DNS Tunneling Detection by Cache-Property-Aware Features

Naotake Ishikura, Daishi Kondo, *Member, IEEE*, Vassilis Vassiliades, Iordan Iordanov, and Hideki Tode, *Member, IEEE*

***Abstract*—Many enterprises are under threat of targeted attacks aiming at data exfiltration. To launch such attacks, in recent years, attackers with their malware have exploited a covert channel that abuses the domain name system (DNS) named DNS tunneling. Although several research efforts have been made to detect DNS tunneling, the existing methods rely on features that advanced tunneling techniques can easily obfuscate by mimicking legitimate DNS clients. Such obfuscation would result in data leakage. To tackle this problem, we focused on a "trace" left by DNS tunneling that cannot be easily hidden. In the context of data exfiltration by DNS tunneling, the malware connects directly to the DNS cache server and the generated DNS tunneling queries produce cache misses with absolute certainty. In this study, we propose a DNS tunneling detection method based on the cache-property-aware features. Our experiments show that one of the proposed features can efficiently characterize the DNS tunneling traffic. Furthermore, we introduce a rule-based filter and a long short-term memory (LSTM)-based filter using this proposed feature. The rule-based filter achieves a higher rate of DNS tunneling attack detection than the LSTM one, which instead detects the attack more quickly, while both maintain a low misdetection rate.**

***Index Terms*—Cache-property-aware features, data exfiltration, DNS tunneling, targeted attacks.**

I. Introduction

VARIOUS protocols exist on the Internet, and by exploiting their vulnerabilities, attackers using their malware launch targeted attacks that cause data exfiltration [2], [3]. One of the ways attackers can exfiltrate data from an enterprise network commences with sending emails that include the malware as seemingly harmless attachments to the employees of targeted enterprises. By opening these malicious emails, the employees unfortunately infect their computers with malware. This mistake establishes communication channels between the attackers and their malware. Then, the attackers can remotely control the malware and steal confidential information from the infected enterprises. This data leakage puts enterprises at a great disadvantage and affects profitability drastically.

In recent years, attackers with malware have launched this form of attack by exploiting a covert channel that abuses the domain name system (DNS) known as DNS tunneling [4]. DNS tunneling is a security threat used to tunnel data and commands by exploiting a domain name in DNS queries and the corresponding DNS responses. It is one of the top DNS-based attacks [5]. Between April and September 2014, the attackers stole 56 million debit and credit card numbers from the American retailer, Home Depot [6], and several attacks were launched against a Middle Eastern government organization in August 2018 [7]. In general, enterprises enforce access control of ports and protocols that are not usually utilized (e.g., peer-to-peer (P2P) file sharing such as BitTorrent) for employees. In addition, in a quarantine network that installs trusted middleboxes, end-to-end encrypted communications can be decrypted and inspected by middleboxes [8] that can identify malicious activities. However, because DNS is an indispensable protocol for implementing many services, such as content distribution, its use is not restricted and is poorly managed. Therefore, the DNS operation unfortunately provides attackers with malware an opportunity to realize targeted attacks through DNS tunneling.

To detect DNS tunneling, several countermeasures have been proposed [9]–[24]. Indeed, these methods are effective for detecting tunneling traffic from malware, such as `Morto worm` [25], or DNS tunneling tools such as `dnscat2` [26]. However, these countermeasures are built using features that can be easily obfuscated by advanced DNS tunneling techniques. For instance, steganography can hide leaked data in the fully qualified domain name (FQDN) of the tunneling query, which makes the FQDN look legitimate and invalidates filters relying on its features. Thus, this obfuscation would result in data leakage.

To address this problem, we focus on the nature of DNS tunneling. To successfully exfiltrate data attached to the domain name of a DNS query, the DNS cache server to which the malware connects directly must avoid producing a cache hit in the server; otherwise, the data cannot be leaked outside of

Manuscript received October 30, 2020; revised March 18, 2021 and April 28, 2021; accepted May 4, 2021. Date of publication May 10, 2021; date of current version June 10, 2021. This project has received funding from JSPS KAKENHI, Grant Number JP19K24351, the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 739578 and the Government of the Republic of Cyprus through the Deputy Ministry of Research, Innovation and Digital Policy. This article was presented in part at the ICIN [1]. The associate editor coordinating the review of this article and approving it for publication was C. Fung. *(Corresponding author: Daishi Kondo.)*

Naotake Ishikura and Daishi Kondo are with Department of Computer Science and Intelligent Systems, Osaka Prefecture University, Sakai 599-8531, Japan (e-mail: ishikura@com.cs.osakafu-u.ac.jp; daishi.kondo@cs.osakafu-u.ac.jp).

Vassilis Vassiliades is with the CYENS Centre of Excellence, 1500 Nicosia, Cyprus (e-mail: v.vassiliades@cyens.org.cy).

Iordan Iordanov is with Corpy & Co., Tokyo 113-0033, Japan (e-mail: iordan@corpy.co.jp).

Hideki Tode is with the Department of Computer Science and Intelligent Systems, Osaka Prefecture University, Sakai 599-8531, Japan (e-mail: tode@cs.osakafu-u.ac.jp).

Digital Object Identifier 10.1109/TNSM.2021.3078428

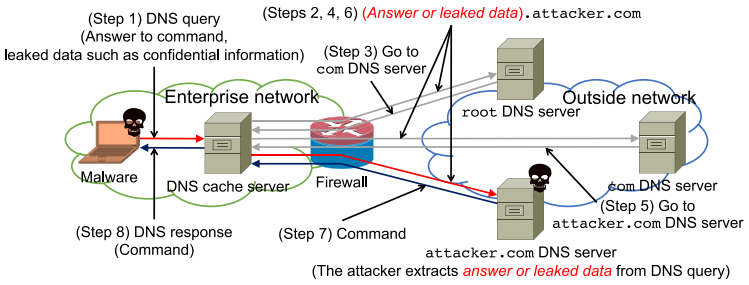


Fig. 1. An overview of DNS tunneling.

the enterprises. In other words, leaking data through DNS tunneling would trigger a cache miss on the DNS cache server. However, cache servers exist to exploit the natural tendency of humans to request the same information multiple times. We hypothesize that the number of queries satisfies Zipf's law [27]. Based on this hypothesis, DNS tunneling violates normal human behavior because it requires the cache to be bypassed, a clear indication that *cache misses are the actual footprint of a DNS tunneling attack*. Therefore, we believe that this *cache property* is more tolerant than the features used in conventional methods to counter feature obfuscation.

Considering the above facts, we propose three features derived from the cache property: *cache hit ratio*, *access hit ratio*, and *access miss count*. Through extensive experiments, we demonstrate that the access miss count addresses some shortcomings of the hit ratios of both the cache and access and clearly characterizes DNS tunneling traffic. Therefore, it is useful for designing and implementing a solid DNS firewall against DNS tunneling. Based on this knowledge, we introduce a rule-based filter and a long short-term memory (LSTM) [28]-based filter using the proposed feature. The rule-based filter achieves a higher detection rate of DNS tunneling attack than the LSTM filter, which instead detects the attack faster, while both maintain a low misdetection rate.

To the best of our knowledge, our previous work [1] was the first to analyze cache-property-aware features, and this paper is an extended version. We extend the previous work with the following contributions.

- performing a comprehensive survey of DNS tunneling research in terms of attack and detection methods,
- introducing a new cache-property-aware feature, access miss count, and comparing this feature with the cache hit ratio and access hit ratio,
- proposing a rule-based filter and an LSTM filter based on the access miss count against DNS tunneling, and
- evaluating the performance of the filters created by a large legitimate training dataset composed of more than 350,000 DNS queries on the test dataset including legitimate queries and DNS tunneling ones.

The remainder of this paper is organized as follows. Section II is the summary of the basics of DNS tunneling and several existing studies on the attack and detection methods of DNS tunneling. Section III proposes cache-property-aware features for DNS tunneling detection while Section IV performs monitoring and analysis of these features. Using one of the features of DNS tunneling, a rule-based filter and an LSTM filter are proposed in Section V. The performance of the filters is evaluated in Section VI. We discuss our findings in Section VII and conclude the paper in Section VIII.

II. Background

A. DNS Tunneling Basics

DNS tunneling bypasses firewalls to send and receive data by exploiting the domain names included in the DNS query and the corresponding DNS response. The data and commands are tunneled between the malware and the attacker in the context of targeted attacks causing data exfiltration (Fig. 1). Assuming a domain name `attacker.com` is shared to create a covert channel between the attacker and the malware that has infiltrated the enterprise network, to obtain a command from the attacker to search confidential information in the enterprise network, the malware generates an FQDN *(get_command)*`.attacker.com` and sends it as a DNS query to the DNS cache server in the enterprise network (Step 1). Following the usual process of resolving an FQDN, the DNS cache server iteratively queries the `root` (Steps 2 and 3), the `com` (Steps 4 and 5), and the `attacker.com` DNS server (Step 6). Then, the `attacker.com` DNS server obtains the request (*get_command*) and replies with a suitable DNS response containing the command to the malware via the DNS cache server (Steps 7 and 8). After repeating the process of obtaining a new command and sending an answer to the command, the malware eventually leaks the confidential information collected by the attacker in the same manner (i.e., by including the information to be leaked in the domain name).

When a DNS client resolves the domain name by sending the DNS query, the query first reaches the DNS cache server. If the corresponding DNS response is cached in the server, it is a cache hit, that is, the response is directly returned from the server; otherwise, it is a cache miss, that is, the DNS query is forwarded to the upstream DNS servers (Fig. 2). For the malware to send malicious DNS queries to the attacker effectively, the queries must not cause a cache hit on the DNS cache server; this is a fundamental characteristic of DNS tunneling. In this study, we assumed that the exfiltrated data are similar to credit card information (such an attack scenario is also considered in [18], [21], [23]), and all the FQDNs generated to leak such data are unique.

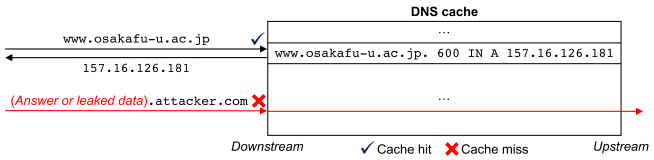


Fig. 2. Structure of DNS cache.

