function considers the maximum value of that region. Then, such a function creates a new output matrix. Each element of this matrix is the maximum of one region in the original input.
- *Mean (average)-pooling function.* For each region below the filter, this function considers the average value of that region. Then, such a function creates a new output matrix. Each element of this matrix is the average of one region in the original input.

On the other hand, the strongly connected component is usually a classic neural network, which takes as input the features generated by the last layer of the loosely connected component to return the prediction result (See Fig. 1).

When properly trained, the CNN progressively acquires knowledge about the optimal values for the filtering matrices allowing the extraction of the more meaningful features from the input feature map.

3. Related work

The detection of anomalous DNS traffic is a deeply-felt and widely-explored research field.

There are mainly two techniques for finding tunnels that leverage DNS queries to communicate: payload analysis (Born & Gustafson, 2010) and traffic analysis (Allard, Dubois, Gompel, & Morel, 2011). However, conventional detection techniques, based purely on a statistical analysis of network traffic or packet content, are not always effective in dealing with many situations (Aiello, Mongelli, & Papaleo, 2015; Dusi et al., 2009). In particular, techniques that rely on the size or entropy of labels in DNS

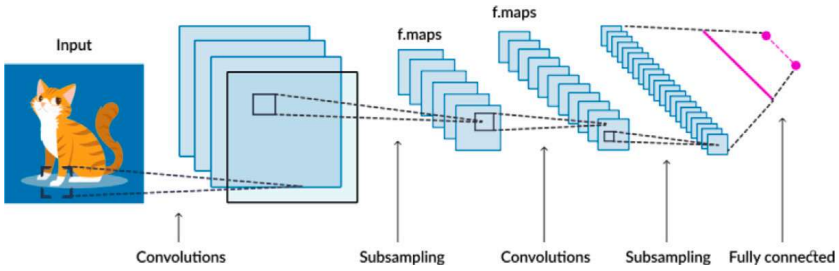

Fig. 1. A typical convolutional neural network (CNN).

queries (Homem, Papapetrou, & Dosis, 2017, 2018; Khodjaeva & Zincir-Heywood, 2021) are not effective with respect to legitimate websites with a very long domain name or services that use geographically-dispersed network infrastructure as *content delivery networks (CDNs)* (Homem et al., 2017). For example, until a few years ago, Facebook used the Akamai CDN for static resources such as CSS files reachable from a path like https://fbstatic-a.akamaihd.net/rsrc.php/v2/yF/r/ADbV1q22oVf.css. It is easy to observe that this path has high entropy since it is composed, at first sight, of purely random characters. Instead, suppose the technique is based on identifying a high traffic volume towards a specific IP address (Ellens et al., 2013). In that case, this technique can be circumvented through IP spoofing mechanisms. Again, when the query uses uncommon resource records such as 'TXT', we cannot be sure that communication occurs using a DNS tunnel. Thus, DNS tunnel detection mechanisms have evolved towards the use of Machine Learning techniques. These techniques enable us to consider various characteristics of the DNS queries before carrying out the classification of such queries.

Yu, Smith, Threefoot, and Olumofin (2016) use neural networks to create a behavior analysis-based method to detect and block tunneled malicious traffic. Nadler, Aminov, and Shabtai (2019) tackle the problem of detecting low-throughput data exfiltration via DNS, using the *Isolation Forest*. Almusawi and Amintoosi (2018), using a *Multilabel Support Vector Machine* algorithm, propose a system able to identify a DNS tunnel and classify the type of tunneled protocol. Buczak et al. (2016) use a *Random Forest* to detect DNS tunnels in network traffic and use various DNS tunneling tools to generate the traffic then used to train and test the classifier. Moreover, other solutions use the *Ensemble Learning* technique, which consists of building a classifier starting from the union of other classifiers. For example, Shafieian, Smith, and Zulkernine (2017) propose a solution based on the union of three completely different algorithms: *Random Forests*, *k-Nearest Neighbor (k-NN)*, and *Multi-Layer Perceptron (MLP)*. Aiello, Mongelli, and Papaleo (2014) propose a two-level classifier based on the ensemble learning technique, called *Majority Voting*. Again, Aiello et al. (2015) introduce a method that calculates the mean, variance, kurtosis, and asymmetry of the time interval between inbound and outbound queries and their sizes. In this way, the authors train four types of classifiers: *k-Nearest Neighbor (k-NN)*, *Neural Networks (NN)*, *Support Vector Machines (SVM)*, *Bayes classifier*, and then use the voting-based ensemble technique. Finally, Sammour, Hussin, and Othman (2017) provide a comparative analysis of three ML techniques, such as SVM, *Naive Bayes (NB)* and *J48*. The authors used a benchmark dataset for the DNS tunneling in such an analysis. This work shows that SVM performs better than other classifiers for detecting DNS tunnels.

