Resolution Without Dissent: In-Path Per-Query Sanitization to Defeat Surreptitious Communication Over DNS

Daiping Liu, Ruian Duan, Jun Wang
Palo Alto Networks
Santa Clara, California
{*dpliu, rduan, junwang*}*@paloaltonetworks.com*

***Abstract*—As one of the most fundamental Internet components, DNS has served various purposes and thus DNS traffic usually exhibits diverse patterns and is probably the least blocked by network administrators. These make DNS an attractive channel for attackers to establish surreptitious communications (*i.e., DNS tunneling*). In fact, such a surreptitious channel has been widely abused for command and control (C2) and enterprise-unapproved virtual private network (VPN). Existing approaches exclusively rely on the statistical characteristics of a sequence of DNS queries to detect DNS tunneling. Unfortunately, these approaches by nature cannot guarantee *zero* data leakage and can be evaded when the stolen data is exfiltrated over many root domains. As a result, state-of-the-art approaches are more suitable for threat investigation and forensic analysis, but not for DNS tunneling prevention.**

To fill this protection gap, we propose `TunTight`, the first system that is able to achieve *in-path per-query* DNS tunneling prevention. Our key insight is that DNS tunneling domains have unique characteristics in their authoritative nameservers, domain usage, and domain name patterns. Based on these characteristics, a set of unique features are defined and extracted which are fed to a machine learning model. To validate the efficacy of `TunTight`, we integrate it into the cloud backend of an enterprise firewall product by one of the largest security vendors. In our two-months real-world deployment, `TunTight` has successfully detected 349 confirmed tunnels at the very first query with negligible false positives and negatives. We also conduct the first large-scale study of DNS tunneling activities in the wild. One interesting finding is that most DNS tunneling traffic in enterprise networks come from public tunneling tools and enterprise-unapproved VPN services.

1. Introduction

Surreptitious communication over DNS (also known as *DNS tunneling*) has become one of the most severe security threats that enterprises have to tackle. When cybercriminals sneak into an enterprise's internal network and gain access to sensitive data, they can exfiltrate the stolen data out of the enterprise over DNS traffic [1] [2]. Another common use case of DNS tunneling is to set up a bi-directional communication channel through which attackers can establish command and control (C2) with the compromised internal hosts and conduct further reconnaissance. Such a bi-directional communication channel can also be abused as a VPN to bypass network security and control policies of an enterprise, *e.g.*, obtaining free Internet services [3]. This kind of abused VPN is referred to as enterprise-unapproved.

Compared to other surreptitious communication channels, DNS has several unique characteristics and properties attractive to attackers. As a fundamental component of the Internet, DNS is used by the vast majority of network applications. Therefore, modern enterprise networks, including the most security-conscious ones, often have to allow hosts to send and receive DNS traffic. Even worse, some enterprise networks fail to implement proper control policies, allowing arbitrary egress and ingress DNS traffic. Moreover, a client on average can send around 20K DNS queries per day [68] and an enterprise network can contain thousands of clients. Such a huge amount of DNS traffic serves as a great cover for DNS tunneling activities.

Many efforts have been devoted to DNS tunneling detection in the past [51] [52] [58] [67] [46]. All of these approaches are based on the fact that one DNS message can only convey a limited amount of data and thus it is inevitable for attackers to send out many DNS queries within a short time window. As such, the statistical characteristics of a sequence of DNS queries for the fully qualified domain names (FQDNs) under the same root domain[1] can provide highly promising detection capability. Unfortunately, these volume-based statistical approaches suffer two shortcomings. First, they by nature cannot prevent data being leaked in the first few queries. In practice, even partially leaked sensitive data (*e.g.,* a few passwords) could pose a perennial threat and incur irreparable losses to enterprises. Second, attackers can exfiltrate data using many different root domains, with one single query to each in the extreme case. Such a sophisticated evasion strategy has been adopted in advanced persistent threats (APT) [4]. Due to these shortcomings, existing approaches are more suitable for threat investigation and forensic analysis, but not for real-time detection and prevention.

In this paper, we develop `TunTight`, a machine learning based detector with the primary goal to detect and block

1. A root domain is also known as a bare/base/naked/apex domain that is separated from the top level domain by a dot, e.g., foo.example.

DNS tunneling activities at the *very first* query for enterprises. To achieve this, TunTight leverages the characteristics derived solely from every single query. We formulate the DNS tunneling characteristics from three aspects. First, we notice that attackers commonly set up their own authoritative nameservers (aDNS) so that the tunneling channel is reliable and supports bi-directional communications. These aDNS servers are usually dedicated and have low reputation. By contrast, the vast majority of non-tunneling domains prefer managed aDNS services such as those provided by domain registrars and cloud platforms. Accordingly, four unique features are defined for our machine learning model. Second, legitimate domains with tunneling-like traffic can often be attributed to specific Internet services owned by known organizations. Tunneling domains owned by attackers instead generally have no indicator of benignity. Based on this observation, we extract two additional unique features. Finally, DNS tunneling in practice dominantly encodes data in the query name field of a DNS message. This leads to distinguishing differences in the queried domain names between tunneling and non-tunneling traffic. For instance, tunneling domains often comprise a long sequence of random characters, compared to a short sequence of meaningful words in non-tunneling domains. To characterize this pattern, we compute three features, with one being first proposed in TunTight. These nine features are then used to train an unsupervised machine learning model, specifically Isolation Forest (IF) [62].

To validate the efficacy of TunTight, we perform comprehensive and in-depth evaluations. During our offline analysis using labeled datasets, we demonstrate that with an appropriate detection threshold for our IF model, TunTight can accurately detect all tunneling queries that are generated by both APT malware and VPN services. In addition, TunTight incurs negligible latency and thus it is efficient enough to be deployed on recursive resolvers for *in-path* protection. To the best of our knowledge, TunTight is the first system that is able to achieve *in-path per-query* DNS tunneling prevention.

We further integrate TunTight into the cloud backend of an enterprise firewall product, Advanced DNS Security by Palo Alto Networks [44]. In our two-months real-world deployment, TunTight inspects 253 billion queries from about 30K organizations across the globe. The results are quite promising, with 358 tunneling root domains being detected and confirmed. More importantly, all of these tunneling activities are detected within six queries, with 349 being detected at the very first one. Our manual analysis finds that most of these tunneling traffic comes from public tunneling tools and enterprise-unapproved VPN. There are 63 false positives (FPs) and 20 false negatives (FNs). All 63 FPs are low-profile domains and most of them could be eliminated with improved algorithms of feature computation. The 20 FNs are actually a subset of the 358 detected domains. This indicates that TunTight might miss some tunneling queries, but eventually all tunneling activities can be detected.

The remainder of this paper is organized as follows.

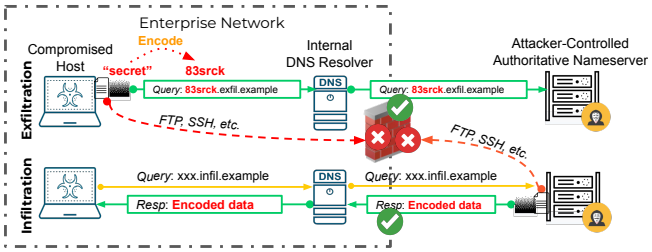


Figure 1: Two basic forms of surreptitious communication over DNS: DNS exfiltration and infiltration. Non-DNS traffic such as FTP and SSH is often strictly restricted in enterprise networks.

In §2, we briefly introduce the technical background of DNS tunneling. After that, we define the threat model in §3. Then, we give a high-level overview of our system in §4 and present the feature extraction and machine learning model in §5. In §6, we describe the implementation details. We validate the efficacy of our system using both labelled datasets and a large-scale real-world deployment in §7. In §8, we discuss the limitations and potential improvements of our detector. Finally, we survey related work in §9 and conclude in §10.

2. Surreptitious Communication over DNS

Since most enterprise networks implement relatively permissive policies for DNS traffic, surreptitious communication over DNS has become a usual modus operandi by adversaries to bypass enterprises' network security and control policies. Specifically, adversaries can tunnel malicious communications (*e.g.,* C2 and data stealing) through normal DNS queries and responses. Therefore, surreptitious communication over DNS is also commonly referred to as DNS tunneling.

As shown in Figure 1, the bi-directional communication over DNS tunneling is called DNS exfiltration and infiltration, respectively. In both cases, there is a compromised host, either hacked by external attackers or abused by insiders, within the victim enterprise network. The compromised host usually conducts domain resolution via an internal DNS resolver owned by the enterprise. Furthermore, a firewall is commonly deployed on the edge of an enterprise network for security enforcement and traffic control. In order to bypass firewalls and exfiltrate the data stolen from an enterprise network, an adversary first registers a root domain (*e.g.,* exfil.example in Figure 1) and sets up the aDNS for it or its subdomains. Then, the stolen data is encoded in DNS queries which are subsequently sent to the internal resolver, the same as other normal DNS queries. The internal resolver conducts iterative resolution and eventually forwards the data-encoded queries to the attacker-controlled aDNS. Finally, the adversary can decode the DNS queries collected at aDNS and obtain the stolen data.

Regarding the data encoding in DNS queries, there are various schemes. Among them, the most widely

used is to encode data in query names, *e.g.,* `83srck` in `83srck.exfil.example`. Alternatively, adversaries can encode data in query type or the arriving time of queries at aDNS. Adversaries can even use two domains (*e.g.,* `mx.exfil.example` and `ns.exfil.example`) to convey bit `0` and `1`, respectively, and set a low time-to-live (TTL) value (*e.g.,* `1` second) to avoid the interference of DNS caching. However, these alternatives either have extremely low throughput or are quite unreliable in practice. For example, many DNS resolvers in the wild do not honor the TTL value received from aDNS [54] [69] and there could be random or replayed queries from various crawlers and bots. Therefore, a low TTL value cannot remarkably improve the tunneling throughput.

Adversaries can also infiltrate data to enterprise networks, *i.e.,* DNS infiltration. For instance, shellcode and C2 commands could be encoded in DNS responses and transmitted to compromised hosts within enterprise networks. Same as DNS exfiltration, an adversary still needs to register a root domain (*e.g.,* `infil.example` in Figure 1) and controls its authoritative nameserver (aDNS). Theoretically, almost all types of DNS responses could be used for infiltration, but the most widely used ones include `A`, `AAAA`, `TXT`, `SOA`, `CNAME`, `SRV`, and `MX`. The infiltration data encoding often complies with the format of the specific types of DNS responses so that the infiltration traffic looks the same as normal DNS responses. In certain situations, such as when the infiltrated data is identical for all victims and remains unchanged for a period of time, the compromised hosts can download the data by sending just one single query to a specific domain (*e.g.,* `download.infil.example`).

DNS exfiltration and infiltration can further be combined to set up a bi-directional channel for interactive communications, *e.g.,* various public virtual private network (VPN) services leveraging DNS tunneling [5] [6] [7] [8]. In fact, these services are very popular, as evidenced by millions of downloads on platforms like Google Play. While these VPN services mainly intend to support the freedom of expression from restrictive networks, unfortunately, they are also often abused to bypass network security and control policies of enterprises, e.g., obtaining free Internet [3].

