Domain Name System (DNS) Tunnelling Detection using Structured Occurrence Nets (SONs)

Talal Alharbi[1,2][0000−0001−8252−9271] and Maciej Koutny[1][0000−0003−4563−1378]

[1] School of Computing, Newcastle University, Newcastle upon Tyne, UK
[2] Faculty of Computer Science and Engineering, University of Hai'l, Hai'l, Saudi Arabia
{talal.alharbi,maciej.koutny}@ncl.ac.uk

Abstract. Today, serious warnings regarding the increasing number of DNS tunnelling methods are on the rise. Attackers have used such techniques to steal data from millions of accounts. The existing literature has thoroughly demonstrated the extent of the damage which DNS tunnelling can achieve on any given DNS server. However, through SONs - Petri net-based formalisms which portray the behaviour of complex evolving systems, such threats can be alleviated. As a concept, SONs are originally grounded in Occurrence Nets (ONs) and already yielded results in terms of successful cybercrime analysis. For instance, adding of alternates to SONs initially used in [10] was extended to in [15] in order to model and analyse system activities such as cybercrime or accidents, which may show contradictory or uncertain evidence in terms of actual activity. The current paper proposes the use of SON features with the purpose of detecting DNS tunnelling, in the event of an actual attack.

Keywords: DNS tunnelling , Structured Occurrence Nets , Detection of DNS Attacks

1 Introduction

In the last decades, internet usage and spread has grown dramatically, expanding to include everything and anything, from small, online businesses, to large company websites. As a consequence, what was initially "the Web" became "Web 2.0", considering the number of applications which has been developed, based on the internet. It is the Domain Name System or DNS, which allows applications to use names instead of numbers (IP addresses), which are considerably more difficult to deal with. There are several works [4,5,9,12] warning of the increasing number of DNS tunnelling methods, and attackers have used these techniques to compromise millions of accounts [6]. The existing literature has demonstrated the damage which DNS tunnelling can make on any deployed DNS server. DNS tunnelling allows hackers to transfer data in a way which violates established system security policies [4]. The danger which such an attack brings is that it

can occur without triggering any alarms. It is actually shown as a legitimate activity, because it uses a DNS protocol to transmit information in the original way, instead of focusing on DNS vulnerabilities or abusing or exploiting the system. The reason behind using SONs to record DNS traffic are the different features that SONs provide. For example, a SON combines multiple related ONS by using various formal relationships in particular, for communication. For instance, SON events can be used to model various types of packets, such as sending a query, receiving a query, sending a response and receiving a response, as well as asynchronous communication. By means of these features, a SON is able to depict and analyse a DNS case. SONs [6] are a Petri net-based formalism which portray the behaviour of complex evolving systems. Their concept is originally grounded in Occurrence Nets (ONs) and already yielded results in terms of successful cybercrime analysis. The approach which we propose is to use SON features to detect DNS tunnelling in the event of a real attack.

DNS Tunnelling DNS tunnelling is a covert communication channel, which allows encapsulating the traffic of other protocols (E.g.: HTTP, Telnet, FTP, SSH, etc.) within DNS packets. This results in concealing the real protocol by making the traffic look like ordinary DNS traffic. DNS tunnelling is very suitable to be used for malicious activities such as data exfiltration as well as command and control call-backs from within restricted internal networks. Different from other tunnelling methods such as SSH Tunnelling, a distinctive property of DNS Tunnelling is that, it uses the legitimate DNS servers of the internal network to hop its packet through, to reach the final destination, which is the attacker-controlled server DNS Tunnelling encapsulates the data in DNS queries and responses. In this way, the outer protocol remains as DNS; however, the encapsulated data payload belongs to another protocol. [14]. Figure 3 explains how DNS tunnelling is performed. The powerful advantage of DNS tunnelling is that, if a compromised computer is trying to send "secret" data to the attacker-controlled server on the external network (the Internet), even if such direct communications get blocked by the network firewall, DNS Tunnelling would bypass this restriction. The compromised computer would send the secret data within a Base64-encoded message, for example "c2VJcmV0". It will send the query to the internal DNS server, to ask the corresponding IP address, in the form of a basic DNS query for c2VJcmV0.talalncl.com. The internal DNS server would forward this DNS query to the authoritative DNS server for the DNS Zone "talalncl.com". Since the authoritative DNS server is actually the attacker-controlled external server, the message gets delivered to its destination. The talalncl.com DNS server will then respond and return a dummy answer, such as 127.0.0.1. Afterwards, the talalncl.com DNS server will analyse the query as follows:
Step 1: c2VjcmV0.talalncl.com,
Step 2: Base64Decode,
Step 3: c2VjcmV0 -> secret
which means the data extracted from within the DNS query. Finally, a single message within DNS tunnelling looks normal. This message may look exactly

