Practical Comprehensive Bounds on Surreptitious Communication Over DNS

Vern Paxson, *University of California, Berkeley, and International Computer Science Institute;* Mihai Christodorescu, *Qualcomm Research;* Mobin Javed, *University of California, Berkeley;* Josyula Rao, Reiner Sailer, Douglas Lee Schales, and Marc Ph. Stoecklin, *IBM Research;* Kurt Thomas, *University of California, Berkeley;* Wietse Venema, *IBM Research;* Nicholas Weaver, *International Computer Science Institute and University of California, San Diego*

This paper is included in the Proceedings of the 22nd USENIX Security Symposium.

August 14–16, 2013 • Washington, D.C., USA

ISBN 978-1-931971-03-4

Open access to the Proceedings of the 22nd USENIX Security Symposium is sponsored by USENIX

Practical Comprehensive Bounds on Surreptitious Communication Over DNS

Vern Paxson[◇*] Mihai Christodorescu[†] Mobin Javed[◇] Josyula Rao[‡] Reiner Sailer[‡]
Douglas Schales[‡] Marc Ph. Stoecklin[‡] Kurt Thomas[◇] Wietse Venema[‡] Nicholas Weaver[*§]
[◇]*UC Berkeley* [*]*ICSI* [†]*Qualcomm Research* [‡]*IBM Research* [§]*UC San Diego*

Abstract

DNS queries represent one of the most common forms of network traffic, and likely the least blocked by sites. As such, DNS provides a highly attractive channel for attackers who wish to communicate surreptitiously across a network perimeter, and indeed a variety of tunneling toolkits exist [7, 10, 13–15]. We develop a novel measurement procedure that fundamentally limits the amount of information that a domain can receive surreptitiously through DNS queries to an upper bound specified by a site's security policy, with the exact setting representing a tradeoff between the scope of potential leakage versus the quantity of possible detections that a site's analysts must investigate.

Rooted in lossless compression, our measurement procedure is free from false negatives. For example, we address conventional tunnels that embed the payload in the query names, tunnels that repeatedly query a fixed alphabet of domain names or varying query types, tunnels that embed information in query timing, and communication that employs combinations of these. In an analysis of 230 billion lookups from real production networks, our procedure detected 59 confirmed tunnels. For the enterprise datasets with lookups by individual clients, detecting surreptitious communication that exceeds 4 kB/day imposes an average analyst burden of 1–2 investigations/week.

1 Introduction

Some of the most serious security threats that enterprises face concern the potential use of surreptitious communication (Figure 1). One such scenario takes the form of *exfiltration*, when an attacker with internal access aims to transmit documents or other substantive data out of the enterprise to a remote location [4]. Another scenario arises in the context of *interactive remote access*: an attacker who has patiently compromised a local system subsequently interacts with it over the network in order to assay the information it holds and employ it as an internal stepping stone for further probing of the enterprise.

DNS plays a pervasive role in Internet communication; indeed, the vast majority of *any* Internet communication ultimately begins with DNS queries. Even sites that are highly security-conscious will find that they still must allow internal clients to issue DNS queries and receive the replies. Unless sites can restrict their systems to only intra-enterprise communication, some of these queries will necessarily reach external systems, giving attackers the opportunity to piggyback their actual communication over seemingly benign DNS traffic. Thus, DNS provides a highly-attractive target for attackers seeking a means of surreptitious communication.

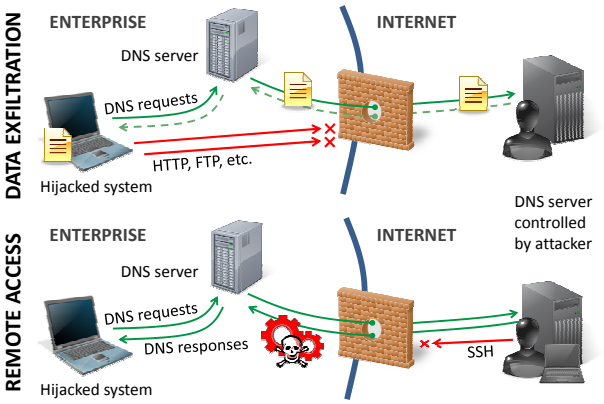

Figure 1: Two examples of surreptitious communication via DNS tunnels through perimeter firewalls.

We note that such communication fundamentally cannot be detected at the level of individual DNS queries. For example, an attacker could exfiltrate only one bit of information per day by having a local system under their control each day issue a single query for either www.attacker.com or mail.attacker.com, where the label used (www or mail) conveys either a 0 bit or a 1 bit.[1] It will prove intractable for a site's security analysts (or any detection tool) to tell that such requests reflect adversarial activity, absent a great deal of additional information.

In this work we develop a principled means—rooted in assessments of information-theoretic entropy and free from false negatives—by which sites can analyze their

[1] We assume that the attacker controls the attacker.com DNS zone.

DNS activity and detect the presence of surreptitious communication whose volume exceeds a configurable bound. Simpler metrics, such as volume of DNS traffic, are not useful to distinguish tunnels from normal query traffic, because large-scale traffic naturally exhibits a high degree of diversity (§ 5.2). Approaches that focus on specific syntactical patterns [29] will miss communication with different encodings. Our configurable bound on the volume of surreptitious communication over DNS allows sites to trade off analysis burden (detections requiring investigation) versus assurance that such communication does not exceed a considerably low level.

We formulate this detection problem as having three main components. The first concerns constructing a sound, fairly tight estimate of the amount of information potentially encoded in a stream of DNS queries. Here we need to comprehensively identify all potential *information vectors*, i.e., aspects of DNS queries that can encode information. The second regards ensuring that we can compute such estimates with reasonable efficiency in the face of very high volumes of DNS activity (tens of millions of lookups per day). Finally, we need to assess to what degree benign DNS query streams encode significant amounts of information, and formulate effective ways of minimizing the burden that such benign activity imposes on a site's security assessment process.

Thus, we conceptualize our overall goal as providing a site's security analysts with a high-quality, tractable set of domains for which the corresponding DNS lookups potentially reflect surreptitious communication. We view it as acceptable that the analyst then needs to conduct a manual assessment to determine which of the candidates actually reflects a problem, *provided* that we keep the set small and the process of eliminating a benign candidate does not require much attention.

This work makes the following contributions:

- We introduce a principled means of detecting the presence of surreptitious communication over DNS, parameterized by a (configurable) bound on the amount of information conveyed.
- Our approach is *comprehensive* because we root our estimates of information conveyed in DNS lookups in lossless compression of entire query streams.
- We perform an in-depth empirical analysis of mostly-benign DNS traffic on an extensive set of traces comprising 230 billion queries observed across a variety of sites. For enterprise datasets with lookups by individual clients, we find that a bound of 4 kB/day per client and registered domain name imposes an operationally viable analysis burden. Thus, we argue that our procedure proves practical for real-world operational use.

After a summary of our information measurement procedure, we define the threat model in § 3. In § 4 we present the extensive datasets used in our study. We discuss information vectors potentially present in DNS queries and ways to estimate their volume in § 5, and explore implementation issues, including filtering techniques for reducing the resources required, in § 6. We evaluate the efficacy of our procedure in § 7, present a real-time detector in § 8, discuss findings, limitations, and future work in § 9, and review prior work in § 10.

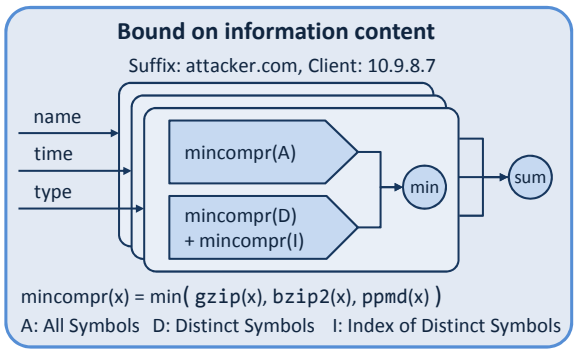

Figure 2: The information measurement procedure, summarized in § 2. Figure 5 shows the full detection procedure.

2 Summary of the information measurement procedure

As explained in § 6.3, we analyze DNS queries per client and per registered domain name. For example, we aggregate queries with names ending in $site_1$.com, $site_2$.co.uk, and so on. We also aggregate PTR queries, but ignore them here for clarity.

We measure the information in query name, time and record-type sequences separately (§ 5.3). For example, we transform a sequence of query names A to a sequence of indices I into a table with distinct names D, and then compress I and D with *gzip*. The size of the output then gives us a measurement of the information in the input sequence.

The key insight is that *we will never under-estimate the information in a query sequence as long as the transformation and compression are reversible,* i.e., we can recover the original input sequence. Taking advantage of this insight, we subject each query attribute sequence to multiple (transformation, compressor) alternatives and use the minimal result as the tightest (upper) bound.

Figure 2 illustrates our measurement procedure. For each client and registered domain we compress both the original and transformed query name sequences with *gzip*, *bzip2* and *ppmd* [23], and take the size of the smallest output. We apply the same procedure to the record-type sequences and 32-bit inter-query arrival time distances, and from these compute a combined score.