Finally, it is important to note that in the cases mentioned above, we assume the DNS resolvers are situated within the internal networks of enterprises and can therefore be regarded as trustworthy. However, compromised hosts can be instructed to communicate directly with *malicious resolvers* under the control of adversaries. In such a situation, adversaries can tunnel data over *random* domains, including legitimate ones such as `google.com` and even invalid ones like `dnscat.840b0[...redacted...]1fafec8e2`. Multiple DNS tunneling tools [9] [10] have implemented this paradigm. But, malware in the wild and real-world attacks rarely adopt this design because *i)* all invalid DNS traffic, *e.g.,* with malformed domains in queries, can usually be safely blocked; and *ii)* enterprise networks can block all DNS traffic to untrusted DNS resolvers or enforce that all outbound DNS traffic must originate from trusted internal resolvers.

Malicious *v.s.* Benign Tunnels. So far, we have solely described the DNS tunneling used by adversaries. However, in practice, DNS tunneling is also exploited by benign applications for various legitimate services. For instance, eSoft employs DNS tunneling to report CPU load, machine up time, *etc.*, and McAfee makes use of DNS queries for file reputation checks on behalf of their Global Threat Intelligence File Reputation Service [56]. Essentially, we cannot easily tell whether a detected tunneling case is malicious or not from its traffic pattern. However, built upon a couple of characteristics unique to malicious tunneling (§5 and §6), our system can significantly mitigate this issue.

3. Threat Model

Our goal is to protect enterprise environments against the malicious activities exploiting DNS tunneling. We do not aim to block the DNS tunneling for censorship purposes. Based on this targeted scenario, our threat model assumes the adversary controls *i)* at least one internal host within a victim's network, *ii)* a root domain for tunneling (*e.g.*, `attack.example`), and *iii)* the external aDNS of the root domain. The internal host initiates DNS queries to benign resolvers owned or trusted by the enterprise, and these queries comply with DNS specifications. DNS tunneling via malicious resolvers is out of scope because such cases are not popular in the wild and can be prevented using simple policies as described in §2. We further assume the tunneling data is transmitted over the root domains owned by adversaries, which is the case in all known attacks. Tunneling over hijacked legitimate domains is rarely seen in the wild and thus is out of scope.

Same as previous works [51] [52] [58] [67] [46], we focus on protecting against DNS exfiltration, one of the most concerned data leakage threats for enterprises. Note that interactive tunneling can often be disrupted too if the exfiltration is blocked. Depending on the size of the exfiltrated data, there could be a sequence of queries or only one tunneling query from the internal host, as in recent supply chain attacks [11]. In case of a large chunk of stolen data, adversaries could adopt multiple strategies to stay as stealthy as possible. For example, the stolen data could be split into small slices and exfiltrated from multiple internal hosts to different root domains [4], with one single query to each domain in the extreme case. Moreover, arbitrarily long intervals between two tunneling queries can be injected. None of the state-of-the-art methods [51] [52] [58] [67] [46] can effectively detect such sophisticated exfiltration activities because they exclusively rely on anomalous traffic behaviors observed across multiple queries over a relatively long time-window.

Furthermore, we focus on DNS exfiltration that uses queried domain names to encode data because it is the most popular and reliable exfiltration scheme as discussed in §2. Adversaries could transmit the stolen data either in plain-text or encoded forms. Actually, VPN services and open-source tunneling tools often encode data in `BASE32` and malware may adopt custom encoding schemes.

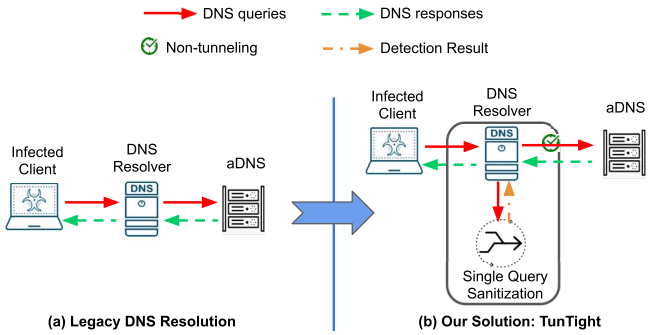


Figure 2: TunTight conducts in-path single query sanitization to disrupt DNS tunneling. Before a DNS query is sent to aDNS, TunTight checks if the query belongs to a tunneling session. If so, the query is blocked.

Finally, we do not assume that a DNS query can be linked to its initiating end-host. As a result, our approach could be deployed at not only end-hosts and DNS resolvers (in a similar way to [52] [67]) but also firewalls which sometimes could only see aggregated DNS traffic across many end-hosts.

4. Overview of TunTight

Our solution is motivated by the fact that, even partially leaked sensitive data (*e.g.*, passwords) could pose a perennial threat and incur irreparable losses to enterprises. Therefore, blocking from *the very first* DNS tunneling query is critical for data loss prevention. Unfortunately, existing DNS tunneling detection approaches [51] [52] [58] [67] [46] exclusively rely on the statistical features of a sequence of DNS queries and thus cannot prevent data being leaked in the first few tunneling queries. To fill this protection gap, we propose *in-path single query sanitization*, the first solution capable of blocking every DNS query carrying stolen data. Note that, our primary goal is to block all potential tunneling queries without affecting benign traffic. We view it as acceptable that some non-tunneling malicious domains get detected and blocked in practice. Essentially, our solution is designed to be a protection solution that complements existing systems.

While the solution sounds enticing, it is non-trivial to instantiate a real-world defense system. The main challenge lies in achieving an optimal trade-off among detection latency, accuracy and financial cost. In this paper, we design and implement a practical system, named TunTight. For real-world deployment, TunTight is quite versatile. We consider a recursive resolver as the best and most straightforward point to deploy TunTight, as shown in Figure 2. Meanwhile, TunTight can also be deployed on various other middleboxes, *e.g.,* firewalls and gateways. Next, we describe the design details of each component in TunTight.

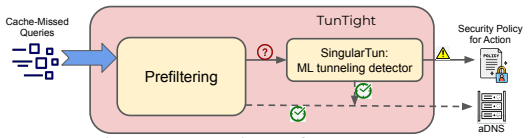


Figure 3: Design of TunTight.

5. Fast In-Path Single Query Sanitization

Compared to previous tunneling detectors, TunTight faces multiple unique challenges. The main one lies in accurately deciding if a *single* DNS query belongs to a tunneling session. None of the existing DNS tunneling detection approaches [58] [52] [51] [45] [66] [67] [70] is feasible because they all rely on the statistical characteristics of a sequence of DNS queries. Meanwhile, to achieve in-path detection, TunTight must be extremely fast to avoid any user-perceivable resolution latency. Finally, TunTight needs to be cost-efficient to scale in order to become a practical solution for large enterprise networks. To address these challenges, we design TunTight as shown in Figure 3. Only cache-missed queries need to be handled here because cache-hit queries are not forwarded to aDNS while tunneling queries have to reach adversary-controlled aDNS to exfiltrate data. The cache-missed queries are first fed to a prefiltering component which filters out any uninteresting traffic. For the remaining queries, we design a machine learning classifier, SingularTun, to identify the potential tunneling ones. The queries detected as tunneling are handled based on users' security policy and the non-tunneling ones are resolved as normal.

5.1. Prefiltering

Our prefiltering component first classifies the queried domains whose top level domain (TLD) is not open to public registration (Trusted TLD) as non-tunneling. These domains are commonly used exclusively within a local organization. For instance, the TLD chrome is intended for registration and use by Google only. Similarly, domains without a valid TLD are considered as non-tunneling (Invalid TLD), *e.g.,* example.local. In real-world traffic, we also observe a high volume of queries for FQDNs whose root domains are not registered (NXDomain). Moreover, DNS queries for root domains (Empty data) and subdomains with popular hostnames (Pop subdomain, *e.g.,* www.example.com) [12] are prefiltered. As there is no special data encoded in such queried domain names, the likelihood that such domains are used for tunneling is negligible. Finally, we classify the following two categories of domains as benign: *i)* the root domains in Tranco top 1 million list [59] with known malicious ones filtered out; *ii)* a FQDN all of whose aDNS servers are in our trusted list. The trusted list of aDNS servers is known to be benign and is curated by our partners who are security experts from a well-known security company. The list contains about 50 nameservers with no one from major cloud providers such as Amazon Route53 [42] being included.

5.2. SingularTun

5.2.1. Characterizing Tunneling Queries. The core of SingularTun comprises a set of unique features that can be computed from a single DNS query, as listed in Table 1. These features can reliably identify potential tunneling traffic. In particular, a couple of features like those derived from aDNS characteristics, benign software communication and potential known organization domains can effectively rule out benign tunnels.

Next, we describe each feature in details. To ease the presentation of feature definitions, we first introduce our notation: let `d` be the queried domain, `r` be the root domain of `d`, $\mathbb{P}$=$\{p_1, p_2, ..., p_n\}$ be the set of upper-level domains of `d` up to `r`, and $\mathbb{A}$=$\{a_1, a_2, ..., a_m\}$ be the set of aDNS servers of all domains in $\mathbb{P}$. For example, given `d=a.sub.foo.example`, `r` is `foo.example`, and $\mathbb{P}$=$\{$`sub.foo.example, foo.example`$\}$.

aDNS characteristics. Nowadays, more and more users prefer hosting their domains on managed DNS services for easier management, better performance and higher reliability. Users store their authoritative DNS records on the aDNS servers managed by a service provider. By contrast, adversaries have to set up their own aDNS for tunneling. First, many managed DNS services, *e.g.,* the one provided by GoDaddy (godaddy.com), do not export raw DNS logs to users. As a result, adversaries are not able to obtain the exfiltrated data and thus they could not leverage this kind of managed service. Second, for those allowing users to access raw DNS logs, they still lack flexibility for adversaries to dynamically customize DNS responses. In real-world DNS tunneling campaigns, adversaries commonly need to dynamically generate responses for the sake of remote control and reliable communications. Actually, our analysis shows that all publicly revealed advanced persistent threat (APT) campaigns [13] [14] [15] [16] [17] [18] [19] and VPN services over DNS (*e.g.,* Astrill [6] and HA-Tunnel [8]) set up their own dedicated aDNS servers. Figure 4 presents the aDNS of tunneling domains used in APT campaigns and VPN services. As a comparison, we analyze all the TLD zone files (1,000+) from CZDS [20] and find that around 24.6% root domains are hosted on managed DNS services by GoDaddy and Amazon Route53.

```
;; Case 1: Oilrig APT
  winodwsupdates.me. NS ns1.winodwsupdates.me.
  winodwsupdates.me. NS ns2.winodwsupdates.me.
  ns1.winodwsupdates.me. A 136.243.214.247
  ns2.winodwsupdates.me. A 136.243.214.247
;; Case 2: Saitama Backddor APT
  asiaworldremit.com. NS ns1.asiaworldremit.com.
  asiaworldremit.com. NS ns2.asiaworldremit.com.
  ns1.asiaworldremit.com. A 64.44.98.164
  ns2.asiaworldremit.com. A 64.44.98.164
;; Case 3: TLS Tunnel VPN
  3.ed2.host. NS ca1.d23.host.
;; Case 4: HA Tunnel Plus VPN
  ns-ca1.hat53.com. NS a-ca1.hat53.com.
;; Case 5: Your Freedom VPN
  s02.q82.be. NS ns1.s02.q82.be.
```

Figure 4: The aDNS of example domains used in APT campaigns and VPN services.

