Cache-Property-Aware Features for DNS Tunneling Detection

Naotake Ishikura[*], Daishi Kondo[*], Iordan Iordanov[†], Vassilis Vassiliades[‡], and Hideki Tode[*]
[*]Osaka Prefecture University, Japan [†]Corpy & Co., Japan
[‡]Research Centre on Interactive Media, Smart Systems and Emerging Technologies (RISE), Cyprus
Email: {ishikura@com., daishi.kondo@, tode@}cs.osakafu-u.ac.jp, iordan@corpy.co.jp, v.vassiliades@rise.org.cy

***Abstract*—A lot of enterprises are under threat of targeted attacks causing data exfiltration. As a means of performing the attacks, attackers and their malware have exploited DNS tunneling in recent years. Although there are many research efforts to detect DNS tunneling, the previously proposed methods rely on features that the malicious entities can easily obfuscate by mimicking legitimate ones. Therefore, this obfuscation would result in data leakage. In order to mitigate this issue, we focus on a trace of DNS tunneling, which cannot be easily hidden. In the context of DNS data exfiltration, malware connects directly to the DNS cache server, and a DNS tunneling query produces a cache miss with absolute certainty. In this work, we propose features derived from this cache property. Our extensive experiments show that one of the proposed features can clearly distinguish DNS tunneling traffic, which makes it useful to design and implement a solid DNS firewall against DNS tunneling.**

I. Introduction

There exist various protocols in the Internet, and by exploiting their vulnerabilities, attackers and their malware perform targeted attacks causing data exfiltration [1], [2]. This data leakage puts enterprises at great disadvantage and has a drastic negative impact on profitability. In recent years, to perform this kind of attacks, attackers and their malware have abused DNS tunneling [3], which is a security threat used to tunnel data and commands by exploiting a domain name in the DNS queries and the corresponding DNS responses. In general, an enterprise enforces access control of ports and protocols that are not usually utilized (e.g., Peer-to-Peer (P2P) file sharing like BitTorrent) for the employees. However, since DNS is an indispensable protocol to implement many services, such as content distribution, the DNS use is not restricted. Therefore, this DNS operation unfortunately provides attackers and their malware with an opportunity to realize targeted attacks through DNS tunneling.

Against DNS tunneling, several countermeasures have been proposed [4]–[15]. Indeed, these methods are effective to detect tunneling traffic from malware, such as `Morto worm` [16], or DNS tunneling tools, such as `dnscat2` [17]. However, these countermeasures are built based on features that the malicious entities can easily obfuscate by mimicking benign ones. For instance, steganography can hide leaked data in a Fully Qualified Domain Name (FQDN) of the tunneling query, which makes the FQDN look legitimate and invalidates filters relying on features of the FQDN. Thus, this obfuscation would result in data leakage.

To cope with this problem, we focus on the *nature* of DNS tunneling, which can be observed in the DNS cache server where malware connects directly as a source of tunneling query. To exfiltrate successfully data attached to the domain name of a DNS query, a source such as the one described above must avoid producing a cache hit to the query on the DNS cache server; otherwise, the data cannot be leaked outside of the enterprise. In other words, leaking data through DNS tunneling definitely produces a cache miss on the DNS cache server. This cache miss is a "trace" of the data leakage. We believe that this *cache property* is more tolerant than the features used in the conventional methods against feature obfuscation since *a cache entry is expected to be used to handle a future query that had been requested previously (i.e., a cache hit), which is directly opposed to the nature of DNS tunneling*.

Considering the above fact, this paper proposes two features derived from the cache property: *cache hit ratio* (*CHR*) and *access hit ratio* (*AHR*). Our extensive experiments show that *AHR* addresses some shortcomings of *CHR* and clearly characterizes DNS tunneling traffic. Therefore, *AHR* is useful to design and implement a solid DNS firewall against DNS tunneling. To the best of our knowledge, this is the first research work to investigate cache-property-aware features for DNS tunneling detection.