Site	*Features*	*Vantage point*	*Notes*	*Time span*	*Daily statistics: Average (Daily peak)*		
					Clients	*Total lookups*	*Distinct lookups*
INDLAB	L,N,Q,T	I		1,212 d	10k (16k)	47M (164M)	310k (2.4M)
LBL	N,Q	I		2,776 d	6.8k (11k)	28M (154M)	867k (2.2M)
NERSC	N,Q	I		1,642 d	1.3k (3.3k)	9M (59M)	44k (114k)
UCB	N,Q,T	E	**a**	45 d	2.1k (5.1k)	38M (52M)	3.3M (4.4M)
CHINA	N,Q,T	I/E	**b**	5 d	61k (101k)	13.9M (15.7M)	468k (670k)
SIE	N	A	**c**, **d**	53 d	123* (123*)	1.45B (1.84B)	110M (129M)

Table 1: Summary of data sources. The available features are: `0x20`-encoding [27] (C), caching lifetime derived from reply time-to-live (L), query name (N), query type (Q), and timing (T). Sensor vantage points are: aggregated across multiple sites (A), external to site (E), lookups associated with individual clients as seen at internal name servers (I), a mixture of these last two (I/E).
a The raw UCB dataset includes resolvers that employ `0x20`-encoding [27] as well as a single system conducting high-volume DNS lookups for research purposes. We preprocessed this dataset by removing lookups from the research system (totaling more than 250M) and downcasing lookups from `0x20`-resolvers (cf. § 5.3). (Note that the dataset has 3 days with only partial information.)
b This dataset's first day starts at 7AM local time rather than midnight. The other days are complete.
c "Clients" in the SIE dataset instead reflect site resolvers, each with potentially thousands of actual clients.
d As discussed later, we omit from our evaluation the `PTR` reverse lookups in this data, which comprise about 10% of the lookups.

3 Threat Model

Our basic model assumes that the attacker controls both a local system and a remote name server. The local system will communicate with the remote name server solely by issuing lookups for DNS names that the site's resolver will ultimately send to the attacker's name server. The attacker inspects both the content of these queries (i.e., the names and the associated query type, such as `TXT` or `AAAA`) and their arrival timing.

We further assume that the internal system under the attacker's control makes standard queries, either because the site's firewalling requires internal systems to use the site's own resolvers, or because non-standard queries made directly to the public Internet could expose the communication's anomalous nature.

For the investigation we develop in this work, we focus on communication outbound from local systems. (We briefly discuss inbound communication encoded in DNS replies in § 9.) We view the outbound direction as the most apt when concerned about exfiltration threats. In addition, for the interactive communication scenario, the outbound direction corresponds to the replies generated by a local login server in response to keystrokes sent by a remote client. The outbound traffic volume to the login client is typically 20 times larger than the incoming traffic [21], making DNS queries embedding outbound traffic the larger target for that scenario, too.

We do not consider here communication that an attacker spreads across multiple remote domains or multiple remote name servers (such as attacker$_1$.com, ..., attacker$_n$.com), nor spread thinly across multiple local clients. We discuss these and other evasion issues in § 9.

4 The Data

For our analysis we draw upon datasets that together comprise 230 billion queries. The data was collected at multiple locations across the US and China, with vantage points ranging from internal DNS servers to network perimeters. We summarize each dataset and its daily traffic statistics in Table 1.

INDLAB: an industrial research laboratory. Collected with a network sniffer near an internal DNS server, this dataset contains queries from internal clients, the reply time-to-live, and microsecond-resolution time stamps.

LBL: a national research laboratory. This dataset contains DNS queries from local clients received by several internal DNS servers. Covering a time span of 7.5 years, this is the largest data set in our analysis.

NERSC: a super-computer center. The dataset contains queries from local clients to the site's DNS servers.

UCB: a university campus. This data was collected on a perimeter network, providing an aggregate view of (outbound) DNS query traffic. This site includes servers that use `0x20` encoding [27], which nearly doubles the number of distinct lookup names.

CHINA: a caching server for several university networks in China, with visibility of individual client IP addresses.

SIE: the Security Information Exchange of the Internet Software Consortium [24]. In this collaboration of infrastructure providers, law enforcement, security companies and researchers, participants[2] mirror their DNS reply traffic from name servers across the Internet. (Note that each reply contains a copy of the query.)

With a combined average of 1.5 billion replies a day, SIE has by far the highest data rate in our collection.

[2]Heavily dominated by a single large U.S. ISP.

However, we note that we use it as a means of assessing to what degree our detection procedure indeed can find actual instances of surreptitious communication over DNS; we do *not* claim our procedure is tenable for actual operational use in this environment, which is hugely aggregated across (likely) millions of actual clients.

5 Establishing Communication Bounds

In this section we develop a principled approach for bounding the amount of information possibly conveyed by local systems to remote name servers. The next section then presents a number of filtering steps that reduce the resources required for detecting communication that exceeds these bounds.

5.1 Information Vectors

We first frame the basic communication mechanisms an attacker could employ. In general terms, we consider an attacker who wishes to communicate a significant quantity of information by sending DNS queries to a remote domain (say D.com) whose name server(s) the attacker controls. Such queries provide a number of *information vectors* that the attacker can exploit to surreptitiously embed data within the stream of queries.

We note that attackers can potentially employ multiple vectors at the same time. We emphasize that our detection scheme does *not* presume use of particular encodings for a given vector; the encodings we give here are just meant to illustrate the possibilities.

Query name-content vector. A conceptually straightforward way to embed data is for the attacker to devise a data encoding that conforms with the requirements imposed on DNS labels, limiting each to no more than 63 bytes in length, and complete DNS names to no more than 255 bytes [18]. For example, one could use Base-64 encoded data strings as such as VVNFQwo.D.com.

To our knowledge, all available tunneling-over-DNS tools reflect this style of approach.

Query name-codebook vector. Rather than using each DNS query to reflect a message many bytes long, attackers can encode messages using a fixed alphabet of symbols and then transmit those symbols one at a time using a series of queries. For example, to convey the bitstring 00101111 one bit at a time, a client could issue the queries: z.D.com, z.D.com, o.D.com, z.D.com, o.D.com, o.D.com, o.D.com, o.D.com. They could of course also use larger alphabets to obtain greater efficiency.

Encodings using this vector will in general generate many more lookups of the same names over time compared to those using the query name-content vector.

Query type vector. Along with the query name, clients include in their requests the *type* of DNS Resource Record they wish to resolve, such as `PTR` for reverse-IP-address-to-hostname mappings, or `AAAA` to look up IPv6 addresses. Attackers can encode a modest amount of information per query using this 16-bit field.

Query timing vector. A more subtle information vector exists in the specific timing of queries. For example, if the attacker can resolve the arrival times of queries to 1 sec precision, then the attacker can use the number of seconds between successive queries as a means of conveying information.[3]

A key issue for timing vectors concerns clock precision. With an extremely precise clock (and sufficiently low jitter), intervals between queries can convey several bytes of information without requiring very large inter-query delays. For example, transmitting one query every second using a clock with 1 msec precision can convey $\lg 10^3$ bits per query, totaling more than 108 KB per day.

Other information vectors. Inspecting the DNS query format reveals several additional fields possibly available for communicating information: query identifiers, a number of flags, options, the query count, and the 16-bit address class field included in each query. We argue that none of these provide a reliable end-to-end information vector for an attacker, given the assumption in our threat model that the attacker's client must relay its queries via a site's standard (non-cooperative) resolver. Such relayed queries will not preserve query identifiers. The flags either do not survive the relaying process (e.g., Recursion Desired) or will appear highly anomalous if they vary (e.g., requesting DNSSEC validation), and likewise DNS options (`EDNS0`) do not survive relaying, as unknown options return an error [26], and the current options themselves are generally implemented on a hop-by-hop basis. Similarly, query counts other than 1 would appear highly anomalous and likely fail to actually propagate through the site's name server. Likewise, use of any address class value other than `IN` (*Internet*) would be readily detectable as anomalous.

5.2 Challenge: Diversities Seen in Practice

A natural starting point when attempting to detect surreptitious DNS communication is to posit that the encodings used for the communication will stand out as strikingly different than typical DNS activity. If so, we can target the nature of the encoding for our detection.

What we find, however, is that while potential encodings may differ from *typical* DNS activity, they do not sufficiently stand out from the *diverse range* of benign activity. When we monitor at a large scale—such as analyzing the traffic from the 1000s of systems in an enterprise—we observe a striking degree of fairly extreme forms of DNS lookups.

[3] In addition, the specific query received after the given interval could also convey additional information using one of the previously described vectors.

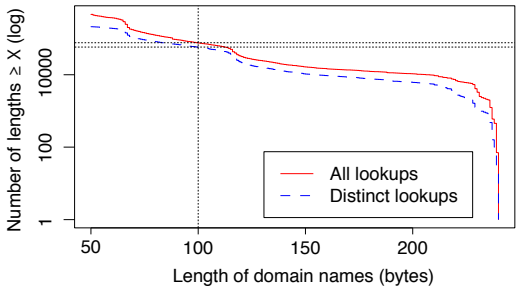

Figure 3: Distribution of the lengths of all individual (solid) or all distinct (dashed) domain name prefixes queried during a sample day of data from LBL. The horizontal lines mark that 76K (all) and 58K (distinct) lookups were $\geq$ 100 bytes. Lengths do not include the *registered domain* targeted by the lookup. Note that the plot shows the upper 1% of all queries, but the upper 18% of all distinct queries.