In the last decade, techniques based on Deep Learning have given an effective solution for DNS tunneling detection. In particular, Zhang et al. (2019) have proposed a detection method based on deep learning that does not use DNS behavior features or network traffic features. The strength of this method is that it can detect DNS tunneling activity before the data exfiltration. This method uses query payloads as predictive variables in the model, which are then processed using natural language processing (NLP). Furthermore, this method to make its classification decisions uses common deep learning models such as recurrent neural network (RNN), dense neural network (DNN), and one-dimensional convolutional neural network (1D-CNN). In the approach proposed by Palau, Catania, Guerra, Garcia, and Rigaki (2020), a Convolutional Neural Network (CNN) is used. The main strength of this approach is that it is based on an architecture that has low complexity. Furthermore, in this work, a new dataset produced by the authors to evaluate the DNS tunneling activity is also presented and described. Liu, Dai, Cui, and Lin (2019) have proposed a method based on deep learning called Byte-level CNN. This method considers DNS packets as bytes and can learn information about the entire DNS packet, including sequential, structural, and statistical information. Finally, Lai, Huang, Huang, Mao, and Lee (2018) propose a *feature-free* solution, which is based exclusively on packet bytes. This solution trains a neural network only through benign queries without malicious queries or specified features.

4. The proposed solution

In this section, the proposed solution is shown in detail. More precisely, we show how several features, extracted from the basic DNS traffic, are arranged to implement bi-dimensional images and how spatial features can be extracted from these images through a CNN. Finally, such spatial features are used to perform traffic classification by using a fully connected neural network.

Table 1
Complete list of used features.

QuestionTypeA	*QuestionTypeAaaa*	*QuestionTypeAll*	*QuestionTypeCname*
QuestionTypeDnskey	*QuestionTypeMx*	*QuestionTypeNs*	*QuestionTypeNull*
QuestionTypePtr	*QuestionTypeSoa*	*QuestionTypeSrv*	*QuestionTypeTkey*
QuestionTypeTxt	*QuestionTypeUnknown*	*IsReplySuccess*	*QuestionLength*
QuestionEntropy	*QuestionInfoBits*	*ResponseAnswerLength*	*ResponseAnswerInfoBits*
ResponseInfoBits	*ResponseEntropy*		

4.1. Modeling the DNS traffic

There are many tools for collecting DNS traffic data, such as DNS Statistics Collector (DSC) (Wessels & Lundstrom, 2021) and DNSCAP (DNS-OARC, 2021). Unfortunately, no specific format has been standardized for storing DNS packet captures, so in most of the cases, PCAP (The Tcpdump Group, 2021) or PCAP Next Generation (PCAP-NG) (commit 3c35b6a, 2021) are used. However, these formats, typically developed for packet capture, include a lot of packets and frame-level additional information that may not be useful in the observation of DNS transactions, which unnecessarily increments the amount of the captured data. Although the collection of full packet traces of the DNS transactions may provide a most comprehensive view of DNS activity, it should be noted that DNS traffic may consist of thousands of queries per second, also in the case of a very small network. To avoid analyzing a great amount of network traffic, only relevant features that characterize the traffic of interest should be individuated, extracted from the raw packet traces, and used for further analysis.

In this paper, the features used are the following:

- *Question_Type:* resource records used for the DNS request (e.g., 'A', 'AAA', 'CNAME', 'TXT', etc.);
- *Is_Reply_Success:* indicates whether there was an error in the response;
- *Question_Length:* length in bytes of the DNS request;
- *Question_Info_Bits:* an estimate of the bytes needed to encode the DNS request;
- *Question_Entropy:* entropy of the DNS request;
- *Response_Answer_Length:* length in bytes of the DNS response;
- *Response_Answer_Info_Bits:* an estimate of the bytes needed to encode the DNS response;
- *Response_Info_Bits:* length in bytes of the DNS response;
- *Response_Entropy:* Entropy of the DNS query response.

Some features can be extracted directly from DNS traffic packets at the collection time. In contrast, others need to be derived from these features. More precisely, some of the derived features are *Response_Entropy* and *Question_Entropy*. These features assume a very important role in distinguishing DNS malicious traffic. Indeed, the content of legitimate DNS queries includes valid and commonly used words. Conversely, the content of DNS queries generated by DNS tunneling tools is usually Base32 or Base64 encoded. This encoding operation increases the entropy of the DNS queries, significantly impacting the discriminatory ability to distinguish legitimate traffic from the illegal (or malicious) one.

Table 1 shows the complete set of used features. Note that the *Question_type* feature has been coded by using the *one-hot encoding* technique.

As depicted, there are 22 different features. In order to derive a bi-dimensional image from them, the distribution represented in Table 1 was used. That is, each image is represented by a (6×4)-dimensional matrix, whose entries are arranged as reported in Table 1. Notice that there are two missing entries, which were replaced with an arbitrary constant. The value of this constant does not influence the effectiveness of the proposal because it is not considered discriminating.

4.2. CNN and spatial features

A CNN was used to mine useful insight from DNS-images. Notice that to the best of our knowledge, the proposed model has never been used in the literature for DNS tunnel detection. Our goal was to evaluate the ability of CNNs in addressing this task using input data that are neither images nor audio.

Generally, a CNN-based architecture is composed of a series of convolutional and pooling levels used to extract features from the input image. More specifically, each convolutional level acts by sliding several different filters of fixed dimensions over such images to extract a set of feature-maps, as shown in Figs. 2 and 3, respectively.

On the other hand, as shown in Fig. 4, the pooling level aims to reduce the size of feature-maps returned by the convolutional level. This operation is capable of speeding up the subsequent phases by focusing on the most characterizing features.