The remainder of this paper is organized as follows. Section II summarizes the basics of DNS tunneling and several existing works about detection methods of DNS tunneling. Section III proposes two cache-property-aware features to detect DNS tunneling, while Section IV evaluates our proposed features. Section V discusses our findings, and, finally, Section VI concludes this paper.

II. Background

A. DNS Tunneling Basics

DNS tunneling exploits the domain name in DNS queries and the corresponding DNS responses. Data and commands are tunneled between the malware and the attacker in the context of targeted attacks causing data exfiltration (Fig. 1). Assume that the malware and the attacker have already shared a domain name `attacker.com` for their covert channel. 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 to resolve an FQDN, the DNS cache server iteratively asks 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 sending an answer to the command and obtaining a new command, eventually the malware leaks the collected confidential information to the outside attacker in the same manner (i.e., adding the leaked information in the domain name). In order for the malware to send malicious DNS queries to the attacker safely, the queries must not cause the corresponding cache hit on the DNS cache server, which is the characteristic of DNS tunneling. In this paper, we assume that exfiltrated data is like credit card information (such an attack scenario is considered also in [11], [14], [15]) and all the generated FQDNs to leak such data are unique.

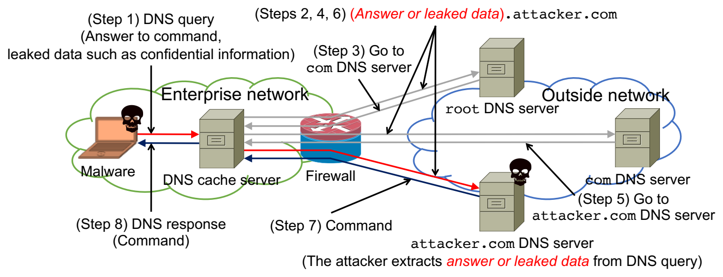


Fig. 1. An overview of DNS tunneling.

B. Related Work

In general, there are two types of analysis 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, while the traffic analysis is an evaluation of DNS traffic over a monitoring period (e.g., in terms of time, number of samples, etc.). The features of interest in payload analysis are, for example, unigram or bigram character frequencies of domains or subdomains [4], [5], FQDN length [11], [15], entropy [11], [13], [15], the first 512 bytes of a DNS response [12], and the number of labels [15]. Regarding traffic analysis, the usual features of interest are, for example, the number of flows [6], Jensen-Shannon divergence computed from DNS query payloads [7], access counts of resource records [8], average time interval between a query and its response [9], a combination of Principal Component Analysis (PCA) and mutual information [10], the number of queries per domain [13], and average query length per domain [14]. These features are fed to machine learning models or used as thresholds to build countermeasures for DNS tunneling detection. The proposed detection methods, however, 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 by steganography. Therefore, this obfuscation would accomplish data leakage.

III. Cache-Property-Aware Features for DNS Tunneling Detection

Fujiwara *et al.* [18] report that *CHR* on the DNS cache server[1] inside the University of Tsukuba in November 2011 was 75.1%. When DNS tunneling is employed, a decrease of *CHR* is expected since the generated malicious DNS queries to exfiltrate data cause cache misses, as discussed in Section II-A. To the best of our knowledge, such a characteristic of DNS tunneling has not yet been investigated in the related works introduced in Section II-B. In this section, we propose two cache-property-aware features 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 that is defined by the following formula: $CHR^n = N^n_{CH}/n$. 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 to the queries within n. Note that in this paper we define the cache hit as a state in which the response to a query of the DNS client is discovered in the connected DNS cache server without sending any queries to authoritative DNS servers. Experimental results regarding time series data are reported in Section IV-B. We show plots of *CHR* derived from all generated queries by using the latest n queries in a sliding window manner.