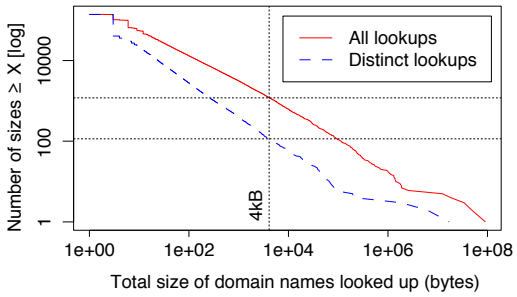

Figure 4: Distribution of the total length of domain name prefixes sent to different registered domains, computed as the sum of all names (solid) or distinct names (dashed). The horizontal lines mark that 1,186 registered domains received $\geq$ 4kB of names, while 114 received $\geq$ 4kB of distinct names.

To illustrate, we consider DNS activity observed on a sample day in 2011 at LBL. It includes 35M queries issued from 9.4k hosts. These queries in total span 1.2M distinct names, and if we discard the first component of each name, 620K distinct subdomains. These subdomains are themselves rooted in 137K distinct *registered domains* (i.e., one level under com or co.uk).

One natural question concerns the frequency with which operational DNS traffic exhibits peculiarly long query names, since many natural encodings for surreptitious communication will aim to pack as much information into each query as possible. Figure 3 shows the distribution of domain name prefix lengths $\geq$ 50 bytes (i.e., characters) looked up in our sample day. We see that queries with names even larger than 100 bytes occur routinely: while rare in a relative sense (only 0.2% of query names are this large), 76,523 such queries occurred on that day. Restricting our analysis to distinct names (dashed line) does not appreciably lower this prevalence.

For concreteness, here are some examples of what such queries look like:

```
JohnsonHouse\032Officejet....sonhouse1.members.mac.com
www.10.1.2.3.static.becau....orant.edu.za.research.edu
awyvrvcataaaegdid5tmr7ete....ilu.license.crashplan.com
g63uar2ejiq5tlrkg3zezf2fk....emc6pi88tz.er.spotify.com
5.1o19sr00ors95qo0p73415p....7rn92.i.02.s.sophosxl.net
```

where we have elided between 63 characters (first example) and 197 characters (last example). See Appendix A for the complete names.

Thus, simply attempting to detect queries that include unusually large names does not appear viable. Similarly, the examples above illustrate that benign traffic already includes DNS queries that use opaque encodings, so we do not see a promising angle to pursue with regard to recognizing surreptitious communication due to the syntax of its encoding.

A different perspective we might pursue is that if only a small number of remote name servers receive the bulk of the site's queries, then we might be able to explicitly examine each such set of traffic. Figure 3, however, shows that large volumes of queries are spread across numerous remote name servers. The plot shows how many registered domains received a given total size of queries (the sum of the lengths of all of the prefixes sent to that domain). If we restrict our view to the total size of *distinct* queries that a registered domain receives, more than 100 registered domains each received in excess of 4kB of query names. If we include prefixes for repeated lookups, the figure is ten times higher.

Surprising query diversity also manifests in other dimensions. For example, surreptitious communication that leverages the transmission of repeated queries in a codebook-like fashion requires using low-TTL answers to prevent local caching from suppressing queries. However, we find that in benign traffic, low TTLs are not unusual: in a day of queries for external names that we examined, a little under 1% of the answers had TTLs of 0 or 1, and 38% are $\leq$ 60 sec. We also find instances of large numbers of repeated queries arising from benign activity such as misconfigurations and failures.

In summary, the variations we find operationally are surprisingly rich—enough so to illustrate that our problem domain will not lend itself to conceptually simple approaches due to the innate diversity that benign DNS lookups manifest when observed *at scale*.

To illustrate the difficulty, we evaluated the performance of a naive detector that simply sums up the volume of lookups sent to each domain, alerting on any client sending the domain more than 4,096 bytes in one day. In steady-state (using the same methodology as in § 7, including the *Identified Domain List* discussed below), this detector produces 200x more alerts than our actual procedure. If we alter the detector to only sum the volume of distinct lookups, we still must abide 5x more

alerts (and lose the ability to detect codebook-style encodings). We emphasize that because our actual procedure has no false negatives, all of these additional alerts represent false positives.

5.3 Establishing Accurate Bounds on Query Stream Information Content

Given that simple heuristic detection approaches will not suffice due to the innate diversity of DNS queries, we now pursue developing principled, direct assessments of upper bounds on the volume of data a given client potentially transmits in its queries.

A key observation is that—provided we do not *underestimate* the potential data volume—we can avoid any false negatives; our procedure will indeed identify any actual surreptitious communication of a given size over DNS. Given this tenet, the art then becomes formulating a sufficiently *tight* upper bound so we do not erroneously flag lookups from a client to a given domain as reflecting a significantly larger volume of information than actually transmitted.

We can obtain tight bounds by quantifying the size of carefully chosen *representations* of a client's query stream. If we obtain these representations in a *lossless* fashion (i.e., we can recover the original query stream from the representation), then the bound is necessarily conservative in the sense of never underestimating the true information content of the queries. At the same time, the representation must be compact enough to reduce any redundancy from the query stream as efficiently as possible in order to obtain a tight estimate. Thus, the task we face is to determine a representation of the query stream that efficiently captures its elements, but does so in a reversible fashion. In general, we seek forms of *lossless compression* with high compression ratios.

Conceptually, the **heart of our approach** is to take encoded query streams and feed them to compression algorithms such as *gzip*, using the size of the compressor's output as our estimate. While simple in abstract terms, pursuing this effectively requires (1) care in encoding the streams to try to achieve as tight a bound as possible, and (2) structuring the analysis procedure to execute efficiently given a huge volume of data to process.

For the rest of this section, we address the first of these issues. We then discuss execution efficiency in § 6.

Character casing. The first question regarding encoding query streams concerns the most obvious source of variation, namely the particular names used in the queries. For these, one significant encoding issue concerns *casing*. While the DNS specification states that names are treated in a case-insensitive manner, in practice resolvers tend to forward along names with whatever casing a client employs when issuing the query to the resolver.

Together, these considerations mean that, for example, a query for foo.D.com and FoO.D.COM will both arrive at the same D.com name server, with the casing of the full query name preserved. Accordingly, we must downcase query name suffixes in order to correctly group them together (i.e., to account for the fact that the same name server will receive them), but preserve casing in terms of computing information content, since indeed the attacker can extract one bit of information per letter in a query (including the domain itself) depending on its casing.

0x20-encoding. Preserving casing in queries can raise a difficulty for formulating tight bounds on information content due to the presence of `0x20`-encoding [27], which seeks to artificially increase the entropy in DNS queries to thwart some forms of blind-spoofing attacks. While the presence of arbitrary casing due to use of `0x20`-encoding does indeed reflect an increase in the actual information content of a stream of queries, this particular source of variation is not of use to the attacker; they cannot in fact extract information from it.

We found that unless we take care, our UCB dataset, which includes queries from a number of resolvers that employ `0x20`-encoding, will indeed suffer from significant overestimates of query stream information content. The presence of such resolvers however means that their clients cannot exploit casing as an information vector, since the resolver will destroy the client's original casing. Accordingly, we developed a robust procedure (details omitted due to limited space) for identifying queries emanating from resolvers that employ `0x20`-encoding. For those query sources we downcase the queries to accurately reflect that casing does not provide any information.

This procedure identified 205 clients in the UCB dataset. Other than those clients, we left casing intact.

Employing codepoints. General compressors such as *gzip* do not make any assumptions about the particular structure of the data they process. However, our particular problem domain has certain characteristics that can improve the compression process if we can arrange to leverage them. In particular, we know that DNS query streams often repeat at the granularity of entire queries. We can expose this behavior to a general compressor by constructing *codepoints*, as follows. We preprocess a given client's query stream, replacing each distinct query with a small integer reflecting an index into a table that enumerates the distinct names. For example, this would reduce a query stream of foo.X.com, bar.X.com, bar.X.com, foo.X.com, bar.X.com to the stream 1, 2, 2, 1, 2, *plus* a dictionary that maps 1 to foo.X.com and 2 to bar.X.com. The particular encoding we use employs 24-bit integers (we take care in our information-content estimation to include the dictionary size).

Representing query types. For datasets that include query types, we construct a separate, parallel compres-

sion stream for processing the corresponding 16-bit values, i.e., we do not intermingle the query types with the query names.

Representing timing. Individual query timings offer only quite limited information content. Thus, for an attacker to make effective use of timing, they will need to send a large number of queries. This means that we likely will benefit from capturing not absolute timestamps but intervals between queries. We compute such intervals as 32-bit integers representing multiples of $\mathfrak{R}$, our assumed lower bound on the timing resolution the attacker can achieve. Again we construct a separate, parallel compression stream for processing these.

