# Information-Based Heavy Hitters for Real-Time DNS Data Exfiltration Detection

Yarin Ozery
Ben-Gurion University of the Negev
Akamai Technologies, inc.
yarinoz@post.bgu.ac.il

Asaf Nadler
Ben-Gurion University of the Negev
asafnadl@post.bgu.ac.il

Asaf Shabtai
Ben-Gurion University of the Negev
shabtaia@bgu.ac.il

***Abstract*—Data exfiltration over the DNS protocol and its detection have been researched extensively in recent years. Prior studies focused on offline detection methods, which although capable of detecting attacks, allow a large amount of data to be exfiltrated before the attack is detected and dealt with. In this paper, we introduce Information-based Heavy Hitters (*ibHH*), a real-time detection method which is based on live estimations of the amount of information transmitted to registered domains. *ibHH* uses constant-size memory and supports constant-time queries, which makes it suitable for deployment on recursive DNS servers to further reduce detection and response time. In our evaluation, we compared the performance of the proposed method to that of leading state-of-the-art DNS exfiltration detection methods on real-world datasets comprising over 250 billion DNS queries. The evaluation demonstrates *ibHH*'s ability to successfully detect exfiltration rates as slow as 0.7B/s, with a false positive alert rate of less than 0.004, with significantly lower resource consumption compared to other methods.**

## I. Introduction

Data exfiltration is performed by malicious actors in order to steal data from a network. Once the data is collected, adversaries often utilize various techniques, including compression and encryption, to obfuscate the information and evade detection when removing it. The methods used to extract data from a targeted network typically involve transmitting it through a covert communication channel. Adversaries may also impose limitations on the data transmission size to minimize the risk of detection. The data exfiltration stage usually marks the final phase in the malware lifecycle [1].

The *Domain Name System* (DNS) protocol [2] is a crucial network protocol, which is primarily used to translate easy-to-remember domain names into corresponding IP addresses. While enterprises and government organizations employ various defensive measures to safeguard their users and networks against malicious actors, the DNS protocol is typically left unblocked and inadequately monitored due to its critical role in facilitating users' Internet access [3]. Given the vulnerable and exploitable characteristics of the DNS protocol, malicious actors targeting enterprise and government organizations for data theft often choose to exploit it as a means of data exfiltration and covert communication. This practice is known as *DNS exfiltration* [4].

Initiating DNS exfiltration is a straightforward and cost-effective process, as the attacker simply needs to register a domain (e.g., *attacker.com*) with a domain name registrar and assign an authoritative name server under their control to that domain. This allows the malware installed on the compromised host to exfiltrate data by encoding it within DNS packets directed towards the registered domain. In the DNS query resolution process, the queries are forwarded from the client to the authoritative name server associated with the queried domain. As a result, queries aimed at *attacker.com* are forwarded to the attacker-controlled authoritative name server, enabling successful data exchange between the malware and the attacker. In addition, the attacker can utilize DNS responses to encode messages sent to the malware, enabling a bidirectional covert communication channel through the DNS exfiltration tunnel. This channel can be exploited for other purposes, such as command and control (C&C) operations.

There are various ways to exfiltrate data over the DNS: (1) encoding data within the DNS query name (i.e., the target domain name to be resolved); (2) using the DNS query type field, which indicates the type of DNS resource record the client is trying to resolve, in order to encode a small amount of information (up to 16 bits) in the DNS packet; and (3) timing-based exfiltration, in which the query arrival time is used as a means of conveying information [5]. Numerous instances of malicious actors leveraging the DNS for data exfiltration have been documented, including activities by state-sponsored threat groups [6], [7], [8], [9], [10], as well as by ransomware actors [11]. Therefore, it is no surprise that significant research attention has been dedicated to DNS exfiltration and its detection in recent years [12].

Many DNS exfiltration methods have been proposed to date, ranging from rule-based techniques [13], [14], statistical-based techniques [15], [16], supervised machine learning techniques [17], unsupervised machine learning techniques [18], [19], and even deep learning techniques [20], [21], [22].

Despite the extensive body of research on DNS exfiltration, limited attention has been directed towards real-time detection methods. The majority of existing solutions are ill-suited for online deployment, and the proposed approaches have predominantly operated in an offline manner. Offline detection methods are a major concern as they allow for substantial data exfiltration to occur before the attack is identified and thwarted.

Network and Distributed System Security (NDSS) Symposium 2024
26 February - 1 March 2024, San Diego, CA, USA
ISBN 1-891562-93-2
https://dx.doi.org/10.14722/ndss.2024.24388
www.ndss-symposium.org

In contrast, real-time DNS exfiltration detection enables a rapid response from network operators, effectively reducing the potential damage inflicted by breaches. In order to provide real-time detection capabilities, a solution should not rely on an external data collection process but rather be executed directly on the DNS queries stream resolved by the resolver. By integrating the detection functionality within the resolver itself, the solution can effectively analyze and classify queries without introducing delays or disruptions to the resolution process.

Given that a real-time detection solution should run directly on the resolver, it is crucial to ensure that the DNS resolution process remains unaffected by the detection mechanism. An effective solution should therefore possess low memory and computational demands while maintaining the ability to process and classify a large volume of queries per second.

In this paper, we introduce *Information-based Heavy Hitters* (*ibHH*), a novel and interpretable method for real-time DNS exfiltration detection. Our approach utilizes a threshold-based method to quantify the unique information transmitted through subdomains within DNS queries and raise alert if the suspected amount of data exfiltrated exceeds that threshold, making it transparent and explainable. To achieve real-time detection, we employ a fixed-size data structure that efficiently processes a continuous stream of DNS queries, inspired by the concept of identifying heavy hitters in data streams [23].

*ibHH* incorporates the *HyperLogLog* sketching algorithm [24] from the field of big data [25] and leverages weighted sampling techniques. This combination enables our method to accurately estimate the volume of information transmitted from clients to registered domains through subdomains. By comparing this estimated quantity against a predefined and configurable detection threshold, *ibHH* can raise an alert when the transmitted information surpasses the threshold. An overview of the method, along with a possible DNS exfiltration scenario, is presented in Figure 1. In order to enable reproducibility of our results and allow further research, we provide an open-source [1] Python implementation of our proposed solution.

Our experiments demonstrate *ibHH*'s high performance and its ability to process over 600,000 queries per second. This makes it well-suited for deployment in large-scale networks where real-time processing and performance are crucial. Additionally, *ibHH* maintains its efficacy in resource-constrained environments with limited computational and memory resources. Therefore, it can be effectively employed in small-scale environments without straining available resources.

An additional advantage of *ibHH* is that its operation does not rely on labeled training data. This characteristic makes the need for data annotation redundant and facilitates easier deployment and maintenance of the method. To handle false positive cases, we propose two reputation-based allowlisting approaches, which are described in Section IV-E.

We performed a thorough evaluation of *ibHH* to assess its effectiveness in detecting DNS exfiltration and its ability to handle false positive cases and compare it with three state-of-the-art (SOTA) methods proposed in prior studies. To ensure the reliability of our results, we collected a real-world DNS query dataset containing over 50 billion queries. This extensive dataset serves as a robust foundation for our evaluation, allowing us to make reliable assessments of *ibHH*'s performance. In addition to the real-world dataset, we performed the evaluation on a publicly available dataset. This enables reproduction of our results and external validation of our findings.

To further enhance the validity and robustness of our evaluation, we concluded the assessment by using a second real-world dataset. This additional dataset consists of over *250 billion queries*, collected over a period of three weeks. Our evaluation on this dataset is discussed in Section V-L. We also evaluated the resource utilization of the compared methods, specifically focusing on memory consumption and compute time (see Section V-M). To the best of our knowledge, this evaluation is the most extensive and rigorous evaluation conducted in the field.

To summarize, the contributions of our work are: (1) *ibHH* - A lightweight and simple real-time DNS exfiltration detection method, which is appropriate for large-scale high throughput networks as well as small networks; (2) Evaluation of our method against SOTA methods on large-scale real-world datasets as well as a publicly available dataset; and (3) Open-source Python implementation of our method, which will enable reproduction of our results and support further research.

## II. Background

### A. DNS stream model

We model DNS stream queries on the data stream model presented in [26]:

*Definition 1 (Data Stream):* A **Data Stream** $S$ is an ordered set of elements $x_1, x_2, ...x_n$ where each element is observed exactly once.

Given definition 1, in the scope of this research, each element $x_i$ is comprised of a pair of values ($k_i$, $v_i$), where the values are taken from domains $K$ and $V$, respectively. $k_i$ is called the key of the pair, while $v_i$ is called the subkey of the pair. More specifically, each element in the DNS stream is a DNS query's qname [2], extracted to $(domain, subdomain)$ pair, where $domain$ is the second-level domain (which we denote as primary domain, or just domain, for the rest of the paper) and $subdomain$ is the concatenation of the rest of the labels of the qname. For example, given the qname **a.b.example.com**, the domain is **example.com.**, the subdomain is **a.b**, and the DNS query stream element is (**example.com.**, **a.b**).

### B. Real-time DNS exfiltration detection

The DNS protocol has traditionally been low-demanding [27]; therefore, DNS resolvers are usually deployed on limited hardware [28]. Given that knowledge, we define criteria which must be satisfied in order for a DNS exfiltration detection algorithm to be considered appropriate for real-time detection: (i) Given a DNS stream of length $n$, the amount of space required should be sublinear with regard to $n$, i.e., have space complexity of $o(n)$. (ii) The classification of a given DNS query should have time complexity of $\Theta(1)$. These criteria are designed to ensure that a real-time DNS exfiltration detection solution can run on the DNS resolver

[1] Source code available at https://github.com/akamai/Information-based-Heavy-Hitters-for-Real-Time-DNS-Exfiltration-Detection

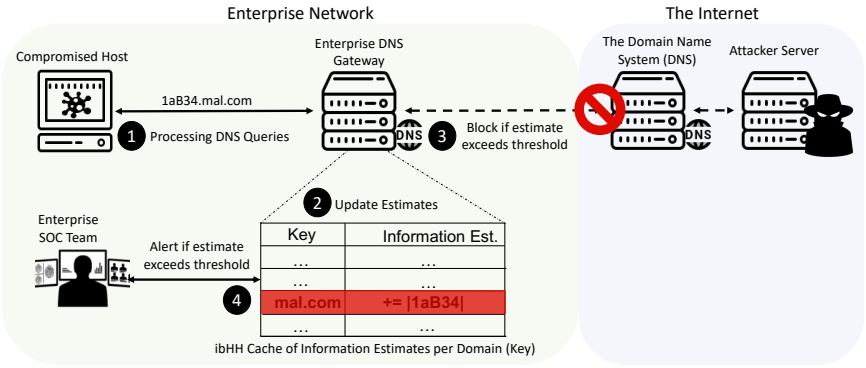



Fig. 1: Overview of *ibHH*. When a compromised host performs DNS queries (1) the enterprise DNS gateway intercepts the query, estimates the amount of information it contains and updates its internal information estimate cache (2). After the update, if the amount of information exceeds a predefined threshold, the query is blocked from reaching the attacker's server and (3) an alerted is raised for the enterprise's SOC team (4).

server, without impacting the DNS resolution throughput of the server or have a large memory footprint, which is needed for the DNS protocol's caching mechanism [2].