CHR is quite a naive feature derived from the cache property to identify DNS tunneling traffic, and it has two shortcomings. The *first* is that caches of resource records that a client query rarely looks up do not contribute to improve *CHR*. According to [18, Table 2], 90.7% of queries from the DNS clients were for `A` and `AAAA` records, and therefore, caching these resource records can increase *CHR*. However, resource records such as `NS` record, which are not often looked up by DNS clients, are also cached to mitigate the load of authoritative DNS servers. Such records do not contribute to detecting DNS tunneling (i.e., unnecessary caches for DNS tunneling detection), so they might induce a decrease of *CHR*.

The *second* is that caches are evicted based on their Time to Live (TTL), besides caching algorithms such as Least Recently Used (LRU). Fujiwara *et al.* [18] show that setting a low TTL value ($\leq$300), which realizes DNS-based wide-area load balancing, decreases *CHR*. Moreover, the TTL itself essentially causes a cache miss and is not a factor useful to characterize DNS tunneling traffic. It is therefore difficult to distinguish whether a decrease in *CHR* is owed to DNS tunneling or to these inherent shortcomings.

B. Access Hit Ratio

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

TABLE I
A Comparison between a Cache Entry and an Access Entry

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

We propose now a second feature, which we call the *access hit ratio* AHR^n, defined by the following formula: $AHR^n = N^n_{AH}/n$. Here, n is the number of queries under observation (i.e., a window size), and N^n_{AH} is the number of successful access hits to the queries within n. Experimental results as to time series data in Section IV-B plot *AHR* derived from every generated query by using the latest n queries in a sliding window manner.

[1]The authors define *CHR* as (Total # of client queries that do not cause any queries to authoritative DNS servers)/(Total # of client queries).

IV. Experiments

A. Experimental Setup

For our DNS traffic monitoring experiments, we install a DNS cache server on the local network of our laboratory at Osaka Prefecture University and capture 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. Note that we assume that the tunneling client is legitimate, but unfortunately infected by malware (i.e., installed a DNS tunneling client). So, the client produces both legitimate and tunneling traffic. **In our experiments, while generating the tunneling traffic, the client produces legitimate DNS traffic by performing web browsing and launching some background applications such as Slack.** The authoritative DNS server delegates the domain name for DNS tunneling to the DNS tunneling server, which causes the DNS cache server to forward malicious queries generated by the tunneling client to the tunneling server. We exploit `dnscat2` [17] as a DNS tunneling tool in the tunneling client and the tunneling server. Note that due to the nature of DNS tunneling (i.e., cache misses occur to exfiltrate data) we would have obtained the same results with different tools. Before performing tunneling experiments, we create a list of cache entries and access entries by capturing DNS traffic from 17 clients in our laboratory for **31 days**. Table II summarizes the parameters for our tunneling experiments. We prepare three data exfiltration scenarios in terms of tunneling query transmission interval: scenarios 1, 2, and 3 with transmission intervals of 1, 10, and 100 seconds, respectively. We use a list of cache entries only for scenario 1 since this scenario demonstrates clearly enough the effectiveness of *AHR* against the shortcomings of *CHR*.

TABLE II
Parameters for tunneling experiments

Parameter	Value		
	Scenario 1	Scenario 2	Scenario 3
Tunneling query transmission interval	1 sec	10 sec	100 sec
# of clients (incl. one tunneling client)	16	16	15
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		

B. Results

Fig. 2 shows the scatter plot of the time series data for *CHR* collected from all the clients in scenario 1, and *AHR* collected from all the clients in scenarios 1, 2, and 3, for $n = 100$. Fig. 3 shows the traffic of the tunneling client extracted from Fig. 2. To compute *CHR* and *AHR* of all the clients, a memory to store the latest n queries is prepared for each client, and the first *CHR* and *AHR* are calculated after n queries arrive. The red curve in Figs. 2 and 3 indicates that DNS tunneling traffic is generated by the tunneling client during the tunneling traffic generation period. These figures illustrate that both of *CHR* and *AHR* decrease approximately when the DNS tunneling traffic is produced.