Clearly, the value of $\mathfrak{R}$ can significantly affect the amount of information the attacker can extract from the timing of queries; but $\mathfrak{R}$ will be fundamentally limited by network jitter. To formulate a defensible value of $\mathfrak{R}$, we asked the authors of [17] regarding what sort of timing variation their measurements found for end systems conducting DNS queries. Using measurements from about a quarter million distinct IP addresses, they computed the maximum timing difference seen for each client in a set of 10 DNS queries it issued. The median value of this difference across all of the clients was 32 msec. Only a quarter of the clients had a difference under 10 msec. Accordingly, for our study we have set $\mathfrak{R}$ to 10 msec.

Constructing unified estimates. As described above, we separately process the query names, types, and timing. Formulating a final estimated bound on a query stream's information content then is simply a matter of adding the three corresponding estimates. We note, though, that by tracking each separately, we can identify which one contributes the most significantly (per Figure 6 below).

Bakeoffs. Finally, as outlined above we have several potential choices to make in formulating our upper-bound information estimates: which compressor should we employ? Should we use codepoints or allow the compressor to operate without them (thus not imposing the size of the dictionary)? We note that we do not in fact have to make particular choices regarding these issues; we can *try each option separately*, and then simply choose the one that happens to perform best (generates the lowest information estimates) in a given context. Such "bakeoffs" are feasible since we employ lossless techniques to construct our estimates; we know that each estimate is sound, and thus the lowest of a set is indeed the tightest upper bound we can obtain.

The drawback with trying multiple approaches, of course, is that it requires additional computation. In the next section we turn to how to minimize the computation we must employ to formulate our estimates.

6 Implementation

The previous section described our approach to developing an accurate bounds on the amount of information conveyed using DNS queries to a given domain's name server(s). Computing these estimates and acting upon their corresponding detections, however, raises a number of issues with regards to reducing the resources required for employing this approach.

In this section we discuss practical issues that arise when implementing our detection approach. One significant set of these concern *filtering*: either restricting the DNS queries we examine in order to conserve computing (or memory) resources, or reducing the burden that our detection imposes on a site's security analysts. The key property of these filtering stages is their efficacy in concert, which is crucial for the scalability of our approach. Figure 5 shows the different stages of processing in our detection procedure and how they pare down in several steps the volume of both the queries that we must examine and the number of domain name suffixes to consider.

We describe our detection procedure as implemented for off-line analysis here, and discuss our experiences with a real-time detector in § 8.

6.1 Cached Query Filter

A query from a DNS client system cannot exfiltrate information unless it is forwarded by the recursive resolver. Thus a highly useful optimization for the internal vantage point (as discussed in § 4) is to model the recursive resolver's cache and not consider any query where the resolver obtained the result from its cache.

We can accomplish this by observing the replies with the TTL field. We maintain a shadow cache based on the query attributes (contained in the reply) and the reply TTL values, and do not consider later queries until their information expires from the shadow cache.

The result of this filtering is to eliminate the disadvantage of the internal vantage point, as this filter ensures that later stages only process uncached requests. With the IndLab dataset, this reduces the number of detections by about 2x for the timing vector, and about 10% for query names. Unfortunately not all of our datasets support this filtering.

6.2 Uninteresting Query Filter

We remove lookups that target domain names within the local organization itself, or within closely-related organizations. Due to their relatively high volume, we find that such lookups can result in a large number of detections, but the likelihood that someone will actually use a DNS tunnel between such domains will be negligible. Likewise, we remove lookups of PTR (address-to-name) records for local and reserved network address ranges.

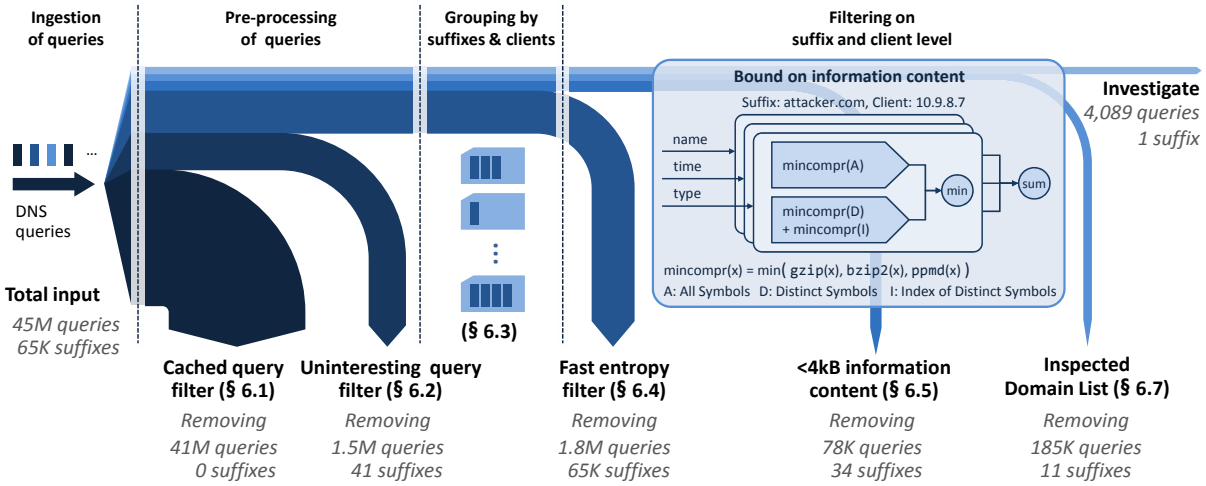

Figure 5: The full detection procedure. The numbers (grey) reflect a day at the INDLAB network for which the detection procedure flagged a new domain name (a relatively rare event).

Finally, we exclude names without a valid global top-level domain. This eliminates numerous queries from systems that are misconfigured or confused.

6.3 Grouping by Suffix and Client

In this stage of our detection procedure, we compute statistics per (lookup name suffix, client)-pair that will serve as input to the lightweight filter described in § 6.4.

Due to the voluminous nature of our data, we aggregate these statistics at the level of registered domain names (e.g., one level under com or co.uk). With IPv4 PTR lookups we aggregate at two and three labels under in-addr.arpa (corresponding with /16 or /24 network ranges), and with IPv6 PTR lookups we aggregate at 12 labels under ip6.arpa (corresponding with /48 network ranges). The reasoning behind these choices is that shorter PTR suffixes will in general represent large blocks that are parents to multiple organizations; thus, the presence of tunneling associated with such suffixes would require compromise of a highly sensitive infrastructure system. In our results for PTR lookups we find no indications of surreptitious communication.

We then compute for each query suffix and client the numbers of unique and distinct lookup names including that suffix, as well as the combined length of those lookup names. We group suffixes in a case-insensitive manner, but count as distinct any lookup names that differ only in case (cf. § 5.3).

6.4 Fast Filtering of Non-Tunnel Traffic

The very high volume of DNS queries means we can obtain significant benefit from considering additional measures for pre-filtering the traffic before we compute the principled bounds described in § 5. For each domain suffix, we use computationally lightweight metrics that overestimate the information content present in the information vectors described in § 5.1. We then compare the sum of these metrics across all information vectors against a *minimum-information content threshold*, $\mathfrak{I}$. If the sum total (guaranteed to not underestimate) lies below the threshold, the traffic for the corresponding domain suffix cannot represent communication of interest. This approach allows us to short-circuit the detection process and eliminate early on numerous domain suffixes.

Fast filter for the query name vector. We consider the following quantities from a sequence of lookups made by some host during one day: the total number of lookups L, the number of distinct query names D_{name} in those lookups, and the total number of bytes C_{name} in those distinct query names. We remark that we can determine all three quantities with minimal computational and memory overhead.

Query name tunnels encode information in terms of the characters and the repetition patterns of the names looked up. Each character in a name may convey up to 1 byte of information, contributing up to C_{name} bytes in total. According to Shannon's law, the number of bits conveyed per lookup amounts to at most $\log_2 D_{name}$. Therefore the combined upper bound on information conveyed in bytes by such a tunnel amounts to:

$$I_{\text{name}} = C_{name} + L \cdot \frac{\log_2 D_{name}}{8}$$

Fast filter for the query type vector. We filter the query type vector similarly. Again, we consider a sequence of DNS lookups with a given suffix made by some host during one day. If we use D_{type} to denote the number of distinct query types in those lookups and C_{type} the total number of bytes in those distinct query types, we have:

$$I_{\text{type}} = C_{type} + L \cdot \frac{\log_2 D_{type}}{8}$$

Fast filter for the query timing vector. The timing vector is more complicated because we need to discretize the time information and create symbols representing the encoded data as it appears in the timing vector. We parametrize this process by the *time resolution* $\Re$ that the network environment affords to the attacker.

Intuitively, for a given number of lookups L observed over a day, the amount of potential information encoded in time is maximal when the number of distinct inter-arrival times, k, is maximal. This is due to the fact that, without knowing the distribution of inter-arrival times, the empirical entropy from the inter-arrival times may be upper-bounded by $L \cdot \log_2 k$, where $\log_2 k$ is the number of bits encoded by a single lookup.

As a consequence, to assess the upper-bound on the information content for a fixed L and an assumed time-slot size (expressed as time resolution $\Re$), we need to determine into how many distinct inter-arrival times k we can partition one day into, while imposing as uniform a distribution of inter-arrival times as possible (i.e., leading to maximal entropy).