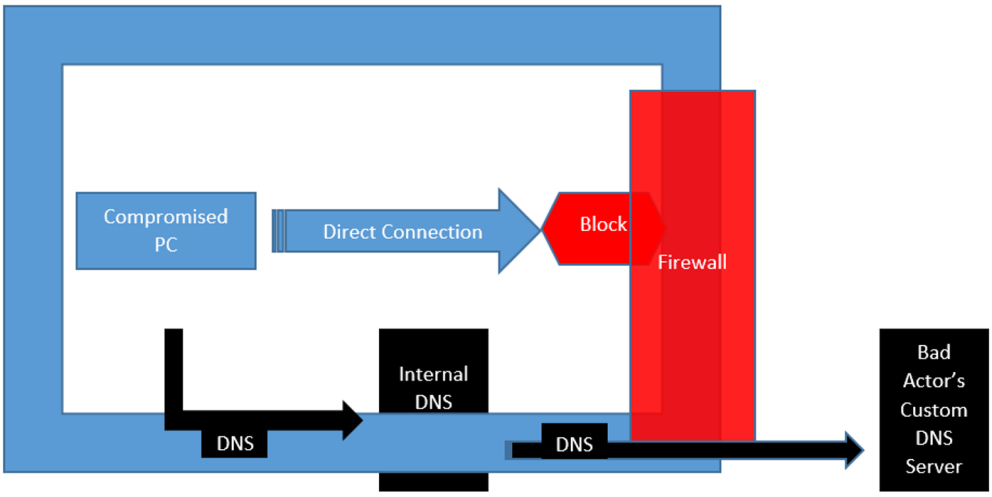

Fig. 1: DNS Tunnelling

like a normal DNS query. This is actually the weakness of DNS Tunnelling: although a single packet may look like a normal DNS query, thousands of queries for sub-domains within the same domain will help to detect the DNS tunnel. Within the next section (section 5), the novel approach which we propose is discussed (i.e., a novel way through which such types of attacks are detected). In addition, an algorithm and its implementation are also discussed. The paper is organised as follows: Section 1 focuses on introduction Section 2 describes the basic features of SONs, as well as background research concerning DNS and the DNS tunnelling phenomenon. Within Section 3 the related work discussed. Section 4 describes DNS experimentation and data preprocessing. Within Section 5 describes the novel solution used to detect DNS tunnelling. Section 6 focuses on the implementation of the solution, as well as on the results and a critical assessment of our algorithm. Section 7 provides concluding remarks.

Terminologies

Table 1: Terminologies

No.	Term	Description
1	SONs	Structured Occurrence Nets
2	ONs	Occurrence Nets
3	DNS	Domain Name System
4	HTTP	HyperText Transfer Protocol
5	FTP	File Transfer Protocol
6	SSH	Secure Shell
7	IP	Internet Protocol
8	iodine	Software that lets you tunnel IPv4 data through a DNS server

2 Background

2.1 Structured occurrence nets (SONs).

The SONs [6,11] concept is originally grounded in occurrence nets (ONs), which are directed acyclic graphs which show causality and concurrency of information concerning a single execution of a system [6,11]. Figure 2 (a) shows an occurrence net (ON). A SON consists of multiple ONs which are associated with each other through various types of formal relationships. SONs are used for recording information concerning the actual behaviour of a complex system, and any particular evidence which can be collected in terms of analysing its past behaviour [4]. The most useful way to use SONs is within a detailed analytical investigation, although there is still a lack of investigation support systems which rely on SONs. The significance of SONs results from the fact that their structuring reduces complexity compared to that of any equivalent representation, and they provide a direct means of modelling evolving systems.