TABLE I
DNS TUNNELING TOOLS AND MALWARE DATASET

DNS tunneling tools and malware dataset	References
`Backdoor.Win32.Denis` [35]	[21]
`BernhardPOS` [36]	[23]
`Cobalt Strike` [37]	[17], [24]
Dataset collected at laboratory, university, enterprise, `alexa.com`, etc.	[9]–[13], [15]–[24]
`DET` [38]	[20], [23], [24]
`DNScat` [39] or `DNScat2` [26]	[12], [17], [18], [20], [22], [24]
`DNSExfiltrator` [40]	[20], [24]
`DNSMessenger` [41]	[23]
`DNSpionage` [42]	[23]
`dns2tcp` [33]	[9], [15], [16], [21], [22], [24]
`FrameworkPOS` [43]	[21], [23]
`iodine` [32]	[9]–[11], [17], [19]–[24]
`OzymanDNS` [44]	[22], [24]
`Pick Pocket`	[17]
`Reverse DNS Shell` [45]	[22], [24]
Security vendor dataset	[14]
`TCP-over-DNS` [46]	[9]

Some researchers [29]–[31] have evaluated the performance (e.g., throughput) of DNS tunneling tools such as `iodine` [32] and `dns2tcp` [33]. Raman *et al.* [34] proposed a DNS tunneling method and measured the maximal throughput.

B. Conventional Attack Methods

The existing DNS tunneling tools and malware datasets examined in related works [9]–[24] are summarized in Table I. In the attacks detected and evaluated in these related works, the time interval between consecutive tunneling queries was sometimes short; furthermore, in some of the works, anomalous/malware datasets were presented without description. For instance, Ellens *et al.* [10] customized `iodine` and performed a simulation to generate 36,389 flows of Command & Control (C&C) communication in 1.55h, or more than 6.5 C&C flows per second. In addition, some of the attacks examined produce encrypted FQDNs that were longer than average. These characteristics make it easy to detect the existing DNS tunneling. However, it is questionable whether conventional detection methods are effective against stealthy DNS tunneling attacks with low throughput and legitimate-looking FQDNs.

Meanwhile, several researchers have discussed DNS tunneling produces queries slowly and does not exhibit characteristics such as encryption. Xu *et al.* [13] converted commands for C&C communication in anomalous FQDNs to seemingly legitimate labels used in general services (e.g., `www`, `mail`, and `ftp` that correspond to the commands). Such an attack method can achieve C&C communication by obfuscating the features related to the FQDN used to detect attacks, as described in Section II-C. Paxson *et al.* [11] embedded one-bit information in an FQDN (e.g., `www.attacker.com` transfers 0 and `mail.attacker.com` transfers 1 at an interval of one query per day). This method can obfuscate the features derived from the time interval and FQDN for detecting the attack, as described in Section II-C. Then, data exfiltration can be realized by adding the sequence number replaced with a general character string to the above FQDN. It is difficult to distinguish these DNS tunneling attacks using the existing methods.

TABLE II
CLASSIFIERS

Classifiers	References	Types
Rule-based method	[9]–[14], [16], [20]	R
Bayes	[15]	S
Convolutional neural network	[22]	S
k-nearest neighbor	[15]	S
Logistic regression	[18]	S
Long-short term memory	[24]	S
Neural network	[15], [19]	S
Random forest	[17]	S
Support vector machine	[15]	S
Isolation forest	[21], [23]	U
k-means clustering	[18]	U

R: rule-based, S: supervised learning, U: unsupervised learning

C. Conventional Detection Methods

DNS tunneling is detected using classifiers and features. A summary of the classifiers used in the related works [9]–[24] is presented in Table II. The most common method is rule-based detection based on a defined threshold for a certain feature. However, rule-based models can neither create complicated rules nor deal with tunneling tools designed to bypass the defined rules. In conventional research, machine learning techniques are often adopted as classifiers. In the case of supervised learning, the detection method is effective only for specific malware and DNS tunneling tools, which limits the versatility of the models. In recent years, unsupervised learning has been adopted to detect DNS tunneling, as it is significantly more suitable for anomaly detection than supervised learning because the training process requires only legitimate data and it is applicable to a wider range of problems. However, DNS tunneling cannot be detected without using a feature that can distinguish anomalous traffic from legitimate ones because unsupervised learning only detects the outliers for legitimate traffic.

The features (the inputs for the classifiers) proposed in the related works [9]–[24] are summarized in Table III. In general, there are two types of analyses for DNS tunneling detection: payload analysis and traffic analysis. The payload analysis is an evaluation of a DNS query and/or the corresponding DNS response and traffic analysis is an evaluation of DNS traffic over a monitoring period (e.g., in terms of time and number of samples). These features are used as thresholds or fed to machine learning models to build countermeasures for DNS tunneling detection. However, the above-mentioned detection methods rely on features that attackers and their malware can easily obfuscate by mimicking benign entities. For instance, an analysis of character frequencies and entropy can be bypassed using steganography. Such an obfuscation would result in data

TABLE III
FEATURES

Features	References	Types
Answer counts in a response	[17]	P
Entropy of content of the TXT record in the response	[18]	P
Entropy of FQDN	[20], [23]	P
First 128 characters of FQDN	[24]	P
First 300 bytes of a DNS request	[22]	P
First 512 bytes of a DNS response	[19]	P
FQDN length	[23]	P
Label counts in FQDN	[17], [23]	P
Longest label length in FQDN	[17], [23]	P
n-gram character frequencies of domains or subdomains	[9], [12]	P
Queries that hit the local DNS resolver cache	[11]	P
Query length	[17]	P
Query types	[11], [17], [20]	P
Ratio of numerical characters to total characters in FQDN	[17]	P
Time interval between query and response	[17]	P
Time interval between responses of specific domain	[17]	P
Access counts of resource records	[14], [17]	T
Averages and variances of query size, response size, and time interval between a query and its response	[15], [16]	T
Combination of principal component analysis (PCA) and mutual information	[16]	T
Entropy of FQDN per domain	[20], [21]	T
Flow-based features such as bytes per flow and average bytes in a packet calculated per flow	[10]	T
Frequencies per domain	[20]	T
Jensen-Shannon divergence computed from the DNS payload	[13]	T
Length of concatenated FQDN	[18]	T
Longest meaningful word over average domain length per domain	[21]	T
Query length per domain	[20], [21]	T
Unique query ratio and volume per domain	[21]	T
Volume of queries per domain	[20]	T

P: payload analysis, T: traffic analysis

TABLE IV
CRITERIA FOR MAKING FEATURE VECTORS (INPUTS) FOR CLASSIFIERS

Criteria	References
Client	[13], [17]
Domain	[11], [14], [18], [20], [21]
Flow	[10]
Query and/or response	[9], [12], [15], [16], [19], [22]–[24]

leakage. Therefore, it is necessary to propose a resilient feature to detect DNS tunneling by focusing on a "trace" of DNS tunneling that cannot be easily concealed.

The output of the classifiers in the related works [9]–[24] is the label (i.e., legitimate or malicious/anomalous). The classifiers are evaluated based on performance metrics, such as the true positive rate and accuracy, which are computed on the basis of the confusion matrix. The criteria for making the feature vectors (inputs) for classifiers in the related works are summarized in Table IV. The focus in most existing studies is on a query and/or response and their legitimateness.

III. CACHE-PROPERTY-AWARE FEATURES FOR DNS TUNNELING DETECTION

Fujiwara *et al.* [47] reported that the cache hit ratio on the DNS cache server[1] inside the University of Tsukuba in November 2011 was 75.1%. During a DNS tunneling attack, the cache hit ratio is expected to decrease because the malicious DNS queries generated to exfiltrate data cause cache misses, as discussed in Section II-A. To the best of our knowledge, this characteristic of DNS tunneling has not been investigated in related works, as demonstrated in the review in Section II-C. In this section, we propose three features based on the cache property to identify DNS tunneling traffic.

A. Cache Hit Ratio

The first feature we propose is the *cache hit ratio* CHR^n on the DNS cache server, which is defined as follows:

$$CHR^n = \frac{1}{n} \cdot N^n_{CH}.$$

Here, n is the number of queries under observation (i.e., a window size), and N^n_{CH} is the number of successful cache hits within the n observed queries. Note that in this paper, we define the cache hit as a state in which the response to a query from the DNS client is discovered in the connected DNS cache server without sending any queries to authoritative DNS servers. Experimental results using time-series data are reported in Section IV-B, where the plots of *CHR* derived from all generated queries in a sliding window manner are shown.

CHR is a naive feature derived from the cache property to identify DNS tunneling traffic; it has two shortcomings. *First*, *CHR* is not improved by the caches of resource records that a client query rarely looks up. According to [47, Table 2], 90.7% of queries from the DNS clients were for the A and AAAA records, and therefore, caching these resource records can increase *CHR*. However, resource records, such as NS

[1]The authors defined the cache hit ratio as (Total # of client queries that do not cause any queries to authoritative DNS servers)/(Total # of client queries).

TABLE V
COMPARISON BETWEEN A CACHE ENTRY AND AN ACCESS ENTRY

	Purpose of creation	Stored information	Entry eviction policy
Cache entry	DNS response	Resource record	TTL + LRU
Access entry	DNS query	FQDN	LRU

records, which are not often looked up by DNS clients, are also cached to mitigate the load of authoritative DNS servers. Such records play no role in DNS tunneling detection (i.e., unnecessary caches for DNS tunneling detection); therefore, they might induce a decrease in *CHR*.

Second, in addition to the use of caching algorithms, such as least recently used (LRU), caches are evicted based on their time to live (TTL). Fujiwara *et al.* [47] show that defining a low TTL value ($\leq$300 sec), where DNS-based wide-area load balancing is achieved [48], [49], decreases the *CHR*. Furthermore, the TTL itself essentially causes a cache miss and does not aid in characterizing the DNS tunneling traffic. It is therefore difficult to determine whether a decrease in the *CHR* is attributable to DNS tunneling or the inherent shortcomings of the *CHR*.

B. Access Hit Ratio

To compensate for the shortcomings of the *CHR* described in Section III-A, we first propose an *access entry* that inspects client queries and stores minimal information, only the FQDNs. The entry eviction policy is the LRU. When an FQDN in the client query is found in a given list of access entries, this can be considered as an *access hit*. Table V summarizes a comparison between the cache entry and the access entry.

We propose a second feature, which we call the *access hit ratio* AHR^n, defined by the following formula:

$$AHR^n = \frac{1}{n} \cdot N_{AH}^n.$$

Here, n is the number of queries under observation (i.e., a window size), and N_{AH}^n is the number of successful access hits within n queries. Experimental results obtained from the time-series data are reported in Section IV-B, showing the plots for the *AHR* derived from every generated query in a sliding window manner.

However, malware can intentionally increase the *AHR* by sending a large volume of legitimate DNS queries whose FQDNs are normally expected to be stored in the access entry. In this case, the malware can send tunneling queries while hiding its activity, which is a shortcoming of *AHR*.

C. Access Miss Count

To compensate for the shortcoming of the AHR described in Section III-B, we focus only on the access misses and propose a third feature, which we call the *access miss count* AMC^t, defined by the following formula:

$$AMC^t = N_{AM}^t.$$

Here, t is a time interval whose unit is seconds, and N_{AM}^t is the number of access miss queries in that time interval.