Based on such insights, we define four features to model the aDNS characteristics of tunneling domains. The first feature, `F1`, is set to 1 if any aDNS server of domains in $\mathbb{P}$ is likely under the control of attackers. Specifically, two types of aDNS servers are considered potentially controllable by attackers: *i)* an aDNS server a_i has the same root domain as `d`; *ii)* an aDNS server is hosted on a dynamic DNS service [21]. Otherwise, `F1=0`.

The second feature is built upon the intuition that the more domains an aDNS server hosts, the more likely it belongs to a managed DNS hosting service. Thus, this feature tries to model the popularity of aDNS. Specifically, for a_i in $\mathbb{A}$, we first get all domains that are delegated to it and then count the number of unique root domains. This number is denoted as `H(`a_i`)`. We further count the number of distinct root domains that are resolved to the IP addresses of a_i, which is denoted as `I(`a_i`)`. In case of multiple IP addresses of a_i, we choose the maximum. Now, the popularity is computed as $P_{\mathbb{A}} = \min_{a_i \in \mathbb{A}}\{\max\{H(a_i), I(a_i)\}\}$. Instead of using $P_{\mathbb{A}}$ as a feature directly, we find that normalizing it yields much better performance. In particular, we define this feature as `F2=1` when $P_{\mathbb{A}}$ is greater than a threshold and otherwise `F2=0`. We empirically set the threshold to be 100 because our analysis using passive DNS (pDNS) data shows that 100 is a good threshold to identify managed DNS services. In other words, `F2=1` indicates that all aDNS servers in $\mathbb{A}$ are likely managed services.

The third feature measures the reputation of aDNS domains. We consider domains with high traffic volume or long history as high reputation, and thus the set of high reputation domains combines i) Tranco top 1 million along with the root domains of their aDNS, and ii) domains whose ages are longer than 5 years. Known malicious domains are filtered out. Then, this feature is defined as `F3=1` if the root domains of all aDNS servers in `A` fall into this set. Otherwise, `F3=0`.

Finally, we observe that a tunneling domain usually either has just one aDNS server or all aDNS servers are resolved to the same IP as the cases shown in Figure 4. We conjecture that attackers adopt this design for two main reasons. First, attackers want to minimize the footprint and cost in their attack infrastructures. Second, using a single aDNS server can simplify the implementation and improve the maintainability of tunneling tools. This is because sophisticated tunneling tools often need to establish *stateful* communication channels for control purposes. However, the resolution behavior is unpredictable in the presence of multiple aDNS servers and an intermediate resolver might randomly pick one aDNS server to forward a DNS query. Therefore, tunneling tools using multiple aDNS servers on different IPs must synchronize each aDNS server in real-time to maintain stateful communication. In contrast, non-tunneling domains typically rely on *stateless* DNS resolution, using multiple aDNS servers on different IPs to enhance reliability and reduce resolution latency. Even for benign tunneling domains, there are often multiple aDNS IPs, as exemplified in Figure 5. Therefore, we compute the

Feature	Description	Unique
aDNS characteristics (F1∼F4)	The aDNS of a tunneling domain often has unique characteristics.	✓
Potential legitimate organization domain (F5)	Is the root domain potentially owned by a valid organization?	✓
Likely hosting benign services (F6)	Is this domain visited by benign software?	✓
Length of transferred data (F7)	Number of bytes that are potentially exfiltrated.	
Ratio of meaningful words (F8)	Ratio of meaningful words in the subdomain name.	✓
Suspicious characters (F9)	Does domain name contain characters not in {a-z,-,_,0-9}?	

TABLE 1: Features used by SingularTun for fast single query tunneling detection.

fourth feature as F4=1 if any domain in P has only one aDNS IP. Otherwise, F4=0.

```
;; Sophos
  i.03.h.sophosxl.net. NS ns.4.sophosxl.net.
  i.03.h.sophosxl.net. NS ns-2.4.sophosxl.net.
  ns.4.sophosxl.net. A 44.215.34.91
  ns.4.sophosxl.net. A 23.21.234.144
  ns-2.4.sophosxl.net. A 44.215.34.91
  ns-2.4.sophosxl.net. A 23.21.234.144
```

Figure 5: The aDNS of a benign tunneling case.

Potential legitimate organization domain. Intuitively, if a domain is owned by a legitimate organization, it is unlikely that the domain will be used for malicious tunneling. Unfortunately, it is quite challenging to compile a complete mapping between legitimate organizations and all domains owned by them. We address this challenge in two steps. First, we do best effort to collect known organizations and their main domains as much as possible. Then, we aggressively search for the domains potentially owned by these known organizations using Algorithm 1. The high level idea is to extend a seed domain by enumerating all TLDs to replace its original TLD and checking the domain owner using registrant name and emails in Whois database. If there is a match with the seed domain's Whois information, the enumerated domain is considered owned by the same organization and thus added into the generate list. Note that anonymous and invalid registration information is excluded using a similar method as in [63]. Moreover, domains with registrars MarkMonitor [22] and CSC [23] are also deemed as organization-owned because they are commonly defensively registered by a legitimate organization. The value of feature F5 is then set to 1 if the root domain is in the generated domain list. Otherwise, F5=0.

Likely hosting benign services. In order to avoid false positives caused by legitimate applications and services that exhibit tunneling-like behaviors [56] [67], one train of thought is to collect as much benign software as possible and monitor their runtime behaviors. Domains visited by these benign software are likely benign, too. Similarly, if a Web application provides legitimate services and contents, the hosting domains are unlikely used for malicious tunneling activities. The main challenge lies in the fact that, it is almost impossible to collect a full list of benign applications or determine precisely whether an application is benign.

To address this, we rely on two sources to collect the domains likely hosting benign services. First, a couple of public and propriety commercial services, such as VirusTotal [40] and ANY.RUN [37], provide dynamic code sample analysis and detect the malicious ones. These services usually have a huge repository of application samples. Second, there are also many security services that determine a domain's category based on the hosted content. For example, google.com is categorized as search engines by three vendors on VirusTotal [40]. For this feature, our current implementation simply considers a domain d as likely hosting benign services if any domain in P is visited by a benign application or categorized as a legitimate service. An application is considered as benign if it is not classified as malicious and it has a valid digital signature if it is a Windows binary. Therefore, this feature is defined as F6=1 if a domain is likely hosting benign services. Otherwise, F6=0.

Algorithm 1 Extend known organization domain list

Input:
SeedDoms ← Root domains of known organizations
WhoisDB ← Whois database
TLDList ← Public TLD list from publicsuffix.org
Output: OrgDoms ← Potential known organization domain list

```
1: procedure EXTEND_ORG_DOMAIN_LIST(SeedDoms, WhoisDB, TLDList)
2:     OrgDoms := emptyset
3:     OrgRegistrants := emptyset
4:     for dom ∈ SeedDoms do
5:         OrgDoms.add(dom)
6:         domname := trim_tld(dom)
7:         for tld ∈ TLDList do
8:             newDom := add_tld(domname, tld)
9:             if is_registered(newDom) then
10:                OrgDoms.add(newDom)
11:        recOrig := WhoisDB.get(dom)
12:        OrgRegistrants.add(recOrig.registrant)
13:    for rec ∈ WhoisDB do
14:        if rec.registrant ∈ OrgRegistrants then
15:            OrgDoms.add(rec.domain)
16:        if rec.registrar ∈ [MarkMonitor, CSC] then
17:            // Defensively registered domains often belong to an organization
18:            OrgDoms.add(rec.domain)
```

Length of transferred data. Due to the data exfiltration nature of DNS tunneling, a straightforward feature is to measure how much data is transferred. We simply assume the hostname part of a subdomain contains exfiltrated data. Therefore, we define feature F7 as the number of bytes of the hostname part. Note that, the dot "." in the hostname part is omitted. For instance, given host.name.foo.example, the concatenation of host and name is considered as the transferred data.

Ratio of meaningful words. The data exfiltrated in DNS tunneling is commonly encoded and thus the hostname part of a tunneling FQDN rarely contains meaningful words. However, legitimate subdomains prefer meaningful words, *e.g.,* webmail and forum. Based on this observation, we compute the ratio of meaningful words in the hostname part of a FQDN. Given a hostname h, let M(h) be the total

length of the meaningful words and `L(h)` be the total length of `h`. This feature is then defined as `F8`=$\frac{M(h)}{L(h)}$.

A main challenge here is to efficiently split hostnames into separate words so that the resulted `F8` is the largest. For instance, the hostname `mailserver` can be split into either `mail|server` or `mails|erver`. In the former case, the ratio is 1.0 while the latter is 0.5. Although there are algorithms to compute the optimal result, it can be too expensive to be used in our latency-sensitive scenario. Thus, we design a custom $O(n)$ algorithm to approximate the result and our empirical results show that this algorithm works quite well for hostnames. First, two lists containing different types of words are collected. One contains valid words in natural languages (`WordList`) [24] [25] [26] and the other consists of popular subdomain names (`SubdomList`) [27] [28]. For both lists, the included words must satisfy three criteria: 1) the word has at least four characters; 2) must contain at least one character in {`a-z`}; and 3) only contain characters in {`a-z,0-9`}. Then, `F8` is computed using Algorithm 2. Basically, a hostname is split into separate strings using three splitters {`-,_,.`}. If a string exactly matches a word in `SubdomList`, we consider it as a meaningful word. Otherwise, it is matched against `WordList` maintained in Trie using two modes (Line 10 and 11 Algorithm 2). For example, in case of `mbappliance`, `match_left_to_right()` greedily matches against the Trie from left to right, resulting in no match. `match_right_to_left()` tries to find the longest word from right to left, resulting in `mb|appliance`. Finally, only when more than half of a string is matched in either mode, it is counted in the final result. This can exclude the accidental partial matches for tunneling FQDNs. This heuristic is especially important to avoid false negatives for tunneling FQDNs with relatively short hostnames.