More precisely, in Fig. 3, we show the creation of Feature Maps. The creation of Feature Maps is based on the use of filters. Such filters extract small sections of the image, where each section has a different characteristic. In detail, the filters slide over the image using convolution operation and generate a feature map. Each feature map captures different features from the same image. In general, by using more filters, we may generate more features.

Fig. 4 shows a 4×4 matrix representing the initial input. Assume applying a 2×2 filter to that input. In the case of *max pooling*, using this filter on the initial input, for each of the regions characterized by the filter, we will take the maximum value of that

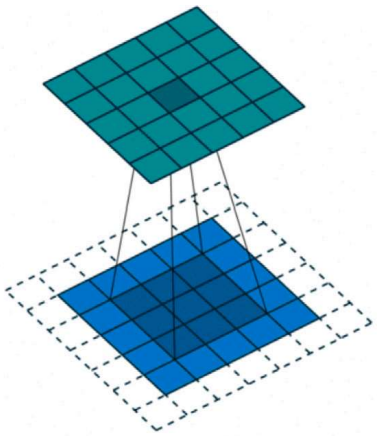

Fig. 2. An example of filter sliding.

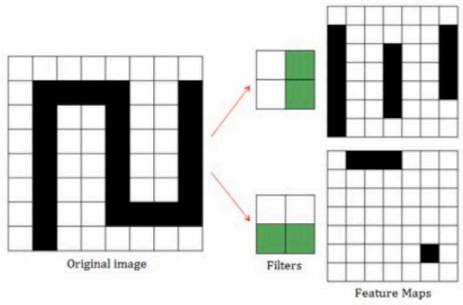

Fig. 3. Creation of feature-maps by sliding the filters on the image.

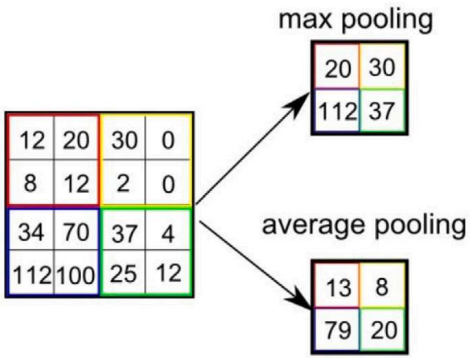

Fig. 4. Result of two pooling operations: the *max pooling* and the *average pooling*.

region and create a new output matrix in which each element is the maximum value of a region of the input initial (i.e., 30, 20, 112 and 37 in Fig. 4). Instead, in the case of average pooling, for each of the regions characterized by the filter, we will calculate the average of the values of that region and create a new output matrix in which each element is the average value of a region of the initial input (i.e., 8, 13, 79 and 20 in Fig. 4).

CNNs operate in layers. In the lowest layer (i.e., the one closest to the input), the network extracts basic features, such as arcs, lines, etc. While, in the next layers (i.e., layers far from the input), the network uses the features of the previous layers to extract increasingly complex features.

Finally, a fully connected neural network followed by a *softmax* layer is used to classify the features extracted by the CNN and, in turn, to perform the network traffic classification.

In order to allow a better understanding of the effectiveness of the proposal, in the following, some mathematical considerations are given.

Let $X \in R^{m \times n}$ be the $(m \times n)$-dimensional matrix representing a generic DNS-Image (note that in our case $m = 6$ and $n = 4$), while let $K^f \in R^{A \times B}$ be the generic convolution filter, then, as is known, the convolution operation between the DNS-Image and F filters

provides as an outcome a new matrix (Y), whose component ($Y_{i,j}$) are given by the following:

$$Y_{i,j} = \sum_{f=1}^{F} \sum_{p=1}^{A} \sum_{q=1}^{B} K_{p,q}^{f} X_{i+p-1,j+q-1} \tag{1}$$

The size of this new matrix in terms of number of rows (Y_{row}) and columns (Y_{col}) is completely defined by the way the convolution is performed through the stride (S) and padding (P), that is:

$$\begin{aligned} Y_{row} &= \frac{m - A + 2P}{S_x} + 1 \\ Y_{col} &= \frac{n - B + 2P}{S_y} + 1 \end{aligned} \tag{2}$$

where S_x and S_y refer to the stride used on x and y axes, respectively.

As it can be observed, Eq. (1) is able to explicit, through a mathematical formalism, the relationship existing among all the DNS-features. Indeed, each component of the new matrix Y can be expressed as a linear combination of the considered features ($X_{i,j}$) weighted by coefficients that are the elements ($K_{p,q}$) of the filters (K^f). The value of these coefficients is entrusted to the CNN training phase. Nevertheless, such a relationship among DNS features can be made more representative if multiple layers of CNN are used. Indeed, as shown below, adding a new CNN layer leads to the following equations:

$$Z_{r,s} = \sum_{l=1}^{L} \sum_{t=1}^{G} \sum_{u=1}^{H} K_{t,u}^{l} Y_{r+t-1,s+u-1} \tag{3}$$

replacing Eq. (1) into (3), it yields:

$$Z_{r,s} = \sum_{l=1}^{L} \sum_{f=1}^{F} \sum_{t=1}^{G} \sum_{u=1}^{H} \sum_{p=1}^{A} \sum_{q=1}^{B} K_{t,u}^{l} K_{p,q}^{f} X_{r+t+p-2,s+u+q-2} \tag{4}$$

where L is the number of filters used in the second layer, while G and H are the sizes of these filters on x and y axes, respectively.