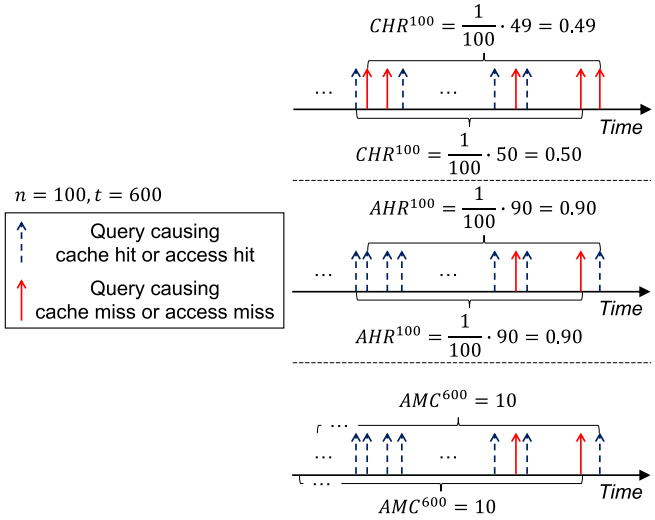


Fig. 3. Calculation example of *CHR*, *AHR*, and *AMC* ($n = 100$, $t = 600$).

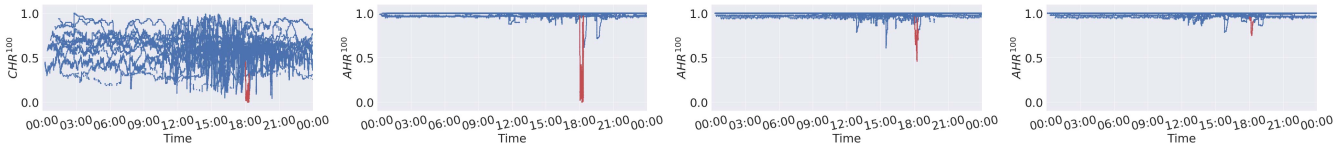
Experimental results from the time-series data are reported in Section IV-B; the plots for the *AMC* derived from every generated query are shown in a sliding window manner.

Fig. 3 is the summary of a calculation example of *CHR*, *AHR*, and *AMC*. These experimental results are shown in Section IV-B. As described in Section III-A, not only the tunneling queries but also the legitimate ones cause cache misses, and it is difficult to distinguish whether a decrease in the *CHR* is caused by a malicious event such as DNS tunneling. By proposing an access entry and *AHR*, we compensate for the shortcomings of the *CHR*. However, malware can intentionally replay legitimate DNS queries to increase the *AHR*. In addition, both ratio-based features normalize the number of cache hits and access hits, respectively, which means that the number of access misses itself cannot be evaluated based on these features. Finally, *AMC*, a count-based feature, solves these shortcomings.

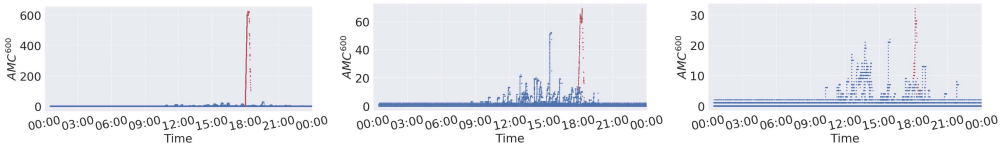
IV. MONITORING AND ANALYSIS OF CACHE-PROPERTY-AWARE FEATURES

A. Experimental Setup

For our DNS traffic monitoring experiments, we installed a DNS cache server on the local network of our laboratory at Osaka Prefecture University and captured the DNS traffic generated on the cache server by laboratory members. To produce DNS tunneling traffic in our laboratory, we set up an authoritative DNS server, a DNS tunneling client, and a DNS tunneling server. We assumed that the tunneling client was legitimate. However, unfortunately, it was infected by malware (i.e., installed a DNS tunneling client). Therefore, the client produced both legitimate and tunneling traffic. **In our experiments, while generating the tunneling traffic, the client produced legitimate DNS traffic by browsing the Web and launching some background applications such as Slack.** The authoritative DNS server delegated the domain name for DNS tunneling to the DNS tunneling server, which

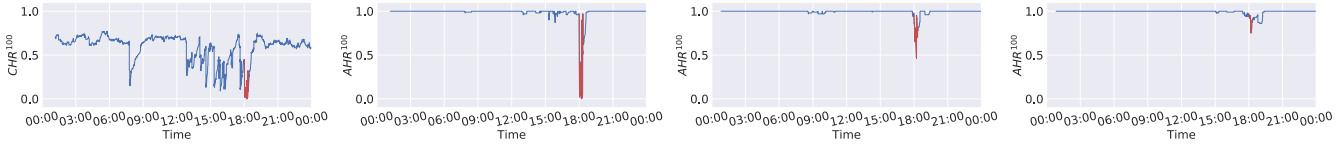


(a) CHR^{100} of all the clients in Scenario 1. (b) AHR^{100} of all the clients in Scenario 1. (c) AHR^{100} of all the clients in Scenario 2. (d) AHR^{100} of all the clients in Scenario 3.

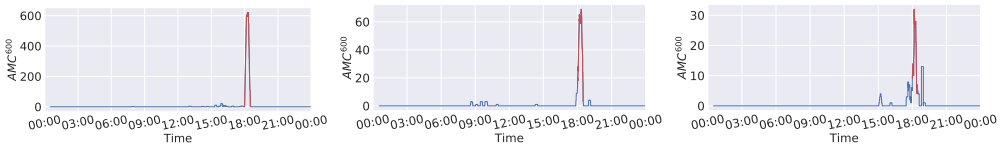


(e) AMC^{600} of all the clients in Scenario 1. (f) AMC^{600} of all the clients in Scenario 2. (g) AMC^{600} of all the clients in Scenario 3.

Fig. 4. Time-series data of CHR^{100} of all the clients in Scenario 1 and AHR^{100} and AMC^{600} of all the clients in Scenarios 1, 2, and 3.



(a) CHR^{100} of the tunneling client in Scenario 1. (b) AHR^{100} of the tunneling client in Scenario 1. (c) AHR^{100} of the tunneling client in Scenario 2. (d) AHR^{100} of the tunneling client in Scenario 3.



(e) AMC^{600} of the tunneling client in Scenario 1. (f) AMC^{600} of the tunneling client in Scenario 2. (g) AMC^{600} of the tunneling client in Scenario 3.

Fig. 5. Time-series data of CHR^{100} of the tunneling client in Scenario 1 and AHR^{100} and AMC^{600} of the tunneling client in Scenarios 1, 2, and 3.

made the DNS cache server forward malicious queries generated by the tunneling client to the tunneling server. We used `dnscat2` [26] as a DNS tunneling tool for both the tunneling client and server. **Note that because of the nature of DNS tunneling (i.e., data is exfiltrated via cache misses), we would have obtained the same results (*CHR*, *AHR*, and *AMC*) using different tools when the parameters of the tunneling query transmission interval and the tunneling traffic generation period were set to the same values.** Before performing the experiments on tunneling, we created a list of cache entries and access entries by capturing DNS traffic from 21 clients in our laboratory for **31 days**. The parameters for the experiments are presented in Table VI. We prepared three data exfiltration scenarios in terms of the tunneling query transmission interval: Scenarios 1, 2, and 3, with transmission intervals of 1, 10, and 100 s, respectively. We used a list of cache entries only for Scenario 1, which demonstrated the effectiveness of *AHR* and *AMC* against the shortcomings of the *CHR*. We omitted the experiments for Scenarios 2 and 3 because the results for Scenario 1 (the "easiest" case for DNS tunneling detection) already indicated the shortcomings of the *CHR*.

TABLE VI
PARAMETERS FOR TUNNELING EXPERIMENTS

Parameters	Values		
	Scenario 1	Scenario 2	Scenario 3
Tunneling query transmission interval	1 s	10 s	100 s
# of clients (incl. one tunneling client)	17	18	17
Monitoring period	1 day (weekday)		
Tunneling traffic generation period	20 mins from 18:00 JST		
Size of the list of cache entries	1 MB	N/A	
Size of the list of access entries	1 MB		
n	10, 20, . . ., 300		
t	100, 200, . . ., 1200		

B. Results

The scatter plot of the time-series data for the *CHR* collected from all the clients in Scenario 1, and the *AHR* and *AMC* collected from all the clients in Scenarios 1, 2, and 3, for $n = 100$ and $t = 600$ are shown in Fig. 4. Fig. 5 shows the traffic of the tunneling client extracted from Fig. 4. To compute the *CHR* and *AHR* of all the clients, a memory to store the latest n queries was prepared for each client, and the first *CHR* and *AHR* are calculated after the arrival of n queries. To compute the *AMC* of all the clients, a memory to store the latest queries within t was prepared for each client, and the first *AMC* was calculated after t s. The red curve in Figs. 4 and 5 indicate that the *CHR*, *AHR*, and *AMC* were affected by the DNS tunneling traffic generated by the tunneling client. These figures illustrate that both the *CHR* and *AHR* decreased, whereas the *AMC* increased when DNS tunneling traffic was produced.

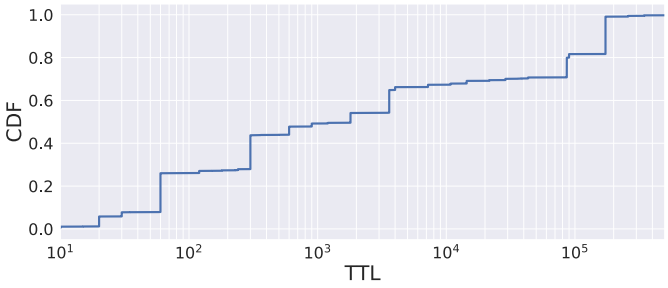


Fig. 6. Cumulative distribution functions (CDF) of TTL (s) of unique A and AAAA records in our 31-day dataset.

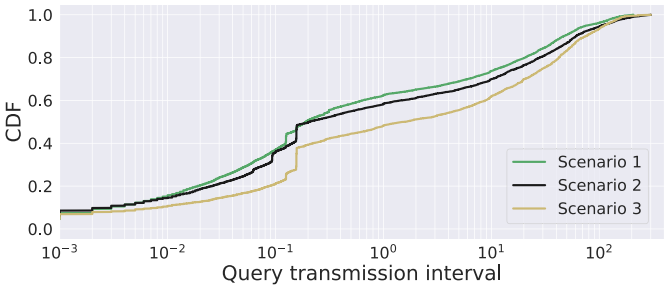


Fig. 7. CDFs of query transmission interval (s) of the tunneling client in Scenarios 1, 2, and 3, which exclude the tunneling query transmission interval.