Algorithm 2 Compute meaningful word ratio

Input:
`WordList` ← Valid words in natural languages
`SubdomList` ← Popular subdomain names
`hostname` ← A hostname, e.g., `mailserver` in `mailserver.foo.tld`
Output: `F8` ← Meaningful word ratio

```
1:  procedure COMPUTE_WORD_RATIO(hostname, WordList, SubdomList)
2:      mlen := 0
3:      tlen := 0
4:      StringSet := split_with_splitters(hostname, [" − ","_","."])
5:      for string ∈ StringSet do
6:          tlen := tlen + len(string)
7:          if string ∈ SubdomList then
8:              mlen := mlen + len(string)
9:              continue
10:         l1 := match_left_to_right(string, WordList)
11:         l2 := match_right_to_left(string, WordList)
12:         l := max(l1, l2)
13:         if l/len(string) > 0.5 then
14:             mlen := mlen + l
        return mlen/tlen
```

Suspicious characters. Finally, tunneling tools usually implement encoding schemes that result in suspicious characters in the queried domains. For example, `BASE64` is a popular encoding scheme among DNS tunneling tools. As a result, the domains in these tunneling queries can contain characters + and = which are rarely seen in non-tunneling traffic. Leveraging this observation, this feature is defined as `F9=1` if a domain name contains characters not in {`a-z,-,_,*,0-9`}. Otherwise, `F9=0`.

5.2.2. Detecting Tunneling Queries. Built upon the aforementioned characteristics, a machine learning model is then developed to detect tunneling queries. In general, both supervised models (trained on labeled data) and unsupervised models (detecting deviations from "normal" behaviors) have been demonstrated effective to scale up the detection and classification in various security problems. We choose an unsupervised model, specifically Isolation Forest (IF) [62], in SingularTun mainly for two reasons. First, it is quite challenging to curate a representative labeled dataset for tunneling traffic. Attackers can adopt various strategies to obfuscate and conceal the communications, and thus the behavioral patterns of tunneling traffic are diverse. By contrast, it is much easier to collect a large set of non-tunneling traffic to train our detector. Second, IF is memory and computation efficient to handle large-scale datasets and thus able to achieve in-path real-time detection.

6. `TunTight` Implementation

`TunTight` is implemented in Go so that it can achieve good performance and can be easily integrated as a plugin to CoreDNS [38], a popular open source DNS server. To efficiently filter out non-registered domains (§5.1), the zone files provided by CZDS [20] are loaded into Redis and the data is updated every four hours following the update schedule of CZDS. For zone files of other TLDs (*i.e.* ccTLDs) not available in CZDS, we conservatively consider all observed domains as registered. The pDNS dataset used in our implementation combines the Farsight DNSDB [35] and the DNS data collected by Palo Alto Networks, one of the largest global security vendors. From these two sources, around 4 billion unique DNS records are collected every day. The pDNS data is streamingly ingested into Bigtable [55]. For feature `F2`, we precompute the popularity of all nameservers in our pDNS data and store the results in Redis. The popularity is updated weekly. At runtime, for any newly observed nameservers, we set `F2=0.01`. For feature `F5`, we currently leverage the daily dump from Crunchbase [34] as the seed to build the list of known organizations along with their main domains. Our WhoisDB contains the full data from WhoisXML [41] and the latest data is continuously ingested from WhoisXML. As a result, ∼136M domains potentially owned by organizations are collected. This list of domains is updated daily. The dynamic code analysis engine and domain categorization engine used for `F6` are two commercial services provided by our industry partner. In total, ∼420 million root domains are collected from these two engines, and the new data from these two engines are continuously synchronized into our system in real-time. The domain lists for `F5` and `F6` are loaded into Redis for fast feature computation.

7. Evaluation

In this section, we present an extensive evaluation of our approach's effectiveness, accuracy and performance. Our evaluation aims to answer three questions:

- Can SingularTun accurately identify tunneling queries?
- How efficient and scalable is `TunTight` and how much extra latency would be incurred for DNS resolution?
- Is `TunTight` effective in the real world?

To answer these questions, we assess `TunTight` in two setups. First, known benign and tunneling traffic datasets are compiled for offline evaluation. This setup mainly facilitates the training and tuning of SingularTun. It also demonstrates that SingularTun can accurately identify various types of malicious tunneling queries. Then, `TunTight` is deployed online to inspect the real-world DNS queries from thousands of organizations around the world. With billions of DNS queries per day, this large-scale deployment serves as a solid demonstration of the efficacy of `TunTight`.

7.1. Offline Evaluation and Tuning

7.1.1. Dataset. To train and tune our IF model in SingularTun, we collect three types of data.

Known benign traffic as training dataset. In SingularTun, the IF model is used for detecting anomalies in new DNS queries. Therefore, an uncontaminated dataset is expected for model training and such a training dataset is crucial for detection accuracy. For our evaluation, we collect 1-day's DNS traffic from the internal resolver deployed at the office site of a Software & Tech company with around 3K employees. The data was collected on a regular work day in 2019. The FQDNs are deduped and the invalid ones are filtered out using the rules in §5.1. Then, the dataset, denoted as D_{train}, is manually examined to make sure that it contains no tunneling FQDNs. Eventually, 125K unique FQDNs are included in D_{train}.

Known malicious tunneling traffic. To tune the IF model and evaluate the generality of SingularTun, we collect the malicious tunneling traffic from attack campaigns, VPN services and publicly available DNS tunneling tools. Table 2 summarizes the dataset, denoted as $D_{malicious_tun}$. For attack campaigns, we collect their C2 domains from public reports and then search for tunneling queries in pDNS. For each root domain, we randomly sample up to 10K unique FQDNs that were observed during the active period of an attack campaign. Manual examination shows that some non-tunneling FQDNs are included, *e.g.*, `www.winodwsupdates[.]me` and `ns1.asiaworldremit[.]com`. This background traffic does not exfiltrate data and thus can be excluded. We rely on the reverse engineered formats of tunneling messages to filter out the non-tunneling FQDNs. Such information can be found in public technical blogs of these APT campaigns. For VPN services, we run their clients on our machines and conduct two operations, browsing `google.com` and downloading a PDF file from a website. For public tools, we set up servers and clients on two machines, and a domain is also purchased if needed. For each tool setup, a PDF file is exfiltrated.

Group	# of Cases	# of Root Domains	# of FQDNs
Attack	7	30	97,379
VPN Service	9	18	28,456
Public Tool	6	5†	4,212

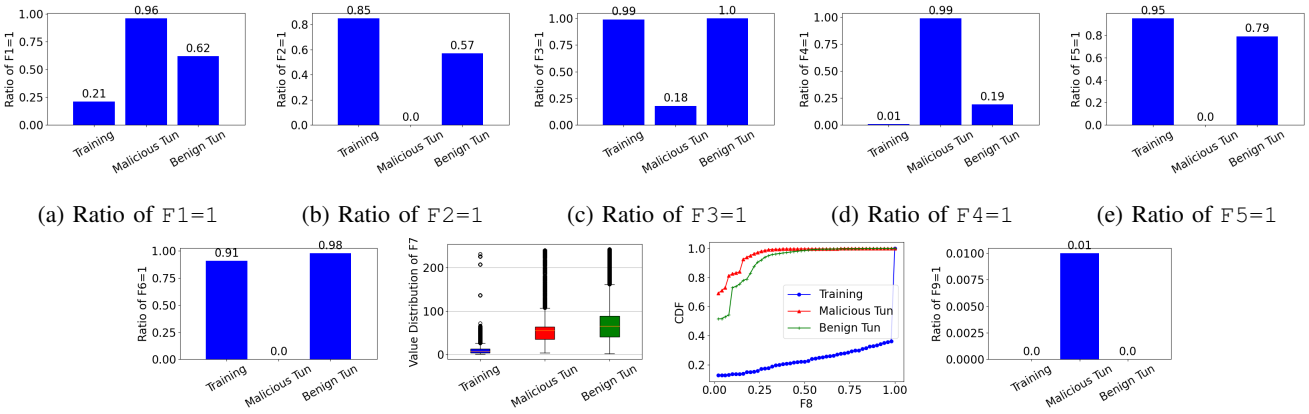
TABLE 2: Known malicious tunneling dataset. Full list of the case names can be found in Table 7 Appendix §A. † There are fewer root domains than cases because two of the tools shared one domain in our setup.

contentkeeper.io	dscsl.net	e5.sk
sophosxl.net	nessus.org	dsipsl.net
mcafee.com	kr0.io	

TABLE 3: Domains of known benign tunneling services.

Known benign tunneling cases. In order to evaluate the capability to distinguish between malicious and benign tunneling, we also collect the DNS traffic of legitimate services with tunneling-like behaviors. Table 3 lists the eight services we collected and they are often falsely detected in previous works [66] [67]. For each service, 10K FQDNs are randomly sampled from pDNS and this dataset is denoted as D_{benign_tun}.

7.1.2. Quantitative Analysis of Features for IF Model. Feature quality is crucial for machine learning models, especially for an anomaly detector. We have qualitatively characterized malicious tunneling in §5.2.1 and now we quantify the effectiveness of the extracted features. Figure 6 shows the distribution of each feature's value in non-tunneling (D_{train}), malicious tunneling ($D_{malicious_tun}$), and benign tunneling (D_{benign_tun}) traffic, respectively. We can see that non-tunneling traffic exhibits totally different distributions in almost every feature. In particular, 99% of DNS queries in D_{train} have `F3=1` while only 18% in $D_{malicious_tun}$ have `F3=1`. Similarly, the distributions of `F4=1` are almost the opposite between D_{train} and $D_{malicious_tun}$. Another interesting case is `F9`. All queries in D_{train} and D_{benign_tun} contain only valid characters while about 1% of DNS queries in $D_{malicious_tun}$ contain invalid characters. We want to emphasize that this should not be seen as a foolproof rule: just because a DNS query contains invalid characters, it does not mean it is definitely DNS tunneling. Although our D_{train} data does not include invalid characters, a significant number of legitimate, non-tunneling queries in real world do. Conversely, most malicious tunneling traffic also lacks invalid characters. Therefore, relying on this feature alone can result in both high false positives and high false negatives. Finally, the data also shows that `F3, F4, F5, F6,` and `F9` serve as a promising set of features to differentiate benign tunneling queries from malicious ones. In summary, every feature used in our IF model yields remarkable capability to tell malicious traffic from benign one. Although



(a) Ratio of `F1=1` (b) Ratio of `F2=1` (c) Ratio of `F3=1` (d) Ratio of `F4=1` (e) Ratio of `F5=1`

(f) Ratio of `F6=1` (g) Distribution of `F7` (h) CDF of `F8` (i) Ratio of `F9=1`

Figure 6: Feature analysis of D_{train}, $D_{malicious_tun}$, D_{benign_tun}

each of them can still incur unacceptably high false positive rate when used individually, the combination of all features results in reliable detection of malicious tunneling.