## III. Related Work

The topic of detecting data exfiltration over the DNS protocol has been the subject of nearly 30 recent studies [12].

**Offline detection methods by design.** There is a wide variety of methods whose design limits them from being applied in real time. The most notable design limitation is time-based aggregation feature extraction; for example, Nadler et al. [18] proposed an anomaly-based isolation forest [29] model to detect both high and low throughput DNS exfiltration, based on features extracted over a sliding window of size $\lambda$ hours; classification is done based on the latest $n_s$ windows, meaning up to $n_s * \lambda$ hours can pass by the time of detection. This also means that at any given moment, the number of DNS queries that need to be stored in memory is $\Omega(\lambda \cdot n_s)$, and therefore it inherently cannot run in real time. Ishikura et al. [14] proposed a DNS exfiltration detection solution based on what they called cache-property-aware features. For each client in an enterprise network, they suggested maintaining a list of the last $n$ Fully Qualified Domain Names (FQDNs) the client accessed, in order to calculate a property called Access Miss Count, which indicates the number of FQDNs queried by the client in the last $t$ seconds. The authors proposed both a rule-based model and a LSTM based model to identify DNS exfiltration activity. The memory requirements of the solution grow linearly with the amount of clients in the network (both for the access list and the LSTM state of each client). Moreover, the solution does not identify which FQDN is suspected as being used for DNS exfiltration but rather only determines if exfiltration has occurred in a given time window (which ranged between 100 and 1,200 seconds in their experiments); this makes it difficult for security operations teams to identify the malicious domain.

Paxson et al. [13] presented an information-based method that provides an upper bound on the amount of information that can possibly be transmitted via DNS queries. The method groups DNS queries per primary domain and DNS source IP on a daily basis, which is followed by lossless compression of the different possible information vectors (query name, query timing, and query type); the minimal value of these is output as the upper bound on the amount of information. The upper bound is then compared with a predefined threshold, and alerts are raised for any communication which exceeds this value. This method is designed to run on window-aggregated data of size $w$ and has the benefit of being able to detect DNS exfiltration regardless of the information vector. However, both the time and space complexity of the method are $\Omega(w)$, because all of the queries in $w$ need to be kept for the information estimation stage, and the time complexity is $\Omega(w)$ due to the compression performed to estimate the information.

**Compute-intensive detection methods.** In recent years, deep learning-based (DL) DNS exfiltration detection methods have been proposed [30], [20], [21]. Chen et al. [21] proposed a DL architecture based on the combination of a CNN and LSTM models. Their model is then trained on labeled DNS queries (benign and malicious). Wu et al. [22] proposed TDAE, an autoencoder DL DNS exfiltration detection method based on semi-supervised learning, which means it requires some labeled data. While DL models generally provide high accuracy and automatic feature extraction, they come with the cost of requiring larger training datasets than traditional machine learning methods. More importantly, they are known to be hardware-intensive [31], [32], which makes them unsuitable for deployment on the network perimeter.

**Supervised learning methods.** Several methods that rely on labeled data for training [17], [33], [34], [35] have been proposed. This is reasonable for identifying a predetermined set of known DNS exfiltration tools, but as shown in [18], the absence of high-quality publicly-available datasets prevents these methods from identifying unfamiliar DNS exfiltration malware. Our proposed method does not require any labeled data for training.

**Real-time detection methods.** To the best of our knowledge, only two previous studies focused on real-time detection [15], [19]. Qi et al. [15] suggested a detection technique based on bigram (subsequent pairs of characters) frequency statistics. The authors described a score mechanism based on the expected value of the bi-gram character frequency as the

score of the primary domain. The model is trained on labeled benign and malicious data to determine the score threshold that will make the classifier produce the least number of false positive alerts, which is then used by the classifier in the online phase. While it is reasonable to expect this kind of classifier to be run in real time, it suffers from the same limitation of other supervised learning methods – it has difficulty generalizing to unfamiliar DNS exfiltration techniques. Ahmed et al. [19] proposed an unsupervised isolation forest model, which is based on classifying queries on a per packet basis. Many features are extracted and used, such as the length of the query name, length of the query subdomain, query name entropy [36], count of numerical characters in the query name, count of uppercase letters in the query name, number of DNS query labels, maximum label length, and average label length. While the isolation forest model [29] is relatively lightweight in terms of classification time and memory requirements, it is questionable how this method can scale to large-scale networks which may reach millions of queries per second [37], given the number of features that need to be extracted to perform classification. A table summarizing related studies and their methods' compliance with the real-time criteria defined in Section II-B is provided in Table I.

TABLE I: Comparison of previously proposed methods for detecting DNS exfiltration (RB-rule-based, MB-model-based).

| Study | Year | Technique | Requires Labeled Training Data | Requires Data Aggregation | Classification in Constant Time | Suitable For Real-Time Detection |
|---|---|---|---|---|---|---|
| Paxson et al. [13] | 2013 | $RB$ | ✗ | ✓ | ✗ | ✗ |
| Kara et al. [38] | 2014 | $RB$ | ✗ | ✓ | ✗ | ✗ |
| Buckzak et al. [33] | 2016 | $MB$ | ✓ | ✓ | ✗ | ✗ |
| Almusawi et al. [17] | 2018 | $MB$ | ✓ | ✗ | ✗ | ✗ |
| Homem et al. [16] | 2018 | $MB$ | ✗ | ✓ | ✓ | ✗ |
| Nadler et al. [18] | 2019 | $MB$ | ✗ | ✓ | ✗ | ✗ |
| Ahmed et al. [19] | 2019 | $MB$ | ✗ | ✗ | ✓ | ✓ |
| Palau et al. [20] | 2020 | $MB$ | ✓ | ✗ | ✗ | ✗ |
| Wu et al. [22] | 2020 | $MB$ | ✓ | ✗ | ✗ | ✗ |
| Yang et al. [34] | 2020 | $MB$ | ✓ | ✗ | ✗ | ✗ |
| Ishikura et al. [14] | 2021 | $MB/RB$ | ✗ | ✓ | ✗ | ✗ |
| Ruiling et al. [35] | 2022 | $MB$ | ✓ | ✗ | ✗ | ✗ |
| *ibHH* | 2022 | $RB$ | ✗ | ✗ | ✓ | ✓ |

## IV. Information-Based Heavy Hitters for DNS Exfiltration Detection

### A. Definitions

*Definition 2 (Information Weight):* Given a stream of elements $S$, the information weight of an element $(k_i, v_i)$, $I_{k_i,v_i}$ is the quantity of information conveyed by $v_i$.

*Definition 3 (Distinct Information Heavy Hitter):* In a stream of elements $S$, for a given $k_i \in K$, the distinct information weight $I_{k_i}$ is the total information conveyed by distinct elements with key $k_i$, i.e., $I_{k_i} = \sum_{\{v|(k_i,v)\in S\}} I_{k_i,v}$.

Key $k_i$ is a **distinct information heavy hitter** if its information weight $I_{k_i}$ is at least $\epsilon$-fraction of the total distinct information weight of the stream, where $\epsilon \in (0,1)$, i.e., $I_{k_i} \geq \epsilon \cdot \sum_{y\in K} I_y$.

### B. Information-based Heavy Hitters

Information-based Heavy Hitters (*ibHH*) is a novel method for real-time detection of DNS exfiltration, which is based on identifying domains associated with a large amount of distinct information conveyed through subdomains in a DNS query stream and inspired by the work of Afek et al. [39] in which distinct heavy hitter detection algorithms were proposed for the detection of DDoS attacks. Identifying heavy hitters in a data stream refers to the problem of finding the most frequent items in a stream, while identifying distinct heavy hitters focuses on finding the keys with the largest number of distinct subkeys in a stream. Despite the usefulness of solving these two problems for various tasks, such as DDoS detection [39] and traffic load balancing [40], they do not fully capture the complexity of DNS exfiltration detection where there is a need to account for the amount of data being exfiltrated via DNS queries. In order to model this complexity, we introduce the new concepts of *information weight* (see Definition 2) and *distinct information-based heavy hitters* (see Definition 3), which describe elements associated with large amounts of unique information in a stream. The problem of finding distinct information-based heavy hitters can be seen as a generalization of the distinct heavy hitter problem proposed in [39], where each distinct element in the stream has a weight. *ibHH* quantifies the amount of information conveyed through DNS query subdomains to domains, identifies domains associated with large amounts of unique information, and raises alerts for these domains as suspected of DNS exfiltration.

*1)* ibHH *Components:* The input for *ibHH* is a stream of DNS queries, such that for each DNS query *subdomain.example.com*, the domain and subdomain are extracted to obtain the element: $(example.com, subdomain)$. *ibHH* consists of a fixed-size cache ($Counters$) whose size ($k$) is a parameter of the algorithm; a random hash function $Hash \sim U[0,1]$ that allows us to sample the distinct DNS query stream; $detection\_threshold$, which is a parameter of the algorithm (whenever the information estimation of a cached domain exceeds it, an alert is raised); and a threshold value $\tau$ (initialized to 1 but may be updated over time), which represents the probability of a domain's inclusion in the cache. Each entry in $Counters$ stores an information counter, which is the total unique information weight (the information quantity) of $domain$ ($I_{domain}$) and $seed_{domain}$ (whose value is the minimum $Hash(domain, subdomain)$ of all elements with key $domain$ in the stream).

*2) Information Quantification:* According to Definition 3, we need to quantify the amount of unique information encoded in subkeys (subdomains) and calculate this amount per key (domain). In order to do so, we need a function $I$: $V \rightarrow I_V$, where $I(subdomain)$ is the information weight of $subdomain$. In this research, we define the information weight as $I(subdomain) = |subdomain|$, i.e., the information weight of a subkey is its length. We acknowledge that this choice does not provide an exact quantification of the information encoded in the subdomain, but it imposes an upper bound on the quantity of information that can be conveyed through it, and, as will be shown in the experiments described in Section V, it provides an adequate approximation for practical purposes. We also experimented with using entropy [36] to estimate the information, but the results were inferior.