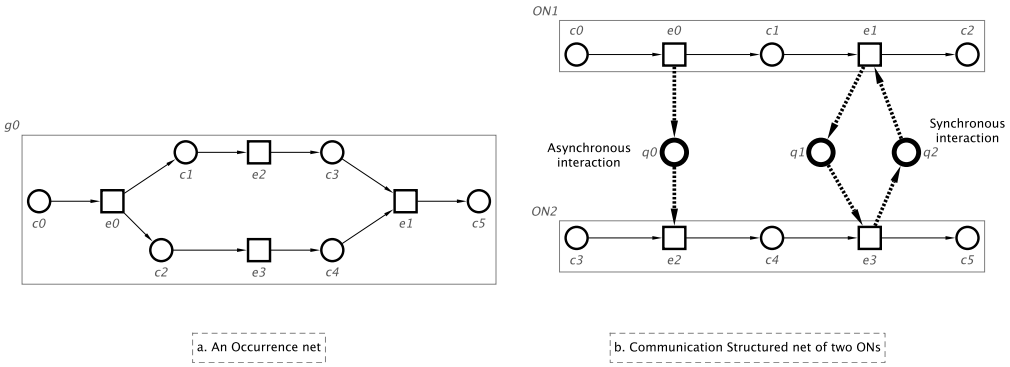

Fig. 2: a. Occurrence net. b. Communication Structured Net.

2.2 Communication in (SONs).

Communication in SONs is of two types: asynchronous and synchronous [8,11]. Figure 2 (b) shows a communication structured occurrence net (C-SON) which consists of two occurrence nets, namely $ON1$ and $ON2$. Figure 2 (b) depicts asynchronous communication which is represented through the dashed arc between two events in different ONs, e.g., ($e0$) and ($e2$). The second type of communication within SONs is synchronous communication which is represented by the two arcs between events ($e1, e3$) via the two arcs via channel place ($q1$ and $q2$) [8, 11]. Thus, in any execution consistent with the causal relationships captured by (C-SON), ($e0$) will never be executed after ($e2$), although the two events can be executed simultaneously, whereas ($e1$) and ($e3$) will always be executed simultaneously. SONs are underpinned by causal structures which extend causal partial orders with additional ordering, called *weak causality*. Two

events ordered in this way can be executed in the order given or simultaneously. Moreover, if two events are weakly ordered in both directions(this means, in particular, that weak causality is not assumed to be acyclic), then they can only be executed simultaneously. In SONs, weak causality results from passing tokens through channel places, whereas the non-channel places introduce the standard causality, as in ONs. In Figure 2 (b), ($e0$) weakly causes ($e2$), and the events ($e1$) and ($e3$) form a weak causality cycle.

2.3 Domain Name System (DNS)

A DNS or Domain Name System assigns domain names and then maps each of them in accordance with the existing name servers, which host each respective domain [13]. At the same time, due to various reasons, network administrators are allowed to delegate authority over some sub-domains of a given name space to other name servers. This permission was given so that the existence of a single, large, centralised database would not occur; at the same time, the resulting services are fault-tolerant [13]. Additionally, DNS also defines the technical parameters of each database service which is situated at its core. These parameters are part of what is known as the DNS protocol, meaning a detailed specification of both the data structures within the DNS, as well as communication between these various structures. Both these elements make up the Internet Protocol Suite. As a broad rule, two main name spaces are maintained across the internet, namely name hierarchy [13] and IP or Internet Protocol address spaces. DNS accomplishes two tasks: on one hand, it ensures the domain name hierarchy is kept, while providing translation services between it and a given address space, on the other hand. To function properly, DNS relies on multiple servers as well as on a communication protocol. Records for a specific domain are stored on a particular type of DNS server called a DNS name server; it answers to queries addressed to databases within it. Once a user types in any web address in the browser, for instance www.google.com, the current, local DNS server (resolver) will check whether or not the address is stored within it and then sends a response to the user, with the IP address of that particular website. However, if the resolver does not recognise that domain, it will send the query to the DNS root name server. As a result, the DNS root server will send a response back to the resolver, telling it the domain name (google.com) belongs to name server for .com LTD. Afterwards, the resolver will send the query to the DNS server google.com, asking about the IP address of www.google.com. As a result, the .com name server will respond with an IP address for google and the resolver will receive it and open the website for the user [6, 13]. Figure 3 illustrates this process.

Name servers The maintenance of Domain Name Systems is ensured through distributed database systems, which use client - server models. In this cases, database nodes consist of name servers. At the same time, for each existing domain, one or more given DNS servers provide data concerning the respective