7.1.3. SingularTun Tuning. We now assess the effectiveness of the IF model in SingularTun. There are four important parameters to set in the IF model, the number of trees (N_{tree}), sub-sampling size ($S_{subsample}$), contamination ratio (C_{ratio}), and threshold of anomaly scores (T_{score}). Since we have manually confirmed that D_{train} only contains non-tunneling queries, we simply set $C_{ratio} = 0$. To tune N_{tree} and $S_{subsample}$, we perform hyperparameter search which aims to minimize the overlap of anomaly score ranges between $D_{train} \cup D_{benign_tun}$ and $D_{malicious_tun}$. It turns out that $N_{tree} = 500$ and $S_{subsample} = 1000$ render the best detection performance and reliability. Unless otherwise specified, we shall use these values as the default parameters in our experiment. Under this setting, the distribution of the anomaly scores for D_{train}, D_{benign_tun} and $D_{malicious_tun}$ is shown in Figure 7. We can see that, there is a relatively large gap, specifically range $[-0.26, -0.22]$, between $D_{train} \cup D_{benign_tun}$ and $D_{malicious_tun}$. As a result, T_{score} can be set with a value in this range to reliably identify malicious tunneling queries with high accuracy. For instance, when $T_{score} = -0.258$, no single query in $D_{train} \cup D_{benign_tun}$ is identified as tunneling (*i.e.,* no false positive) and two queries in $D_{malicious_tun}$ are classified as non-tunneling (*i.e.,* two false negatives). The anomaly scores of the two false negatives are about -0.252, which are close to the threshold. The main cause of these false negatives lies in `F8`. The hostname accidentally contains meaningful English words, *e.g.,* `44heazy8.asiaworldremit[.]com`.

In conclusion, these results confirm that SingularTun can reliably identify malicious tunneling queries. In practice, security operators can decide the threshold based on their specific scenarios and security requirements. Generally, a loose threshold (*i.e.,* larger value for T_{score}) results in more queries to be detected as tunneling, and thus could reduce false negatives but potentially cause more false positives.

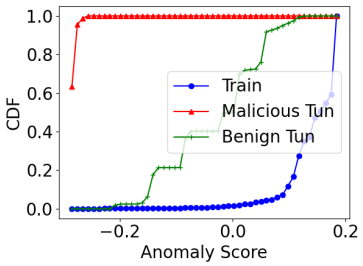


Figure 7: Cumulative distribution function (CDF) of anomaly scores from offline evaluation datasets.

7.1.4. Efficiency of SingularTun. Since SingularTun is designed to be deployed in the path of DNS resolution, its efficiency is critical to avoid incurring high latency to DNS resolution. The prefiltering component requires a couple of reads to hash tables in memory. The cost of such operations are often negligible and it takes ∼25 microseconds per query for the whole prefiltering module. We next mainly focus on the performance of the IF model. Our measurement is conducted on a virtual machine (VM) instance of e2-highmem-16 type from Google Cloud Platform (GCP). The VM has 16 virtual CPUs of Intel Broadwell and 128GB memory. As described in §6, all data for feature extraction is loaded into Redis. In our evaluation, the Redis instance is co-located with SingularTun on the same VM to avoid the noise from network jitter. Similarly, the aDNS data required for `F1` and `F4` are preloaded into memory. On average, for one DNS query, it takes $0.17ms$ to extract the feature vector and $0.1ms$ for the IF model to compute the anomaly score. Compared to the $20ms$ resolution speed of popular modern DNS resolvers [29], the extra $0.27ms$ latency incurred by SingularTun is trivial. While our testing is conducted on a small-scale dataset and does not consider concurrent requests, the real-world deployment aligns with the above latency. In our deployment, there is a cluster of detector instances and each instance sequentially processes

DNS queries. The cluster auto-scales up and down based on the traffic volume. The stateless per-query detection in TunTight makes the autoscaling easy and straightforward.

7.2. Real-World Operation

Our offline evaluation has achieved encouraging results. We further deploy TunTight in real-world networks for more practical and comprehensive evaluation.

7.2.1. Deployment of TunTight. TunTight is integrated into the cloud backend of an enterprise firewall product by one of the largest security vendors, namely Advanced DNS Security by Palo Alto Networks [44]. The firewalls are deployed in tens of thousands of organizations across the globe. The firewalls intercept all DNS queries which are then sent to TunTight on the cloud. Throughout our evaluation, TunTight runs in *proof-of-concept (PoC)* mode where DNS queries are sent to TunTight and aDNS servers simultaneously. As a result, potential tunneling queries are not blocked inline. However, TunTight can instruct the firewalls to block subsequent FQDNs under the detected tunneling root domain. Based on the results in §7.1.3, we set the threshold of SingularTun as $T_{score} = -0.258$ in this deployment. As a result, given a DNS query from a firewall, if its anomaly score is smaller than -0.258, the root domain of the FQDN in the query is classified as tunneling and this verdict is sent back to the firewall. The domain verdicts received from the cloud backend are cached on firewalls and the caching policy is: *i)* benign domains are cached for 1 hour and the firewalls conduct exact-match (*i.e.,* subdomains under these benign domains are still sent to our detector on the cloud for detection); *ii)* tunneling domains are cached for 5 minutes and the firewalls conduct parent-match (*i.e.,* once a domain has a tunneling verdict in cache, all of its subdomains will be classified as tunneling and will not be sent to our detector in the cloud.)

7.2.2. Ethical Considerations. Our research follows the rules and policies by our legal department.

Our data collected for manual investigation does not contain any personally identifiable information (PII). Four types of data are used in this work: query type, queried FQDN, rdata in responses, and serial number of the firewall where a DNS query is seen. The query type, queried FQDN, and rdata have been classified as non-PII by our legal department. And, such data can be accessed by researchers in the company. This is explicitly described in the Privacy Policy of the service provided to the organizations whose data is collected [39]. The serial number is only used by the detectors at runtime to filter out testing traffic from non-production firewalls, and it is not logged or available during manual investigation. Thus, a DNS query cannot be attributed to a specific organization or user.

For DNS traffic collected in countries and regions with specific data policies, *e.g.*, General Data Protection Regulation by the European Union, we have detector instances deployed in those regions to process the traffic.

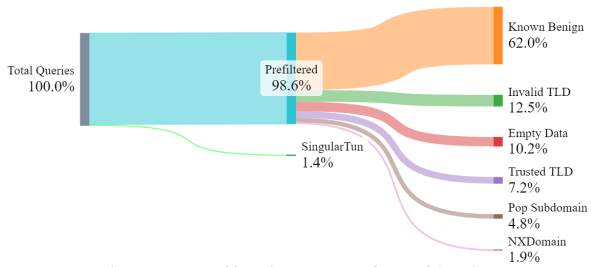


Figure 8: Effectiveness of prefiltering.

In addition, all data processing and analysis are performed on secure internal servers owned by our company. And, for detected tunneling FQDNs, we never try to decode them to obtain the exfiltrated data.

Finally, the organizations where the firewalls are deployed are aware of the block action of detected tunneling domains. Actually, they purchase our DNS Security service to detect and block DNS tunneling from their networks. They can configure the action, *e.g.*, allow/alert/drop/sinkhole, for each type of malicious domains including the tunneling ones. There is also an official channel through which organizations can submit reports if they have concerns about a potential false positive (FP). We have not received any such kind of complaints for the detected domains with tunneling verdict returned to firewalls.

7.2.3. Efficacy of TunTight. TunTight has been deployed for two months from 2024-03-19 to 2024-05-19. The results from our large-scale deployment with 253 billion queries from about 30K organizations demonstrate that the efficacy of prefiltering and SingularTun is quite promising and TunTight is on standby for production deployment.

Efficacy of prefiltering. Figure 8 shows how many DNS queries are prefiltered using each rule described in §5.1. The prefiltering rules are checked in the order as shown in Figure 9. Once a rule decides that a DNS query is not tunneling, the subsequent rule checks are skipped. In other words, if these rules are checked in a different order, the statistics could vary significantly. We can see that the prefiltering is quite effective in practice and about 98.6% of cache missed DNS queries could be prefiltered. It is expected that the rule "*Known Benign*" is the most effective, covering ∼62% of the DNS queries. The second most effective rule is "*Invalid TLD*". Considering that our DNS queries are collected from enterprise private networks, this result is not surprising. Actually, most of the domains prefiltered using this rule have invalid TLDs that are commonly used in private networks, *e.g.,* local and internal. Moreover, about 10.2% of queries try to resolve root domains. Note that, many of these root domains are actually DGA-like (Domain Generation Algorithm) NXDomains and they are counted in "*Empty Data*" instead of "*NXDomain*" due to our rule check order in Figure 9. The remaining three prefiltering checks ("*Trusted TLD*", "*Pop Subdomain*", and "*NXDomain*") are relatively less effective. However, they are still able to filter out ∼13.9% combined.

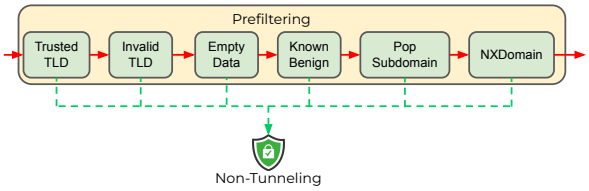


Figure 9: The order of prefiltering rules that a DNS query is checked against. Once a rule decides a DNS query to be non-tunneling, the subsequent rule checks are skipped.

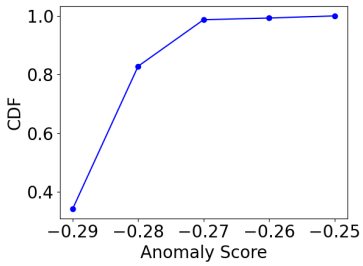


Figure 10: Distribution of anomaly scores in real-world deployment. Since more than 99.8% of scores are greater than -0.25 which are highly unlikely tunneling, we only show the distribution of scores less than -0.25.

Efficacy of SingularTun. After prefiltering, about 3.58 billion DNS queries are fed to SingularTun. More than 99.8% of these queries have anomaly scores greater than -0.25. Our offline evaluation has shown that such queries are highly unlikely tunneling. Therefore, we focus on the the queries with anomaly scores smaller than -0.25. The distribution of their anomaly scores is presented in Figure 10. Among these queries, 5.7 million (*i.e.*, 0.002% of the total traffic) have anomaly scores less than -0.258, the threshold we decided in §7.2.1. In our PoC deployment, we purposely add delays (ranging from hours to days) before tunneling verdicts are pushed to firewalls to block further DNS tunneling queries. This intended delay is mainly for the sake of detection verification and false positive prevention. Meanwhile, it also enables us to collect more tunneling queries for evaluation. As a result, the number of tunneling queries sent to SingularTun is much higher than the number of unique tunneling root domains detected. We next dive deep into the results to evaluate detection accuracy and completeness.

7.2.4. Detection Result Analysis. As mentioned in §7.2.1, when the anomaly score of a queried FQDN is smaller than -0.258, the root domain of the FQDN will be tagged with a tunneling verdict. If a detected root domain is confirmed as tunneling, we call it a *true positive (TP)*. On the contrary, if a detected root domain turns out be to benign, we call it a *false positive (FP)*. It is also possible that a detected root domain is not tunneling but serving other malicious or illegitimate activities (e.g., phishing and scam, with full list in Appendix B). We refer to these cases as *good-to-block (G2B)*. For the remaining detected root domains, we categorize them as

	# of Queries	# of Root Domains
Total Detection	5,742,689	834
TP	5,530,388	358
G2B	48,825	309
FP	159,658	63
Unknown	3,818	104
Candidate FN	833,495	3213
TN	801,872	3175
FN	31,541	20
Potential FN	82	18

TABLE 4: Number of domains detected by our system and potential false negatives during our real-world deployment from 2024-03-19 to 2024-05-19.