*3) Optimized Counting with HLL++:* To calculate the amount of unique information, for each domain we need to store a set of all the associated subdomains in a stream and sum up their lengths, resulting in a linear space complexity. Since our solution is intended to run on DNS resolvers with limited memory capabilities, we cannot use exact information weight counters. Instead, we employ count-distinct approximation algorithms from the world of big data. The count-distinct

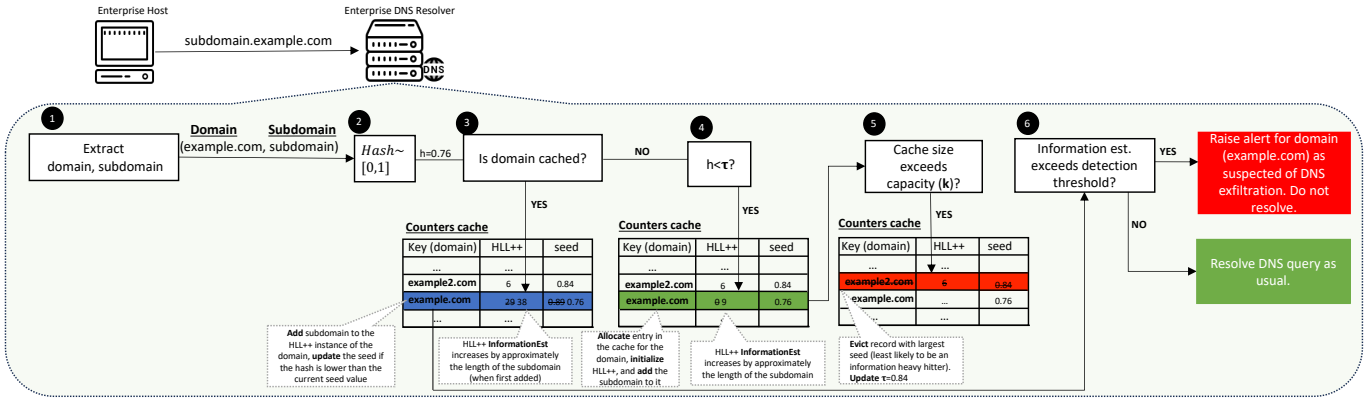



Fig. 2: *ibHH* processing a query. When an enterprise host sends a DNS query (1) the DNS gateway intercepts the query and extracts the domain and subdomain. (2) Then, the hash function is calculated for the pair ($h$). (3) If the domain is cached in $Counters$, the subdomain is inserted into the HLL++ instance of that domain, and the seed is calculated and updated. Otherwise (4), if $h$ is below the threshold $\tau$, the domain is added to $Counters$. If $Counters$ exceeds its capacity $k$ (5), the domain with largest seed value is evicted, and $\tau$ is updated to be that domain's seed value, ensuring that only the most likely information heavy hitters are in the cache. If the information estimation for the domain exceeds the detection threshold (6), an alert is raised.

problem is a well-studied problem [41], and many extremely accurate and high-performing approximation algorithms have been proposed to solve it. One of the state-of-the-art solutions for this problem is *HyperLogLog* [24].

Essentially, HyperLogLog (*HLL*) takes advantage of a clever property of multisets that the cardinality of a multiset (the number of distinct elements) of uniformly distributed random numbers can be estimated by calculating the maximum number of leading zeros in the binary representation of each number in the set. If the maximum number of leading zeros observed is $l$, then the number of distinct elements in the multiset is approximately $2^l$. In order to obtain a uniformly distributed random number multiset, a hash function is applied to the elements in the original multiset, which is the DNS stream in this case. *HLL*'s data is stored in counter arrays, which are called registers, and the size of the arrays depends on the number of bits allocated for the registers $p$. In this research, we fix $p$ at 12, in order to achieve optimal memory consumption while achieving highly accurate cardinality approximations. In the proposed method, we use a variation of the original *HLL*, known as *HyperLogLog++* (*HLL++*) [42], which provides better accuracy and uses less memory than the original design.

*HLL++* is designed to approximate the distinct number of elements in a stream, while our goal is to approximate the amount of distinct information in a stream. Therefore, we modify the insertion operation of *HLL++*, such that for each DNS query stream element $(domain, subdomain)$ instead of adding $subdomain$ to $domain$'s *HLL++* instance, for each integer $i$ in the range of $(0, length(subdomain))$, we add the concatenated string $subdomain||i$ to $domain$'s *HLL++* instance; thus, we are able to approximate the amount of distinct information associated with $domain$. The element insertion operation of the modified HLL++ structure is denoted as $Add$, while the information estimation operation is denoted as $InformationEst$.

*4) Processing Elements with* ibHH*:* Figure 2 presents an example in which *ibHH* processes a DNS query. When a DNS query, for example *subdomain.example.com*, is processed by the DNS resolver, *ibHH* first extracts the domain, *example.com*, and subdomain, *subdomain* (Step 1 in Figure 2). Then, *ibHH* calculates the hash value $h = Hash(example.com, subdomain)$ (Step 2). If *example.com* is already cached in *Counters* (Step 3), *ibHH* adds *subdomain* to the HLL++ instance of *example.com* (as described in Section IV-B3) and updates $seed_{example.com}$ to be the minimum of the current $seed$ value and $h$; in Figure 2, HLL++'s *InformationEst* is increases by (approximately) 9, from 29 to 38, which is the length of *subdomain*. If *example.com* is not cached and $h < \tau$ (Step 4), *ibHH* allocates a new entry for it, initializes an HLL++ instance, adds *subdomain* to it, and sets $seed_{example.com}$ to be $h$; in Figure 2, HLL++'s *InformationEst* is (approximately) 9, which is the length of *subdomain*. Next, *ibHH* checks if $Counters$ has exceeded its capacity $k$ (Step 5). If it has, *ibHH* evicts the cached domain with the largest $seed$ value, denoted as $seed_{max}$, and updates $\tau$ to be $seed_{max}$. Finally, *ibHH* estimates the information using the $InformationEst$ operation described in Section IV-B3. If the estimated information exceeds the detection threshold, an alert is raised for *example.com* (Step 6); otherwise, the DNS resolver continues.

$\tau$ indicates the probability of inclusion in the *ibHH* cache. Whenever the size of $Counters$ exceeds $k$, $\tau$ decreases. Thus it becomes "harder" for non-cached domains to be added to it. Whenever a query of a cached domain is processed by *ibHH* and the hash value of the *(domain, subdomain)* pair is lower than the current $seed$ value of the domain, the $seed$ is updated, ensuring that evicted domains are least likely to be the information-heavy hitters, increasing the likelihood that the cache primarily contains the most significant domains in terms of information volume. This is based on the sampling scheme introduced by Gibbons and Matias [43].

*ibHH*'s pseudocode is provided in Algorithm 1.

**Algorithm 1** *ibHH* pseudocode

**Input** $k$ - cache size, $d$ - detection threshold, stream of DNS queries

```
τ ← 1
Counters ← {}
for stream element (domain, subdomain) do
    N ← |subdomain|
    h ← Hash(domain, subdomain)
    if domain is in Counters then
        for i=0; i<N; i++ do
            subdomain_i ← subdomain||str(i)
            Counters[domain].Add(subdomain_i)
        end for
        seed_domain ← min{seed_domain, h}
        if Counters[domain].InformationEst > d then
            raise alert for domain
        end if
    else
        if h < τ then
            Counters[domain] ← newHLLPlusPlus
            seed_domain ← h
            for i=0; i<N; i++ do
                subdomain_i ← subdomain||str(i)
                Counters[domain].Add(subdomain_i)
            end for
            if |Counters| > k then
                ToDel ← argmax_{dom∈Counters} seed_dom
                τ ← seed_ToDel
                Delete Counters[toDel]
                Delete seed_ToDel
            end if
        end if
    end if
end for
```

### C. ibHH *Space and time complexity analysis*

One of *ibHH*'s benefits is the fact that it has sublinear (in fact, logarithmic) space complexity in the DNS stream length $n$, as well as constant query classification time complexity, which means that it satisfies the real-time criteria defined in Section II-B:

*1) Memory Analysis:* For our cache structure, we store k *HLL++* instances. Using the $(\epsilon, \delta)$ model [42], each *HLL++* instance requires $\mathcal{O}(\epsilon^{-2}loglog(m_{dom}) + log(m_{dom}))$ space [42], where $m_{dom}$ is the number of distinct elements associated with key $dom$. We denote $m = max_{dom \in Counters} m_{dom}$ for readability; thus the total space complexity of *ibHH* is $\mathcal{O}(k \cdot \epsilon^{-2}loglog(m_) + log(m))$, which is logarithmic in the cardinality of the data stream and therefore logarithmic in the entire data stream size, given that $m = O(n)$.

*2) Time Analysis (processing a query):* When processing a DNS stream element (domain, subdomain), we calculate the hash value $h = Hash(domain, subdomain)$, which has a time complexity of $\mathcal{O}(1)$. If **domain** is already cached, we proceed with adding **subdomain** to **domain**'s HLL++ instance. The add operation of the HLL++ algorithm has a time complexity of $\mathcal{O}(1)$, and given that a domain name is limited to 255 characters, as described in the original DNS RFC [2], the subdomain is also necessarily limited to 255 characters; therefore the time complexity of the add operation is $\mathcal{O}(1)$. Following the add operation, *ibHH* performs a $count$ operation to determine if an alert should be raised for the domain. The $count$ operation has a time complexity that depends on the number of register bits allocated to the *HLL++* instance $p$. We fix $p$ at 12; therefore this operation has a time complexity of $\mathcal{O}(1)$. We conclude that processing a query and classifying it has a constant time complexity.

### D. Reset mechanism

Given the conditional probabilistic nature of the method (which derives from the need to calculate a hash function and compare it to the value of $\tau$), domains that appear earlier in the stream are much more likely to be included than later ones (due to the decreasing nature of $\tau$); thus, the confidence interval of the counters depends on $\tau$ and decreases over time. This was also mentioned in [39], however the authors did not introduce a way of dealing with the counters' decreasing confidence intervals. In order to avoid missing information heavy hitters that appear later in the stream, we propose a reset mechanism, where the *ibHH* cache is flushed and reset at constant intervals, which allows us to identify DNS exfiltration events that occur long after the deployment of the *ibHH* algorithm in the network.

### E. Allowlists

Patterns of legitimate use resembling that of DNS exfiltration makes it difficult to distinguish between benign and malicious DNS traffic. Anti-malware agents are known to use the DNS protocol to send signatures of suspected files to the DNS zone of the anti-malware service provider for inspection [44]. Other services, such as search engines, social networks, and streaming services are known to use disposable domains for purposes of signaling [45], [46]. In order to handle false positive domains and avoid raising many false alarms, we present two simple lightweight allowlisting approaches based on the concept of global and local reputation, as described in [47]. Globally reputable domains are domains listed in publicly available lists associated with benign domains (such as TRANCO) and therefore should be trusted. Locally reputable domains, are domains queried by a large portion of hosts in the local enterprise network and therefore should be trusted. Using the allowlists in a pre-filtering phase is also important for preventing known benign domains that have many distinct subdomains from being cashed and ensuring that they do not take up cache space. In our evaluation, we apply the allowlists as a post-filter (instead of a pre-filter), in order to evaluate the effectiveness of the proposed approaches on the false positive rate of *ibHH* and the compared methods.

*1) TRANCO:* The use of top-ranking domain lists for allowlist purposes is very common in DNS exfiltration detection [19], [18], [13]. In this paper, we use TRANCO [48], an approach for ranking websites' popularity, to generate a top 1M list that allows us to filter out popular websites, reducing the number of false positive domains.