From Figs. 2(a) and 3(a), during the monitoring period, *CHR* cannot distinguish clearly whether the decrease of *CHR* is caused by the tunneling because of the two drawbacks of cache entries discussed in Section III-A. In our 31-day dataset, 99.7% of the queries from all clients were for `A` and `AAAA` records while 51.2% of cache entries were for `A` and `AAAA` records. Fig. 4 shows the cumulative distribution functions (CDF) of TTL of unique `A` and `AAAA` records in our 31-day dataset, and the CDF indicates that 43.7% of `A` and `AAAA` records were with TTL of less than 300 sec. These statistics are related to the two drawbacks discussed in Section III-A.

On the other hand, Figs. 2(b), 3(b), 2(c), and 3(c) clearly identify the decrease of *AHR*, which is caused by the tunneling. These figures describe that *AHR* can handle these drawbacks and successfully distinguish the tunneling traffic. From Figs. 2(d) and 3(d) we see that during the tunneling traffic generation period *AHR* does not decrease drastically, so *AHR* cannot distinguish the tunneling traffic when the tunneling query transmission interval is large; this is a vulnerability of *AHR*.

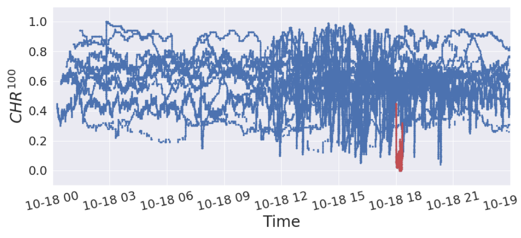
Fig. 5 illustrates the CDFs of query transmission interval of the tunneling client in scenarios 1, 2, and 3, excluding the tunneling query transmission interval. From the CDFs, 62.4%, 58.3% and 48.3% of the intervals have length less than 1 sec in scenarios 1, 2, and 3, respectively. Considering a higher threshold, 73.7%, 69.9%, and 61.8% of the intervals have duration less than 10 sec, for each scenario, and also, 96.3%, 94.5% and 93.3% of the intervals have duration less than 100 sec for each scenario. Therefore, our parameter settings generate tunneling queries with a reasonable time interval compared to general legitimate query traffic.

Fig. 6 shows the minimum AHR^n for tunneling and legitimate traffic in scenarios 1, 2, and 3. Here, the minimum AHR^n for tunneling traffic is calculated based on (a) the traffic produced by the tunneling client for 20 mins during the tunneling traffic generation period, and (b) the first n queries after generating the last tunneling query for each scenario. The minimum AHR^n for legitimate traffic in scenarios 1, 2, and 3 is computed based on traffic from all clients except the above traffic. The minimum AHR^n for tunneling traffic in scenarios 1 and 2 is lower than the one for legitimate traffic in scenarios 1, 2, and 3, which means that these traffic can be easily classified. We heuristically compute a detection threshold (y) from the minimum *AHR* by fitting a polynomial of degree 3 between the two curves for the legitimate traffic in scenarios 1, 2, and 3 and tunneling traffic in scenario 2:

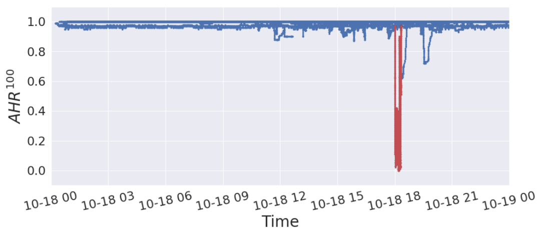
$$y = \frac{n^3 - 609n^2 + 125907n - 372267}{12500000}, \quad 0 < n < 300.$$

See Fig. 6 for an illustration. Thus, if $AHR < y$, the traffic can be classified as tunneling. By contrast, in scenario 3 (large tunneling query transmission interval), it is impossible to classify traffic as the corresponding minimum *AHR* values of legitimate traffic are smaller than those of tunneling traffic.