Unknown. Finally, for an undetected root domain (*i.e.,* the anomaly score is larger than -0.258), if it is non-tunneling, we call it a *true negative (TN)*. Otherwise, if it turns out to be tunneling, we call it a *false negative (FN)*. The cases that cannot be definitely decided are referred to as *potential FN*. To evaluate the false negatives, due to the huge traffic volume, we investigate three cases: *i)* the queries for the FQDNs of known tunneling root domains whose anomaly scores are < -0.258; *ii)* the queries whose anomaly scores are in range $(-0.258, -0.25]$; *iii)* a set of 1,500 randomly sampled root domains that are not detected as tunneling. These queries are referred to as *candidate FNs*.

During our deployment, 834 unique root domains are detected as tunneling and 3,213 unique root domains are candidate FNs (including the 1,500 sampled ones). To verify the detected domains, we manually analyze each case and assign them to the above categories. The detailed verification workflow is described in Appendix B. Table 4 presents our analysis results. Overall, our detection accuracy is quite promising, with 667 out of 834 detected cases categorized as TP and G2B. These domains can be safely blocked in practice. We next dive deep into the detection results.

TP. We successfully confirmed 358 TPs which account for 5.5M DNS queries. This is reasonable because VPN services based on DNS tunneling usually generate a huge number of DNS queries to achieve high-throughput bidirectional communication. Actually, about 4.9M queries come from a single case `codered-servers[.]online` which turns out to be a tunneling tool built upon V2Ray [30]. As the primary motivation of `TunTight` is to block the very first tunneling query, a key metric is how many tunneling queries are missed before a tunneling activity is detected. We consider a tunneling activity to be detected once one FQDN is flagged as tunneling by `TunTight`. This is based on our detection policy in production that, once a FQDN is detected as tunneling, its root domain is immediately pushed to the firewall so that all subsequent FQDNs are blocked as tunneling. In our PoC deployment, however, after a FQDN is detected by `TunTight`, subsequent FQDNs are not immediately blocked on firewalls. First, to avoid potential severe FPs, a detection by `TunTight` needs to manually verified before being published to firewalls. This usually takes hours to days. Second, collecting more data can help us comprehensively evaluate potential FNs. Among

the 358 TPs, 349 are detected at the very first query, 6 are at the second query, and the rest 3 are detected within six queries. Furthermore, once a TP has been manually confirmed, we start to return the tunneling verdict to firewalls. Up to 2024-09-15, we have not received any complaint of FPs from the organizations where our firewalls are deployed. The evaluation results demonstrate that TunTight can effectively and accurately prevent data being leaked over DNS tunneling.

G2B. TunTight detects 309 G2B domains as tunneling in two months. Compared to thousands of new malicious domains everyday [36], this result indicates that the features used by TunTight are specific to DNS tunneling and do not model the typical characteristics of non-tunneling malicious domains.

FP. 63 detected root domains are verified to be FPs. During our manual investigation, no evidence shows that these FPs are high-profile domains or used for critical services. Further analysis shows that most of these FPs can be categorized into the following groups. First, while the FQDNs of most FP cases look like tunneling (*i.e.,* long hostname with random characters), there are 28 cases whose FQDNs look unlike tunneling, *e.g.,* liuhe2014shengxiaoboban.htt[.]pw and www.declaraciones2024.web-com[.]cl. This suggests that further improvements are needed for F8, including expanding the list of meaningful words (*e.g.,* Chinese Pinyin in the case of htt[.]pw) and increasing the accuracy of Algorithm 2. Second, a single FQDN for each root domain is repetitively queried in 17 cases. For instance, om2v9[...redacted...]qr.jabiaksi[.]com is the only observed FQDN under jabiaksi[.]com and it is queried five times in three days. Therefore, one potential mitigation is that, for FP-sensitive environments, we do not block a domain until two unique FQDNs are observed within a predefined time-window. Moreover, 9 FP domains have been registered and actively used for many years. Motivated by these cases, we plan to incorporate the domain age information into our detectors to avoid such FPs. Finally, there are 6 FPs whose aDNS turn out to be trusted after manual analysis. In summary, these FPs probably will not cause severe production issues in practice and most of them could be mitigated with future improvements.

Unknown. There are 104 cases where we cannot decide the purpose of queries at a confident determination. The FQDNs of these cases look like tunneling and there is no useful information on the Internet. For example, the domain 9mn[.]lat has a number of tunneling-like FQDNs, *e.g.,* dpdf3d[...redacted...]skpx.9mn[.]lat and dc81[...redacted...]cof.9mn[.]lat. The only useful information we found is that 9mn[.]lat was registered on 2023-12-23 and will expire on 2024-12-23. Actually, 70 of the 104 domains are registered within one year and the expiration dates are also within one year. We posit that blocking these low-profile new domains should have trivial business impact on enterprise networks.

FN. 20 FNs are observed in our evaluation. We find that all of these FNs overlap with TPs, which means while some tunneling queries could have been missed, all tunneling activities are eventually detected. In particular, there is no FN from the 1,500 sampled root domains whose 54,619 FQDNs are manually checked. 14 of the 1,500 domains fall into G2B. Regarding the root causes of these FNs, our analysis identifies three features as the main factor, specifically 7 FNs having F2=1, 7 having F4=0, and 6 having F6=1. As described in §5.2.1, tunneling queries usually have a different value in these three features. Moreover, all 31K queries of the 20 FNs have short FQDNs and the FQDNs have relatively high meaningful word ratio.

Potential FN. Finally, 82 FQDNs under 18 root domains are classified as potential FN, such as:

```
25c71bcb[...redacted...]24654c6.trk.scorvira.com
aaed543d[...redacted...]699593fd.trk.tyblyr.com
```

These FQDNs look like tunneling and their root domains are mostly newly registered. We cannot decide with high confidence how to classify these DNS traffic.

	TXT	A	AAAA	MX	CNAME	NS	Other
Traffic Ratio (%)	93.4	4.6	1.6	0.2	0.05	0.005	0.1
Adoption Ratio (%)†	35.2	60.3	38.5	8.1	5.0	12.0	8.7

TABLE 5: The traffic ratio breakdown by query types and the ratio of root domains adopting each query type. † Each root domain can use multiple types of queries for tunneling.

7.2.5. Analysis of Real-World Tunneling. We follow the steps described in Figure 13 to conduct best-effort attribution for the detected tunneling domains. Our results show that out of the 358 TPs, 44 can be attributed to known attacks and 44 are attributed to known VPN services and public tools. In particular, the traffic originating from the 44 VPN services amounts to 5.2M queries, namely, 93.4% of the total tunneling traffic. This is because VPN services are bandwidth-intensive, compared to attack campaigns where attackers usually try to minimize their footprint.

To understand query types used in tunneling, we characterize and aggregate the tunneling traffic using two metrics, Traffic Ratio and Adoption Ratio as shown in Table 5. The Traffic Ratio row lists the percentage of tunneling queries of each query type. We can see that 93.4% of tunneling queries are TXT. A further look reveals that Traffic Ratio is biased towards codered-servers[.]online which accounts for the majority of the detected tunneling traffic. Hence, we use Adoption Ratio to present the ratio of root domains adopting each query type for tunneling purposes. This metric is a better depiction of the choices of different threat actors. It is evident from Adoption Ratio that TXT, A and AAAA are the mostly query types, with A being the most popular and adopted by 60.3% domains.

Figure 11 shows the CDF of the number of configured aDNS IPs for each tunneling root domain. 77% of these domains are configured to use only one aDNS IP, which aligns with our intuition of F4. The rest of the tunneling domains use more than one aDNS IPs, with the extreme case of the VPN domain srvdns[.]us

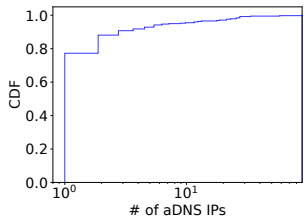


Figure 11: CDF of the number of aDNS IPs for the detected tunneling root domains.

Top ISPs by # of aDNS IPs		Top ISPs by # of C2 aDNS IPs	
DigitalOcean	80	Linode	38
Linode	66	DigitalOcean	24
Contabo GmbH	53	Akamai	14
CloudFlare	26	Google	11
Akamai	18	Hetzner Online GmbH	11
IPXO	17	Vultr Holdings	10
Amazon	16	CloudFlare	10
Google	16	The Constant Company	9
Vultr Holdings	15	Amazon	9
OVH SAS	14	Vercel	9

TABLE 6: Top ISPs adopted by generic tunneling domains and C2 domains, respectively.

using 89 aDNS IPs. These domains usually use different subdomains (*e.g.,* `ns-br1dns.srvdns[.]us` and `ns-ge1dns.srvdns[.]us`) for different tunneling sessions, with each subdomain having one single aDNS (*e.g.,* `br1dns.srvdns[.]us` being the aDNS of `ns-br1dns.srvdns[.]us`). This is another evidence that DNS tunneling domains commonly use one single aDNS to achieve reliable stateful communications.

Additionally, we characterize the ISP of the tunneling aDNS IPs. Table 6 lists the top ISPs adopted by generic tunneling domains and C2 domains, respectively, where the latter is a subset of the former. They are ranked by the number of aDNS IPs from their corresponding networks. The table shows that Linode and DigitalOcean are favored by both generic tunneling domains, as well as C2 domains. Meanwhile, Contabo GmbH are frequently used for generic tunneling, but rarely used by attackers. The behavioral differences in ISP selection could be affected by many factors such as pricing strategies and privacy policies. We leave such qualitative analysis for future work.

Finally, we study the distribution of creation dates of the detected tunneling root domains in Figure 12. While the plot shows that newer domains are more-frequently used for tunneling purposes, it is important to notice that older domains can be used as well. For example, `vpnoverdns[.]com` is registered in 2012 (*i.e.,* domain age of 12-years) and is used to serve the VPN-over-DNS [31]. Hence, the domain age, as a potential feature to reduce FPs, needs to be combined with other features to avoid FNs.

8. Discussion & Limitations

We have demonstrated that `TunTight` can effectively block the very first DNS tunneling query with high accuracy. The low latency and high-throughput detection enables `TunTight` to be deployed inline on resolvers. However,

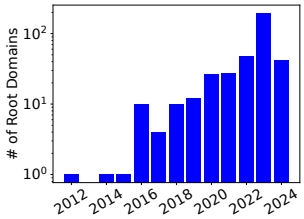


Figure 12: Distribution of the creation dates for the detected tunneling root domains.

`TunTight` also has a couple of limitations and we discuss them in this section.

This paper only covers how `TunTight` detects the data encoded in FQDNs, the dominating information vector in DNS tunneling. However, attackers can encode data in other fields of DNS protocol, including query type, address class, *etc*. Moreover, data could also be encoded in the timing of queries. One future work is to improve `TunTight` to defeat these data encoding schemes.