TABLE II: A summary of the datasets used in this study.

| Dataset Name | # DNS Queries | # Unique 2LD | # Org | # Hosts |
|---|---|---|---|---|
| $DS_f$ | 50,853,030,033 | 43,310,209 | 753 | Unknown |
| $DS_p$ | 5,069,006,334 | 668,456 | 223 | 129,528 |
| Ziza et al. [49] | 35,074,149 | 12,844 | N/A | 35,989 |
| $DS_r$ | 255,750,980,779 | 463,122,409 | 753 | Unknown |

*2) Peacetime/Wartime:* Using the peacetime/wartime model, which was first introduced in [39], we execute *ibHH* in a non-enforcing mode for a limited period of time. During this execution (called *peacetime* (PT)), we assume that the presence of DNS exfiltration traffic in the network is negligible (inspired by the idea of [18]), and therefore any domain that is detected as malicious by *ibHH* during this time is actually benign and should be allowlisted. We collect these domains in an allowlist called the *peacetime allowlist*. After that, *ibHH* is deployed in an enforcing mode (called *wartime* (WT)), filtering out domains that appear in the peacetime allowlist. This approach is model-agnostic; therefore we use it for *ibHH* and the compared methods in our evaluation. As will be shown, this approach is simple yet very effective. The amount of time the algorithm should run in *peacetime* mode depends on the network. In Section V-G, we describe an experiment performed to determine the frequency at which the peacetime list should be generated, as well as the amount of time it should run for.

## V. Evaluation

### A. Overview

The evaluation is divided into two parts, namely parameter tuning and comparison with other methods.

**Parameter tuning.** We compare the effect of different detection threshold values on the number of alerted queries and domains, as well as the effect of employing the proposed allowlist techniques. The method of Paxson et al. is tuned similarly and serves as a baseline.

In addition, we present two deployment settings for *ibHH*; one simulates consolidation of data to a single point and its analysis (similar to other offline methods), and the other is a deployment setting that simulates *ibHH*'s execution right on the enterprise DNS gateway, and compare their performance.

**Comparison with other methods.** In this step, we evaluate our method's detection capabilities and compare it with the capabilities of methods proposed in earlier studies, namely the studies of Paxson et al. [13], Nadler et al. [18], and Ahmed et al. [19]. Our evaluation includes both detection efficacy comparison (the ability to properly identify DNS exfiltration domains and avoid misclassification of benign DNS domains), as well as a performance evaluation, comparing the classification time and memory use of the compared methods.

### B. Datasets

The DNS traffic datasets used in this study were collected from DNS servers operated by a large CDN (content delivery network) provider. These datasets were used with the permission of the CDN's enterprise customers, and measures to ensure privacy were taken (e.g., identities of organizations and IP addresses of source machines were hashed and anonymized properly). In addition, a publicly available dataset published by Ziza et al. in 2022 [49] was used as an independent dataset.

**First dataset.** The first dataset, denoted as $DS_f$, consists of 50.85 billion DNS queries from 753 real-world enterprise organizations whose traffic is monitored by the CDN provider. The queries were collected over the course of eight days, beginning on December 28, 2021. Accordingly, the average number of queries per hour is 260 million. The dataset contains one domain suspected of DNS exfiltration (as alerted by the CDN provider's proprietary DNS exfiltration detection algorithm), joinsanjose[.]com. Due to the scarcity of data exfiltration events and the fact that the dataset consists of monitored data, we presume that the rest of the dataset has at most just a negligible amount of malicious traffic, and we thus treat it as benign. Note that while the queries can be associated with a specific source enterprise network, not all of the queries can be associated with the specific machine they were generated by.

**Identifiable dataset.** We sample a subset of $DS_f$, denoted as $DS_p$, which contains 5.06 billion DNS queries. In contrast to the full dataset, all of the DNS queries in $DS_p$ can be attributed to a specific host among nearly 130,000 host IP addresses. The IP addresses are hashed to preserve the privacy of the end users. The $DS_p$ subset is generated to evaluate the methods' ability to detect compromised hosts (see the evaluation presented in Section V-J). Although $DS_p$ accounts for 10% of the total traffic obtained, this dataset is still larger than most datasets used in previous studies.

**Real-world dataset.** The second dataset, denoted as $DS_r$, consists of 255 billion DNS queries collected in a similar fashion to the collection process of $DS_f$ and is provided by the same source. The queries were collected over the course of 21 days, beginning on February 28, 2023. Accordingly, the number of queries per hour is 507 million; to the best of our knowledge, this is the largest dataset ever used to evaluate DNS exfiltration detection methods. This dataset will be used in our real-world evaluation of *ibHH* and the compared methods in Section V-L.

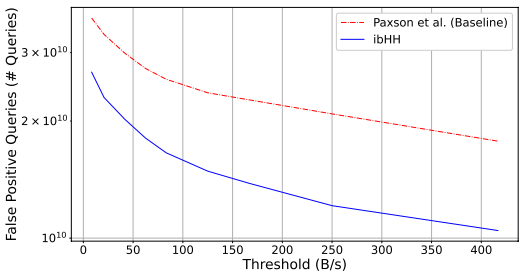
**Public dataset.** We perform our evaluation on a third, publicly available dataset [49], denoted as $ZIZA$. This dataset was constructed by collecting more than 35M DNS queries from an Internet service provider's (ISP) DNS server over the course of 26 hours. Accordingly, the average number of queries per hour is 1.9 million. The dataset also contains exfiltration queries to three distinct domains. The queries are generated by the Iodine [50] and DNSExfiltrator tools [51], both freely available on GitHub. A summary of the datasets is provided in Table II.

### C. Parameter tuning

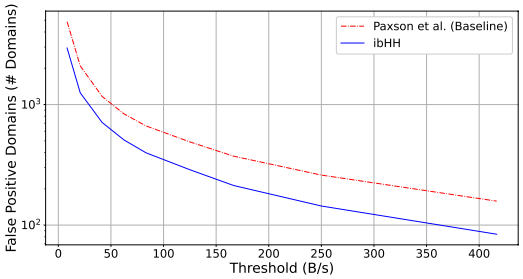
The objective of this tuning phase is to find the lowest detection threshold that produces a practical number of false positive domains. While this definition may vary between different enterprises, in this study we consider 15 false alerts per week (just over two alerts per day) to be practical.

The detection threshold was tuned between 0 bytes/sec (B/s) and 400 B/s. As a baseline, we chose the method of Paxson et al. [13], an information-based threshold detection method, which is the SOTA for such methods. *ibHH*'s cache size was fixed at 1,000, and the reset interval was fixed at 120 seconds. The parameters were set based on the sensitivity analysis performed in Section V-F, which shows that a combination of a window size of 120 seconds and a cache size of 1,000 records obtains the best performance in terms of true positive and false positive alerts.

In addition, we examine the effect of using the allowlisting methods described in Section IV-E. To do so, we generate a PT

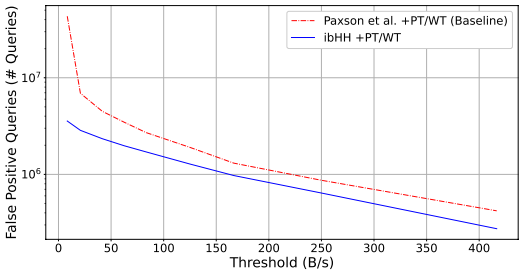



(a) False positive queries

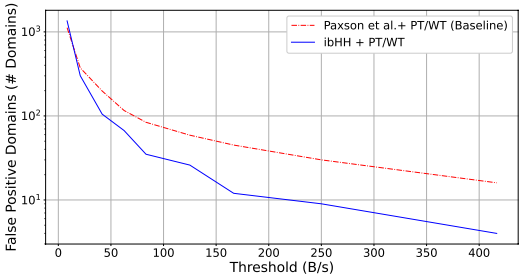



(b) False positive domains

Fig. 3: Parameter tuning without allowlists.




(a) False positive queries



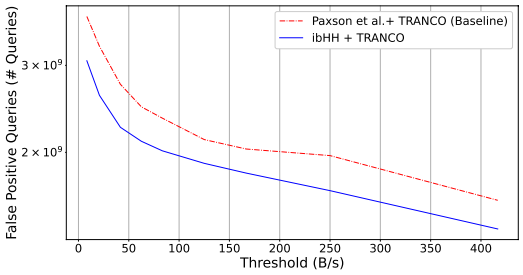

(b) False positive domains

Fig. 4: Parameter tuning with peacetime allowlist.

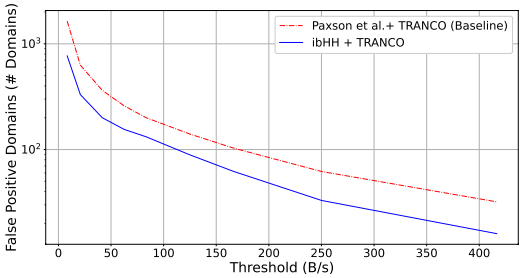
allowlist for both *ibHH* and the method of Paxson et al. method using data from the first day. As can be seen in Figures 3-6, *ibHH* consistently produces less false positive detections than the method of Paxson et al., across all allowlist combinations. *ibHH* with the TRANCO and PT/WT allowlists obtains a total of 10 false alerts over all 753 organizations, but it also results in a high threshold of 250 B/s. In Figure 6, it can be seen that a much lower threshold of 15 B/s results in about 80 alerts over the course of seven days (an average of 0.015 alerts per day per organization), which is an acceptable alert rate for many organizations. We can also see an exponential growth in the number of alerted domains in the lower threshold values.

### *D. Analysis of alerted domains*

Based on the results presented in Section V-C, we manually inspect the domains for which *ibHH* raised an alert (Table IV summarizes our findings). This analysis is performed on the positive alerts produced by the*ibHH* + TRANCO + PT/WT with a detection threshold of 250 B/s configuration. As noted, 10 domains out of 43 million unique domains in the dataset (representing 0.00002% of the unique domains in the dataset) were marked as suspected for DNS exfiltration. Out of the 10 domains, six domains are registered and operated by known security vendors. sophosxl[.]com, appsechcl[.]com, barracudabrts[.]com, dnsbl[.]org, and softsqr[.]com are domains used by DNS anti-malware list (DNSAML) service providers [44]. cnr[.]io is a domain used for honeypot [52] services. Security vendors' AV clients send DNS requests with their current signature ruleset version or suspicious file hashes encoded within the DNS request subdomains, which results in a high number of subdomains [44].