From Figs. 4(a) and 5(a), during the monitoring period, *CHR* cannot clearly establish whether the decrease in the *CHR* was caused by the tunneling because of the two drawbacks of the cache entries discussed in Section III-A. In our 31-day dataset, 99.7% of the queries from all the clients were for A and AAAA records, whereas 51.2% of the cache entries were for A and AAAA records. Fig. 6 shows the CDF of the TTL of unique A and AAAA records in our 31-day dataset, and the CDF indicates that the TTL of 43.7% of the A and AAAA records were less than 300 s. These statistics are related to the two drawbacks discussed in Section III-A.

By contrast, Figs. 4(b), 5(b), 4(c), and 5(c) clearly identify the decrease in the *AHR* caused by tunneling. These figures indicate that the *AHR* effectively addressed the drawbacks of the *CHR* and can successfully identify tunneling traffic. As shown in Figs. 4(d) and 5(d), we observed that during the tunneling traffic generation period, the *AHR* did not decrease drastically; therefore, the *AHR* could not identify the tunneling traffic when the tunneling query transmission interval was large, which is a vulnerability of *AHR*.

As described in Section III-B, one shortcoming of the *AHR* is that it can be increased by malware that sends a large volume of legitimate DNS queries with FQDNs that are normally expected to be stored in the access entry, thus concealing the tunneling queries. To overcome this vulnerability, we focused only on the access misses and computed the *AMC*. Figs. 4(e), 5(e), 4(f), 5(f), 4(g), and 5(g) clearly show the increase in the *AMC* owing to the tunneling. Compared to the *AHR*, the *AMC* can characterize tunneling traffic more effectively because it ignores legitimate traffic that increases *AHR* during the tunneling traffic generation period. These figures indicate an increase in the *AMC*, even when only legitimate traffic was generated. This was caused by the fact that new Web content that were not captured among the DNS traffic for the 31 days were accessed. Accessing new content sends a certain number of queries that are not included among the access entries in a short period of time, which drastically increases the *AMC*. In addition, we confirmed that some websites install crawlers to resolve domain names. For instance, the website, `http://www.guide2research.com/` resolves many FQDNs of conference Web sites and might retrieve some information about the conferences.

Fig. 7 illustrates the CDFs of query transmission interval of the tunneling client in Scenarios 1, 2, and 3, excluding the tunneling query transmission interval. From the CDFs, we

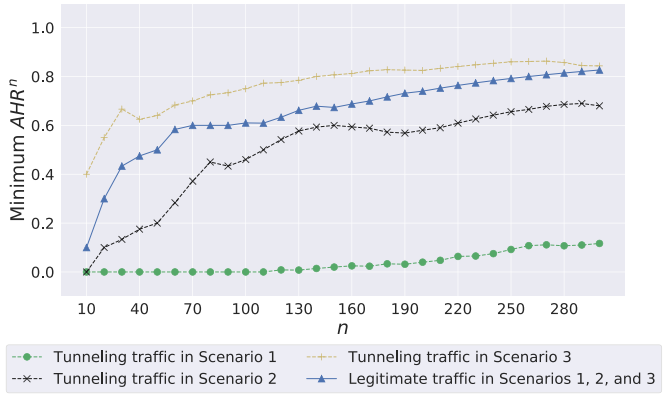


Fig. 8. Minimum AHR^n for legitimate and tunneling traffic in Scenarios 1, 2, and 3.

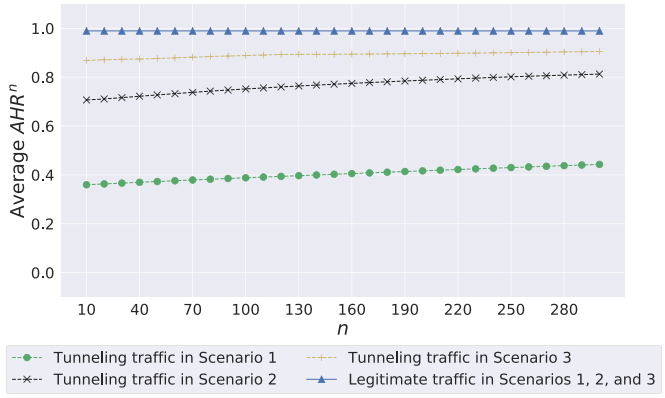


Fig. 9. Average AHR^n for legitimate and tunneling traffic in Scenarios 1, 2, and 3.

observed that 62.4%, 58.3%, and 48.3% of the intervals were less than 1 s in Scenarios 1, 2, and 3, respectively. Considering a higher threshold, 73.7%, 69.9%, and 61.8% of the intervals were less than 10 s for each scenario, and 96.3%, 94.5%, and 93.3% of the intervals were less than 100 s for each scenario. Therefore, our parameter settings generated tunneling queries at a reasonable time interval, compared to general legitimate query traffic.

Figs. 8 and 9 show the minimum and average AHR^n for tunneling and legitimate traffic in Scenarios 1, 2, and 3. Here, the minimum and average AHR^n for the tunneling traffic were calculated based on (a) the traffic produced by the tunneling client for 20 min during the tunneling traffic generation period,

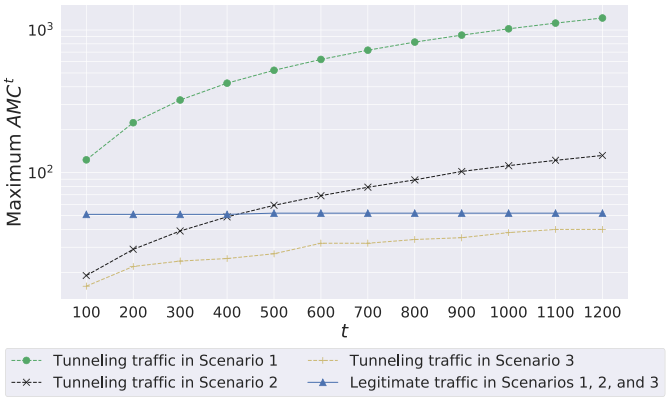


Fig. 10. Maximum AMC^t for legitimate and tunneling traffic in Scenarios 1, 2, and 3.

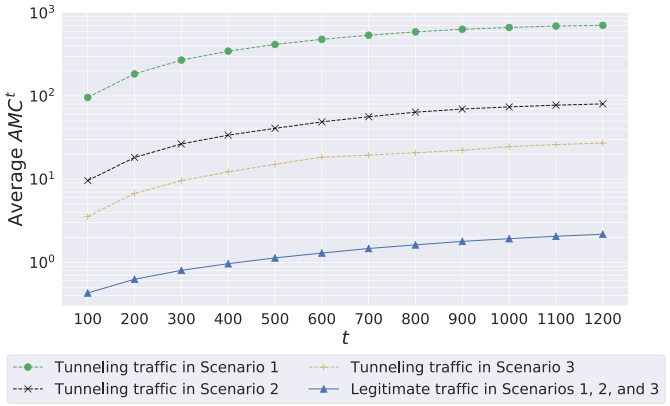


Fig. 11. Average AMC^t for legitimate and tunneling traffic in Scenarios 1, 2, and 3.

and (b) the first *n* queries after generating the last tunneling query. The minimum and average AHR^n for legitimate traffic in Scenarios 1, 2, and 3 were computed based on traffic from all the clients, except the above (a) and (b) traffic. The minimum AHR^n for tunneling traffic in Scenarios 1 and 2 was lower than that for legitimate traffic, which means that the traffic can be easily classified. By contrast, in Scenario 3 (large tunneling query transmission interval), it was impossible to classify the traffic, as the corresponding minimum *AHR* values of legitimate traffic were less than those of the tunneling traffic. The average AHR^n for the tunneling traffic in Scenarios 1, 2, and 3 was lower than that for legitimate traffic, which indicates that the AHR^n decreased because of the tunneling traffic.

Figs. 10 and 11 show the maximum and average AMC^t for the tunneling and legitimate traffic in Scenarios 1, 2, and 3. Here, the maximum and average AMC^t for the tunneling traffic are calculated based on (a) the traffic produced by the tunneling client for 20 min during the tunneling traffic generation period and (b) the queries produced within *t* seconds after the tunneling traffic generation period. The maximum and average AMC^t for the legitimate traffic in Scenarios 1, 2, and 3 were computed based on traffic from all the clients, except the above (a) and (b) traffic. The maximum AMC^t for the tunneling traffic in Scenario 1 was higher than that for

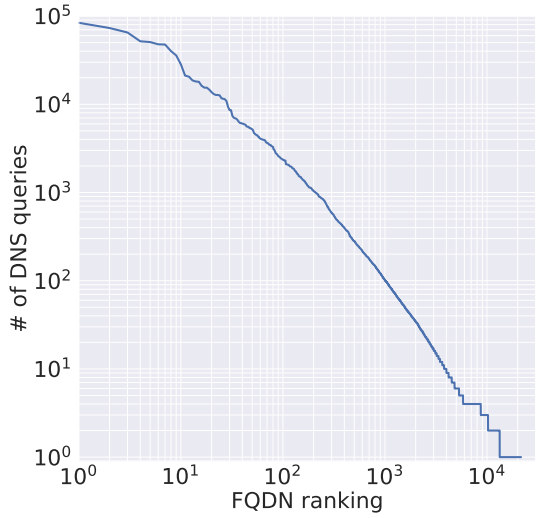


Fig. 12. FQDN ranking vs. the number of DNS queries.

the legitimate traffic in Scenarios 1, 2, and 3, which indicated that the traffic could be easily classified. When *t* increased ($t \geq 500$ s), the maximum AMC^t for tunneling traffic in Scenario 2 was higher than that for the legitimate traffic in Scenarios 1, 2, and 3. By contrast, in Scenario 3 (large tunneling query transmission interval), it was impossible to classify the traffic, as the corresponding maximum *AMC* values of the legitimate traffic was higher than those of the tunneling traffic. The average AMC^t for the tunneling traffic in Scenarios 1, 2, and 3 was higher than that for legitimate traffic in Scenarios 1, 2, and 3, which indicates that the AMC^t increased because of the tunneling traffic. From the average AMC^t for the tunneling traffic in Scenarios 1, 2, and 3, the average number of access misses for 20 min was approximately 2.

Fig. 12 shows the queried FQDN ranking versus the number of DNS queries from all the clients, which can be obtained by analyzing our 31-day dataset. Each FQDN was ranked based on the number of DNS queries that contained it. Fig. 12 indicates that popular FQDNs are repeatedly requested by the clients, roughly adhering to the Zipf's law [27]. This fact verifies the hypothesis introduced in Section I and supports the results for the *AHR* and *AMC* shown in this section (i.e., access misses do not often happen based on the typical human behavior, and once tunneling queries are generated, the corresponding access misses occur; consequently, the *AHR* decreases and *AMC* increases, which indicates DNS tunneling traffic).