Attackers can also leverage DNS tunneling for infiltration only. For instance, attackers can encode C2 commands and shellcode in DNS responses, and a compromised host can download the data by simply issuing a DNS query for a non-tunneling like FQDN (*e.g.,* `account.infil[.]example`). Such pure infiltration activities have been observed in the wild [32]. This kind of threat is out of the scope of `TunTight`.

The prefiltering rules can potentially be exploited to bypass detection. For instance, attackers can obtain the domains in the Tranco list. Attackers can achieve this through domain drop-catch [43] or buying domains from original owners. One future improvement is to track the ownership change of known benign domains. Similarly, the Trusted TLD and Invalid TLD lists should be kept up-to-date.

As described in §7.2.4, a couple of our feature extraction algorithms could be improved to address the FPs and FNs. There are also multiple additional potential improvements. For example, for `F6`, if we know definitely an application is benign, then any domain visited by the application can be considered benign and can even be added to the prefilter. Additionally, we will continue to study if there are better ways to model the unique characteristics of tunneling domains' aDNS. Further, inspired by the email server reputation based approaches to combat spamming, we can investigate if a similar idea could be applied to DNS tunneling detection and prevention. For instance, DNS security community can collaborate and build a reputation database for aDNS servers so that users can optionally block DNS queries for domains hosted on disreputable servers.

As shown in Table 4, a non-trivial number of detected domains fall into G2B category. Since `TunTight` is designed to be an inline prevention system, it is acceptable and fine to block these domains in practice. However, it increases the manual investigation burden if users (*e.g.,* Security Operations Center (SOC) teams) are especially interested in true DNS tunneling traffic. For such scenarios,

existing DNS tunneling detection systems [51] [52] [58] [67] [46] can facilitate the identification of interesting tunneling activities from the blocked DNS queries.

Finally, our current detection assumes that attackers own the root domains. However, attackers can leverage legitimate services such as dynamic DNS [21], subdomain leasing [33], and compromised domains [61]. Since no real-world case has been observed, detecting such cases remains as our future work.

9. Related Work

Our study is particularly analogous to two areas of prior work: covert communication and DNS tunneling detection.

DNS tunneling is a form of covert communication. According to the taxonomy in [65], DNS tunneling via query content (*e.g.,* queried domain names and query types) and query timing could be treated as a storage and timing channel, respectively. The general problem of covert communication over networks has received extensive studies in the literature. Cabuk et al. [53] developed the first IP covert timing channel which transmits a 1-bit and 0-bit by sending and not sending a packet during an interval, respectively. One innate property of such a scheme is that the variance of the inter-packet delays remains relatively constant, and therefore the covert channel could be detected by identifying the artificially introduced regularity. Gianvecchio et al. [57] used conditional entropy measures to detect various covert timing channels in Internet traffic. Detection of storage covert channels has also been studied before. Built upon the insight that most network traffic contains fixed protocol data and the data that could be derived according to protocol specifications, Borders et al. [50] showed how to quantify the true volume of information flows in HTTP requests. Adversaries could also encode data into popular multimedia applications such as Skype to set up covert channels [60] [64] [47] [49]. Barradas et al. [48] demonstrated that such covert channels could be detected by machine learning techniques utilizing features computed from network packet lengths and packet inter-arrival times. Compared to these works, we take advantage of DNS specific properties and do not rely on packet-level features.

Existing approaches for DNS tunneling detection exclusively examine the statistical characteristics of DNS traffic. Karasaridis et al. [58] observed that DNS tunneling traffic exhibits a distinguishable packet-size distribution, and thus proposed a detection approach that computes the hourly Kullback-Leibler distance between the packet size distributions of the observed DNS traffic and a baseline. Butler et al. [52] proposed to detect DNS tunneling by computing the Jensen-Shannon divergence of byte distributions in normal and tunneling DNS payloads. The detection approach in [51] computed the 1-, 2-, and 3-gram fingerprints of the domain names in observed DNS queries. If the fingerprints follow Zipf distributions, the traffic is legitimate. Otherwise, it is detected as DNS tunneling.

Multiple recent works [45] [66] trained machine learning models to identify DNS tunneling traffic. These models were built upon a list of features extracted from the querying behavior of each domain, *e.g.,* character entropy of domain names and number of unique domain names, *etc.*

Another line of research tries to filter out non-tunneling traffic as much as possible so that security analysts can manually investigate the remaining suspicious DNS queries. Paxson et al. [67] proposed a principled means that picks out DNS tunneling communications by estimating the amount of information potentially conveyed in a stream of DNS queries. Specifically, the DNS query stream is fed into compression algorithms such as gzip and the size of the compressor's output is used as the estimate. Security analysts can adjust the configurable bound to trade off detection completeness and analysis burden. The authors in [70] filtered out normal traffic using a list of manually created rules based on general characteristics of DNS tunneling (*e.g.,* number of subdomains and size of response). Our idea differs fundamentally from these works. Instead of detecting DNS tunneling for forensic or post-mortem protection, we aim to prevent DNS tunneling by *blocking the very first* exfiltration query so that no data is leaked. Actually, as discussed in §8, our work can complement existing systems to achieve inline DNS tunneling prevention and provide better forensic capabilities.

10. Conclusion

This paper has presented `TunTight`, the first system that achieves in-path per-query DNS tunneling prevention. Built upon a set of unique features derived from the unique characteristics of DNS tunneling activities, `TunTight` can accurately detect tunneling traffic with negligible false negatives. In particular, by avoiding using any traffic volume based statistical features, `TunTight` is able to detect and block the *very first* tunneling query. We demonstrate the efficacy of `TunTight` by integrating it into the cloud backend of an enterprise firewall product by one of the largest security vendors. In our two-months real-world deployment, `TunTight` has successfully detected 358 confirmed tunnels, with 349 detected at the very first DNS query. We also conduct the first large-scale study of the characteristics of DNS tunneling activities in the wild, and find that illegitimate VPN is the main source of tunneling traffic in enterprise networks.

11. Acknowledgements

We would like to thank the anonymous reviewers and our shepherd for their insightful and constructive comments. The authors would also like to thank Wanjin Li, Xing Wang, and Fan Fei for building the infrastructure and providing support for pDNS and Whois. Any opinions, findings, and conclusions expressed in this material are those of the author(s) and do not necessarily reflect the views of Palo Alto Networks.

References

[1] https://unit42.paloaltonetworks.com/dns-tunneling-in-the-wild-overview-of-oilrigs-dns-tunneling/. Accessed: 2022-12-09.

[2] https://www.darkreading.com/attacks-breaches/report-air-gapped-networks-vulnerable-dns-attacks. Accessed: 2022-12-09.

[3] https://crew-center.com/hundreds-crew-members-fired-using-free-internet-access-app-onboard. Accessed: 2022-12-09.

[4] https://unit42.paloaltonetworks.com/xhunt-campaign-backdoors/. Accessed: 2022-12-14.

[5] https://play.google.com/store/apps/details?id=com.in.troiddns&hl=en_US&gl=US. Accessed: 2022-11-06.

[6] https://play.google.com/store/apps/details?id=com.astrill.astrillvpn&hl=en_US&gl=US. Accessed: 2022-11-06.

[7] https://play.google.com/store/apps/details?id=com.one.vpnapp&hl=en_US&gl=US. Accessed: 2022-11-06.

[8] https://play.google.com/store/apps/details?id=com.hatunnel.plus&hl=en_US&gl=US. Accessed: 2022-11-06.

[9] https://github.com/m57/dnsteal. Accessed: 2022-11-06.

[10] https://github.com/iagox86/dnscat2. Accessed: 2022-11-06.

[11] https://web.archive.org/web/20181214053140/https://blog.autsoft.hu/a-confusing-dependency/. Accessed: 2022-12-14.

[12] https://github.com/bitquark/dnspop/tree/master. Accessed: 2022-10-14.

[13] https://www.mandiant.com/resources/blog/cyber-espionage-apt32/. Accessed: 2022-12-14.

[14] https://unit42.paloaltonetworks.com/dns-tunneling-in-the-wild-overview-of-oilrigs-dns-tunneling/. Accessed: 2022-12-14.

[15] https://www.socinvestigation.com/how-the-apt34-uses-saitama-backdoor-for-dns-tunnelling/. Accessed: 2022-12-14.

[16] https://www.sentinelone.com/labs/fin6-frameworkpos-point-of-sale-malware-analysis-internals/. Accessed: 2022-12-14.

[17] https://unit42.paloaltonetworks.com/unit42-new-wekby-attacks-use-dns-requests-as-command-and-control-mechanism/. Accessed: 2022-12-14.

[18] https://blog.talosintelligence.com/dnsmessenger/. Accessed: 2022-12-14.

[19] https://blogs.infoblox.com/cyber-threat-intelligence/decoy-dog-is-no-ordinary-pupy-distinguishing-malware-via-dns/. Accessed: 2023-09-15.

[20] https://czds.icann.org/. Accessed: 2022-12-14.

[21] https://en.wikipedia.org/wiki/Dynamic_DNS. Accessed: 2023-01-17.

[22] https://www.markmonitor.com/. Accessed: 2024-06-06.

[23] https://www.cscglobal.com/cscglobal/home/. Accessed: 2024-06-06.

[24] https://www.mit.edu/~ecprice/wordlist.10000. Accessed: 2023-09-11.

[25] https://www-personal.umich.edu/~jlawler/wordlist.html. Accessed: 2023-09-11.

[26] https://github.com/dwyl/english-words/blob/master/words.txt. Accessed: 2023-09-11.

[27] https://wordlists-cdn.assetnote.io/data/automated/httparchive_subdomains_2023_08_28.txt. Accessed: 2023-09-11.

[28] https://wordlists-cdn.assetnote.io/data/manual/best-dns-wordlist.txt. Accessed: 2023-09-11.

[29] https://www.dnsperf.com/#!dns-resolvers. Accessed: 2023-10-08.

[30] https://github.com/v2fly/v2ray-core. Accessed: 2024-05-11.

[31] http://vpnoverdns.com/home.html. Accessed: 2024-06-04.

[32] https://research.checkpoint.com/2023/tunnel-warfare-exposing-dns-tunneling-campaigns-using-generative-models-coinloader-case-study/. Accessed: 2024-5-29.

[33] https://7labs.io/technology-news/subdomain-leasing.html. Accessed: 2024-05-29.

[34] Crunchbase. https://data.crunchbase.com/docs/crunchbase-basic-getting-started. Accessed: 2022-12-15.

[35] Farsight dnsdb 2.0. https://www.farsightsecurity.com/solutions/dnsdb/.

[36] Newly registered domains: Malicious abuse by bad actors. https://unit42.paloaltonetworks.com/newly-registered-domains-malicious-abuse-by-bad-actors/, 2021.

[37] Any.run. https://any.run/, 2023.

[38] Coredns. https://coredns.io/, 2023.

[39] Dns security service privacy. https://www.paloaltonetworks.com/resources/datasheets/dns-security-service-privacy, 2023.

[40] Virustotal. https://www.virustotal.com, 2023.

[41] Whoisxml. https://www.whoisxmlapi.com, 2023.

[42] Amazon route 53. https://aws.amazon.com/route53/, 2024.