As depicted, Eq. (4) expresses a most complex relationship among features. Indeed, each component ($Z_{r,s}$) of the output matrix of the second CNN-level is computed by a linear combination of the DNS-features weighted on the product of the filter-components belonging to both layers, that is $K_{t,u}^{l}$ and $K_{p,q}^{f}$. Hence, Eq. (4) can be most representative of the complex relationship existing among DNS-features.

4.3. Parameters setting

In the following, we report some basic parameters used for implementing the proposed neural network:

- *Kernel_size:* denotes the size of the filters. In our model, this parameter was set to '3'.
- *Activation:* this parameter was set to '*relu*' (*REctified Linear Unit*). In detail, the rectifier is an activation function defined as the positive part of its argument. It was demonstrated that the *rectifier* provides better deep neural networks training.
- *Padding*: this parameter was set to '*same*'. In this way, the convolutional layers use padding if necessary.

The proposed model is shown in Fig. 5. Ultimately, the proposed model consists of a first convolutional level with an '*input_shape*' equal to $[6, 4, 1]$, where the input matrix is (6×4)-dimensional, and "1" black and white channel. Again, 8 (3×3)-dimensional filters were used in the first convolutional layer, which provides as output 8 feature-maps, each created with a different filter. These feature-maps were the input for the next level, i.e., a further similar convolutional level. Next, a pooling level was used to reduce the (6×4)-dimensional matrix into a (2×3)-dimensional matrix by setting the 'pool_size' equal to 2. Besides, we set up another convolutional level, in which the number of filters is increased to 16. Indeed, it is common to double the number of filters after each pooling level. The doubling operation occurs because each level of this type reduces the size of the feature-maps. In particular, in our model, the reduction occurs by a factor of 2. Therefore, we can double the number of filters without incurring an excessive increase in the number of parameters, memory use, or computational calculation. The subsequent levels of the model included the fully connected neural network. As a consequence, a *flattening* level was also added. We remark that flattening converts the two-dimensional output of the convolutional level into a one-dimensional array. This array was fed into the fully connected NN, including 2 dense hidden levels and a softmax output level. These levels are interspersed with two *dropout levels*, with the dropout value set at 50%, to reduce overfitting.

We set the '*binary_crossentropy*' as *loss* because we deal with a binary classification problem that can only belong to two classes: legitimate query or illegitimate query. Once the model is compiled, we train it using the '*fit*' method.

The following parameters were used during the training:

- *Epochs:* this parameter denotes the number of iterations the model performs on the training set. We set this parameter to 20.
- *Batch size:* this parameter denotes the number of samples to be processed before the update of the model. We set this parameter to 100. This assignment causes the network weights to be updated after processing blocks of 100 samples. Notice that this operation speeds up network training.

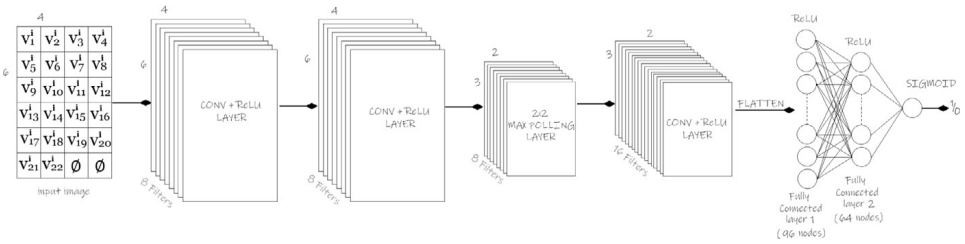

Fig. 5. Graphic representation of the proposed model.

- *Validation_split:* this parameter denotes the percentage of the training data to use as validation data. We set this parameter to 0.2. This assignment means that 20% of the training set is used for data validation. Therefore, the model does not train on this fraction of the training data and evaluates the loss and any model metrics on this data at the end of each epoch.
- *Verbose:* this parameter allows us to display the details of each epoch on the screen. We set this parameter to 2.

5. Experiments and discussion

This section shows the experimental results obtained by evaluating the proposed solution. In particular, in this section, we first provide the details related to the technologies used to create a simple but effective proof of concept. Next, we describe the dataset used for the experimental evaluation. Finally, we describe the metrics used in such evaluation and the results obtained by assessing the application of the proposal on the dataset mentioned above. We emphasize that as done in Liu et al. (2019), Palau et al. (2020) and Zhang et al. (2019), we used for the evaluation a standard testing pattern together with the common metrics defined for the assessment of machine learning models.

5.1. Implementation details

The proof of concept prototype of the detection framework was implemented using the *Python* programming language.

In detail, we used the *NumPy* libraries (Oliphant, 2006) for n-dimensional data manipulation, the *pandas* library (McKinney et al., 2011) for implementing convenient data structures, such as *DataFrame*, and the *Matplotlib* library (Hunter, 2007) to create and manipulate graphics. We remark that these libraries belong to the *SciPy* ecosystem (Virtanen et al., 2020), which provides many useful tools.