V. Filters Based on Cache-Property-Aware Features Against DNS Tunneling

In this section, based on the results presented in Section IV-B, we implement a rule-based filter and an LSTM filter using the *AMC* as a cache-property-aware feature. The proposed monitoring and filtering system (Fig. 13) should be deployed on the DNS cache server in the enterprise network. We assume that the DNS clients are expected to connect to the DNS cache server installed in the enterprise to monitor and manage their activities in terms of risk hedge.

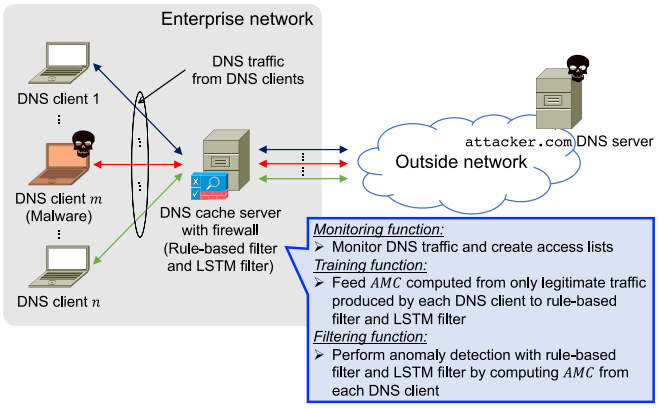


Fig. 13. Overview of the proposed monitoring and filtering system.

A. Rule-Based Filter

Our rule-based filter is based on an *anomaly detection* model that defines a threshold value for *AMC*. Specifically, the filter classifies the DNS query as anomalous when the *AMC* exceeds a preconfigured threshold value. To extract the threshold values, we collect only *legitimate* DNS traffic, create access lists, and compute the *AMC* for the DNS query generated by each DNS client. The *AMC* values are collected (from all the clients) into a training dataset. The classifiers' thresholds are determined by taking user-defined percentile values from this dataset. For example, the threshold computed for the 99th percentile corresponds to the minimum *AMC* from the top 1%.

B. Long Short-Term Memory Filter

The filter is modeled using LSTM networks [28], a type of recurrent neural network. LSTMs differ from feedforward neural networks (which are universal, nonlinear function approximators [50]), as they also have feedback connections that serve as a type of "memory" for what the network has already seen. Such networks are suitable for use when the input consists of data sequences such as network traffic. Compared to standard recurrent neural networks, LSTMs can handle very long sequences, which makes them ideal for this study. They are widely deployed in the industry.

The proposed LSTM filter is based on an *anomaly detection* pipeline for temporal data. The rationale behind the design of this pipeline is that a predictive model of normal behavior has a low prediction error when fed with normal input and a higher prediction error for abnormal input. We implement the pipeline as follows. In the same manner as described in Section V-A, we first collect only *legitimate* DNS traffic, create access lists, and compute the *AMC* for the DNS query generated by each DNS client. Then, we collect the *AMC* values into a training dataset. We use the *AMC* dataset to train an LSTM model and obtain its next-step predictions. We then compute the prediction error based on the model's predictions and the actual *AMC* values. Because legitimate DNS traffic itself contains outliers, the prediction error tends to be higher at these outliers and can be used to create a filter. Therefore, we create a filter by constructing a binary classifier that takes the prediction error as the input. The classifier's threshold above which the input is deemed an "anomaly" is computed by a user-defined percentile value. For example, the threshold computed for the 99th percentile corresponds to the minimum prediction error from the top 1%. During deployment, the filter monitors the DNS traffic, computes the *AMC* for each DNS client, and predicts the next step *AMC* using the trained LSTM. When the prediction error exceeds the threshold computed for the binary classifier, the filter produces an alert because such queries can be anomalous.

To summarize, the difference between the rule-based and LSTM-based filters is that the former uses a threshold computed directly from the *AMC*, whereas the latter uses a threshold computed from the prediction error of an *AMC*-forecasting LSTM model.

C. Guides to Deploy Filters

Note that multiple thresholds can be computed to achieve flexibility and easily customize the sensitivity of the filter during deployment. In general, there is a tradeoff when choosing a threshold value (increasing both the true and false positive rates); therefore, there is no optimal threshold value. It all comes down to the false positive rate deemed acceptable by the enterprise network operators.

The proposed monitoring and filtering system should be implemented on the DNS cache server, and the number of trained models equals the number of DNS cache servers in the enterprise network. For instance, if an enterprise network installs several DNS cache servers for each department, corresponding models should be created for each of them. This model creation policy stems from the fact that the locality of the DNS traffic for each DNS cache server was different [51], which makes it necessary to tune the model for each of them. Because our proposed system monitors multiple DNS clients in parallel, the system has to create a number of instances for the trained model equal to the number of clients. The required storage for an LSTM-based filter depends on the number of units, weights and clients (see Appendix A for a detailed explanation).

VI. Evaluation of Filters

A. Experimental Setup

In this section, we describe the procedure for creating and evaluating the rule-based and LSTM filters. To create the filters, we used the 31-day DNS traffic dataset introduced in Section IV-A. The initial access entries were created using the data of the first 24 days. The access entries were then used to calculate $AMC^t(t = 100, 200, \ldots, 1200)$ for each DNS query generated by each DNS client for the remaining seven days. The training set consisted of a large dataset of legitimate queries, which were more than 350,000. AMC^t was not computed for the first t s because the data were not sufficient. The computed AMC^t vectors, which were time-series data, comprised the training dataset for creating the filters.

The additional procedures for building the LSTM filters can be explained as follows. We preprocessed the dataset using the

standard scaler[2] to the dataset. The training dataset obtained from each client was divided according to the batch size, and data that were not consistent with the batch size were excluded. The training dataset was fed into a stateful LSTM, which can remember past trends better than the stateless LSTM. As for the parameters of the LSTM, the timestep was set to 100, the batch size was set to 32, the input dimension was set to 1 because only AMC^t was used, and the number of LSTM cells was set to 1, 2, 4, 8, and 16. We used the Adam Optimizer [52]. The loss function was the mean squared error, and when the difference between the current error and the previous error was less than 10^{-3} or the number of epochs reached 30, we terminated training. The LSTM was trained on a client-by-client basis, and the state of the model was reset every time the client was switched.

The evaluation method of the filters is described below. For the rule-based filters, as mentioned in Section V-A, we determined multiple thresholds for the training dataset to classify a query as an attack or not on the basis of the AMC^t. In our experiments, we considered 10,000 thresholds (from 0.01% to 100% with 0.01% increments). For the LSTM filters, we used the trained LSTM model to predict the training set and calculate the prediction error, that is, the squared error between the actual and predicted AMC^t. As mentioned in Section V-B, multiple thresholds can be used to classify a prediction error as an attack or not. In our experiments, we considered 10,000 thresholds (from 0.01% to 100% with 0.01% increments). We then extracted the AMC^t for each DNS query from the DNS traffic data generated in Scenarios 1, 2 and 3, which were described in Section IV-A. The initial access list used to obtain AMC^t was created from the 31-day DNS traffic dataset described in Section IV-A. Similar to the training, the AMC^t was not computed for the first t s because of insufficient data. The test dataset for evaluating the data was the computed AMC^t, which was time-series data. The rule-based filters directly classified the activity of the test dataset using their computed thresholds. On the other hand, the LSTM-based filters classified the activity by first predicting it, then computing the prediction error and finally comparing the error with their corresponding thresholds. For both filters, we evaluated the receiver operating characteristic (ROC) curve (this curve was plotted using the true positive rate[3] and false positive rate[4]), area under the curve (AUC) score, and accuracy.[5] The speed of the attack detection was defined as follows:

$$x^{\{filter,t\}} = N_{FN}^{\{filter,t\},1} + N_{FN}^{\{filter,t\},2} + N_{FN}^{\{filter,t\},3} \text{ and}$$

$$Speed^{\{filter,t\}} = 1 - \frac{x^{\{filter,t\}} - x^{min}}{x^{max} - x^{min}}.$$

Here, $x^{\{filter,t\}}$ is the summation of $N_{FN}^{\{filter,t\},1}$, $N_{FN}^{\{filter,t\},2}$, and $N_{FN}^{\{filter,t\},3}$, which are the number of false negatives required for the model {*filter*, *t*} (there were 72 models because we prepared one rule-based model and five LSTM models for 12 values of *t*) **to create the first alarm** (as the filter initially predicted the input as legitimate) in Scenarios 1, 2, and 3, respectively, and x^{min} and x^{max} were the minimum and maximum of $x^{\{filter,t\}}$, respectively. When $Speed^{\{filter,t\}}$ is 1.0, the model {*filter*, *t*} is the one that raises the earliest detection alarm. Note that in this paper, positives indicate that the queries are (a) the ones produced by the tunneling client for 20 min during the tunneling traffic generation period or (b) the ones produced within *t* seconds after the tunneling traffic generation period (the red curve illustrates the queries in Figs. 4 and 5). The other queries were labeled negatives.

[2]We also used the power transformer to preprocess the dataset; however, the models created using the preprocessed dataset exhibited worse prediction performance (see Appendix B).

[3](# of true positives)/(# of true positives + # of false negatives).

[4](# of false positives)/(# of false positives + # of true negatives).

[5](# of true positives + # of true negatives)/(# of queries).

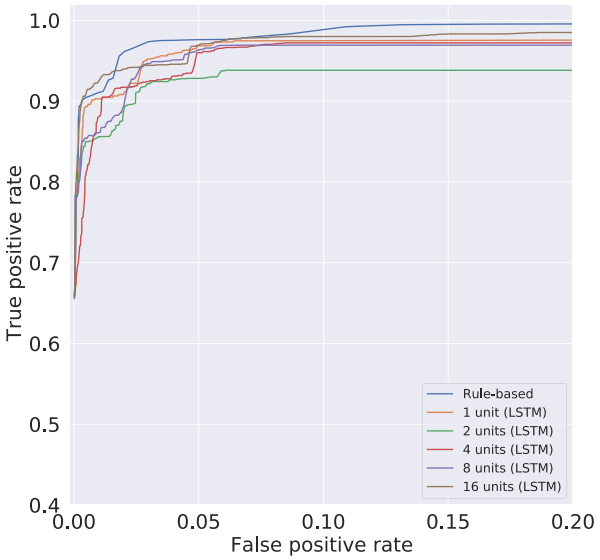


Fig. 14. ROC curves for the test dataset of Scenarios 1, 2, and 3 (AMC^{600}).