By maximizing k subject to the constraint that the distribution of distinct inter-arrival times is uniform (omitting details for brevity), and upper-bounding k by $L-1$ (the number of intervals), we find that we can express the upper bound on the information amount in the timing vector as:

$$I_{\text{time}} = L \cdot \log_2 \left(\min \left(L-1, \left\lfloor \frac{2M}{L-1} \right\rfloor + 1 \right) \right)$$

where $M = \frac{86{,}400}{\Re}$ denotes the number of time slots with resolution $\Re$ over one day (86,400 seconds).

Unified fast filter. From the above equations, we can now formulate the following unified test condition to handle all types of information vectors:

> If $I_{\text{name}} + I_{\text{type}} + I_{\text{time}} < \Im$, the suffix is not a candidate tunnel.

We then eliminate from further detailed analysis the name suffixes that are not candidate tunnels.

Choosing the thresholds. The fast filter relies on two parameters, the information content threshold $\Im$ and the time resolution $\Re$. In order to select security-relevant values for these parameters, we measured their impact on the analyst's workload. (Note that in § 5.3 we also framed empirical evidence that $\Re = 10$ msec appears fairly conservative.) It is clear that both reducing the information content threshold and reducing the time resolution can increase the false positive rate, and relatedly the analyst's workload.

Figure 6 shows how varying these parameters affects the analyst for INDLAB data. One can see, for example, that decreasing the information content threshold $\Im$ from 4,096 to 256 bytes (and potentially increasing security) would increase the number of domain name suffixes passed to the analyst for manual inspection 50-fold. The plot also shows a clear power-law relationship between analyst workload and $\Im$, with the former scaling as approximately $x^{-1.38}$ in the latter.

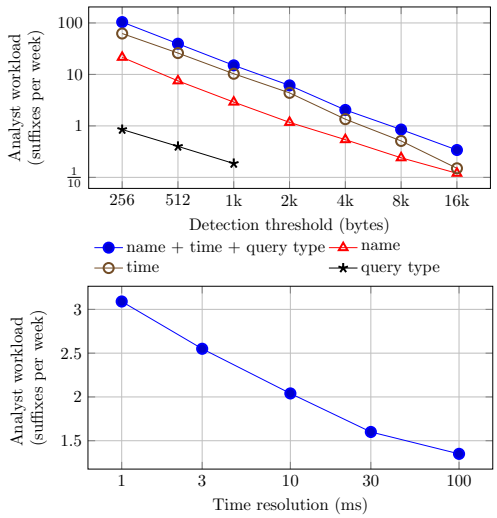

Figure 6: The impact of the information content threshold $\Im$ and the time resolution $\Re$ on the number of suffixes to validate manually per week for the INDLAB dataset. The top chart reflects a value $\Re = 10$ msec, and the bottom chart $\Im = 4,096$ bytes.

Setting the information content threshold $\Im$ to 4,096 bytes and the time resolution $\Re$ to 10 ms thus provides a good balance between analyst workload and potential detections. Sites might of course revisit these parameters based on their particular threat models and networking environments.

6.5 Bounding Information Content

For each (suffix, client)-pair that remains after the preceding filter steps, we compute the size of *gzip*, *bzip2* and *ppmd* [23] compression for the series of all corresponding lookup names, selecting the lowest value. We also assess a codepoint-oriented analysis (§ 5.3), computing the *gzip*, *bzip2* and *ppmd* compression sizes for the series of distinct (unique) lookup names, selecting the lowest value, and adding the lowest value of the *gzip*, *bzip2* and *ppmd* compression sizes for the corresponding distinct lookup name indices. Given these two assessments, we choose the smaller as the best (tightest) upper bound on the amount of information potentially transferred through lookup names to the given domain suffix (cf. box "Bound on Information Content" in Figure 5).

Next, we apply the same procedure to the corresponding inter-query arrival times (in $\Re$ = 10 msec units) and query record types, if this information is available. Fi-

nally, we add up the results from the lookup name, time and type information vectors, and if their sum lies below $\mathfrak{I}$, we discard the (suffix, client)-pair.

6.6 Inspected Domain List

We expect sites to employ our analysis procedure over an extended period of time. For example, once a site sets it up, it might run as a daily batch job to process the last 24 hours of lookups. An analyst inspects the traffic associated with any domains flagged by the procedure and renders a decision regarding whether the activity appears benign or malicious.

An important observation is that the same benign domains will often reappear day after day, due to the basic nature of their lookups. However, the analyst needn't reexamine such domains, as the verdict will prove the same. (See § 9 for further discussion of this point.) Given this, we presume the use of an *Inspected Domain List* (IDL) that accumulates previous decisions regarding domains over time. For a given day's detections, we omit flagging for the analyst any that already appear on the IDL. Once populated, such a dynamic list can greatly reduce the ongoing burden that our detection procedure places on a site's analysts.

A final issue regarding the IDL concerns its granularity. For example, if our procedure flags s1.v4.ipv6-exp.l.google.com and we put that precise domain on the IDL, then this will not spare the analyst from having to subsequently investigate i2.v4.ipv6-exp.l.google.com.[4] However we note that the analyst's decision process will focus heavily on *registered domains*. In this example, the analyst will likely quickly decide to mark the detection as benign because for it to represent an actual problem would require subversion of some of Google's name servers, which would represent an event likely significantly more serious than an attacker communicating surreptitiously out of the site. In addition, the analyst will reach this conclusion simply by inspecting the registered domain google.com, rather than studying all of the subdomains in depth.

Accordingly, once an analyst inspects a detection, we place on the IDL the corresponding registered domain, which we compute by consulting Mozilla's *Effective TLD Names* list [20]. In this example, com appears on the list (meaning that any domain directly under it will reflect a registration), so we add google.com to the IDL. Any subsequent matching against the IDL likewise employs trimming of names using the same procedure.

We note that we could implement the IDL with finer granularity than described above. In particular, we could frame it in terms of per-client filtering, or using custom entropy thresholds. We leave exploring these refinements for future work.

[4] Both of these are actual detections.

Exfiltration Scenario	*Estimated Data Volume*			
	Total	Name	Timing	Type
Query name-content	111%	110%	0.4%	0.01%
Query name-codebook	109%	103%	5.6%	0.1%
Timing	105%	0.8%	104%	0.2%
Query type	111%	0.6%	6.8%	104%

Table 2: Estimates of data volumes produced by our procedure measured against specific exfiltration scenarios, showing the total estimate, and the individual contributions from the query name, timing, and type information estimation.

7 Evaluation

In this section we evaluate the efficacy of our detection procedure in terms of assuring that it can detect explicit instances of communication tunneled over DNS (§ 7.1) and investigating its performance on data from production networks (§ 7.2). For this latter, we assess both the procedure's ability to find actual surreptitious communication, and, just as importantly, what sort of burden it imposes on security analysts due to the events generated.

7.1 Validating on Synthetic Data

To validate our procedure's ability to accurately measure communication embedded in DNS queries, we assessed what sort of estimates it produces for scenarios under which we fully control the DNS communication used for exfiltration. Table 2 summarizes the results, comparing the information vector used for exfiltration vs. the estimates of the volume of data present in the corresponding lookups, both in total and when restricted to just considering a single information vector. All values are percentages of the actual exfiltration size, so a value of 105 indicates an estimate that was 105% of the true size (i.e., the estimate was 5% too high). Naturally, estimators that focus on information vectors different from those used in a given exfiltration scenario can greatly underestimate the data volume if used in isolation, highlighting the need to combine such estimators into a final comprehensive sum.

Regarding the scenarios reflected in the table, to assess tunnels based on encoding information directly in query names, we recorded Iodine [10] queries while sending a 99,438-byte compressed file with *scp*. The 11 % difference (shown in the "Query name-content" row) between measured content and actual content is nearly all due to tunnel encapsulation overhead (SSH, TCP/IP headers, Iodine framing). As we are not aware of any available tunneling tools that leverage repeated (codebook-style) queries, timing, or varying query types, we wrote simple proof-of-principle implementations for testing purposes. The codebook-style implementation used 16 distinct names that each convey four data bits per query,

Type Of Activity \ Dataset	IndLab	LBL	NERSC	UCB	CHINA	SIE	SIE^{UNIQ}
Lookups (days)	57B (1,212)	73B (2,565)	12B (1,642)	1.7B (45)	69M (5)	77B (53)	12B (53)
Detection threshold	4kB	4kB	4kB	10kB	10kB	10kB	10kB
Confirmed DNS channel	0	**2**	0	0	0	**57**	**57**
Benign use	286	306	29	200	41	4,815	1,088
Malware	2	2	0	5	2	74	73
Misconfiguration	49	62	5	126	8	310	182
IPv4 PTR	11	29	4	26	3	N/A	N/A
IPv6 PTR	0	5	0	1	0	N/A	N/A
Unknown	14	27	0	13	13	1	1
Total	**362**	**433**	**38**	**371**	**67**	**5,256**	**1,401**
Domains flagged (first week)	16	5	3	199	(67+)	3,002	798
Domains flagged (typical week)	2.0	1.1	0.15	32	N/A	358	97

Table 3: Number of domains flagged in each dataset, broken out by the type of activity that the use of the domain represents. The IndLab, UCB and CHINA analyses cover all information vectors: LBL and NERSC incorporate query names and types, but not timing; SIE considers only query names; and SIE^{UNIQ} only the contents of query names (not repetitions). SIE and SIE^{UNIQ} analyses includes additional considerations discussed in the Appendix.

while the timing-interval implementation used one name and 16 distinct time intervals spaced 10 ms apart. The query-type implementation used one name and 16 distinct query types. In addition, these tunnels used five distinct query names for command and control. We exfiltrated a 10,000-byte compressed file and found that the difference between the estimated exfiltration volume and the actual size ranged from 5–11 %.