[43] Domain drop catching. https://en.wikipedia.org/wiki/Domain_drop_catching, 2024.

[44] Palo alto networks advanced dns security. https://www.paloaltonetworks.com/network-security/advanced-dns-security, 2024.

[45] Jawad Ahmed, Hassan Habibi Gharakheili, Qasim Raza, Craig Russell, and Vijay Sivaraman. Real-time detection of dns exfiltration and tunneling from enterprise networks. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2019.

[46] Nadler Asaf, Nadler Asaf, and Shabtai Asaf. Information-based heavy hitters for real-time dns data exfiltration detection. In *Annual Network and Distributed System Security Symposium (NDSS)*, 2024.

[47] Diogo Barradas, Nuno Santos, and Luís Rodrigues. DeltaShaper: Enabling unobservable censorship-resistant TCP tunneling over video-conferencing streams. *Privacy Enhancing Technologies*, 2017.

[48] Diogo Barradas, Nuno Santos, and Luís Rodrigues. Effective detection of multimedia protocol tunneling using machine learning. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[49] Diogo Barradas, Nuno Santos, Luís Rodrigues, and Vítor Nunes. Poking a hole in the wall: Efficient censorship-resistant internet communications by parasitizing on webrtc. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020.

[50] Kevin Borders and Atul Prakash. Quantifying information leaks in outbound web traffic. In *2009 30th IEEE Symposium on Security and Privacy*, 2009.

[51] Kenton Born and David Gustafson. Detecting dns tunnels using character frequency analysis. In *Proceedings of the 9th Annual Security Conference*, 2010.

[52] Patrick Butler, Kui Xu, and Danfeng (Daphne) Yao. Quantitatively analyzing stealthy communication channels. In *Applied Cryptography and Network Security*, 2011.

[53] Serdar Cabuk, Carla E. Brodley, and Clay Shields. Ip covert timing channels: Design and detection. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS)*, 2004.

[54] Thomas Callahan, Mark Allman, and Michael Rabinovich. On modern dns behavior and properties. *SIGCOMM Comput. Commun. Rev.*, 43(3), 2013.

[55] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. In *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI 06)*, 2006.

[56] Yizheng Chen, Manos Antonakakis, Roberto Perdisci, Yacin Nadji, David Dagon, and Wenke 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*, 2014.

[57] Steven Gianvecchio and Haining Wang. Detecting covert timing channels: An entropy-based approach. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS)*, 2007.

[58] Anestis Karasaridis, Kathleen Meier-Hellstern, and David Hoeflin. Detection of dns anomalies using flow data analysis. In *IEEE Globecom 2006*, 2006.

[59] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. In *26th Annual Network and Distributed System Security Symposium (NDSS)*, 2019.

[60] Shuai Li, Michael Schliep, and Nick Hopper. Facet: Streaming over videoconferencing for censorship circumvention. *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, 2014.

[61] Daiping Liu, Zhou Li, Kun Du, Haining Wang, Baojun Liu, and Haixin Duan. Don't let one rotten apple spoil the whole barrel: Towards automated detection of shadowed domains. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017.

[62] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining (ICDM)*, 2008.

[63] Chaoyi Lu, Baojun Liu, Yiming Zhang, Zhou Li, Fenglu Zhang, Haixin Duan, Ying Liu, Joann Qiongna Chen, Jinjin Liang, Zaifeng Zhang, Shuang Hao, and Min Yang. From whois to whowas: A large-scale measurement study of domain registration privacy under the gdpr. In *Proceedings 2021 Network and Distributed System Security Symposium*, 2021.

[64] Richard McPherson, Amir Houmansadr, and Vitaly Shmatikov. CovertCast: Using live streaming to evade internet censorship. *Privacy Enhancing Technologies*, 2016.

[65] I.S. Moskowitz and M.H. Kang. Covert channels-here to stay? In *Proceedings of COMPASS'94 - 1994 IEEE 9th Annual Conference on Computer Assurance*, 1994.

[66] Asaf Nadler, Avi Aminov, and Asaf Shabtai. Detection of malicious and low throughput data exfiltration over the dns protocol. *Computers Security*, 2019.

[67] Vern Paxson, Mihai Christodorescu, Mobin Javed, Josyula Rao, Reiner Sailer, Douglas Lee Schales, Mark Stoecklin, Kurt Thomas, Wietse Venema, and Nicholas Weaver. Practical comprehensive bounds on surreptitious communication over DNS. In *22nd USENIX Security Symposium (USENIX Security 13)*, 2013.

[68] David Plonka and Paul Barford. Context-aware clustering of dns query traffic. In *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement (IMC)*, 2008.

[69] Kyle Schomp, Tom Callahan, Michael Rabinovich, and Mark Allman. On measuring the client-side dns infrastructure. In *Proceedings of the 2013 Conference on Internet Measurement Conference (IMC'13)*, 2013.

[70] Dennis Tatang, Florian Quinkert, and Thorsten Holz. Below the radar: Spotting dns tunnels in newly observed hostnames in the wild. In *2019 APWG Symposium on Electronic Crime Research (eCrime)*, 2019.

Group	List of Cases
Attack	APT 32 [13], Wekby [17], OilRig [14], APT34 (Saitama Backdoor) [15], Decoy Dog [19], FrameworkPOS [16], AndroidAudioRecorder [11]
VPN Service	TunnelGuru, SlowDNS, TunnelCat, VPN-over-DNS, Your-Freedom, TLS Tunnel VPN, HA Tunnel Plus, Astrill VPN, MinaProNet VPN
Public Tool	CobaltStrike, dns2tcp, tcp_over_dns, dnsbin, flightsim, Sliver

TABLE 7: Full list of malicious tunneling cases.

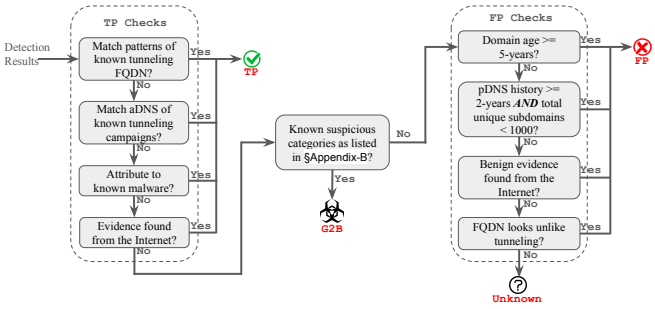


Figure 13: Workflow to verify detection results.

Appendix A. Full List of Malicious Tunneling Cases

Table 7 lists names of the attack campaigns, VPN services and public tools whose DNS tunneling traffic is collected for our evaluation.

Appendix B. Workflow of Detection Result Verification

Figure 13 shows the workflow to verify a detected domain.

TP. To decide if a detection is a TP, we first check if the FQDNs or aDNS servers match any known DNS tunneling campaigns. Our industry partner has collected FQDN patterns and aDNS servers for tens of DNS tunneling tools and C2, all of which have been proven to be accurate in practice. For the remaining cases, we cross check with the database of our industry partner's dynamic malware analysis system and try to attribute to known malware that uses DNS tunneling as C2. Finally, we manually search for the evidence from the Internet by searching the root domains in Google Search.

G2B. For domains that cannot be verified as TP, we check with VirusTotal and our industry partner's security products to decide if they are malicious or fall into other good-to-block categories including C2, phishing, scamming, adult, gambling, parked, and proxy avoidance. Enterprise networks often block these categories by default. Therefore, even though these domains are not tunneling, it is usually good to block them. Actually, our further analysis shows that many of these domains can also be counted as tunneling because they usually transmit user/host identification information (e.g., IP and Mac address) in DNS queries. Leaking

such information can pose security threats to enterprise networks.

FP. After G2B check, we follow four steps to confirm FP cases. First, a domain with long history is usually benign. Thus, we simply classify the domains that are registered five years ago and still active as FP. Similarly, if a root domain has been active since two years ago and has less than 1000 unique FQDNs as subdomains, it is usually not DNS tunneling. The rationale is that DNS tunneling activity usually results in thousands or even millions of unique FQDNs. Next, we search the root domains in Google Search and manually decide if a domain is used for legitimate purposes. If so, the detected domain is considered as a FP. In the last step, we rely on the security analysts from our industry partner to manually decide if a detected FQDN looks like tunneling. If not, we classify a detection as a FP.

Unknown. Finally, the detections that cannot be classified as TP, G2B or FP are referred to as Unknown.

FN. To evaluate the candidate FNs, we employ two different strategies for the two cases. *i)* For the FQDNs of known tunneling root domains, we analyze the FQDN format to decide if a FQDN carries exfiltrated data. This is usually quite straightforward since the data component in the FQDNs is a string of random characters. *ii)* For the queries whose anomaly scores are in range $(-0.258, -0.25]$, we follow a similar workflow as Figure 13. Given a query, if any TP check returns true, the query is a FN; if any FP check returns true, the query is a *true negative (TN)*; otherwise, the query is referred to as *potential FN*.

Appendix C. Meta-Review

The following meta-review was prepared by the program committee for the 2025 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

C.1. Summary

This paper designs, implements, and discusses the deployment of TunTight, a ML-based system for detecting DNS-based tunneling. TunTight is notable for being the first system to block tunnel-based DNS queries on the first query. TunTight uses a prefiltering step to identify obviously benign DNS queries, which is the common class, and then uses an Isolation Forest-based anomaly detection mechanism built on several features identifying key aspects of tunneled traffic. The system was evaluated using both labeled datasets and a large-scale deployment in a real-world setting, integrated into a commercial enterprise firewall product. These experiments demonstrate that TunTight achieves high accuracy in classifying malicious DNS tunnels, including the detection of over 350 confirmed tunnels in the real-world deployment.

C.2. Scientific Contributions

- Creates a New Tool to Enable Future Science
- Addresses a Long-Known Issue
- Provides a Valuable Step Forward in an Established Field

C.3. Reasons for Acceptance

1) The paper creates a new tool to enable future science. Although source code for TunTight is not available, the approaches and features demonstrated by the work can be reproduced in future work. As a result, TunTight provides a footprint for improving the fidelity of measurement studies on tunneling, as well as future work in anti-tunneling measures.
2) The paper addresses a long-known issue. Tunneling over DNS is a well-known problem in computer security, but by performing a focused evaluation on this problem in the enterprise domain and demonstrating the weaknesses of prior approaches, the work improves performance in this space.
3) The paper provides a valuable step forward in an established field. Although tunneling is a known issue, the work demonstrates that one-shot identification and filtering of tunneling is poossible in certain instances. Further, achieving this while minimizing false positives shows that such an approach is feasible in practice.

C.4. Noteworthy Concerns

1) The work identifies facets of DNS covert channels as-deployed. However, these features may be artifacts of current tunneling implementations and not requisite for creating such channels. Several presented features have straightforward bypass techniques by construction.
2) As an ML-based system, there is risk that the work overfits to training data, which does not reflect the imbalance of tunneled traffic in practice. The paper evaluation is lacking sufficient characterization of false negatives to eliminate this possibility.