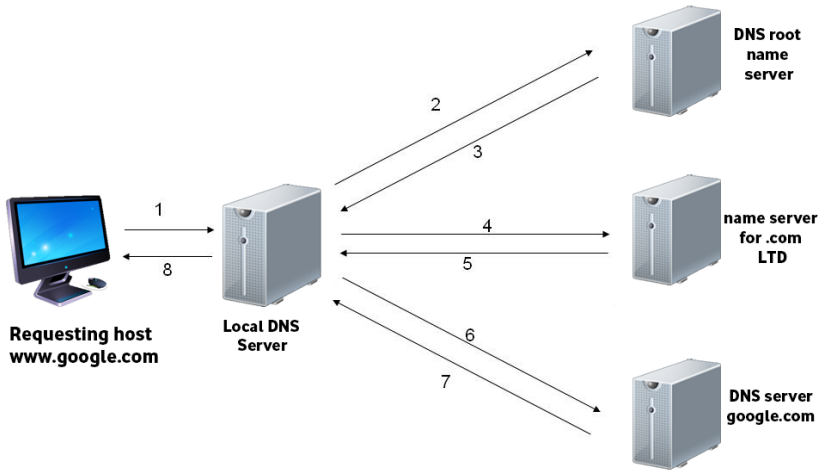

Fig. 3: How DNS Resolves Names and IP Address. (The labels on arcs indicate the steps of DNS protocol)

domain and the servers of domains which are subsequent to that respective domain. At the top of this hierarchy, we find root name servers, which typically receive queries when a TLD must be solved [6, 13].

Location transparency Location transparency helps to identify existing network resources, through names, rather than through location [6,13]. For instance, a user gains access through a unique file; however, actual data is reserved in sectors which are dispersed either across an network or within a local computer. Within such a system, the actual location of a file is not relevant to the user; however, a distributed system is required, so as to ensure a naming scheme for the available resources. One of the essential gains of using location transparency is that location is indeed irrelevant. Considering the given network setup, a user can obtain files from any computer connected to the network. As such, the actual location of a given resource does not matter anymore, thus creating the impression that the whole ensemble is accessible from any computer terminal, further boosting software upgrades [13]. Using location transparency also provides considerable versatility. This means system resources can be shifted from one computer to another, within the network, without disrupting the network.

DNS records The DNS protocol typically uses multiple record types for various purposes. The most common type of such records is A, which essentially switches domain names into IP addresses. For example, AAAA records accomplish a similar role for IPv6 protocols. Other record types, such as CNAME, are known as aliases. This type of record directs towards another defined domain name, but never towards an IP address. At the same time, one IP address can have several domain names or aliases. NS stands for NameServer; it keeps the address of the authoritative NameServer which can resolve any sub-domains for that

respective domain. At the same time, TXT records link any given text with a specified domain name, while MX records provide information regarding mail servers.

3 Related work

The review of the existing literature concerning DNS tunnelling led us to conclude that there are two main types of analyses when it comes to this phenomenon: traffic analysis and payload analysis. The focus of our research pertains to the former.

3.1 Traffic Analysis

DNS traffic volume per IP address Traffic analysis detects attacks by monitoring traffic generated by specific IP addresses (Pietraszek, 2004). The idea is simple: each tunnelled data request takes up to 512 bytes, a complete communication will imply sending many requests (Van Horenbeeck, 2006). As such, a server will continuously send the request, provided the client polls the server [5].

DNS traffic volume per domain The other fundamental method for detecting DNS tunnelling involves analysing the amount of traffic to a particular domain. Since utilities are designed to tunnel data through specific domain names, this method can assess the possibility of DNS tunnelling (Butler, 2011), unless the attack is configured through multiple domain names [5].

Number of Hostnames per Domain Guy (2009) suggests the number of hostnames per domain may indicate the possibility of DNS tunnelling. In such an event, each request has a unique hostname, meaning there are far more hostnames as compared to a legitimate domain name. This method can be tailored to determine an optimal threshold, since various tunnelling methods use various numbers of hostnames [5]. However, our approach is to check each domain and sub-domain, thus identifying the optimal threshold, both through logistic recognition, as well as through a powerful SON feature which optimises causality and communication features.

Domain history Domain history can help detect DNS tunnelling, as it raises suspicions about DNS traffic. It is used to detect tunnelling by determining when A or NS records were added, since a given domain may have recently been acquired for DNS tunnelling purposes, meaning its NS was recently added (Zrdnja, 2007). This method is used to detect domains involved in malicious activities.

Visualisation Visualisation can also be used to detect DNS tunnelling (Guy, 2009). It involves interactive, analyst driven work [5]. By contrast, our SON-based visualisation approach illustrates results automatically [3].