B. Results

Fig. 14 shows the ROC curves for the test dataset of Scenarios 1, 2, and 3, when *t* was 600. From the figure, when the false positive rate was over 0.025, the true positive rate was over 0.91, which indicated that our filters could classify legitimate and malicious DNS queries correctly with high probability. By contrast, in Fig. 15, which shows the ROC curves for the test dataset of only Scenario 3, when *t* was 600, the classification performance deteriorated, compared to Fig. 14. This is because, as mentioned in Section IV-B, the larger the tunneling transmission interval, the harder it was to detect the DNS tunneling attack. From these figures, the rule-based filter outperformed the LSTM-based one. In checking the positives classified as negatives (i.e., legitimate queries) by the filters, the LSTM filter sometimes identified several positives as legitimate queries because it evaluated the queries on the basis of a threshold computed from the prediction error, which means the prediction error could become low at some point, even during the tunneling attack. Meanwhile, the rule-based filter evaluated the queries with a threshold computed directly from the *AMC*. When the prediction error obtained by the LSTM filter fell below the predetermined threshold and the *AMC* exceeded the threshold for the rule-based filter, the rule-based filter classified the queries as anomalous.

TABLE VII
AUC SCORES FOR THE TEST DATASET OF SCENARIOS 1, 2, AND 3

	AMC^{100}	AMC^{200}	AMC^{300}	AMC^{400}	AMC^{500}	AMC^{600}	AMC^{700}	AMC^{800}	AMC^{900}	AMC^{1000}	AMC^{1100}	AMC^{1200}
Rule-based	0.9888	0.9904	0.9914	0.9927	0.9864	0.9936	0.9932	0.9927	0.9933	0.9961	0.9963	0.9965
1 unit (LSTM)	0.9869	0.9872	0.9892	0.9351	0.9816	0.9772	0.9921	0.9714	0.9859	0.9649	0.9789	0.9883
2 units (LSTM)	0.9769	0.9696	0.9754	0.9183	0.9885	0.9682	0.9742	0.9411	0.8883	0.9890	0.9795	0.9910
4 units (LSTM)	0.9694	0.9750	0.9832	0.9663	0.9800	0.9710	0.9126	0.9708	0.9834	0.9682	0.9161	0.9461
8 units (LSTM)	0.9661	0.9410	0.9241	0.9770	0.9820	0.9732	0.9828	0.9886	0.9865	0.9797	0.9552	0.9808
16 units (LSTM)	0.9828	0.9139	0.9310	0.9375	0.9261	0.9827	0.9145	**0.8968**	0.9050	0.9379	**0.8414**	0.9801

TABLE VIII
AUC SCORES FOR THE TEST DATASET OF SCENARIO 3

	AMC^{100}	AMC^{200}	AMC^{300}	AMC^{400}	AMC^{500}	AMC^{600}	AMC^{700}	AMC^{800}	AMC^{900}	AMC^{1000}	AMC^{1100}	AMC^{1200}
Rule-based	0.9499	0.9530	0.9614	0.9747	0.9795	0.9838	0.9809	0.9812	0.9822	0.9850	0.9836	0.9854
1 unit (LSTM)	0.9394	0.9390	0.9578	0.5245	0.9396	0.8582	0.9711	0.8281	0.9543	0.8267	0.9066	0.9603
2 units (LSTM)	0.9137	0.9155	0.8834	0.4296	0.9589	0.8303	0.8745	0.8944	0.3399	0.9595	0.9440	0.9814
4 units (LSTM)	0.8196	0.8427	0.9419	0.7968	0.9260	0.8131	0.4852	0.8351	0.9400	0.9227	0.6386	0.6945
8 units (LSTM)	0.9166	0.4904	0.4313	0.8382	0.9171	0.8707	0.9473	0.9646	0.9431	0.9077	0.7458	0.9526
16 units (LSTM)	0.9242	0.8407	0.4440	0.5077	0.4963	0.8920	0.4153	0.5669	0.4498	0.7693	0.5487	0.9269

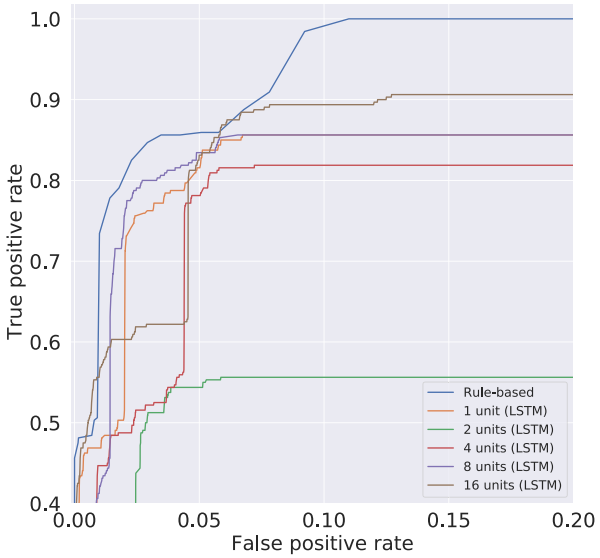


Fig. 15. ROC curves for the test dataset of Scenario 3 (AMC^{600}).

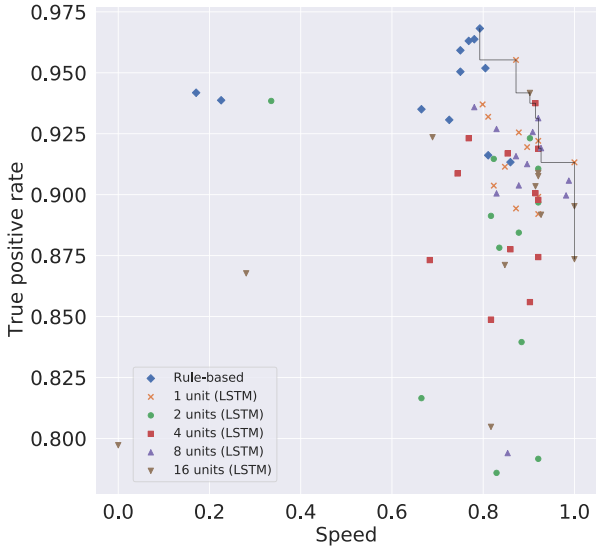


Fig. 16. Pareto front (false positive rate = 0.025). There are 12 models per filter (each with a different value of t $(100, 200, \ldots, 1200)$) and there exist 9 non-dominated solutions. Although the rule-based filters tend to achieve higher true positive rates than LSTM filters, the LSTM filters tend to detect attacks more quickly.

As shown by the results reported in Table VII, almost all of the AUC scores were over 0.90, with a few exceptions (marked in bold). However, from the results in Table VIII, we observed a considerably different situation. Scenario 3 was the most challenging in the context of DNS tunneling detection. The results indicate that the efficiency of the filter not only depended on the number of units in the LSTM model but was also strongly related to the time interval t, which was also the size of the sliding window. It appears that the value of t significantly influenced the quality of the detections. More interestingly, for the same value of t, we observed different results, depending on the number of units in the LSTM model. This clearly implies that for more challenging settings where exfiltration was performed over large periods of time, the model must be able to identify a larger number of features on varying scales. Therefore, one possible future research direction is to investigate the effectiveness of multi-scale ensemble LSTM models for detecting DNS tunneling attacks over large periods of time. Another interesting question is whether there is a correlation between the number of units in the LSTM model and the exfiltration period.

We fixed the false positive rate at 0.025 and discussed the accuracy and speed of attack detection. The accuracy for the entire test dataset for Scenarios 1, 2, and 3 are shown in Table IX. Table X shows the accuracy for the test dataset for only Scenario 3. These tables indicate that both filters have a high accuracy (ranging from 0.9697 to 0.9763).

Fig. 16 illustrates the Pareto front based on the true positive rate and the speed of the attack detection. As shown in Figs. 14 and 15, rule-based filters tended to achieve higher true-positive rates than LSTM filters. Instead, the LSTM filters tend to detect attacks more quickly than rule-based filters. This benefit comes from the fact that the LSTM filters predicted the time-series data and could thus detect anomalies faster. The metric of speed is important to prevent data exfiltration;

TABLE IX
ACCURACY FOR THE TEST DATASET IN SCENARIOS 1, 2, AND 3 (FALSE POSITIVE RATE = 0.025)

	AMC^{100}	AMC^{200}	AMC^{300}	AMC^{400}	AMC^{500}	AMC^{600}	AMC^{700}	AMC^{800}	AMC^{900}	AMC^{1000}	AMC^{1100}	AMC^{1200}
Rule-based	0.9747	0.9755	0.9734	0.9736	0.9758	0.9743	0.9736	0.9738	0.9756	0.9759	0.9735	0.9746
1 unit (LSTM)	0.9740	0.9741	0.9742	0.9742	0.9739	0.9741	0.9749	0.9737	0.9744	0.9745	0.9748	0.9740
2 units (LSTM)	0.9751	0.9735	0.9712	0.9740	0.9739	0.9743	0.9744	0.9722	0.9729	0.9745	0.9728	0.9746
4 units (LSTM)	0.9738	0.9740	0.9741	0.9743	0.9744	0.9743	0.9739	0.9738	0.9731	0.9733	0.9729	0.9733
8 units (LSTM)	0.9725	0.9743	0.9741	0.9743	0.9748	0.9744	0.9745	0.9745	0.9741	0.9739	0.9749	0.9743
16 units (LSTM)	0.9738	0.9727	0.9739	0.9742	0.9747	0.9746	0.9740	0.9734	0.9738	0.9735	0.9725	0.9737

TABLE X
ACCURACY FOR THE TEST DATASET IN SCENARIO 3 (FALSE POSITIVE RATE = 0.025)

	AMC^{100}	AMC^{200}	AMC^{300}	AMC^{400}	AMC^{500}	AMC^{600}	AMC^{700}	AMC^{800}	AMC^{900}	AMC^{1000}	AMC^{1100}	AMC^{1200}
Rule-based	0.9698	0.9733	0.9753	0.9763	0.9750	0.9763	0.9744	0.9751	0.9747	0.9739	0.9711	0.9705
1 unit (LSTM)	0.9701	0.9720	0.9735	0.9708	0.9721	0.9741	0.9739	0.9701	0.9728	0.9751	0.9718	0.9725
2 units (LSTM)	0.9697	0.9710	0.9725	0.9715	0.9716	0.9723	0.9723	0.9709	0.9701	0.9707	0.9718	0.9730
4 units (LSTM)	0.9716	0.9723	0.9722	0.9724	0.9738	0.9724	0.9721	0.9724	0.9727	0.9726	0.9720	0.9701
8 units (LSTM)	0.9714	0.9718	0.9718	0.9720	0.9738	0.9740	0.9725	0.9733	0.9720	0.9726	0.9727	0.9729
16 units (LSTM)	0.9714	0.9719	0.9716	0.9716	0.9719	0.9730	0.9711	0.9721	0.9714	0.9714	0.9705	0.9743

TABLE XI
VALUES OF *t* AND THRESHOLDS OF THE 9 NON-DOMINATED SOLUTIONS SORTED BY TRUE POSITIVE RATE (NOTE THAT THE THRESHOLD OF THE RULE-BASED FILTER IS *AMC* WHILE THAT OF THE LSTM FILTER IS THE PREDICTION ERROR OF THE CORRESPONDING AMC-FORECASTING LSTM MODEL)

Models	t	Thresholds
Rule-based	600	13
1 unit (LSTM)	700	0.06810
16 units (LSTM)	600	0.0007434
4 units (LSTM)	500	0.02775
8 units (LSTM)	600	0.01374
8 units (LSTM)	1100	0.008933
1 unit (LSTM)	900	0.009276
16 units (LSTM)	700	0.001700
16 units (LSTM)	800	0.003331

once the first alarm goes off, the firewall operator can carefully examine the anomalous client and determine whether the client is to be isolated. Finally, the total amount of leaked data can be reduced as much as possible. Table XI summarizes the values of *t* and the thresholds of the 9 non-dominated solutions sorted by true positive rate.