Furthermore, we used the *Keras-Tensorflow* library (Gulli & Pal, 2017; Ketkar, 2017; Manaswi, 2018) to perform all the machine learning-related tasks underlying the proposed model. We remark that Keras is an open-source library for machine learning and neural networks written in Python. It is designed as an interface to other lower-level libraries. Keras supports the TensorFlow, *Microsoft Cognitive Toolkit (CNTK)*, and *Theano* libraries as a back-end. TensorFlow is a library designed to enable rapid prototyping of deep neural networks. Its main goal is to provide ease of use, modularity, and extensibility. Finally, all the work needed to implement the model has been done using *PyCharm* as an integrated development environment (IDE).

5.2. Dataset

The experiments were carried out by using the real traffic dataset provided by Queen's University (Kingston, Canada). This dataset was created to analyze DNS query classification, using an approach based on *Ensemble Learning*.

This dataset includes both benign queries representing regular DNS traffic, and a set of malicious queries representing traffic tunneled into DNS queries. More precisely, the traffic tunneled in DNS queries was generated through three different tools: *Iodine*, *Dnscat*, and *Ozyman*.

The set of benign DNS queries was generated by capturing the DNS traffic of a *corporate network* of geographically-distant offices located in Canada and the US. In detail, packets from March 23, 2016, to May 3, 2016, were captured for a total of 1.5 GB of data. The captured packets represent the regular network traffic of the corporate network.

On the other hand, the set of malicious queries is formed by the union of three subsets, each generated using the previously mentioned DNS tunneling tools. We remark that for the generation of malicious queries, different tools were used to achieve a more significant variability between the records of the dataset and build a robust system capable of addressing the problem of DNS tunneling detection more broadly and carefully. In detail, all DNS packets were physically captured in the used dataset using a *Network Tap* on the Corporate Network. Subsequently, using *Wireshark* and *Weka*'s Java APIs, these packages were processed and sorted.

The dataset constructed in such a way is made up of 600,000 records. We point out that one of the main limitations affecting the datasets used by the solutions described in the Related Work section (Section 3) is that these datasets are quite small, making the contributions proposed by these solutions limited to a fairly narrow set of traffic types.

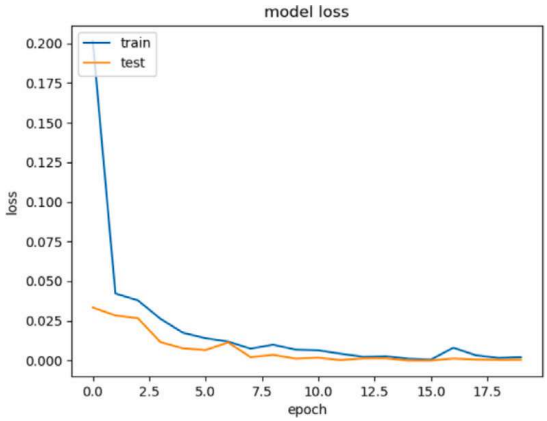

Fig. 6. The trend of the loss function.

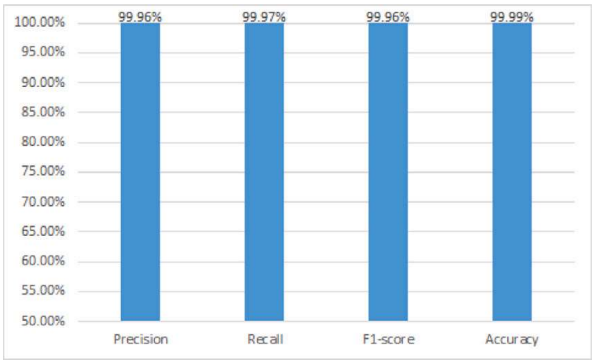

Fig. 7. Results for testing set.

5.3. Evaluation metrics and results

The proposed detection framework was tested by dividing the dataset into two parts, namely the *training* (70%) and *testing* (30%) sets. Each part contains an equal portion of both classes.

To evaluate the performance, the following metrics, derived from the multi-class confusion matrix (Diez, 2018), were used:

$$Accuracy\ (Acc) = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

$$Sensitivity\ (Sens) = \frac{TP}{TP + FN} \tag{6}$$

$$Specificity\ (Spec) = \frac{TN}{TN + FP} \tag{7}$$

$$Precision\ (Prec) = \frac{TP}{TP + FP} \tag{8}$$

$$F\_Measure\ (Fmea) = \frac{2 * Sens * Prec}{Sens + Prec} \tag{9}$$

where TPs (True Positives) are the malign queries correctly classified, FPs (False Positives) are the benign queries incorrectly classified, FNs (False Negatives) are the benign queries incorrectly rejected, and TNs (True Negatives) are the benign queries correctly rejected.

All the metrics were computed by using the *scikit-learn* Python library (Cournapeau, 2007).

Also, the *loss function* was estimated. As it is known, the *loss function* characterizes the discrepancy between the values predicted by the model and the expected values for each instance. Fig. 6 shows the trend of the loss function for both training and testing sets. As depicted, it achieved a very low value, about 0.0012, within a few epochs for both datasets.