4 Preprocessing Datasets

4.1 Data collection

Data collection constitutes the initial stage of our work. The first step consists of discovering how DNS packets work, how they behave and what their structure is. This step actually consists of two stages which have been conducted in parallel: first, the extensive studying of literature pertaining to the subject and second, capturing real data from a local DNS server. Initially, we captured normal DNS packets. Afterwards, we modelled the normal packet in our model SON. Table 2 illustrates the DNS packet and Figure 4 shows the result. This type of data can

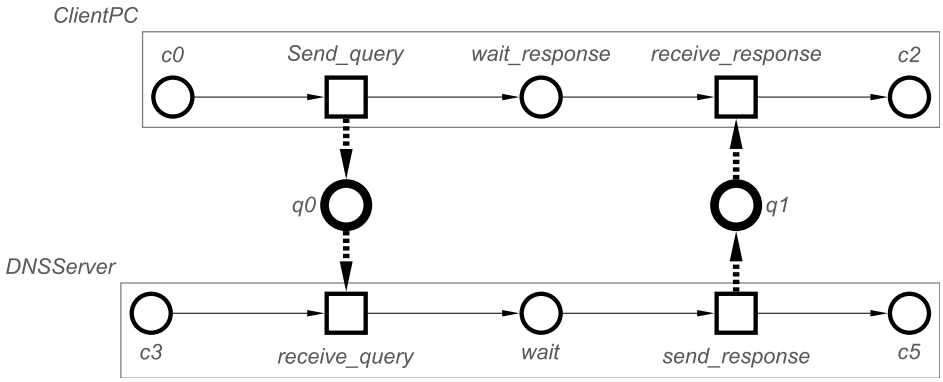

Fig. 4: Normal DNS packet SON model

be modelled in SONs; however, the "Info" field has various information which must actually be split into different fields. The reason behind this split is to distinguish whether the data packet is either query or response. Additionally, this split serves to distinguish the domain names, but also to determine the packet's id (0x81fe), since it is this ID which is our indicator, namely the one which links each packet (query) with its response. For instance, row 26 is the query and it is linked with row 29, which is the response for the ID packet in question (0x81fe).

Table 2: DNS Packets

No.	Source	Destination	Info
26	172.20.10.2	172.20.10.1	query 0x81fe A google.com
29	172.20.10.1	172.20.10.2	response 0x81fe A google.com

We developed a python script [2] to deal with this particular scenario. The main idea of this script is to split the "Info" field into 3 subfields, namely operation (which is either query or response), "PacketID" and finally, "Domain", as shown in the table 3. Afterwards, we modelled the last result of a normal packet, to ensure how it will be represented within the SON, as shown in Figure 4.

Table 3

No.	Source	Destination	Operation	PacketID	Domain
1	172.20.10.2	172.20.10.1	query	0x81fe	google.com
2	172.20.10.1	172.20.10.2	response	0x81fe	google.com

Moreover, during the experiments, we collected attack packets, the result of DNS tunnelling attempts. This was, in fact, the second challenge which we discovered and treated. Table 4 represents one such attack packet. As noticeable, if we attempt to model this packet, it will look the same as the normal packet, described above. Due to this, the point of the research is to discover an idea or algorithm which allows the clear distinguishing between normal packets and attack packets. The idea is one must examine each particular domain and its sub-domains, with the purpose of discovering whether or not a specific packet is an attack packet or not.

Table 4

No.	Source	Destination	Operation	PacketID	Domain
1	172.20.10.2	146.185.138.197	query	0xc7cb	paaac5ay.tunnel.carn.us.to
2	146.185.138.197	172.20.10.2	response	0xc7cb	paaac5ay.tunnel.carn.us.to

As such, the python script was updated to split the fields with which we are dealing with, in this case. The main idea for the updated script version is that it loops on each domain field and links it by a unique ID. It is called GroupID. As a result, afterwards, we can identify each chunk of the domain and its sub-domains by that GroupID, as Table 5 illustrates.

Table 5

No.	Source	Destination	Operation	GroupID	Domain
1	172.20.10.2	172.20.10.1	query	001	google.com
2	172.20.10.1	172.20.10.2	response	001	google.com 216.58.206.206
3	172.20.10.2	172.20.10.1	query	001	maps.google.com
4	172.20.10.1	172.20.10.2	response	001	maps.google.com
5	172.20.10.2	146.185.138.197	query	002	paaac5ay.tunnel.carn.us.to
6	146.185.138.197	172.20.10.2	response	002	paaac5ay.tunnel.carn.us.to
7	172.20.10.2	146.185.138.197	query	002	paaydani.tunnel.carn.us.to
8	146.185.138.197	172.20.10.2	response	002	paaydani.tunnel.carn.us.to

Table 5 illustrates the link between the “Google” domain and its subdomains. For instance, we assume “google.com” and “maps.google.com” have the same group id, namely “001”. However, subdomains (paaac5ay) and (paaydani) have been grouped to ”.us.to” via group id, which is actually “002”. So, the idea is to initially identify the domain, i.e. “google” and loop to find any particular sub-domain of “google” and link it with a unique group id.