VII. DISCUSSION

Based on the evaluation in the foregoing Section VI, we conclude that the rule-based filter achieves a higher rate of the DNS tunneling attack detection than the LSTM, which, however, detects the attack faster, while both maintain a low misdetection rate. From this perspective, enterprise network operators can deploy our proposed monitoring and filtering system with some strategies. For example, first, based on the LSTM filter, the operators identify a suspicious client. After the first alarm raised by the LSTM filter, the operators still allow the client to produce the queries. However, these queries, including the tunneling ones, should be resolved by only the connected DNS cache server without the iterative query process, such that the unresolvable ones do not get forwarded to the outside (i.e., data exfiltration is eliminated at this point). Then, using a rule-based filter, at a defined point, the operators can determine whether the client is infected by malware. This strategy to combine these filters is an enterprise network operation guarding against data exfiltration through DNS tunneling.

In this paper, we proposed a method for detecting DNS tunneling that focuses on DNS clients rather than domain names. This is because our goal was to prevent information leakage by detecting DNS clients infected with malware in targeted attacks. As discussed in Section II-C, some researchers have focused on domain names and proposed filters whose design guideline was to identify the domain names used to perform DNS tunneling [11], [14], [18], [20], [21]. The shortcoming of such methods is that they fail when the attackers and their malware change tactics and utilize several domain names for the attacks. Our proposed filter does not focus on the domain name; rather, it detects attacks based on whether access misses have occurred, thereby effectively addressing the above drawback.

The proposed filters can be easily integrated with the conventional detection methods introduced in Section II-C. For example, when malware attempts to leak a file through DNS tunneling, to detect the attack, a countermeasure adopts length-based features such as the FQDN length and the longest label length, which are used in the payload analysis. RFC 1035 [53] defines the maximum total length of a domain name (dots included) and a label as 255 characters and 63 characters, respectively. In the dataset used in [23], more than 99% of the queries included less than 80 characters. Considering the malware, to improve the information leakage throughput by adding more data, the FQDN becomes longer. However, this type of malicious FQDN should be filtered out by a countermeasure based on the statistics of the FQDN length. Finally, to circumvent the filter, the malware is forced to generate more malicious queries, which causes more access misses, thus including the likelihood of the proposed filter detecting the attack more easily. By combining our proposal with conventional filters, we can create a more resilient firewall.

Our experiments were carried out by utilizing a DNS traffic dataset from 21 clients in our laboratory. We expect that,

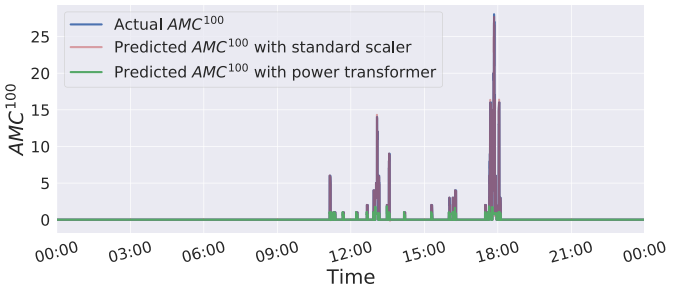


Fig. 17. Comparison of actual AMC^{100}, predicted AMC^{100} with standard scaler, and predicted AMC^{100} with power transformer for one client in one day in the training set.

even if the number of clients increases, the robustness of the cache-property-aware features will be assured. The reason is because the robustness is related to the queried FQDN ranking versus the number of DNS queries from all the clients (see Section IV-B). Popular FQDNs are repeatedly requested by the clients, roughly adhering to Zipf's law, and hence, access misses do not often happen based on the typical human behavior. Hasegawa *et al.* [51] observed Zipf's law even in a large DNS traffic dataset captured at a DNS cache server in a campus network. This indicates that our solution could be applied even for the case that the number of clients increases. Evaluating our proposed system by increasing the number of clients is one of our future works.

However, the proposed method is limited in one regard; it cannot effectively identify low-throughput attacks, as shown in Sections IV and VI. We believe that it is still difficult to detect such attacks, even using the existing methods introduced in Section II-C. To detect low-throughput attacks, one possible approach is to employ the proposed cache-property-aware features. For instance, when the information on the access misses from a DNS client is recorded over a long period of time and the total number of access misses for one client is higher than that of other DNS clients at a certain point, the client could be suspected of being infected with malware. In other words, it is essential to propose a method for detecting low-throughput attacks by long-term monitoring rather than short-term monitoring.

VIII. Conclusion

Various countermeasures against DNS tunneling have been proposed; however, they are based on DNS tunneling features that can be easily obfuscated by malicious entities mimicking legitimate ones. Therefore, conventional approaches are not robust against feature obfuscation. To solve the issue, we focused on the nature of DNS tunneling. When a tunneling client sends a malicious query to the tunneling server, the query definitely causes a cache miss on the DNS cache server to which the client connects. Based on this observation, we proposed cache-property-aware features for DNS tunneling detection. Our extensive experiments revealed that the access miss count can clearly reveal DNS tunneling traffic that generates tunneling queries within a reasonable time interval, compared to general legitimate query traffic. Moreover, we exploit the cache-property-aware features to develop the rule-based and LSTM filters to counter DNS tunneling. The DNS tunneling attack detection rate of the rule-based filter was higher than that of the LSTM, which instead detected the attack more quickly, while both maintained a low misdetection rate.

In future work, we will tackle low-throughput information leakage DNS tunneling attacks. We believe that the proposed cache-property-aware features can be applied for detection based on long-term monitoring; therefore, we will further investigate the "trace" of DNS tunneling against such an advanced attack.

Appendix A
Filter Storage Requirements

The required storage for an LSTM-based filter depends on the number of units, weights and clients. One LSTM unit typically has two stored values for the states: the "cell state" and the "hidden state". The number of weights for 1, 2, 4, 8, and 16-unit LSTMs is 18, 43, 117, 361, and 1233, respectively. Note that the LSTM weights and the threshold can be shared (they do not change for each client), therefore, can be stored only once, whereas the states need to be maintained independently for each client. Thus, assuming that the state, weight and threshold are expressed by 8 bytes, the required storage for a rule-based system is just 8 bytes, while the required storage for the LSTM-based filters is $8 * (2 * N * m + w + 1)$ bytes, where N is the number of clients, m is the number of LSTM units and w is the number of weights. More specifically, 1, 2, 4, 8, and 16-unit LSTMs, require $(16N + 152)$, $(32N + 352)$, $(64N + 944)$, $(128N + 2896)$, and $(256N + 9872)$ bytes, respectively. If, for example, there are 1000 clients, the most memory-intensive model (16 units) requires 265872 bytes ($\sim$260KB).

Appendix B
Data Preprocessing

Similar to the procedure described in Section VI, we performed experiments to verify the influence of two data preprocessing methods: standard scaler and power transformer. Fig. 17 shows the comparison of the actual AMC^{100}, predicted AMC^{100} with standard scaler and power transformer for one client for one day in the training set. From the figure, we can observe that the standard scaling can enable accurate prediction *AMC* (in the figure, blue and pink lines overlap; thus, the overlapped part can be seen as purple), whereas that of the power transformer failed to do this. This fact was most apparent in the case where t was set to a smaller value. Thus, in this study, we adopted a standard scaler as the data preprocessing method.

Acknowledgment

The authors thank the anonymous reviewers for their constructive comments.

References

[1] N. Ishikura, D. Kondo, I. Iordanov, V. Vassiliades, and H. Tode, "Cache-property-aware features for DNS tunneling detection," in *Proc. 23rd Conf. Innovat. Clouds Internet Netw. Workshops (ICIN)*, Paris, France, 2020, pp. 216–220.

[2] *IT Security Risks Survey 2014: A Business Approach to Managing Data Security Threats*. Accessed: Mar. 18, 2021. [Online]. Available: https://media.kaspersky.com/en/IT_Security_Risks_Survey_2014_Global_report.pdf

[3] (2015). *Understanding Targeted Attacks: The Impact of Targeted Attacks*. [Online]. Available: https://www.trendmicro.com/vinfo/us/security/news/cyber-attacks/the-impact-of-targeted-attacks

[4] M. Al-Kasassbeh and T. Khairallah, "Winning tactics with DNS tunnelling," *Netw. Security*, vol. 2019, no. 12, pp. 12–19, 2019.

[5] *IDC 2020 Global DNS Threat Report*. Accessed: Mar. 18, 2021. [Online]. Available: https://www.efficientip.com/resources/idc-dns-threat-report-2020/

[6] (2014). *New FrameworkPOS Variant Exfiltrates Data via DNS Requests*. [Online]. Available: https://www.gdatasoftware.com/blog/2014/10/23942-new-frameworkpos-variant-exfiltrates-data-via-dns-requests

[7] (2019). *DNS Tunneling in the Wild: Overview of OilRig's DNS Tunneling*. [Online]. Available: https://unit42.paloaltonetworks.com/dns-tunneling-in-the-wild-overview-of-oilrigs-dns-tunneling/

[8] D. Naylor *et al.*, "Multi-context TLS (McTLS): Enabling secure in-network functionality in TLS," in *Proc. ACM Conf. Spec. Interest Group Data Commun. (SIGCOMM)*, 2015, pp. 199–212.

[9] K. Born and D. Gustafson, "Detecting DNS tunnels using character frequency analysis," 2010. [Online]. Available: https://arxiv.org/abs/1004.4358.

[10] W. Ellens, P. Żuraniewski, A. Sperotto, H. Schotanus, M. Mandjes, and E. Meeuwissen, "Flow-based detection of DNS tunnels," in *Proc. IFIP Int. Conf. Auton. Infrastruct. Manage. Security (AIMS)*, 2013, pp. 124–135.

[11] V. Paxson *et al.*, "Practical comprehensive bounds on surreptitious communication over DNS," in *Proc. 22nd USENIX Conf. Security*, Aug. 2013, pp. 17–32.

[12] C. Qi, X. Chen, C. Xu, J. Shi, and P. Liu, "A bigram based real time DNS tunnel detection approach," *Procedia Comput. Sci.*, vol. 17, pp. 852–860, 2013.