These results confirm that our procedure can readily detect information that is encoded into query names, timing, or query record types, and that it can provide meaningful upper bounds.

7.2 Evaluation on Operational Data

We now turn to evaluating our detection procedure as applied to the extensive datasets we gathered, comprising 230 billion lookups from the networks listed in Table 1.

A key question for whether our detector is operationally viable concerns the combination of (1) how many domains it flags for analysis, coupled with (2) how quickly an analyst can identify the common case of a flagged domain not in fact posing a threat.

The filtering steps in § 6 aim to address the first issue. Regarding the second issue, as we briefly discussed in § 6.6 we find that often analysts can rely on *fate-sharing* to quickly determine they needn't further investigate a candidate domain. For example, a site's analyst can reason that a detection of google.com or mcafee.com is safe to ignore, because if indeed an attacker has control over those domains' name servers, the site has (much) bigger problems than simply the presence of surreptitious communication to the sites.

Table 3 summarizes the findings across all of the datasets. For each dataset, the row in bold gives the total number of different domains flagged by our detector (many appear in more than one day), and the bottom row reflects the "steady state" burden on an analyst investigating detections for the given environment. We partition the datasets into two groups. The logs for IndLab, LBL and NERSC include individual per-client lookups, and thus these sites represent the sort of environments for which we target our detection, using a threshold of 4 kB/day. The lookups recorded for UCB, CHINA and SIE, on the other hand, are primarily aggregated across many clients, and thus for these datasets we cannot perform per-client analysis. We do not aim to treat these datasets as operational environments for our detection procedure, but rather to assess what sort of surreptitious communication the procedure *can* detect in real traffic. For them, we use a higher threshold of 10 kB/day to limit our own analysis burden in assessing the resulting detections. Finally, the SIE dataset introduces some additional complexities, as discussed in Appendix B.

We classified the detections based on manual analysis to assign each to one of six general categories, as follows.

Confirmed DNS channel reflects domains for which we could amass strong evidence that indeed the detection represents surreptitious communication over DNS. For LBL, both flagged domains correspond to tunnels that staff members acknowledge having set up to obtain free Internet access in WiFi hotspots that allow out DNS traffic without requiring payment. One used DNStunnel [8], the other NSTX [13].

For SIE, we identified 3 types of tunnels. One type (responsible for 42 domains) corresponds to a product offered by Dynamic Internet Technology, a company that builds tools to evade censorship [9]. These tunnels encode most requests in two 31-character labels, using only

alphanumerics, followed by an identifier that appears to identify the tunnel itself. Another 10 domains all have whois information leading to MMC Networks Limited (of Gibraltar), a company that provides a program offering "Free WiFi" using tunneling [28]. The tunneling technology used for these is a variant of Iodine, with the main difference being use of only alphanumeric characters for the encoding. We also found 5 domains that use Iodine, for reasons we have not been able to identify.

Finally, we examined an addition 150 billion DNS records captured in a separate 259 days of monitoring from SIE. Due to monitoring gaps, this expanded data is unsuitable for analyzing long-term analyst burden. But in it (using a somewhat higher detection threshold) we detected 42 new tunnel instances, including a new tunnel type belonging to vpnoverdns.com.

Benign use encompasses a number of different scenarios that we believe would lead an analyst to fairly quickly decide that the corresponding activity does not appear problematic. These scenarios include flagging of: (1) a well-known site (e.g., google.com), for which a name server breach would reflect a catastrophe, so very likely has not occurred (*fate-sharing*). (2) A sister site (e.g., a partner institute), where a similar argument holds. (3) ISPs, for which sometimes local systems look up many hostnames corresponding to end-user systems. For example, in LBL we observe queries for numerous names such as 201-11-50-242.mganm703.dsl.brasiltelecom.net.br. (4) Directory-style services offered over DNS, including blocklists, user-generated content, and catalogs. (5) Software license servers. (6) Cloud-based antivirus services.

Malware indicates lookups associated with malware activity or sites flagged (for example, by McAfee's *SiteAdvisor* service) as malicious. For SIE these also include lookups such as p9b-8-na-5w-2z3-djmu-...-njx-2es.info, i.e., 62-character labels consisting of letters or numbers separated by dashes. We concluded that these lookups reflect malware activity because names following the same pattern appeared in a trace generated by a researcher running bots within a contained environment.

Misconfiguration generally reflect clients making large volumes of lookups due to configuration problems that lead to repeated failures. For example, in one LBL instance we observed more than 60,000 lookups of 33 different names within a single domain, such as _ldap._tcp.standardname-...isi.fhg.de. Other problems we observed include lookups apparently based on email addresses, such as itunes@new-music.itunes.com; subdomains appearing to be IP addresses; repeated failures of names with narrow, rigid structures; and domains in search paths that have lookups encapsulating a client's entire stream of queries sent to other domains.

IPv4 PTR and **IPv6 PTR** reflect lookups under the in-addr.arpa and ip6.arpa zones, respectively. These zones provide a decentralized mapping from numeric IP addresses to domain names. As discussed in § 6.2, we do not flag PTR lookup suffixes that correspond to address ranges that are local to the organization, or that are reserved. As noted in § 6.3, for IPv4 PTR lookups we only flag suffixes corresponding to /16 or /24 netblocks, and for IPv6, /48 netblocks.

Unknown reflects domains for which we could not arrive via manual analysis at a confident determination regarding how to classify the activity. For example, one striking instance concerns a number of domains (primarily seen in CHINA traffic, but also SIE) that issue thousands of lookups such as:

```
wojnlbefrhpfumrupmsn.0ule365.net
jnrlciinsszxahnfrvxe.0ule365.net
okgjeqckeqrxdigktkua.0ule365.net
```

Here, the domain (0ule365.net) is associated with a Chinese gaming site. Other instances following the same pattern appear to be associated with *phishing* sites related to such gaming sites.

Domains flagged in first week and **in typical week** reflect the two extreme behaviors of our *Inspected Domain List* approach (§ 6.6). In the first week of operation our detector reports a peak number of domains; once the list is primed, it flags domains at a much lower rate. (We special-case the figure for CHINA because that entire dataset spans less than a week.)

Finally, the main conclusion we highlight regarding the **Total** row is the low number of events that analysts would have to inspect. (Even for SIE, the average load aggregated across the more than 100 participating sites comes to about 50 detections per day, given $\mathfrak{I}$ increased from 4 kB to 10 kB.)

8 Real-Time Operation

As developed so far, our analysis procedure operates in an offline fashion, processing full days as a single unit. While this suffices to enable analysts to detect DNS exfiltration on a daily basis, real-time detection would enable immediate identification of such activity and thus much quicker response. In this section we explore the viability of adapting our scheme for such detection.

Our real-time variant uses *gzip* and *bzip2* as the compression functions. We can adapt both the cached query filter and the "uninteresting query" filter to streaming operation, with the only consideration being that we modify the cached query filter to actively flush all cache entries as their TTLs expire to minimize statekeeping.

Adapting the fast filter and the compression-based filters takes more consideration, since they naturally process entire sets of activity as a unit. In addition, if we try to use a compressor in a stream fashion, we must deal with the compressor's destructive operation: if we add

data to a stream and call `flush()` to obtain the size of the compressed result, the `flush()` operation changes the compressor's internal state—adding more data and calling `flush()` again can produce a larger output than simply compressing all of the data at once.

Our approach combines the fast filter and the compression measurement for each (domain, client) pair as follows. Initially, for each pair we only track the uncompressed input. Upon receiving new input, we check whether the total message length plus maximum possible entropy contribution from the timing, and query, and query type could possibly lead to the pair generating an alert. If not, we simply append the new information to the list of previously seen queries.

If the total could cross our threshold, we allocate compressors, feed them all of the recorded input, and invoke `flush()`. If the resulting entropy lies below the alert threshold, we simply update the uncompressed data threshold that could possibly generate an alert, discard the compressed data, and continue. Otherwise, we generate an alert, create new compressors, feed them all the previous data, and pass all subsequent data to them as it arrives. These new compressors allow us to compute a full 24-hour entropy total for the (domain, client) pair to aid the analyst. After 24 hours we generate a summary for each pair and discard the associated state.