4.2 Mixing data

In order to simulate a real scenario of DNS server behaviour, the collected data consisted of two main parts. The first part included the data collected during the attack experiment. The second part includes normal data, collected from the university. For this data to make sense, and in order to simulate a real scenario example, we created python script [1] mixing these two kinds of data. This data mix will be evaluated in different stages in the evaluation section.

5 Proposed Solutions

5.1 Detection DNS Tunnelling using (SONs)

The main idea of our algorithm to detect DNS tunnelling is to model DNS data i.e., each particular transaction and the way it will communicate with the local DNS server. We assume that each packet is a unique ON. And the local DNS server is one ON. Each packet (ON) communicates with the local DNS server by channel place (thick circle) via events ($Send - query$ and receive−query), as shown in figure 4 . The client (ONc) sends a query to local DNS server (ONserver), and then DNS server responds to the client as shown in Figure 4. This is the normal packet. When we were to model the attack packet, we found it is the same as the normal one in terms of communication and behaviour. The reason is that we examined each packet as a separate one. However, to detect an attack we need to examine each domain and its sub-domains in order to examine wither we see normal or abnormal packets. For instance, if google.com, map.google.com and developers.google.com are involved, then we need to deal with all google domains and its sub-domains, and model them as one of chunk of domain packets. So, we group each domain and its sub-domains with unique ID as in prepossessing section above. The result is seen in Figure 5, where we can distinguish between normal and abnormal packets (we displayed the attack ON very small to fit the width of the text; however, the ON transaction is actually bigger).

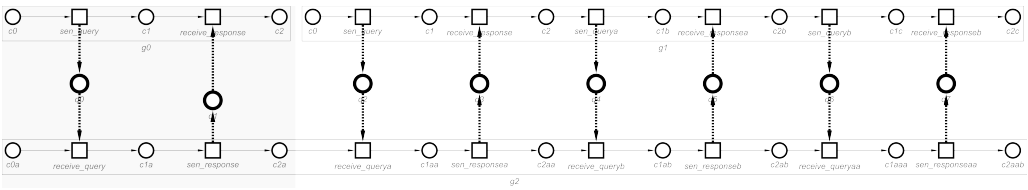

Fig. 5: Normal and Abnormal ONs

5.2 Detection DNS Tunnelling algorithm

The main idea behind the algorithm is first to detect any ON input, whether it is a normal input or an ON representing a DNS attack. Before going more in depth, it should be mentioned that ONs refer to DNS packets. Furthermore, we assume the local DNS local server to be another ON. It does not matter whether inputs are normal or representing a DNS attack. We examine each ON input and count its events which communicate with the DNS Server ON. If the number of those ON events is less than threshold value, we flag it as a normal ON. Otherwise, we flag it as an abnormal ON. Then, we check whether or not we had abnormal ONs; if so, then we know a DNS attack has occurred. Finally, we should mention that the aforementioned threshold was identified via logistic regression, a widely used statistics model. The idea behind this binomial model resides in properly estimating the parameters of the logistics model. From a mathematical standpoint, the model relies on variables with two possible values, for instance "win" or "lose" or "pass" or "fail". As such, we applied logistic recognition to available data in order to estimate the threshold, and estimate how many events within a particular GroupID could be suspicious.

The calculated threshold It is at the core of the logistic regression model. From a mathematical standpoint, logistic regression performs multiple linear regression functions of the required feature, such as the number of events. These functions are based on the dependent, dichotomous variable, i.e. "attack" or "no attack". The aforementioned functions produce a threshold by using the *logit* equation. Finally, this threshold can be used in our algorithm. The mathematical aim of linear regression functions is to build an equation with the following coefficients: (a) and (c), as follows: $y = ax + c$, where (y) represents the probability of an attack and (x) is the number of events. Afterwards, the values for both the number of events and the dependent variable, namely "attack" or "not attack", will be inserted; this will allow us to perform regression analysis, meaning we will be able to calculate the values of (a) and (c), which in turn will increase the likelihood of (y) conforming to ground truth values. Once these values are known, they can be used to calculate the threshold.