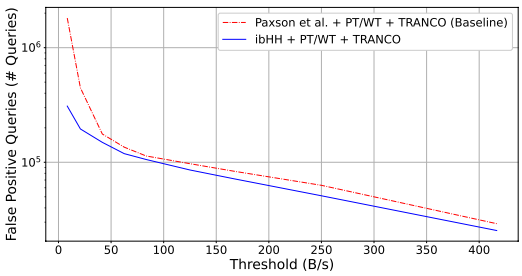

(a) False positive queries



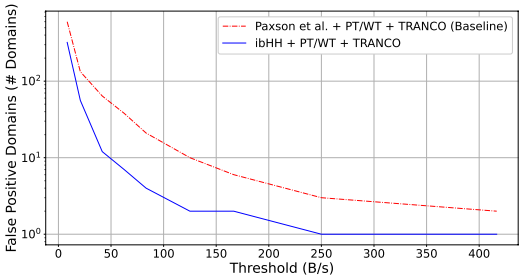

(b) False positive domains

Fig. 5: Parameter tuning with TRANCO allowlist.

Three domains, pldtgroup[.]net, cnsevrx[.]com, and kfcmsp[.]com, are domains associated with a large number of unique (and occasionally, long) subdomains; therefore, the method marked them as suspected DNS exfiltration domains. We looked these domains up with the WHOIS [53] protocol (a query/response protocol that is widely used to obtain information about registered domain names). All three domains were registered at least five years ago (which considerably lowers the likelihood that they have been used for DNS exfiltration), and pldtgroup[.]net is registered to the Philippine Long Distance Telephone Company, a reputable




(a) False positive queries




(b) False positive domains

Fig. 6: Parameter tuning with TRANCO & peacetime allowlist.

company. We thus conclude that these cases are false positive alerts. The last alerted domain, joinsanjose[.]com, was also classified as DNS exfiltration by the CDN provider's algorithm. Being a newly registered domain strengthened suspicion that it was a true positive, however VirusTotal scan did not provide any supportive indications. We were also unable to identify a DNS-tunneling tool that could produce queries like the ones observed in joinsanjose[.]com's subdomains. Therefore, we could not label this case an exfiltration attempt. As such, we classify it as **unknown**.

TABLE III: Sensitivity analysis for *ibHH*.

| Detection window size (sec) | Cache size (# entries) | Num. of alerts (# domains) | Num. of alerts with PT allowlist | Num. of alerts with TRANCO allowlist | Num. of alerts with both allowlists | Num. of true positives |
|---|---|---|---|---|---|---|
| 120 | 100 | 34 | 8 | 15 | 3 | 3 |
| 120 | 1000 | 41 | 11 | 19 | 4 | 3 |
| 120 | 10000 | 41 | 11 | 19 | 4 | 3 |
| 600 | 100 | 23 | 4 | 6 | 3 | 3 |
| 600 | 1000 | 26 | 4 | 6 | 3 | 3 |
| 600 | 10000 | 26 | 4 | 6 | 3 | 3 |
| 1800 | 100 | 13 | 3 | 3 | 2 | 2 |
| 1800 | 1000 | 14 | 4 | 4 | 3 | 3 |
| 1800 | 10000 | 14 | 4 | 4 | 3 | 3 |

### E. Mitigating the need for data consolidation

To provide real-time DNS exfiltration detection, the solution needs to avoid collecting and consolidating data into a single point. In order to demonstrate that *ibHH* satisfies this requirement, we compare two deployment settings:

1) Global - All the enterprise's data is processed by a single instance of *ibHH* (simulates consolidation of data to a single point).
2) Local - An instance of *ibHH* is allocated per enterprise (simulates deployment of *ibHH* right on the DNS gateway of the enterprise, i.e., data is not consolidated).

Similar to Section V-C, we tune the detection threshold and compare the number of alerted domains produced for each deployment setting. TRANCO and PT/WT allowlists are applied in both settings. As can be seen in Figure 8, the results are almost identical; thus, we conclude that both deployment settings are equally viable. Therefore, for *ibHH*, the data does not need to be consolidated, and it can be deployed right on the enterprise DNS gateway.

TABLE IV: Summary of domains alerted by *ibHH*.

| Domain | Category | Frequency | Subdomain Example |
|---|---|---|---|
| cnr[.]io | Honeypot Service | 1 out of 10 | 7zn28g<br>.2.592<br>.ZPQXSURLT7IU5YQFCOS2S76N<br>GVNZHVA2MSZE6JIBXZ2B3EES2BORGPHCI2G6FDC<br>.KQRC5YIEIXH5UBLYRSTQ3C4B<br>ZNNTAD7OVOQUIOD3KEP5JQLEOVGTS2F3HFS7ZGO<br>.YMJULMS26QIC7RLPXGYOZHQA888888.ef584e16 |
| dnsbl[.]org | DNSAML [44] Services | 5 out of 10 | 2FWww.wayfAIr.cOm.IN |
| barracudabrts[.]com | | | 01143c071e01.t-164113145.id135d030.prizelabs.com.d.bl |
| sophosxl[.]com | | | adfba15d2df0b4465475c420f2a2137d.sigv2.vir1 |
| appsechl[.]com | | | v3-query-13018-40e9142bf<br>-95c2-41e5-87e7-d6a7c536a2e4.securityip |
| softsqr[.]com | | | id-c55c08e136af19b29d1a33684e2f7100.<br>4efeac399c6095bf867256d4529f4f1ef257<br>a677c3e2e207472.fde9-0050-06cd<br>-aa55d1-1BS80I039724-61cac413.f |
| joinsanjose[.]com | Unknown | 4 out of 10 | 519b67d9b7487348793fefdc4e9728a12e48b30af66a6ac685e<br>.606.xkgoz0.hex.a10c0458d0 |
| pldtgroup[.]net | | | cqgvrnyjrqlaxcppf.pldt |
| cnsevrx[.]com | | | tb-024753-wan2 |
| kfcmsp[.]com | | | D212146-wan1 |

### F. Sensitivity analysis

We perform a sensitivity analysis of *ibHH*'s detection window size and cache size parameters, while setting the detection threshold at 10 B/s. The values examined for the window size are 120, 600, and 1,800 seconds, and the examined cache size values are 100, 1,000, and 10,000. This analysis complements the detection threshold tuning step described in Section V-C. It can be seen that increasing the detection window causes a reduction in the number of alerts. This is expected, as a longer detection window results in a higher detection threshold for the window. This can have both a positive effect on the detection of false positives (reducing the number of false positive alerts) and a negative effect on the detection of true positives (when the attacker exfiltrates data in short bursts), as can be seen in the case in which the detection window is 1,800 and the cache size is 100. Increasing the cache size also affects the number of alerts. This can be attributed to the fact that in the case of a smaller cache size, a cached domain is more likely to be evicted, and thus it might not remain in the *ibHH* cache long enough to be considered an information heavy hitter. On the other hand, an increased cache size also means that more memory is required to store the *ibHH* cache. Based on this analysis, we recommend setting the detection window in the range of 120 to 600 seconds, and the cache size should be set between 1,000 and 10,000 (see Table III for a summary of the results of our analysis).

### G. Frequency of peacetime list generation

To determine the frequency at which the peacetime list generation procedure should run and the amount of time it should run for, we conducted an experiment using $DS_f$. We examined the rate of new observed domains per enterprise organization as a function of the number of days passed. We found that for 90% of the organizations, over 90% of the domains observed over the period of eight days, were observed in the first day. This means that a peacetime list produced for

one day covers more than 90% of the DNS domain names for the week that follows, making it a good baseline for the PT generation list (i.e., generating one peacetime list every eight days). Figure 7 illustrates our findings.

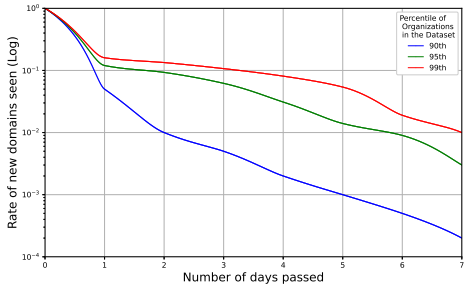



Fig. 7: Rate of new observed domains for 90, 95 and 99% of the organizations in the dataset.

### H. Compared methods

*1) Information-based Heavy Hitters for DNS Exfiltration Detection:* This is the method proposed in this paper. The reset interval was fixed at 120 seconds, and the cache size was set to 1,000 entries, similar to the configuration used in Section V-C.

*2) Practical Comprehensive Bounds on Surreptitious Communication over DNS:* The method of Paxson et al., which was used in the parameter tuning section V-C, is also used to evaluate our proposed method; it will be denoted as *Paxson* for the rest of the paper. *Paxson* is selected as it is the SOTA information estimation-based DNS exfiltration detection technique, and the most similar to our proposed method.

*3) Detection of Malicious and Low Throughput Data Exfiltration Over the DNS Protocol:* Nadler et al. [18] presented an unsupervised anomaly detection model based on the isolation forest algorithm [29] In this method, DNS queries are collected from recursive DNS servers at a frequency of $\lambda$ time units, and feature extraction is performed on a window size of $\lambda * n_s$, and fed to the pre-trained isolation forest model. Despite not being a real-time solution, we chose to compare our method's performance to it, since it has the ability to detect DNS exfiltration campaigns with exfiltration rates as slow as 0.11 B/s, up to six hours after the traffic is collected. To the best of our knowledge, this is the SOTA in terms of near-real-time detection capabilities. In the remainder of the paper, we refer to this method as *IF*. We configured *IF* according to the authors' recommendation, setting $\lambda = 60$ and $n_s = 6$.

*4) Real-Time Detection of DNS Exfiltration and Tunneling from Enterprise Networks:* Ahmed et al. [19] presented an unsupervised anomaly detection model for real-time DNS exfiltration from enterprise networks based on the isolation forest algorithm. As noted in Section III, it is a true real-time method, and to the best of our knowledge, it is the SOTA real-time detection solution. In the remainder of the paper, this method will be referred to as *RT-IF*. We configured *IF* according to the authors' recommendation, fixing the number of trees at two and limiting the tree height to 18. All of the methods described were implemented in Python, and the experiments on the $DS_p$ dataset were performed on Azure Databricks Runtime version 10.3 [54]. *IF* and *RT-IF* were both implemented using SynapseML [55].

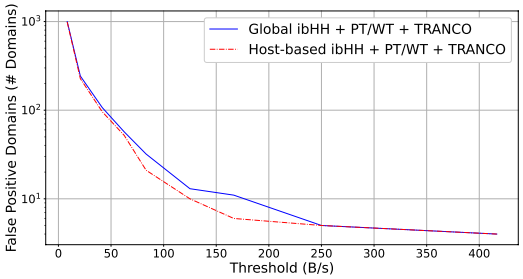



Fig. 8: Global vs local false positive domains for various thresholds (# domains).