[13] K. Xu, P. Butler, S. Saha, and D. Yao, "DNS for massive-scale command and control," *IEEE Trans. Depend. Secure Comput.*, vol. 10, no. 3, pp. 143–153, May/Jun. 2013.

[14] A. M. Kara, H. Binsalleeh, M. Mannan, A. Youssef, and M. Debbabi, "Detection of malicious payload distribution channels in DNS," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Sydney, NSW, Australia, 2014, pp. 853–858.

[15] M. Aiello, M. Mongelli, and G. Papaleo, "DNS tunneling detection through statistical fingerprints of protocol messages and machine learning," *Int. J. Commun. Syst.*, vol. 28, no. 14, pp. 1987–2002, 2015.

[16] M. Aiello, M. Mongelli, E. Cambiaso, and G. Papaleo, "Profiling DNS tunneling attacks with PCA and mutual information," *Logic J. IGPL*, vol. 24, no. 6, pp. 957–970, Dec. 2016.

[17] A. L. Buczak, P. A. Hanke, G. J. Cancro, M. K. Toma, L. A. Watkins, and J. S. Chavis, "Detection of tunnels in PCAP data by random forests," in *Proc. 11th Annu. Cyber Inf. Security Res. Conf. (CISRC)*, 2016, pp. 1–4.

[18] A. Das, M.-Y. Shen, M. Shashanka, and J. Wang, "Detection of exfiltration and tunneling over DNS," in *Proc. 16th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Cancun, Mexico, 2017, pp. 737–742.

[19] C.-M. Lai, B.-C. Huang, S.-Y. Huang, C.-H. Mao, and H.-M. Lee, "Detection of DNS tunneling by feature-free mechanism," in *Proc. IEEE Conf. Depend. Secure Comput. (DSC)*, Kaohsiung, Taiwan, 2018, pp. 1–2.

[20] J. Steadman and S. Scott-Hayward, "DNSxD: Detecting data exfiltration over DNS," in *Proc. IEEE Conf. Netw. Funct. Virtualization Softw. Defined Netw. (NFV-SDN)*, Verona, Italy, 2018, pp. 1–6.

[21] A. Nadler, A. Aminov, and A. Shabtai, "Detection of malicious and low throughput data exfiltration over the DNS protocol," *Comput. Security*, vol. 80, pp. 36–53, Jan. 2019.

[22] C. Liu, L. Dai, W. Cui, and T. Lin, "A byte-level CNN method to detect DNS tunnels," in *Proc. IEEE 38th Int. Perform. Comput. Commun. Conf. (IPCCC)*, London, U.K., 2019, pp. 1–8.

[23] J. Ahmed, H. H. Gharakheili, Q. Raza, C. Russell, and V. Sivaraman, "Monitoring enterprise DNS queries for detecting data exfiltration from internal hosts," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 1, pp. 265–279, Mar. 2020.

[24] S. Chen, B. Lang, H. Liu, D. Li, and C. Gao, "DNS covert channel detection method using the LSTM model," *Comput. Security*, vol. 104, May 2021, Art. no. 102095.

[25] (2011). *Morto Worm Sets a (DNS) Record*. [Online]. Available: https://community.broadcom.com/symantecenterprise/communities/community-home/librarydocuments/viewdocument?DocumentKey=268f079a-2bb8-4775-9ef9-1b02e32ca55d&CommunityKey=1ecf5f55-9545-44d6-b0f4-4e4a7f5f5e68&tab=librarydocuments

[26] *dnscat2*. Accessed: Mar. 18, 2021. [Online]. Available: https://github.com/iagox86/dnscat2

[27] G. K. Zipf, *Human Behavior and the Principle of Least Effort*. Cambridge, MA, USA: Addison-Wesley, 1949.

[28] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[29] T. van Leijenhorst, K.-W. Chin, and D. Lowe, "On the viability and performance of DNS tunneling," in *Proc. 5th Int. Conf. Inf. Technol. Appl. (ICITA)*, 2008, pp. 560–566.

[30] L. Nussbaum, P. Neyron, and O. Richard, "On robust covert channels inside DNS," in *Proc. IFIP Int. Inf. Security Conf. (IFIP SEC)*, 2009, pp. 51–62.

[31] M. Aiello, A. Merlo, and G. Papaleo, "Performance assessment and analysis of DNS tunneling tools," *Logic J. IGPL*, vol. 21, no. 4, pp. 592–602, Aug. 2013.

[32] *iodine*. Accessed: Mar. 18, 2021. [Online]. Available: https://github.com/yarrick/iodine

[33] *dns2tcp*. Accessed: Mar. 18, 2021. [Online]. Available: https://github.com/alex-sector/dns2tcp

[34] D. Raman *et al.*, "DNS tunneling for network penetration," in *Proc. Int. Conf. Inf. Security Cryptol. (ICISC)*, 2012, pp. 65–77.

[35] *Backdoor.Win32.Denis*. Accessed: Mar. 18, 2021. [Online]. Available: https://otx.alienvault.com/pulse/590314fb6575a03746de87a8

[36] *BernhardPOS*. Accessed: Mar. 18, 2021. [Online]. Available: https://otx.alienvault.com/pulse/55a5b4eeb45ff55fb194e69e

[37] *Cobalt Strike*. Accessed: Mar. 18, 2021. [Online]. Available: https://www.cobaltstrike.com/

[38] *DET*. Accessed: Mar. 18, 2021. [Online]. Available: https://github.com/sensepost/DET

[39] *DNScat*. Accessed: Mar. 18, 2021. [Online]. Available: http://tadek.pietraszek.org/projects/DNScat/

[40] *DNSExfiltrator*. Accessed: Mar. 18, 2021. [Online]. Available: https://github.com/Arno0x/DNSExfiltrator

[41] (2017). *Covert Channels and Poor Decisions: The Tale of DNSMessenger*. [Online]. Available: https://blogs.cisco.com/security/talos/covert-channels-and-poor-decisions-the-tale-of-dnsmessenger

[42] (2018). *DNSpionage Campaign Targets Middle East*. [Online]. Available: https://blog.talosintelligence.com/2018/11/dnspionage-campaign-targets-middle-east.html

[43] (2016). *Three Month FrameworkPOS Malware Campaign Nabs ~43,000 Credit Cards From Point of Sale Systems*. [Online]. Available: https://www.anomali.com/blog/three-month-frameworkpos-malware-campaign-nabs-43000-credits-cards-from-poi

[44] (2009). *OzymanDNS—Tunneling SSH Over DNS*. [Online]. Available: https://malicious.link/post/2009/2009310ozymandns-tunneling-ssh-over-dns-html/

[45] *Reverse DNS Shell*. Accessed: Mar. 18, 2021. [Online]. Available: https://github.com/ahhh/Reverse_DNS_Shell

[46] *TCP-over-DNS*. Accessed: Mar. 18, 2021. [Online]. Available: https://analogbit.com/software/tcp-over-dns/

[47] K. Fujiwara, A. Sato, and K. Yoshida, "DNS traffic analysis—CDN and the world IPv6 launch," *J. Inf. Process.*, vol. 21, no. 3, pp. 517–526, 2013.

[48] A. Shaikh, R. Tewari, and M. Agrawal, "On the effectiveness of DNS-based server selection," in *Proc. IEEE INFOCOM Conf. Comput. Commun. 20th Annu. Joint Conf. IEEE Comput. Commun. Soc.*, vol. 3. Anchorage, AK, USA, 2001, pp. 1801–1810.

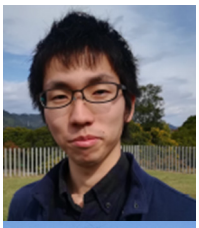
[49] G. C. M. Moura, J. Heidemann, R. D. O. Schmidt, and W. Hardaker, "Cache me if you can: Effects of DNS time-to-live," in *Proc. ACM Internet Meas. Conf. (IMC)*, 2019, pp. 101–115.

[50] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Netw.*, vol. 4, no. 2, pp. 251–257, 1991.

[51] K. Hasegawa, D. Kondo, and H. Tode, "FQDN-based whitelist filter on a DNS cache server against the DNS water torture attack," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, May 2021. [Online]. Available: https://im2021.ieee-im.org/program/poster-sessions

[52] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. [Online]. Available: https://arxiv.org/abs/1412.6980.

[53] "Domain names—Implementation and specification," Internet Eng. Task Force, RFC 1035, 1987. [Online]. Available: https://tools.ietf.org/html/rfc1035



Naotake Ishikura received the B.S. and M.S. degrees in engineering from Osaka Prefecture University, Osaka, Japan, in 2019 and 2021, respectively. His current research interest includes network security.

Daishi Kondo (Member, IEEE) received the B.S. degree in engineering from Osaka University, Osaka, Japan, in 2013, the M.A.S. degree in interdisciplinary information studies from the University of Tokyo, Tokyo, Japan, in 2015, and the Ph.D. degree in computer science from the University of Lorraine, LORIA (CNRS UMR 7503), Inria Nancy-Grand Est, Nancy, France, in 2018. He is currently an Assistant Professor with Osaka Prefecture University. His research interests include information-centric networking, network security, privacy, and peer-to-peer networking.

Vassilis Vassiliades received the B.Sc. degree in computer science from the University of Cyprus (UCY) in 2007, the M.Sc. degree in intelligent systems engineering from the University of Birmingham, U.K., in 2008, and the Ph.D. degree in computer science from UCY in 2015. He is currently a Team Leader with the CYENS Centre of Excellence (formerly known as RISE), Cyprus, and an Associate Research Fellow with UCY. He was a Postdoctoral Fellow and a Research Engineer with Inria Nancy, France, from 2015 to 2018 and a Research Associate with UCY from 2015 to 2019 and RISE in 2019. His research interests lie in the areas of artificial intelligence and robotics, with emphasis on machine learning and evolutionary computation.

Iordan Iordanov received the B.S. degree in applied mathematics and the M.S. degree in applied and computational mathematics from the University of Crete, Greece, in 2013 and 2015, respectively, and the Ph.D. degree in computer science from the University of Lorraine, LORIA (CNRS UMR 7503), Inria Nancy-Grand Est, Nancy, France, in 2019. He is currently a Chief Scientist with Corpy&Co., Inc., Tokyo, Japan. His research interests include applied and computational mathematics, computational geometry, and explainable artificial intelligence.

Hideki Tode (Member, IEEE) received the B.E., M.E., and Ph.D. degrees in communications engineering from Osaka University in 1988, 1990, and 1997, respectively. From 1991 to 2008, he was an Assistant Professor and an Associate Professor with Osaka University. He has been a Professor with the Department of Computer Science and Intelligent Systems, Graduate School of Engineering, Osaka Prefecture University since 2008. His current research interests include architectures and controls for optical networks, wireless multihop networks, future Internet, and content distribution networks. He is a Fellow of the Institute of Electronics Information and Communication Engineers, Japan.