Fig. 7 shows the metrics for the testing set. As depicted, very high scores were achieved. Indeed, they are 99.96%, 99.97%, 99.96%, and 99.99% for Precision, Recall, F1, and Accuracy, respectively. Besides, in Fig. 8 the accuracy trend at different epochs is shown. As depicted, Accuracy achieves a very high score from the earliest epochs.

Finally, in order to show the superiority of the proposed detection architecture with respect to the state-of-the-art, we report in Fig. 9 the comparison of our proposal with respect to the main classification techniques available so far. In particular, we compared

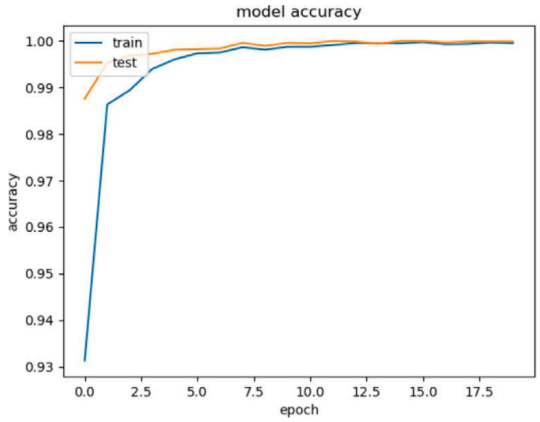

Fig. 8. Graph showing the accuracy trend.

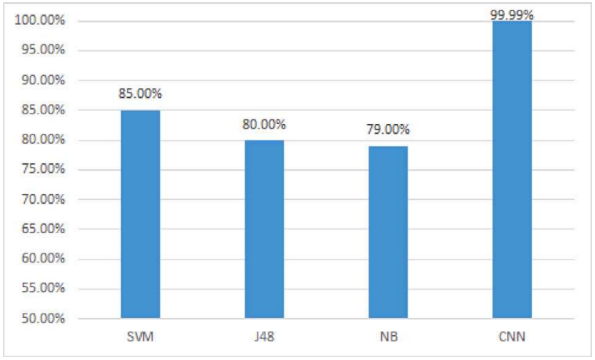

Fig. 9. Accuracy comparison.

our proposal with Support Vector Machine (SVM), J48 decision tree, and Naive Bayes. As depicted, our proposal outperformed all the others.

We remark that the proposed solution guarantees better performance than the work of Zhang et al. (2019) in terms of Precision, Recall, F1, and Accuracy. Furthermore, the evaluation has been performed on a dataset whose size is almost twice, in terms of the number of queries, than the one used by Zhang et al. Again, the approach proposed by Palau et al. (2020) also exhibits a performance that is worse than the one we proposed in terms of Precision, Recall, and F1. In addition, the dataset used in this paper is considerably smaller, in terms of the number of queries, than the one we used for evaluation. Besides, although the method introduced by Liu et al. (2019) ensures performance in terms of Accuracy, Precision, Recall, and F1 that are in line with our approach, it is extremely cumbersome, as it requires expensive operations and shows a high computational complexity. Therefore, this method is not suitable for analyzing and detecting DNS tunnels in real-time traffic. Finally, though the DNS tunneling detector proposed by Lai et al. (2018) provides results comparable with ours, however, for the evaluation of this detector, fewer metrics are used. Again, the dataset used by Lai et al. (2018) is significantly smaller, in terms of the number of queries, than the one used in our evaluation.

6. Conclusions and future directions

DNS is a fundamental protocol for accessing Internet services. As a result, most defensive systems that filter messages over the Internet network do not monitor DNS traffic. For this reason, in the last decade, many tools have been proposed to encapsulate any traffic in DNS queries. This approach allows creating a "two-way" tunnel that can be used to carry out fraudulent activities, such as installing malware, creating a *Command & Control* infrastructure to carry out DDoS attacks, or stealing sensitive user data. Also, all these fraudulent activities are made even easier because firewalls generally do not filter DNS traffic and let it pass freely.

In this work, we have proposed a method to extract meaningful and powerful features from DNS queries, which can be used as a signature of the traffic typology. In particular, some basic features extracted by the DNS traffic are used to derive a behavioral representation of the traffic itself as a bi-dimensional image, referred to as DNS-images.

A stacked neural network including a CNN followed by a fully connected neural network has also been provided for classifying these new features, and in turn, to distinguish legitimate queries from the ones related to DNS tunneling activity. This approach may reveal to be an extremely promising task in predictive security approaches to attack detection.

The experimental results obtained by testing our proposal on a real-world dataset have proven its effectiveness by achieving an accuracy of 99.99%. Besides, a comparison with the state-of-the-art solutions has shown the superiority of the DNS-images in

classifying the DNS queries. Indeed, the accuracy achieved by Support Vector Machine, J48 decision tree, and Naive Bayes was 85%, 80%, and 79%, respectively.

As future developments, we first intend to expand the dataset by generating further illegitimate queries through other DNS tunneling tools that use different approaches. Furthermore, we want to re-train the model based on this new dataset to achieve a better generalization. Again, we intend to create an *Intrusion Prevention System (IPS)* that could be used with firewalls to block fraudulent DNS queries in real-time automatically.