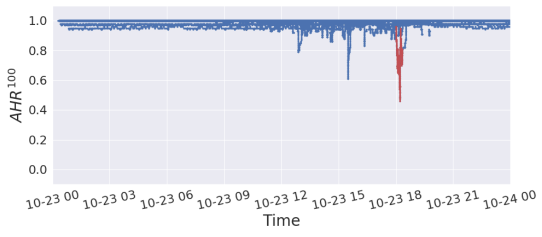
Fig. 7 shows queried FQDN ranking versus the number of DNS queries from all the clients, which can be obtained by analyzing our 31-day dataset. As for the definition of the FQDN ranking, an FQDN is ranked by the number of DNS queries including the corresponding FQDN. Fig. 7 indicates that popular FQDNs are repeatedly requested by the clients, roughly following Zipf's law, and this fact supports the results of *AHR* shown in this section.



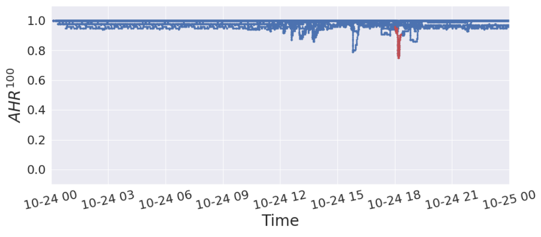
(a) CHR^{100} of all the clients in scenario 1.



(b) AHR^{100} of all the clients in scenario 1.

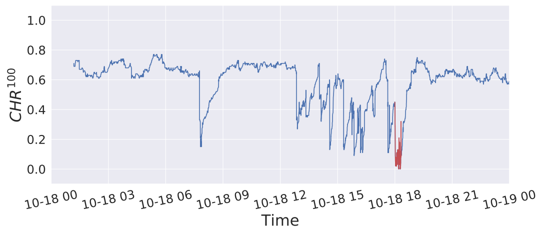


(c) AHR^{100} of all the clients in scenario 2.

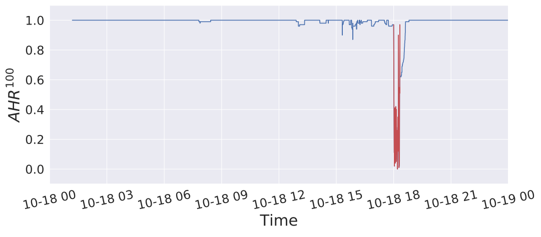


(d) AHR^{100} of all the clients in scenario 3.

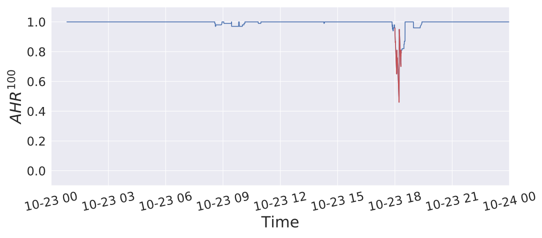
Fig. 2. Time series data of CHR^{100} of all the clients in scenario 1 and AHR^{100} of all the clients in scenarios 1, 2, and 3.



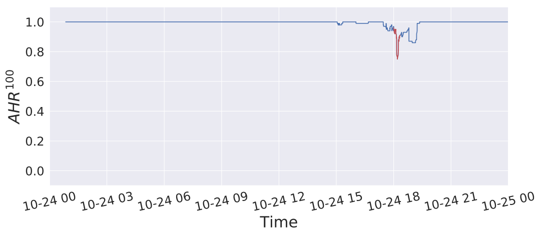
(a) CHR^{100} of the tunneling client in scenario 1.



(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.

Fig. 3. Time series data of CHR^{100} of the tunneling client in scenario 1 and AHR^{100} of the tunneling clients in scenarios 1, 2, and 3.

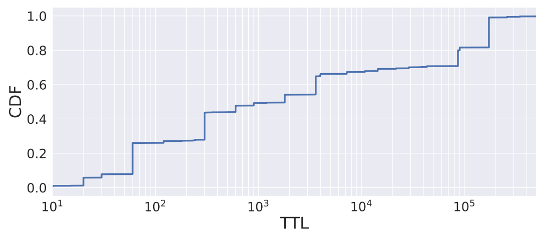


Fig. 4. CDF of TTL (sec) of unique `A` and `AAAA` records in our 31-day dataset.