Algorithm 1: Detection DNS Tunnelling algorithm using SON (Part1)

```
Input: set_of_ons : Set of All Ons
       dns_server_on : DNS Server ON
       attack_max_events : DNS Server threshold to interpret an
       attack
Output: set_of_normal_ons = Set of normal ONs, not presenting a
        potential attack.
        set_of_abnormal_ons = Set of abnormal ONs, presenting a
        potential attack.
Interpretation:
 if size of (set_of_abnormal_ons) > 0 then
    DNS Attack;
else
    No DNS Attack;
end
Start ;
array set_of_normal_ons = [] ;
array set_of_abnormal_ons = [] ;
integer set_of_normal_ons_index = 0 ;
integer set_of_abnormal_ons_index = 0 ;
integer connected_events = 0 ;
for (count_event(current_on)) < attack_max_events)) do
    connected_events_count = 0;
    if (count_event(current_on)) < attack_max_events)) then
        set_of_normal_ons[set_of_normal_ons_index] = current_on;
        set_of_normal_ons_index = set_of_normal_ons_index + 1;
        continue;
    else
        for (Event current_event in current_on's events) do
            if (current_event is connected to dns_server_on) then
                connected_events_count = connected_events_count +1;
            end
        end
        if (connected_events_count >= attack_max_events) then
            set_of_abnormal_ons[set_of_abnormal_ons_index] =
             current_on; set_of_abnormal_ons_index =
             set_of_abnormal_ons_index + 1;
        else
            set_of_normal_ons[set_of_normal_ons_index] = current_on;
             set_of_normal_ons_index = set_of_normal_ons_index + 1;
        end
    end
end
```

Algorithm 1: Detection DNS Tunnelling algorithm using SON (Part2)

```
return structure : {set_of_normal_ons , set_of_abnormal_ons} ;
count_event(ON on) function :;
  Input: on : ON
  Output: events_count : number of events of ON.;
Start ;
integer vents_count = 0 ;
for (Event current_event in current_on's events) do
  | events_count = events_count + 1; ;
end
return events_count ;
End ;
```

6 Result Testing and Evaluation

We considered the following proportions of attacking packets (as percentage of the total packets):

0%, 1%, 5%, 10%, 20%, 40%, 60%, 80%, 90%, 95%, 99% and 100%.

In addition, we used the sensitivity and specificity methods [7], developed to evaluate a system of computer-assisted diagnosis. Hence we used a tried and tested method, with the only change being made to the terminology, as follows:

First, True positive (TP) result: attack transactions were correctly identified as attacks.Then, False positive (FP) result: normal transactions were incorrectly identified attacks.And, True negative result (TN) : normal transactions were identified as normal transactions.Finally, False negative (FN) result: attack transactions were incorrectly identified as normal transactions. In terms of the data mix (of normal and attack packets), we created a Python script [1]. The idea behind the script was to mix the two data sets, namely: the normal and attack packets. Each time we ran the script, we asked the script to mix the data, relying on the percentage from each file, such as 10% normal and 90% attack; as a result, we obtained one file with mixed data. The main data sets for each packet type, i.e., normal or attack packets, were collected during a five-minute time span. The data for both sets consisted of roughly 700 DNS packets. After running the algorithm, we checked the sensitivity and specificity through an evaluation method, obtaining the following results. As evident in (1) in Table 6, in True Positives and False Negatives, we assumed that there was no result because this data set had no attack packets.

However, in (2) in Table 6, when the data set had 1% attack and 99% normal packets, we got a 0% True Positive, which means there were no attacked packets detected as attack packets, although we had 1% attack data. This is because the number of packets in this data set was less than the threshold. Regarding False Negatives, we also got 0% because there were no attack packets detected as normal packets. Likewise, in (3) in Table 6, the same situation applied. However, in (4) in Table 6, the algorithm detected 7.1% from a total of 10% of the attack data; however, it failed to detect 2.9% of the attack packets. After we checked

the data sets manually, we found that most of these packets were only request queries to the DNS server, to which the DNS server did not respond, so our algorithm skipped them. However, from this 2.9%, the algorithm did not attacked them, although they have query and response packets. In future research, we will further investigate this issue in order to discover why this happened. Another interesting point is that, in (4) in Table 6, there is a False Positive, which means the algorithm detected 9.4% of the normal packets as attack packets; after investigating this, we found that Google had sent many requests to the DNS server as unusual requests, and these request events were above our threshold. When we rerun the experiment, we did not face this issue. Moreover, as evident in (7) in Table 6, no normal packets were detected as attack packets (False Positives). Finally, in (15) in Table 6, the False Positives and True Negatives were "N/A"; this was because, in these data sets, there were no normal packets at all.