Finally, as further future research development, we intend to extend the proposed framework to let it assist the forensic investigator in analyzing DNS tunnels to predict which network protocols are hidden within the DNS tunnels. In this way, forensic investigators can quickly identify and manage specific DNS tunneling traffic samples, which more likely may contain network protocols that deserve more interest and deeper inspection for investigative purposes, while discarding others considered less important without manually reconstructing and dissecting the packets. In this way, the analysis process could be further automated, reducing investigative efforts only on the data of interest, which are more likely to contain meaningful information. This improvement would significantly reduce the burden of the forensic investigator while decreasing the possibility of making errors.

References

Aiello, M., Merlo, A., & Papaleo, G. (2013). Performance assessment and analysis of DNS tunneling tools. *Logic Journal of the IGPL*, *21*(4), 592–602.

Aiello, M., Mongelli, M., & Papaleo, G. (2014). Supervised learning approaches with majority voting for DNS tunneling detection. In J. G. de la Puerta, I. G. Ferreira, P. G. Bringas, F. Klett, A. Abraham, A. C. de Carvalho, A. Herrero, B. Baruque, H. Quintián, & E. Corchado (Eds.), *International joint conference SOCO'14-CISIS'14-ICEUTE'14* (pp. 463–472). Cham: Springer International Publishing.

Aiello, M., Mongelli, M., & Papaleo, G. (2015). DNS tunneling detection through statistical fingerprints of protocol messages and machine learning. *International Journal of Communication Systems*, *28*(14), 1987–2002.

Allard, F., Dubois, R., Gompel, P., & Morel, M. (2011). Tunneling activities detection using machine learning techniques. *Journal of Telecommunications and Information Technology*, 37–42.

Almusawi, A., & Amintoosi, H. (2018). DNS tunneling detection method based on multilabel support vector machine. *Security and Communication Networks*, *2018*.

Berg, A., & Forsberg, D. (2019). Identifying DNS-tunneled traffic with predictive models. arXiv preprint arXiv:1906.11246.

Born, K., & Gustafson, D. (2010). Detecting DNS tunnels using character frequency analysis. arXiv preprint arXiv:1004.4358.

Buczak, A. L., Hanke, P. A., Cancro, G. J., Toma, M. K., Watkins, L. A., & Chavis, J. S. (2016). Detection of tunnels in PCAP data by random forests. In *Proceedings of the 11th annual cyber and information security research conference*. New York, NY, USA: Association for Computing Machinery.

commit 3c35b6a (2021). pcapng: PCAP next generation file format specification. URL https://www.tcpdump.org/.

Cournapeau, D. (2007). scikit-learn. URL https://scikit-learn.org/stable/.

D'Angelo, G., & Palmieri, F. (2020a). Discovering genomic patterns in SARS-CoV-2 variants. *International Journal of Intelligent Systems*, *35*(11), 1680–1698. http://dx.doi.org/10.1002/int.22268.

D'Angelo, G., & Palmieri, F. (2020b). Knowledge elicitation based on genetic programming for non destructive testing of critical aerospace systems. *Future Generation Computer Systems*, *102*, 633–642. http://dx.doi.org/10.1016/j.future.2019.09.007, URL https://www.sciencedirect.com/science/article/pii/S0167739X19306193.

D'Angelo, G., & Palmieri, F. (2021). Network traffic classification using deep convolutional recurrent autoencoder neural networks for spatial–temporal features extraction. *Journal of Network and Computer Applications*, *173*, Article 102890. http://dx.doi.org/10.1016/j.jnca.2020.102890, URL https://www.sciencedirect.com/science/article/pii/S1084804520303519.

D'Angelo, G., Palmieri, F., Robustelli, A., & Castiglione, A. (2021). Effective classification of android malware families through dynamic features and neural networks. *Connection Science*, *33*(3), 786–801.

D'Angelo, G., Tipaldi, M., Palmieri, F., & Glielmo, L. (2019). A data-driven approximate dynamic programming approach based on association rule learning: Spacecraft autonomy as a case study. *Information Sciences*, *504*, 501–519. http://dx.doi.org/10.1016/j.ins.2019.07.067, URL https://www.sciencedirect.com/science/article/pii/S0020025519306796.

Diez, P. (2018). Chapter 1 - Introduction. In P. Diez (Ed.), *Smart wheelchairs and brain-computer interfaces* (pp. 1–21). Academic Press, http://dx.doi.org/10.1016/B978-0-12-812892-3.00001-7, URL https://www.sciencedirect.com/science/article/pii/B9780128128923000017.

DNS-OARC (2021). DNSCAP. URL https://www.dns-oarc.net/tools/dnscap.

Dusi, M., Crotti, M., Gringoli, F., & Salgarelli, L. (2009). Tunnel hunter: Detecting application-layer tunnels with statistical fingerprinting. *Computer Networks*, *53*(1), 81–97.

Ellens, W., Żuraniewski, P., Sperotto, A., Schotanus, H., Mandjes, M., & Meeuwissen, E. (2013). Flow-based detection of DNS tunnels. In *IFIP international conference on autonomous infrastructure, management and security* (pp. 124–135). Springer.

Gulli, A., & Pal, S. (2017). *Deep learning with Keras*. Packt Publishing Ltd.