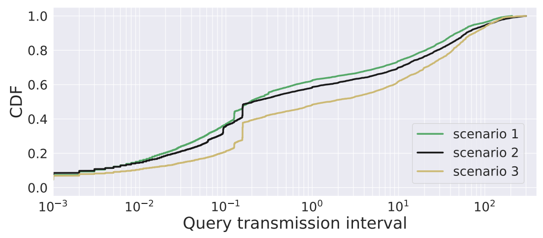


Fig. 5. CDFs of query transmission interval (sec) of the tunneling client in scenarios 1, 2, and 3, which exclude the tunneling query transmission interval.

V. Discussion

From Figs. 2(b), 2(c), 2(d), 3(b), 3(c), and 3(d), we can see a decrease of *AHR* even when only the legitimate traffic was generated. This was caused by new web access that has not occurred at all during capturing our DNS traffic for 31 days. Specifically, such new web access sends a certain number of queries not included in the list of access entries for a short term and this makes *AHR* drastically decrease, especially, in case of smaller n cases by definition. For instance, a client accessing `www.amazon.com` sent out 78 queries within 5 seconds from the beginning of the access. A naive filter derived from *AHR* might give a false alarm for this kind of new web access.

From malware's point of view, malware can send tunneling queries without decreasing *AHR* drastically (i.e., circumvention of a filter using *AHR*) in the following two ways. The first

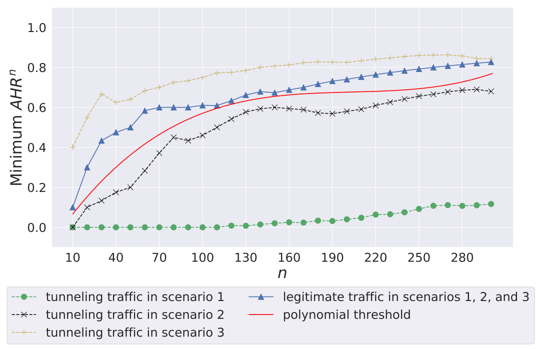


Fig. 6. Minimum AHR^n for legitimate and tunneling traffic in scenarios 1, 2, and 3.

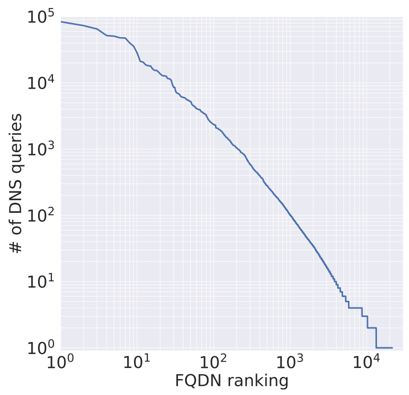


Fig. 7. FQDN ranking vs. the number of DNS queries.

is for malware to send tunneling queries as well as frequent DNS ones whose FQDNs are normally expected to be stored in access entries, which can intentionally increase *AHR*. Indeed, from Figs. 2(b), 2(c), 2(d), 3(b), 3(c), we can observe that an instantaneous increase of *AHR* occurs during the tunneling traffic generation term, which is mainly derived from web browsing (see our assumption about the tunneling client in Section IV-A), though the tunneling traffic is successfully characterized in scenarios 1 and 2. This phenomenon might be exploited for the circumvention by malware. However, we believe that this malware's counterattack cannot improve tunneling throughput. If malware tries to send such previously sent queries frequently, this behavior itself should be identified as anomalous by, for example, a feature of frequency of sending DNS queries. This anomaly should be exposed thanks to our proposed feature, and obfuscation of this anomaly results in low tunneling throughput.

The second is for malware to send tunneling queries very slowly. According to our experiments, when the tunneling query transmission interval is too large, it should be extremely difficult for *AHR* to characterize the tunneling, which is a limitation of our proposed feature. To mitigate the limitation, we need to investigate another cache-property-aware feature (e.g., focusing on access misses themselves, not ratio of access hits) toward proposing a rare event detection method, which is our future work.