For good performance we parallelized this approach, running the cached-query and uninteresting-query filters in a single process that dispatches each (suffix, client) pair to one of 15 distinct child processes. We verified that the implementation produces a consistent analysis by processing the same day of IndLab data using both the original batch implementation (with only *gzip* and *bzip2*) and the real-time variant (70M DNS queries, 36M non-empty replies). They fully agreed, with the real-time implementation requiring 28 minutes and 4.5GB of RAM to process the day of traffic. The execution totaled 53 CPU-core-minutes on a dual processor Intel Xeon X5570 system. Given these results, we conclude that real-time operation is quite viable.

9 Discussion

This paper demonstrates how we can comprehensively measure the information content of an outbound DNS query stream. Our lossless compression-based procedure measures all information that an attacker can effectively send via names, types, and timing, regardless of the actual encoding used. This procedure also has only two tuning parameters, the threshold of detection and the timing precision.

Some minor DNS features remain that we have not included in our analysis procedure. We have omitted these for simplicity, since in their usual (benign) use, they appear almost always to have a single value for a given client. These information vectors include requesting DNSSEC information (single bit) and the query's *class* (which for modern traffic is almost always type `IN`, "Internet"). Similarly, future EDNS0 extensions could appear that recursive resolvers will forward intact, providing a new information vector. For all such features, we can simply employ an additional compressor optimized with the use of a very low-cost special case of using a single bit to indicate that for a given client, the feature never changes.

Attackers can tunnel information in DNS replies as well as in queries, and indeed existing tunnels do so. Since replies can include domain names (returned for example in `CNAME` records) or unstructured byte strings (e.g., `TXT` records), replies can potentially convey large volumes of data. (We remind the reader that in this work we have focused on analyzing DNS queries rather than responses since for the scenarios of particular interest—exfiltration or remote interactive access—the query streams will generally carry the bulk of the data.)

Attackers who can successfully mimic the appearance of benign data-rich query streams (such as blocklist lookup services) can trick analysts into deeming their surreptitious communication as harmless. Similarly, an attacker who compromises a previously benign domain can encode their traffic using the same style of lookups as the domain originally used. These problems are orthogonal to the question of *flagging* the activity.

Attackers aware of our detection procedure can in addition design their tunnels to keep the information content below the 4 kB per day threshold. Given that we aggregate information content metrics per domain, a simple evasion strategy would be to spread the traffic across $K > 1$ domains, and then send < 4 kB per day to each, but in aggregate communicate K times that volume. A possible detection approach we envision pursuing consists of analyzing each client's lookups in their entirety, rather than on a per-destination-domain basis. Coupled with an expanded Inspected Domain List (§ 6.6) to remove the major contributors to DNS traffic, we would aim with this approach to compute a bound on the total information content each client communicates via all of its external DNS queries.

Finally, attackers could spread their exfiltration across multiple compromised clients, so that each client's query stream remains below the detection threshold. Our experiences with external vantage points such as UCB indicates that we still might be able to find the activity of groups of clients, since that vantage point already aggregates multiple clients into a single apparent source. However, a combination of using multiple compromised clients *and* K external name servers might prove exceedingly difficult to detect for the sort of thresholds we have employed in this work.

10 Related Work

Four areas of prior work have particular relevance to our study: covert communication; designing ways of tunneling communication over DNS traffic; detecting such tunneling; and establishing bounds on the volume of covert communication.

We adopt Moskowitz and Kang's classification of covert communication channels [19]. In particular, a storage channel is a covert channel where the output alphabet consists of different responses all taking the same time to be transmitted, and a timing channel is a covert channel where the output alphabet is made up of different time values corresponding to the same response. Accordingly, we treat covert communication via DNS query content (name, type and other attributes) as a storage channel, and covert communication via query timing as a timing channel.

Conventional DNS tunnels are similar in construction: they are bi-directional, directly embedding the outbound information flow in query names, and the inbound flow in server responses. In the absence of outbound data, the client sends low-frequency queries to poll the tunnel server for any pending data. The functionality of these tunnels ranges from a simple client-to-server virtual circuit to full IP-level connectivity. Examples are NSTX [13], dns2tcp [7], Iodine [10], OzymanDNS [15], tcp-over-dns [25], and Heyoka [14]. DNS exfiltration has also been a tool in the attacker's toolbox for a number of years (per [22] and the references therein).

Beyond query names, the DNS message format contains a variety of fields that could be used for embedding data (as we detail in § 5.1). In addition to the DNS-specific message fields, timing (e.g., the timing of queries) provides a rich vector for embedding data. This is not unique to DNS traffic, but present in all Internet traffic, allowing any message to be encoded in the inter-arrival times between packets. Gianvecchio et al. [12] showed how to automatically construct timing channels that mimic the statistical properties of legitimate network traffic to evade detection. Our detection technique avoids such complication by measuring information content rather than particular statistical properties.

One approach for detecting covert communication over DNS examines the statistical properties of DNS traffic streams. Karasaridis et al. propose DNS tunnel detection by computing hourly the Kullback-Leibler distance between baseline and observed DNS packet-size distributions [16]. To defeat such temporal statistical anomaly detectors, Butler et al. propose stealthy half-duplex and full-duplex DNS tunneling schemes [5]. They also propose the use of Jensen-Shannon divergence of per-host byte distributions of DNS payloads to detect tunneled traffic. Their detection technique only flags whether the aggregate traffic contains tunneled communication; it does not identify the potential tunneled domains. In addition, the detection rate depends to a large extent on the ratio of tunneled traffic to normal traffic. In [3], the authors show that domain names in legitimate DNS queries have 1-, 2-, and 3-gram fingerprints following Zipf distributions, which distinguishes them from the higher-entropy names used in DNS tunneling. The evaluations in these works do not particularly address practicality for operational use, however, since the authors validate their hypotheses on short, low-volume benign and synthetic tunneled traces collected using free DNS tunneling tools. As we discuss in § 5.2, large-scale DNS traffic often exhibits extensive diversity in multiple dimensions, which likely will exacerbate issues of false positives.

Our work overlaps with work on algorithmically-generated domain names by Yadav et al. [29]. The most salient difference is that their algorithm assumes a specific model of name construction (distributions of letters and bigrams). Instead of focusing on specific name patterns and missing communication that uses different encodings, we measure the aggregate information content of a query stream regardless of how encodings are generated for the query name, type or timing.

Detection of timing channels has been studied before, and we mention here only a few recent results. Cabuk et al. [6] observe that timing-based tunnels often introduce artificial regularity in packet inter-arrival times and present detection methods based on this characteristic. More generally, Gianvecchio and Wang [11] identify timing-based tunnels in general Internet traffic (not just DNS) by using conditional entropy measures to identify the subtle distortions introduced by the tunnel in packet inter-arrival time distributions. These works use time intervals of 20 msec or more; we use a more conservative 10 msec timing resolution, and do not assume the presence of detectable distortions.

While the general problem of surreptitious communication has received extensive examination in the literature of covert channels and steganography, more closely related to our work is previous research on bounding the volume of surreptitious communication in other protocols. Borders et al. studied this problem for HTTP, observing that covert communication is constrained to the user-generated part of an outgoing request [1, 2]. By removing fixed protocol data and data derived from inbound communication, the authors show how to determine a close approximation to the true volume of information flows in HTTP requests. An analogous approach for our problem domain would be to track the domain names a system receives from remote sources (such as web pages and incoming email), and to exclude lookups for these names as potentially conveying information. Such tracking, however, appears infeasible without requiring extensive per-system monitoring.

11 Summary

We have presented a comprehensive procedure to detect stealthy communication that an adversary transmits via DNS queries. We root our detection in establishing principled bounds on the information content of entire query streams. Our approach combines careful encoding and filtering stages with the use of lossless compression, which provides guarantees that we never underestimate information content regardless of the specific encoding(s) an attacker employs.

We demonstrated that our procedure detects conventional tunnels that encode information in query names, as well as previously unexplored tunnels that repeatedly query names from a fixed alphabet, vary query types, or embed information in query timing. We applied our detection procedure to 230 billion lookups from a range of production networks and addressed numerous challenges posed by anomalous-yet-benign DNS query traffic. In our assessment we found that for datasets with lookups by individual clients and a threshold of detecting 4 kB/day of exfiltrated data per client and domain, the procedure typically flags about 1–2 events per week for enterprise sites. For a bound of 10 kB, it typically flags 50 per day for extremely aggregated logs at the scale of a national ISP. In addition, buried within this vast number of lookups our procedure found 59 confirmed tunnels used for surreptitious communication.

Acknowledgments

Our thanks to Partha Bannerjee, Scott Campbell, Haixin Duan, Robin Sommer, and James Welcher for facilitating some of the data and processing required for this work. Our thanks too to Christian Rossow and the anonymous reviewers for their valuable comments.

This work would not have been possible without the support of IBM's *Open Collaboration Research* awards program. In addition, elements of this work were supported by the U.S. Army Research Office under MURI grant W911NF-09-1-0553, and by the National Science Foundation under grants 1161799, 1223717, and 1237265. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] Borders, K., and Prakash, A. Towards Quantification of Network-Based Information Leaks via HTTP. In *Proceedings of the 3rd USENIX Workshop on Hot Topics in Security* (2008), USENIX Association.

[2] Borders, K., and Prakash, A. Quantifying Information Leaks in Outbound Web Traffic. In *Proceedings of the IEEE Symposium on Security and Privacy* (2009), USENIX Association.