Table 6: Table results

No.	% normal and abnormal packets	TP	FP	TN	FN
1	0% attack, 100 normal	N/A	9.7%	90.92%	N/A
2	1% attack, 99 normal	0%	9.77%	90.92%	0%
3	5% attack, 95 normal	0%	8.9%	91.1%	0%
4	10% attack, 90 normal	7.1%	9.4%	90.6%	2.9%
5	20% attack, 80 normal	82.9%	5.3%	94.7%	17.1%
6	30% attack, 70 normal	86.7%	5.3%	94.7%	13.3%
7	40% attack, 60 normal	88.5%	0%	100%	11.5%
8	50% attack, 50 normal	90.9%	0%	100%	9.1%
9	60% attack, 40 normal	91.5%	0%	100%	8.5%
10	70% attack, 30 normal	91.1%	0%	100%	8.9%
11	80% attack, 20 normal	91.5%	0%	100%	8.5%
12	90% attack, 10 normal	91.8%	0%	100%	8.2%
13	95% attack, 5 normal	92%	0%	100%	8%
14	99% attack, 1 normal	92.3%	0%	100%	7.7%
15	100% attack, 0 normal	92.5%	N/A	N/A	7.5%

7 Concluding remarks

The paper proposes a novel solution to DNS tunnelling detection based on SONs. An detection algorithm has been designed and implemented. Also, data preprocessing and a set of experiments have been discussed. Further work will focus on improving the current algorithm, aiming at allowing it to work automatically in terms of reading with packets in real-time scenarios. In addition, we will model and develop algorithms dealing with more than one DNS server at a time, including all user packets. In addition, we will use other DNS tunnelling tools, instead of "iodine", in order to compare various attack behaviours and check how the algorithm deals with these types of attacks.

Acknowledgements

The authors would like to thank the reviewers for their comments on the paper.

References

1. Alharbi, T.: mix packets script. `https://github.com/talalsm/mix/` (May 2019)
2. Alharbi, T.: split packets script. `https://github.com/talalsm/mix/` (May 2019)
3. Alharbi, T., Koutny, M.: Visualising data sets in structured occurrence nets. PNSE 2018 (2018)
4. Born, K., Gustafson, D.: Detecting dns tunnels using character frequency analysis. arXiv preprint arXiv:1004.4358 (2010)
5. Farnham, G., Atlasis, A.: Detecting dns tunneling. SANS Institute InfoSec Reading Room **9**, 1–32 (2013)
6. Jung, J., Sit, E., Balakrishnan, H., Morris, R.: Dns performance and the effectiveness of caching. IEEE/ACM Transactions on networking **10**(5), 589–603 (2002)
7. Kapp, E.A., Schütz, F., Connolly, L.M., Chakel, J.A., Meza, J.E., Miller, C.A., Fenyo, D., Eng, J.K., Adkins, J.N., Omenn, G.S., et al.: An evaluation, comparison, and accurate benchmarking of several publicly available ms/ms search algorithms: sensitivity and specificity analysis. Proteomics **5**(13), 3475–3490 (2005)
8. Koutny, M., Randell, B.: Structured occurrence nets: A formalism for aiding system failure prevention and analysis techniques. Fundamenta Informaticae **97**(1-2), 41–91 (2009)
9. van Leijenhorst, T., Chin, K.W., Lowe, D.: On the viability and performance of dns tunneling. International Conference on Information Technology and Applications (2008)
10. Li, B., Randell, B., Bhattacharyya, A., Alharbi, T., Koutny, M.: Soncraft: A tool for construction, simulation, and analysis of structured occurrence nets. In: 2018 18th International Conference on Application of Concurrency to System Design (ACSD). pp. 70–74. IEEE (2018)
11. Li, B., Randell, B., Bhattacharyya, A., Alharbi, T., Koutny, M.: Soncraft: A tool for construction, simulation, and analysis of structured occurrence nets. In: 2018 18th International Conference on Application of Concurrency to System Design (ACSD). pp. 70–74. IEEE (2018)
12. Merlo, A., Papaleo, G., Veneziano, S., Aiello, M.: A comparative performance evaluation of dns tunneling tools. In: Computational Intelligence in Security for Information Systems, pp. 84–91. Springer (2011)
13. Paul, M., Dunlap, K.J.: Development of the domain name system. ACM **18**(4) (1988)
14. Qi, C., Chen, X., Xu, C., Shi, J., Liu, P.: A bigram based real time dns tunnel detection approach. Procedia Computer Science **17**, 852–860 (2013)
15. Randell, B.: Occurrence nets then and now: The path to structured occurrence nets. In: Kristensen, L.M., Petrucci, L. (eds.) Applications and Theory of Petri Nets. pp. 1–16. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)