### I. Methodology

$DS_p$ is divided into training, peacetime, and test sets. The training set consists of $790M$ queries from $112K$ unique hosts from the first day in the dataset. The peacetime set consists of 720M queries from the next day in the dataset. The rest of the dataset (six days of data) composes the test set, with a total of $3.8B$ queries from $115K$ unique hosts. Malicious DNS exfiltration traffic is synthetically generated, similarly to previous research [18], [19]. The attacks are generated based on three well-known DNS exfiltration tools and attacks:
1) *Iodine* [50] - Iodine is an open-source DNS tunneling tool, mainly used to bypass Wi-Fi paywalls, like the ones that can be found in hotels. This simulates high throughput DNS exfiltration campaigns.
2) *FrameworkPOS* [7] - the *FrameworkPOS* malware was used in a targeted attack on the American retailer, Home Depot. The malware extracted credit card information from compromised machines' memory, encoded the data, and sent it to a remote server in the following format: $< encoded\_credit\_card > .domain.com$. We generate *FrameworkPOS* queries at a frequency of three queries per second to simulate the original malware's throughput of 56 million credit cards in six months.
3) *Backdoor.Win32.Denis* - The Trojan malware Backdoor.Win32.Denis was used in Operation Cobalt Kitty, a large-scale Asian APT [6], [56]. *Denis* enables an intruder to manipulate the file system and run arbitrary commands and loadable modules. Denis uses the DNS as a bidirectional C&C communication channel with its operator. In this paper, we simulate the malware's keep-alive instructions. We generate the requests every 1.5 seconds, conforming to Cobalt Kitty's operation security report analysis.

In each experiment, 1% of the client hosts (i.e., 1,300 hosts) are sampled. Queries are generated using one of the DNS exfiltration tools, with the sampled client hosts as the source of the DNS queries. We evaluate the detection abilities of each of the methods based on the following metrics: number of overall hosts alerted (i.e., number of hosts suspected as being infected), hosts' $TPR$ (true positive rate, hosts which are truly infected and for which alerts are raised for), hosts' $FPR$ (hosts which are not infected but for which alerts are raised). Each host is infected with a random number of malicious queries in the range of 100 to 10,000, where the queries are

injected at random start times in the test set. To identify the infected hosts, each host is associated with a distinct malicious primary domain. We want to compare the methods' abilities to detect compromised client hosts. Therefore, the compared methods are trained with different acceptable false positive rate ($FPR$) values: 0.01 (1,300 clients), 0.001 (130 clients), 0.0001 (13 clients), 0.00001 (1-2 clients). In each experiment, *IF* and *RT-IF* are trained by setting the isolation forest's contamination rate to be the experiment's acceptable $FPR$ value. For *ibHH* and Paxson, the algorithms are executed on the training dataset, and the detection threshold is tuned to be the minimum value for which the acceptable $FPR$ is achieved. For each of the compared methods, we generate a peacetime allowlist by feeding the peacetime dataset to the trained model. The peacetime allowlist is composed of alerted domains in the peacetime dataset. The TRANCO allowlist is applied to all the compared methods.

### J. Results

Table V presents the compared methods' detection abilities with different acceptable FPR values. For an acceptable FPR of 0.01 (1%), *ibHH*'s detection threshold is 0.7B/s, meaning it can detect exfiltration rates as slow as 0.7 B/s while producing 1% FP alerts on the training set. Based on our evaluation, it can be inferred that the detection time for DNS exfiltration is quick. In fact, considering the Track 2 format commonly used for credit card data, which requires at least 40 bytes of information per credit card [57], our method is capable of detecting and raising an alert within a timeframe that would typically allow for the exfiltration of at most three credit card details. This highlights *ibHH*'s effectiveness and efficiency in limiting the potential impact of data exfiltration incidents.

It should be noted that the FPR on the test set is just under 0.004, which is less than the acceptable FPR of the training set. This difference can be attributed to the allowlists, and it is observed in the rest of the methods. It should be noted that for the very low acceptable FPR value of 0.00001, $RT-IF$ is unable to detect any exfiltration events except for Iodine, while *ibHH* is able to detect the slower exfiltration rate domains used for FrameworkPOS and Denis communication. Unsurprisingly, the lower the acceptable FPR is set, the higher the *detectable exfiltration rate* ($DER$) gets. *ibHH* and Paxson's have a similar DER, which is to be expected, as they are both based on a similar idea of quantifying the amount of information and comparing it against a predefined threshold.

Measuring the $DER$ value of $IF$ and $RT-IF$ is tricky, since they are traffic-based and payload-based machine learning models, respectively. $IF$ collects features in a sliding window of size $\lambda$ time units, and classification is performed based on the last $n_s$ time windows. This means that exfiltration events are detected between $\lambda$ and $n_s \cdot \lambda$ time units after they occur (disregarding the time it takes to consolidate the logs into a single point, as well as the feature extraction and classification time). In the authors' recommended setup, $\lambda$ is set to 60 minutes, and $n_s$ is set to six, meaning up to six hours can pass until the detection time. Still, the theoretical detectable exfiltration rate is as slow as 0.11 B/s, making $IF$ a practical complementary solution for offline analysis.

Because $RT-IF$ operates on single DNS queries, it can detect DNS exfiltration events on the first packet inspected (so theoretically, it can stop DNS exfiltration by the time it starts); in practice we can see that only the model trained for an acceptable FPR of 0.01 achieves a competitive detection rate, and the method becomes less practical for acceptable FPR values of 0.001 and lower. While this method may be useful on small networks where 1% FPR may be acceptable, it is not practical for large-scale networks like the one the dataset used in this research comes from, where an FPR of 1% results in over 40,000 false alerts per week.

### K. Evaluation on the $ZIZA$ dataset

We perform an additional evaluation on a public dataset, $ZIZA$ (described in Section V-B). Because $ZIZA$ was collected over the course of 26 hours, we use the first 10 hours as a training set; the following two hours are used to generate the peacetime allowlist, and the final 14 hours are used as the test set. We train the different detection methods similarly to the way described in Section V-I and evaluate their ability to detect DNS exfiltration domains with different acceptable FPR values. The methods were compared on a single machine in order to evaluate the resource use of each method.

The results on the public dataset are similar to those obtained on $DS_p$. For an acceptable FPR of 1%, the detection threshold of ibHH is 0.6 B/s, and it is able to detect all three DNS exfiltration domains with 62 false positive domains. While this is quite a large number, it is significantly better than that obtained by both $IF$ (140 false positive domains), $RT-IF$ (119), and $Paxson$ (87). For the acceptable FPR of 0.01%, $ibHH$ has only a single false positive while still detecting all the exfiltration domains. $IF$ is able to detect only one malicious domain, and Paxson detects two. $RT-IF$ is unable to detect any malicious activity in this scenario. See Table V for a summary of the results.

### L. Real-world evaluation

We perform a real-world evaluation of the compared methods on the $DS_r$ dataset. We partition the dataset into training, peacetime allowlist generation, and test sets, similar to the partitioning described in Section V-I. The first seven days serve as training data, followed by one day of peacetime allowlist generation, and the rest of the data is dedicated to test the trained models. Because this evaluation is performed on non-labeled real-world DNS queries, we cannot provide TPR and FPR estimations. Instead, we perform manual inspection of alerted domains and queries and determine if the alerts are true positives (TPs) or false positives (FPs) and provide these counts. All methods have been trained under an acceptable FPR of 0.001 (0.1%), as it showed promising results in the synthetic dataset evaluation results for all the compared methods in Section V-J. The detectable exfiltration rate ($DER$) *ibHH* obtained in the training phase is 6 B/s (18 credit card numbers), while *Paxson*'s $DER$ is 11B/s (33 credit card numbers).

*ibHH* generated a total of 17 alerts, out of which two are true positive detections (so, 15 are false positive alerts; this is the lowers number of FP alerts for all the compared methods). *IF* and *Paxson* both successfully detected the two domains, yet with more FP alerts, while *RT-IF* successfully detected only one of the domains. The results show that *ibHH* and *IF* have a similar number of TP queries, where *IF* classifies about 400 more TP queries, however with the cost of over 55,000

TABLE V: Comparison of the evaluated methods based on the TPR and FPR.

| Method | Dataset | FPR=0.01 | | | | FPR=0.001 | | | | FPR=0.0001 | | | | FPR=0.00001 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $TD^1$ | FPR | TPR | $DER^1$ | $TD^1$ | FPR | TPR | $DER^1$ | $TD^1$ | FPR | TPR | $DER^1$ | $TD^1$ | FPR | TPR | $DER^1$ |
| ibHH | $DS_p + I$ | 1734 | **0.0037** | **1.0** | 0.7 | 1420 | **0.001** | **1.0** | 5 | 1343 | <0.001 | **1.0** | 65 | 1300 | 0 | **1.0** | 275 |
| | $DS_p + F$ | 1743 | **0.0038** | **1.0** | 0.7 | 1430 | **0.001** | **1.0** | 5 | 1298 | <0.001 | **0.98** | 65 | 1280 | 0 | **0.97** | 275 |
| | $DS_p + D$ | 1728 | **0.0037** | **1.0** | 0.7 | 1417 | **0.001** | **1.0** | 5 | 1252 | <0.001 | **0.98** | 65 | 1214 | 0 | 0.92 | 275 |
| | $ZIZA$ | 65 | **0.005 (62)** | **1.0 (3)** | 0.6 | 12 | **0.0007 (9)** | 1.0 (3) | 4 | 4 | 0.000085 (1) | **1.0 (3)** | 15 | N/A | N/A | N/A | N/A |
| IF | $DS_p + I$ | 3015 | 0.007 | **1.0** | | 2132 | 0.0012 | **1.0** | | 1342 | <0.001 | **1.0** | | 1300 | 0 | **1.0** | |
| | $DS_p + F$ | 3015 | 0.007 | 0.99 | N/A | 2085 | 0.0012 | 0.96 | N/A | 1267 | <0.001 | **0.98** | N/A | 1279 | 0 | **0.97** | N/A |
| | $DS_p + D$ | 3015 | 0.007 | 0.98 | | 2058 | 0.0012 | 0.94 | | 1240 | <0.001 | 0.97 | | 1183 | 0 | 0.91 | |
| | $ZIZA$ | 143 | 0.012 (140) | **1.0 (3)** | | 24 | 0.0017 (22) | 0.67 (2) | | 1 | **0.0 (0)** | 0.33 (1) | | N/A | N/A | N/A | |
| RT-IF | $DS_p + I$ | 3200 | 0.008 | **1.0** | | 2659 | 0.014 | **1.0** | | 1314 | <0.001 | **1.0** | | 1250 | 0 | 0.96 | |
| | $DS_p + F$ | 3214 | 0.008 | **1.0** | N/A | 2631 | 0.014 | 0.98 | N/A | 1107 | <0.001 | 0.85 | **N/A** | 0 | 0 | 0 | N/A |
| | $DS_p + D$ | 3170 | 0.008 | 0.98 | | 2599 | 0.014 | 0.95 | | 1039 | <0.001 | 0.8 | | 0 | 0 | 0 | |
| | $ZIZA$ | 122 | 0.01 (119) | **1.0 (3)** | | 21 | 0.015 (19) | 0.67 (2) | | 0 | **0.0 (0)** | 0.0 (0) | | N/A | N/A | N/A | |
| Paxson | $DS_p + I$ | 1927 | 0.0041 | **1.0** | 0.9 | 1771 | 0.0023 | **1.0** | 12 | 1314 | <0.001 | **1.0** | 70 | 1300 | 0 | **1.0** | 300 |
| | $DS_p + F$ | 1927 | 0.0041 | **1.0** | 0.9 | 1771 | 0.0023 | **1.0** | 12 | 1249 | <0.001 | 0.96 | 70 | 1270 | 0 | 0.96 | 300 |
| | $DS_p + D$ | 1927 | 0.0041 | 0.98 | 0.9 | 1771 | 0.0023 | **1.0** | 12 | 1230 | <0.001 | 0.95 | 70 | 932 | 0 | 0.72 | 300 |
| | $ZIZA$ | 87 | 0.0071 (84) | **1.0 (3)** | 1 | 14 | 0.0009 (11) | **1.0 (3)** | 6 | 3 | 0.000085 (1) | 0.67 (2) | 32 | N/A | N/A | N/A | N/A |

[1] Total Detections (#Distinct Hosts)
[2] Detectable Exfiltration Rate (B/s)

more FP queries than *ibHH*. An analysis of the TP domains is provided in Section V-L1, and a summary table of the results is provided in Table VI.