VI. Conclusion

Various countermeasures against DNS tunneling have been proposed, but these are built based on DNS tunneling features that the malicious entities can easily obfuscate by mimicking benign ones. Therefore, conventional approaches are not tolerant 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 where the client connects. Based on this observation, we proposed cache-property-aware features for DNS tunneling detection. Our extensive experiments showed that access hit ratio could clearly characterize DNS tunneling traffic that generates tunneling queries with a reasonable time interval compared to general legitimate query traffic. As a next step, we will implement a solid DNS firewall against DNS tunneling by utilizing access hit ratio.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number JP19K24351. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 739578 complemented by the Government of the Republic of Cyprus through the Directorate General for European Programmes, Coordination and Development.

References

[1] IT Security Risks Survey 2014, https://media.kaspersky.com/en/IT_Security_Risks_Survey_2014_Global_report.pdf

[2] Understanding Targeted Attacks: The Impact of Targeted Attacks, https://www.trendmicro.com/vinfo/us/security/news/cyber-attacks/the-impact-of-targeted-attacks

[3] DNS Tunneling in the Wild: Overview of OilRig's DNS Tunneling, https://unit42.paloaltonetworks.com/dns-tunneling-in-the-wild-overview-of-oilrigs-dns-tunneling/

[4] K. Born and D. Gustafson, "Detecting DNS Tunnels Using Character Frequency Analysis," https://arxiv.org/pdf/1004.4358v1.pdf

[5] C. Qi, X. Chen, C. Xu, J. Shi, and P. Liu, "A Bigram Based Real Time DNS Tunnel Detection Approach," in *ITQM* 2013.

[6] W. Ellens, P. Żuraniewski, A. Sperotto, H. Schotanus, M. Mandjes, and E. Meeuwissen, "Flow-Based Detection of DNS Tunnels," in *AIMS* 2013.

[7] K. Xu, P. Butler, S. Saha, and D. Yao, "DNS for Massive-Scale Command and Control," *IEEE TDSC*, vol. 10, no. 3, pp. 143–153, 2013.

[8] A. M. Kara, H. Binsalleeh, M. Mannan, A. Youssef, and M. Debbabi, "Detection of Malicious Payload Distribution Channels in DNS," in *IEEE ICC* 2014.

[9] 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, Jul. 2014.

[10] E. Cambiaso, M. Aiello, M. Mongelli, and G. Papaleo, "Feature Transformation and Mutual Information for DNS Tunneling Analysis," in *ICUFN* 2016.

[11] A. Das, M. Shen, M. Shashanka, and J. Wang, "Detection of Exfiltration and Tunneling over DNS," in *IEEE ICMLA* 2017.

[12] C. Lai, B. Huang, S. Huang, C. Mao, and H. Lee, "Detection of DNS Tunneling by Feature-Free Mechanism," in *IEEE DSC* 2018.

[13] J. Steadman and S. Scott-Hayward, "DNSxD: Detecting Data Exfiltration Over DNS," in *IEEE NFV-SDN* 2018.

[14] A. Nadler, A. Aminov, and A. Shabtai, "Detection of Malicious and Low Throughput Data Exfiltration over the DNS Protocol," in *Computers & Security*, vol. 80, pp. 36–53, Jan. 2019.

[15] J. Ahmed, H. H. Gharakheili, Q. Raza, C. Russell, and V. Sivaraman, "Real-Time Detection of DNS Exfiltration and Tunneling from Enterprise Networks," in *IFIP/IEEE IM* 2019.

[16] Morto Worm Sets a (DNS) Record, https://www.symantec.com/connect/blogs/morto-worm-sets-dns-record

[17] https://github.com/iagox86/dnscat2

[18] K. Fujiwara, A. Sato, and K. Yoshida, "DNS Traffic Analysis – CDN and the World IPv6 Launch," *Journal of Information Processing*, vol. 21, no. 3, pp. 517–526, 2013.