[3] Born, K., and Gustafson, D. Detecting DNS Tunnels Using Character Frequency Analysis. In *Proceedings of the 9th Annual Security Conference* (2010).

[4] Bromberger, S. DNS as a Covert Channel Within Protected Networks. http://energy.gov/sites/prod/files/oeprod/DocumentsandMedia/DNS_Exfiltration_2011-01-01_v1.1.pdf, 2011.

[5] Butler, P., Xu, K., and Yao, D. Quantitatively analyzing stealthy communication channels. In *Proceedings of International Conference on Applied Cryptography and Network Security* (2011).

[6] Cabuk, S., Brodley, C. E., and Shields, C. Ip covert timing channels: design and detection. In *Proceedings of the 11th ACM conference on Computer and communications security* (New York, NY, USA, 2004), CCS '04, ACM, pp. 178–187.

[7] Dembour, O. DNS2tcp. http://www.hsc.fr/ressources/outils/dns2tcp/index.html.en.

[8] DNStunnel. http://www.dnstunnel.de/.

[9] Dynamic Internet Technology. http://www.dit-inc.us/.

[10] Ekman, E., and Andersson, B. Iodine, tunnel IPv4 over DNS. http://code.kryo.se/iodine/, 2011.

[11] Gianvecchio, S., and Wang, H. An entropy-based approach to detecting covert timing channels. *Dependable and Secure Computing, IEEE Transactions on 8*, 6 (Nov/Dec. 2011), 785–797.

[12] Gianvecchio, S., Wang, H., Wijesekera, D., and Jajodia, S. Model-based covert timing channels: Automated modeling and evasion. In *Proceedings of the 11th international symposium on Recent Advances in Intrusion Detection* (Berlin, Heidelberg, 2008), RAID '08, Springer-Verlag, pp. 211–230.

[13] Gil, T. NSTX (IP-over-DNS). http://thomer.com/howtos/nstx.html.

[14] Heyoka. http://heyoka.sourceforge.net/.

[15] Kaminsky, D. OzyManDNS.

[16] Karasaridis, A., Meier-Hellstern, K., and Hoeflin, D. Detection of DNS anomalies using flow data analysis. In *Global Telecommunications Conference (GLOBECOM)* (2006).

[17] Kreibich, C., Weaver, N., Nechaev, B., and Paxson, V. Netalyzr: Illuminating the edge network. In *Proceedings of the ACM Internet Measurement Conference (IMC)* (Melbourne, Australia, November 2010), pp. 246–259.

[18] Mockapetris, P. Domain names—implementation and specification. RFC 1035, Internet Engineering Task Force, Nov. 1987.

[19] Moskowitz, I. S., and Kang, M. H. Covert channels - here to stay? In *Proceedings of the Ninth Annual Conference on Computer Assurance* (1994), pp. 235–244.

[20] Mozilla. Public Suffix List. Published online at http://publicsuffix.org/. Last accessed on May 4, 2012.

[21] PAXSON, V. Empirically-Derived Analytic Models of Wide-Area TCP Connections. *IEEE/ACM Transactions on Networking 2*, 4 (Aug. 1994), 316–336.

[22] RICKS, B. DNS Data Exfiltrationa Using SQL Injection. http://www.defcon.org/images/defcon-16/dc16-presentations/defcon-16-ricks.pdf, 2008.

[23] SHKARIN, D. PPMd. http://www.compression.ru/ds/ppmdj1.rar, 2006.

[24] Security Information Exchange. http://sie.isc.org/.

[25] tcp-over-dns. http://analogbit.com/software/tcp-over-dns.

[26] VIXIE, P. Extension Mechanisms for DNS (EDNS0). RFC 2671 (Proposed Standard), Aug. 1999.

[27] VIXIE, P., AND DAGON, D. Use of Bit 0x20 in DNS Labels to Improve Transaction Identity. Work in progress, Internet Engineering Task Force, 2008.

[28] Wi-Free. http://wi-free.com/.

[29] YADAV, S., REDDY, A. K. K., REDDY, A. N., AND RANJAN, S. Detecting algorithmically generated malicious domain names. In *Proceedings of the 10th annual conference on Internet measurement* (2010), IMC '10, ACM, pp. 48–61.

A Full Names for Examples

For completeness, Figure 7 lists the full names of various DNS lookups that in the main body of the text we elided portions for readability. Note that for some names we introduced minor changes for privacy considerations.

B Issues Evaluating the SIE Dataset

The SIE data's extreme volume and qualitatively different nature necessitated several changes to our analysis procedure. Our access to the data was via a Hadoop cluster, requiring coding of our algorithms in the Pig and Scala languages. These provide efficient support for only a subset of the functionality we employed when analyzing the other datasets. A significant difference in this regard was that we were confined to only using *gzip* for compression; *bzip2* and *ppmd* were not available.

Another important difference concerns the definition of "client". A single large American ISP dominates the SIE data, representing roughly 90% of the traffic. This ISP uses clusters of resolvers to process requests. Thus, a single abstract resolver manifests as multiple "client IP addresses", which we determined come from the same /28 address prefix. Therefore we treat query source IP addresses equivalent in their top 28 bits as constituting a single source.

This extreme aggregation leads to significant increases in detections, as we are now measuring the information volume for queries aggregated across potentially hundreds of thousands of clients. One particular increase in benign alerts arises due to popular names with short TTLs (e.g., www.google.com). With so many clients, every popular name becomes immediately refetched whenever its TTL expires, leading to a steady stream of closely-spaced lookups. This very high level of aggregation also generates such a large volume of detections for

```
5.1o19sr00ors95qo0p73415p3r8r8q777634r5o86osn295ss2rqos
 s3r9601ro3.1r1p7r4719o34393648s2345nn60qnqoop45psos37n
 551s002n80850sr2r8n3.r1105qqq28r7pn82843rp76383qr6344q
 qpq7rpnrp63o957687r980r.rrqs656p04pn614q6n76o97883op73
 r0p787rn92.i.02.s.sophosxl.net
g63uar2ejiq5tlrkg3zezf2fksjrxpxyvro4ce5yz65udnjn.dagbuu
 5pkocwcaxkntmxzwvkbulhg3qlj6ho7jwobeddjqvv.gepxfdwfhu7
 6on6gza2nkringxp35e6g3ftpqlpl5h6uofgo.kukjy4jvybu7jhrl
 hrgxe7es3lmkxdrpmpb4lg7wmbpygjg7.gef2uoemc6pi88tz.er.s
 potify.com
awyvrvcataaaegdid5tmr7eteje2kst35frnnr3kupbfc6hr.gq3dey
 4qnjvqtoltoj2dq5bxnmaauaaeaiaaeg7xa4ut3ilu.license.cra
 shplan.com
www.10.1.2.3.static.because.dul.is.rfc.ignorant.edu.za.
 static.because.dul.is.rfc.ignorant.edu.za.research.edu
JohnsonHouse\032Officejet\032J6400\032\032The\032Johnso
 n\032MacBook._ipp._tcp.johnsonhouse1.members.mac.com
```

(a) Example DNS names with more than 100 bytes in length (cf. § 5.2).

```
1751913.86c0ade0d13143ab83d7e4f60cbd204c.00000000.xello
 .xobni.com
1753942.86c0ade0d13143ab83d7e4f60cbd204c.00000000.xello
 .xobni.com
1756950.86c0ade0d13143ab83d7e4f60cbd204c.00000000.xello
 .xobni.com
1758762.86c0ade0d13143ab83d7e4f60cbd204c.00000000.xello
 .xobni.com
```

(b) Example DNS names with little variation between consecutive queries.

```
p9b-8-na-5w-2z3-djmu-7pk-qy-0-bok-re9-ym-v9h-av-njx-2es
 .info
```

(c) Example DNS name reflecting malware activity (cf. § 7.2).

```
_ldap._tcp.standardname-des-ersten-standorts._sites.dc.
 _msdcs.isi26.isi.fhg.de
```

(d) Example DNS name originating from client misconfiguration (cf. § 7.2).

Figure 7: Full names of examples used in the main text. We line-break each name at 54/55 characters.

reverse lookups that we excluded them from the SIE analysis, which removes about 10% of the queries.

As previously discussed in § 4, we emphasize that the role of the SIE dataset for our evaluation is simply to give us a (huge) target environment in which to validate that we can find actual tunnels. We do not envision our procedure as operationally viable for this environment; nor does such an environment strike us as making sense in terms of conforming with our threat model, which focuses on tightly controlled enterprises, rather than wide-open ISPs.

Given this perspective, to keep our own manual analysis tractable, for SIE we used a detection threshold $\mathfrak{I}$ of 10 kB rather than the 4 kB value we use for the other datasets.

We also explored the effects of other analysis changes. First, we investigated conducting our analysis on the SIE queries reduced to distinct, sorted names. This transformation removes our opportunity of assessing *query name-codebook* information vectors, but preserves our ability to estimate data conveyed through the *query name-content* vector—the only type of encoding employed by known DNS tunneling tools. Table 2 shows this version of the SIE data as SIE^{UNIQ}. The reduction in analyst load is quite significant, more than a factor of three.