*1) True Positive Domain Analysis:* The first TP domain we inspect is *cymulatedlp[.]com* which was detected by all models except *RT-IF*. This domain name is registered by a cybersecurity vendor of a similar name and is used by enterprises to simulate exfiltration campaigns to assess data exfiltration defense mechanisms employed by the enterprises. While this is a simulated attack, we treat it as a true positive detection given the fact that the data is unlabeled. This attack consists of quite short subdomains (of length 64). The average time between two consecutive queries is approximately two seconds. This might explain why *RT-IF* was not able to detect it, as it simulates a rather stealthy DNS exfiltration campaign. The second domain, detected by all methods, is *q2t[.]nl*. The data seems to be base64 encoded, with subdomain's lengths ranging between 30 and 144, and queries are sent quickly one after the other (an average of 0.01 second between consecutive queries). This domain represents a high throughput exfiltration campaign and is detected by all methods. We discovered with WHOIS that the domain is owned by your-freedom[.]net [58], a VPN provider that supports tunneling over DNS, which further supports the classification of the domain as TP.

TABLE VI: Real-world evaluation results.

| Method | FP Domains | TP Domains | FP Queries | TP Queries | $DER$ |
|---|---|---|---|---|---|
| ibHH | **15** | **2** | 2,043 | 17,441 | 6 |
| IF | 31 | **2** | 57,125 | 17,820 | N/A |
| RT-IF | 20 | 1 | 5,093 | 12,391 | N/A |
| Paxson | 17 | **2** | 2,677 | 15,570 | 11 |

### M. Resource Use

We evaluate the average runtime and average memory consumption of each method on a single machine with a 6 core CPU and 16GB RAM, representing a high-performance DNS server hardware specification. *IF* and *RT-IF* were both implemented with the scikit-learn library [59]. We measured the total runtime for each method to train, generate the peacetime allowlist, and classify the test dataset. The $ZIZA$ dataset was used in this evaluation. We assess the runtime by measuring the processing time for each method. The experiments are performed five times, and the average runtime is calculated based on the outcomes of these five runs.

*ibHH* and *RT-IF*, as methods with real-time capabilities, both use about 1.5MB memory, but *ibHH* is significantly faster, with an average runtime of 58 seconds compared to the 857 second runtime of *RT-IF*. This can be explained by the fact that $RT - IF$ needs to generate a large number of features to classify a DNS query.

The offline methods ($Paxson$ and $IF$) have a notable disadvantage in that they need to store all the queries in a specified inspection window. This requirement leads to a considerably larger memory footprint and longer runtime, rendering them unsuitable for real-time deployment on the resolver. Implementing these methods on the resolver would negatively affect the rate at which the DNS resolver performs DNS resolution. A summary of the resource analysis is provided in Table VII.

## VI. Discussion

### A. Limitations

*1) Allowlisting:* To distinguish between benign and malicious data exchange over DNS, we described two allowlisting methods in Section IV-E. As noted in Figure 3, our allowlisting approaches significantly reduce the number of FP alerts. Difficulty in distinguishing between malicious and benign DNS exfiltration traffic is common among DNS exfiltration detection algorithms, and allowisting methods are often employed to cope with this issue [19], [18], [13], [21]. Maintenance of these lists can be automated thus easing the process of incorporating them in the DNS exfiltration detection pipeline.

*2) Resilience against an aware attacker:* An aware attacker can circumvent detection by configuring malware to exfiltrate data at rates below the detection threshold. While this is a valid concern, we show that exfiltration campaigns as slow as 0.7 B/s can be detected by *ibHH* with less than 0.04% of benign domains misclassified. An IT organization that wants to detect very slow campaigns, may choose to lower the detection threshold; this will come with the cost of possibly having to deal with more false alarms. Another challenge is the attacker's ability to use encrypted DNS requests, such as DNS over TLS [60] and DNS over HTTPS [61], to evade detection. Enterprises can deal with this issue by blocking encrypted DNS traffic that is not resolved by an enterprise DNS resolver, as recommended by the National Security Agency [62], and only allow encrypted DNS if it is resolved by the enterprise's internal recursive DNS resolver (which allows inspection of the raw DNS packet).

TABLE VII: Comparison of the evaluated methods based on runtime and memory consumption, on the ZIZA dataset.

| Method | Avg. Runtime (Second) | Avg. Memory Usage (MB) | Avg. Queries per Second |
|---|---|---|---|
| ibHH | **58** | **1.6** | **603,448** |
| IF | 2,738 | 102 | 12,783 |
| RT-IF | 857 | **1.5** | 40,840 |
| Paxson | 3,642 | 237 | 9,610 |

While *ibHH* primarily focuses on detecting DNS exfiltration through the query name, it is important to note that attackers can use other information vectors like query type and timing, thus avoid detection by *ibHH*. However, these vectors have limitations, such as restricted information capacity and vulnerability to inaccuracies. Despite these alternatives, the query name remains the most commonly exploited vector. By effectively detecting information conveyed through the query name, *ibHH* serves as a valuable defense against DNS exfiltration.

To the best of our knowledge, the query name has been the primary (or only) information vector utilized by all publicly available DNS exfiltration tools and known DNS exfiltration campaigns. The method of Paxson et al. is designed to detect DNS exfiltration events regardless of the information vector, which is a strength of that approach.

Another way the attacker can try to bypass *ibHH* is to break the exfiltrated data down into single character queries (e.g., instead of sending "exfiltration.domain.com," the attacker would send "e.domain.com," "x.domain.com," ... , "n.domain.com"). Because *ibHH* only accounts for distinct subdomains, it might miss this exfiltration scenario. However, it should be noted that this approach also results in a significantly lower rate of data exfiltration. In addition, DNS resolvers have a cache structure [2] that stores resolved DNS queries for a limited time. That can be problematic for the attacker, because subsequent requests for the same query will be served from the cache instead of being sent to the authoritative DNS server. Finally, the number of requests required to exfiltrate a given message increases linearly based on the message size. This increases the risk of DNS queries failing to reach the attacker because of DNS throttling, which is widely employed on public DNS servers [63], [64] (and can easily be implemented on internal enterprise DNS resolvers).

The attacker may also intentionally send queries with short subdomain (which have low information weights) before sending the actual DNS exfiltration traffic (which has a high information weight) in an attempt to fill the cache before the actual attack starts. For the attack to succeed, the short subdomain queries' hash values (computed by the uniform hash function) need to be very low in every reset period; in addition, the long queries' hash values need to be over the $\tau$ threshold. Even if the attacker knows the hash function used, there is no certainty that low hash values will be consistently obtained, since this is a matter of chance. Therefore, while it is a clever attack, we deem its impact and the risk it poses to be minor.

*3) Intentional false positives:* An aware attacker could trigger an intentional false positive by sending queries with long random subdomains targeted at a legitimate registered domain. This might cause *ibHH* to trigger an alert that the benign domain is being used for DNS exfiltration, which could prevent the host machines from accessing legitimate services. However, this pertains to domains that are not on the allowlists (peacetime or global allowlists), limiting its impact. This limitation is not unique to our approach and is relevant to all DNS exfiltration detection techniques, as they struggle to distinguish benign and malicious tunnels.

### B. Wildcard subdomain resolution

There are multiple services that use subdomains to host multiple services or deliver user-generated content (UGC). Notable examples include dropbox.com and googledocs.com, which organize their content under different subdomains and URLs for better isolation and network load distribution. These UGC services are sometimes incorrectly classified as DNS exfiltration despite being legitimate due to their extensive use of unique subdomains, as reported by [44]. This is a limitation of all existing methods given their inherent inability to distinguish between legitimate and malicious cases of DNS exfiltration, and it also applies to our proposed method, which attempts to overcome this limitation by using allowlists. This situation is suboptimal but arguably acceptable, since the rate of false alerts reported in our real-world, representative dataset of 750 organizations indicates there are, on average, less than 0.1 cases like this weekly per organization.

### C. Deployment considerations

Given the low time and memory complexity of *ibHH* (theoretically proven in Section IV-C and shown in practice in Section V-M), an organization may benefit from the deployment of multiple *ibHH* instances with different threshold values in order to cover different potential data exfiltration attacks over DNS and improve performance. This approach is also aligned with our evaluation results (see Section V-J), where we present different models with different detection thresholds.

### D. Amount of malicious activity found in real-world traffic

Despite the fact that $DS_f$ contains more than 50B queries, and $DS_r$ comprises over 250B queries, the amount of malicious activity found by *ibHH* and the other methods was relatively small. This is due to the fact that the datasets were collected from security-aware enterprises with various defense mechanisms deployed on their networks. Naturally, those enterprises are less likely to be victims of cyber-threats like data exfiltration. Similar exfiltration frequencies were seen by Nadler et al. [18], where datasets of security-aware enterprises were used as well.

## VII. Conclusions and Future Work

We present *ibHH*, a simple yet effective method capable of both detecting DNS exfiltration events in real time, by estimating the amount of unique information conveyed to registered domains through query subdomains. We perform a comprehensive evaluation, comparing the proposed method's performance to that of prominent SOTA methods, including the real-time machine learning based solution that *ibHH* was

shown to outperform. In the future, we plan to adapt *ibHH* for the detection of cross-domain exfiltration events, for example, by changing the information quantification so that it is per source user IP instead of per target registered domain. We also plan to explore a possible variation of *ibHH* capable of detecting other information vectors used for DNS exfiltration (such as the data exfiltration based on the query type field), as well as consider the DNS response (which can help in the detection of bidirectional communication). We also plan to deploy *ibHH* on DNS resolvers and evaluate its performance on online DNS query streams.

## REFERENCES


[1] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, and C. B. Thomas, "Mitre att&ck: Design and philosophy," in *Technical report*. The MITRE Corporation, 2018.

[2] P. Mockapetris, "Domain names - concepts and facilities," RFC 1034, Nov. 1987. [Online]. Available: https://www.rfc-editor.org/info/rfc1034

[3] M. Lyu, H. H. Gharakheili, and V. Sivaraman, "A survey on dns encryption: Current development, malware misuse, and inference techniques," *ACM Computing Surveys (CSUR)*, 2022.