Homem, I., & Papapetrou, P. (2017). Harnessing predictive models for assisting network forensic investigations of DNS tunnels. In *Annual ADFSL conference on digital forensics, security and law*. ADFSL (Association of Digital Forensics, Security and Law).

Homem, I., Papapetrou, P., & Dosis, S. (2017). Entropy-based prediction of network protocols in the forensic analysis of DNS tunnels. arXiv preprint arXiv:1709.06363.

Homem, I., Papapetrou, P., & Dosis, S. (2018). Information-entropy-based DNS tunnel prediction. In *IFIP international conference on digital forensics* (pp. 127–140). Springer.

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(03), 90–95.

Ketkar, N. (2017). Introduction to Keras. In *Deep learning with Python* (pp. 97–111). Springer.

Khan, S., Gani, A., Wahab, A. W. A., Shiraz, M., & Ahmad, I. (2016). Network forensics: Review, taxonomy, and open challenges. *Journal of Network and Computer Applications*, *66*, 214–235.

Khodjaeva, Y., & Zincir-Heywood, N. (2021). Network flow entropy for identifying malicious behaviours in DNS tunnels. In *The 16th international conference on availability, reliability and security* (pp. 1–7).

Lai, C., Huang, B., Huang, S., Mao, C., & Lee, H. (2018). Detection of DNS tunneling by feature-free mechanism. In *2018 IEEE conference on dependable and secure computing* (pp. 1–2).

Liu, C., Dai, L., Cui, W., & Lin, T. (2019). A byte-level CNN method to detect DNS tunnels. In *2019 IEEE 38th international performance computing and communications conference* (pp. 1–8). IEEE.

Manaswi, N. K. (2018). Understanding and working with Keras. In *Deep learning with applications using python* (pp. 31–43). Springer.

McKinney, W., et al. (2011). pandas: A foundational Python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, *14*(9), 1–9.

Nadler, A., Aminov, A., & Shabtai, A. (2019). Detection of malicious and low throughput data exfiltration over the DNS protocol. *Computers & Security*, *80*, 36–53.

Ogiela, L., & Ogiela, M. R. (2020). Cognitive security paradigm for cloud computing applications. *Concurrency Computations: Practice and Experience*, *32*(8), Article e5316.

Ogiela, U., & Ogiela, M. R. (2021). Predictive intelligence approaches for security technologies. In *International conference on broadband and wireless computing, communication and applications* (pp. 193–196). Springer.

Ogiela, L., Ogiela, M. R., & Ogiela, U. (2016). Efficiency of strategic data sharing and management protocols. In *2016 10th international conference on innovative mobile and internet services in ubiquitous computing* (pp. 198–201). IEEE.

Oliphant, T. E. (2006). *A guide to NumPy: Vol. 1*. Trelgol Publishing USA.

Palau, F., Catania, C., Guerra, J., Garcia, S., & Rigaki, M. (2020). DNS tunneling: A deep learning based lexicographical detection approach. arXiv preprint arXiv:2006.06122.

Sammour, M., Hussin, B., & Othman, M. F. I. (2017). Comparative analysis for detecting DNS tunneling using machine learning techniques. *International Journal of Applied Engineering Research*, *12*(22), 12762–12766.

Schmid, G. (2021). Thirty years of DNS insecurity: Current issues and perspectives. *IEEE Communications Surveys & Tutorials*, *23*(4), 2429–2459.

Shafieian, S., Smith, D., & Zulkernine, M. (2017). Detecting DNS tunneling using ensemble learning. In Z. Yan, R. Molva, W. Mazurczyk, & R. Kantola (Eds.), *Network and system security* (pp. 112–127). Cham: Springer International Publishing.

Sikos, L. F. (2020). Packet analysis for network forensics: A comprehensive survey. *Forensic Science International: Digital Investigation*, *32*, Article 200892.

The Tcpdump Group (2021). PCAP. URL https://www.tcpdump.org/.

Torabi, S., Boukhtouta, A., Assi, C., & Debbabi, M. (2018). Detecting internet abuse by analyzing passive DNS traffic: A survey of implemented systems. *IEEE Communications Surveys & Tutorials*, *20*(4), 3389–3415.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., et al. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261–272.

Wang, Y., Zhou, A., Liao, S., Zheng, R., Hu, R., & Zhang, L. (2021). A comprehensive survey on DNS tunnel detection. *Computer Networks*, *197*, Article 108322.

Wessels, D., & Lundstrom, J. (2021). DSC. URL https://www.dns-oarc.net/tools/dsc.

Xu, K., Butler, P., Saha, S., & Yao, D. (2013). DNS for massive-scale command and control. *IEEE Transactions on Dependable and Secure Computing*, *10*(3), 143–153.

Yu, B., Smith, L., Threefoot, M., & Olumofin, F. (2016). Behavior analysis based DNS tunneling detection and classification with big data technologies. In *Proceedings of the international conference on internet of things and big data*: *Vol. 1*, (pp. 284–290). INSTICC, SciTePress.

Zhang, J., Yang, L., Yu, S., & Ma, J. (2019). A DNS tunneling detection method based on deep learning models to prevent data exfiltration. In *International conference on network and system security* (pp. 520–535). Springer.