[4] S. Bromberger, "Dns as a covert channel within protected networks," *National Electronic Sector Cyber Security Organization (NESCO)(Jan., 2011)*, 2011.

[5] C. G. Girling, "Covert channels in lan's," *IEEE Transactions on software engineering*, vol. 13, no. 2, p. 292, 1987.

[6] A. Dahan, "Operation cobalt kitty: A large-scale apt in asia carried out by the oceanlotus group," https://www.cybereason.com/blog/operation-cobalt-kitty-apt, 2017, [Online].

[7] P. Rascagneres, "New frameworkpos variant exfiltrates data via dns requests," https://www.gdatasoftware.com/blog/2014/10/23942-new-frameworkpos-variant-exfiltrates-data-via-dns-requests, 2014, [Online].

[8] R. Falcone, "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/, 2019, [Online].

[9] F. Gutierrez, "Please confirm you received our apt," https://www.fortinet.com/blog/threat-research/please-confirm-you-received-our-apt, 2022, [Online].

[10] A. Turing, H. Wang, "New threat: B1txor20, a linux backdoor using dns tunnel," https://web.archive.org/web/20220407213839/https://blog.netlab.360.com/b1txor20-use-of-dns-tunneling_en/, 2022, [Online].

[11] D. Fisher, "Ransomware actors leaning on dns tunneling," https://duo.com/decipher/ransomware-actors-leaning-on-dns-tunneling, 2023, [Online].

[12] Y. Wang, A. Zhou, S. Liao, R. Zheng, R. Hu, and L. Zhang, "A comprehensive survey on dns tunnel detection," *Computer Networks*, vol. 197, p. 108322, 2021.

[13] V. Paxson, M. Christodorescu, M. Javed, J. Rao, R. Sailer, D. L. Schales, M. Stoecklin, K. Thomas, W. Venema, and N. Weaver, "Practical comprehensive bounds on surreptitious communication over {DNS}," in *22nd USENIX Security Symposium (USENIX Security 13)*, 2013, pp. 17–32.

[14] N. Ishikura, D. Kondo, V. Vassiliades, I. Iordanov, and H. Tode, "Dns tunneling detection by cache-property-aware features," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1203–1217, 2021.

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

[16] I. Homem, P. Papapetrou, and S. Dosis, "Information-entropy-based dns tunnel prediction," in *Advances in Digital Forensics XIV: 14th IFIP WG 11.9 International Conference, New Delhi, India, January 3-5, 2018, Revised Selected Papers 14*. Springer, 2018, pp. 127–140.

[17] A. Almusawi and H. Amintoosi, "Dns tunneling detection method based on multilabel support vector machine," *Security and Communication Networks*, vol. 2018, 2018.

[18] A. Nadler, A. Aminov, and A. Shabtai, "Detection of malicious and low throughput data exfiltration over the dns protocol," *Computers & Security*, vol. 80, pp. 36–53, 2019.

[19] J. Ahmed, H. H. Gharakheili, Q. Raza, C. Russell, and V. Sivaraman, "Monitoring enterprise dns queries for detecting data exfiltration from internal hosts," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 265–279, 2019.

[20] F. Palau, C. Catania, J. Guerra, S. Garcia, and M. Rigaki, "Dns tunneling: A deep learning based lexicographical detection approach," *arXiv preprint arXiv:2006.06122*, 2020.

[21] S. Chen, B. Lang, H. Liu, D. Li, and C. Gao, "Dns covert channel detection method using the lstm model," *Computers & Security*, vol. 104, p. 102095, 2021.

[22] K. Wu, Y. Zhang, and T. Yin, "Tdae: Autoencoder-based automatic feature learning method for the detection of dns tunnel," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 2020, pp. 1–7.

[23] T. Locher, "Finding heavy distinct hitters in data streams," in *Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures*, 2011, pp. 299–308.

[24] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm," in *Discrete Mathematics and Theoretical Computer Science*. Discrete Mathematics and Theoretical Computer Science, 2007, pp. 137–156.

[25] P. Indyk, "Sketching, streaming and sublinear-space algorithms," *Graduate course notes, available at*, vol. 33, p. 617, 2007.

[26] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2002, pp. 1–16.

[27] Internet Systems Consortium, "Resource requirements," https://bind9.readthedocs.io/en/v9.18.20/chapter2.html, 203, [Online].

[28] CloudBlue Commerce, "Hardware requirements for bind dns servers," https://docs.cloudblue.com/cbc/21.0/DNS-Hosting-Services/Hardware-Requirements-for-BIND-DNS-Servers.htm, 2022, [Online].

[29] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 eighth ieee international conference on data mining*. IEEE, 2008, pp. 413–422.

[30] J. Zhang, L. Yang, S. Yu, and J. Ma, "A dns tunneling detection method based on deep learning models to prevent data exfiltration," in *International Conference on Network and System Security*. Springer, 2019, pp. 520–535.

[31] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[32] A. R. Mohammed, S. A. Mohammed, and S. Shirmohammadi, "Machine learning and deep learning based traffic classification and prediction in software defined networking," in *2019 IEEE International Symposium on Measurements & Networking (M&N)*. IEEE, 2019, pp. 1–6.

[33] 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 *Proceedings of the 11th Annual Cyber and Information Security Research Conference*, 2016, pp. 1–4.

[34] P. Yang, X. Wan, G. Shi, H. Qu, J. Li, and L. Yang, "Naruto: Dns covert channels detection based on stacking model," in *Proceedings of the 2nd World Symposium on Software Engineering*, 2020, pp. 109–115.

[35] G. Ruiling, D. Jiawen, C. Xiang, and S. Shouyou, "A dns-based data exfiltration traffic detection method for unknown samples," in *2022 7th IEEE International Conference on Data Science in Cyberspace (DSC)*. IEEE, 2022, pp. 191–198.

[36] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[37] K. Schomp, O. Bhardwaj, E. Kurdoglu, M. Muhaimen, and R. K. Sitaraman, "Akamai dns: Providing authoritative answers to the world's queries," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 465–478.

[38] A. M. Kara, H. Binsalleeh, M. Mannan, A. Youssef, and M. Debbabi, "Detection of malicious payload distribution channels in dns," in *2014 IEEE International Conference on Communications (ICC)*. IEEE, 2014, pp. 853–858.

[39] Y. Afek, A. Bremler-Barr, E. Cohen, S. L. Feibish, and M. Shagam, "Efficient distinct heavy hitters for dns ddos attack detection," *arXiv preprint arXiv:1612.02636*, 2016.

[40] L. Yang, B. Ng, and W. K. Seah, "Heavy hitter detection and identification in software defined networking," in *2016 25th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2016, pp. 1–10.

[41] A. Rajaraman and J. Ullman, "Mining data streams," in *Mining of Massive Datasets, 2nd ed., Cambridge University Press*, 2014, pp. 165–173.

[42] S. Heule, M. Nunkesser, and A. Hall, "Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm," in *Proceedings of the 16th International Conference on Extending Database Technology*, 2013, pp. 683–692.

[43] P. B. Gibbons and Y. Matias, "New sampling-based summary statistics for improving approximate query answers," in *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, 1998, pp. 331–342.

[44] A. Nadler, R. Bitton, O. Brodt, and A. Shabtai, "On the vulnerability of anti-malware solutions to dns attacks," *Computers & Security*, vol. 116, p. 102687, 2022.

[45] Y. Chen, M. Antonakakis, R. Perdisci, Y. Nadji, D. Dagon, and W. Lee, "Dns noise: Measuring the pervasiveness of disposable domains in modern dns traffic," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2014, pp. 598–609.

[46] Y. Zeng, X. Yun, X. Chen, B. Li, H. Tsang, Y. Wang, T. Zang, and Y. Zhang, "Finding disposable domain names: A linguistics-based stacking approach," *Computer Networks*, vol. 184, p. 107642, 2021.

[47] X. Hu, J. Jang, M. P. Stoecklin, T. Wang, D. L. Schales, D. Kirat, and J. R. Rao, "Baywatch: robust beaconing detection to identify infected hosts in large-scale enterprise networks," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2016, pp. 479–490.

[48] V. L. Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen, "Tranco: A research-oriented top sites ranking hardened against manipulation," *arXiv preprint arXiv:1806.01156*, 2018.

[49] Ziza, Kristijan and Vuletić, Pavle and Tadić, Predrag, "Dns exfiltration dataset," 2023. [Online]. Available: https://data.mendeley.com/datasets/c4n7fckkz3/3

[50] E. Ekman, B. Andersson, "Iodine (ip-over-dns, ipv4 over dns tunnel)," https://code.kryo.se/iodine/, 2022, [Online].

[51] Arno0x, "Iodine (ip-over-dns, ipv4 over dns tunnel)," https://github.com/Arno0x/DNSExfiltrator, 2022, [Online].

[52] A. Mairh, D. Barik, K. Verma, and D. Jena, "Honeypot in network security: a survey," in *Proceedings of the 2011 international conference on communication, computing & security*, 2011, pp. 600–605.

[53] L. Daigle, "WHOIS Protocol Specification," RFC 3912, Sep. 2004. [Online]. Available: https://www.rfc-editor.org/info/rfc3912

[54] Databricks and Microsoft, "Azure databricks," 2022, [Online]. [Online]. Available: https://azure.microsoft.com/en-us/services/databricks/

[55] Microsoft, "Synapseml," 2022, [Online]. [Online]. Available: https://microsoft.github.io/SynapseML/

[56] S. Y. A. Shulmin, "Use of dns tunneling for c&c communications," https://securelist.com/use-of-dns-tunneling-for-cc-communications/78203/, 2017, [Online].

[57] "Magnetic Stripe Track Format," 9 2015. [Online]. Available: https://orangetags.com/smart-card-reader/magnetic-stripe/magnetic-stripe-track-format/

[58] "Your Freedom - VPN, tunneling, anonymization, anti-censorship. Windows/Mac/Linux/Android." [Online]. Available: https://www.your-freedom.net/

[59] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[60] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. E. Hoffman, "Specification for DNS over Transport Layer Security (TLS)," RFC 7858, May 2016. [Online]. Available: https://www.rfc-editor.org/info/rfc7858

[61] P. E. Hoffman and P. McManus, "DNS Queries over HTTPS (DoH)," RFC 8484, Oct. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8484

[62] Agency, N.S., "Adopting encrypted dns in enterprise environments," https://media.defense.gov/2021/Jan/14/2002564889/-1/-1/0/CSI_ADOPTING_ENCRYPTED_DNS_U_OO_102904_21.PDF, 2021, [Online].

[63] Unknown, "Security benefits," https://developers.google.com/speed/public-dns/docs/security, 2023, [Online].

[64] ——, "How can i determine whether my dns queries to the amazon-provided dns server are failing due to vpc dns throttling?" https://aws.amazon.com/premiumsupport/knowledge-center/vpc-find-cause-of-failed-dns-queries/, 2023, [Online].